
AWS Elastic Beanstalk

Developer Guide



AWS Elastic Beanstalk: Developer Guide

Copyright © 2020 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is AWS Elastic Beanstalk?	1
Pricing	1
Where to go next	1
Getting started	3
Setting up: Create an AWS account	3
Step 1: Create	3
Create an application and an environment	3
AWS resources created for the example application	5
Step 2: Explore	5
Step 3: Deploy a new version	7
Step 4: Configure	8
Make a configuration change	9
Verify the configuration change	9
Step 5: Clean up	10
Next steps	11
Concepts	13
Application	13
Application version	13
Environment	13
Environment tier	13
Environment configuration	13
Saved configuration	14
Platform	14
Web server environments	14
Worker environments	15
Design considerations	16
Scalability	16
Security	17
Persistent storage	17
Fault tolerance	18
Content delivery	18
Software updates and patching	18
Connectivity	18
Permissions	20
Service role	20
Instance profile	21
User policy	23
Platforms	25
Platforms glossary	25
Shared responsibility model	27
Platform support policy	28
Retired platform branch schedule	28
Retired platform branches	29
Supported platforms	29
Supported platform versions	30
Linux platforms	30
Linux platform versions	31
List of Elastic Beanstalk Linux platforms	31
Extending Linux platforms	31
Custom platforms	38
Creating a custom platform	39
Using a sample custom platform	39
Platform definition archive contents	43
Custom platform hooks	44

Platform scripts	45
Packer instance cleanup	46
Platform.yaml format	46
Tagging custom platform versions	48
Working with Docker	50
Docker platforms	50
Single Container Docker	51
Multicontainer Docker	58
Preconfigured containers	71
Environment configuration	74
Running containers locally	79
Working with Go	83
Getting started	84
Development environment	86
The Go platform	86
Tutorial for Go	91
Working with Java	94
Getting started	95
Development environment	100
The Tomcat platform	101
The Java SE platform	112
Adding a database	118
Eclipse toolkit	123
Resources	137
Working with .NET	137
Getting started	138
Development environment	140
The .NET platform	141
Tutorial - ASP.NET MVC5	149
Tutorial - .NET core	155
Adding a database	163
The AWS Toolkit for Visual Studio	166
Migrating on-premises application	191
Resources	192
Working with Node.js	192
Getting started	192
Development environment	193
The Node.js platform	195
Tutorial - Express	203
Tutorial - Express with clustering	207
Tutorial - Node.js w/ DynamoDB	216
Adding a database	225
Resources	227
Working with PHP	227
Development environment	228
The PHP platform	230
Tutorial - Laravel	234
Tutorial - CakePHP	241
Tutorial - Symfony	249
Tutorial - HA production	253
Tutorial - HA WordPress	262
Tutorial - HA Drupal	273
Adding a database	285
Working with Python	288
Development environment	288
The Python platform	290
Tutorial - flask	295

Tutorial - Django	301
Adding a database	311
Resources	313
Working with Ruby	314
Development environment	314
The Ruby platform	316
Tutorial - rails	320
Tutorial - sinatra	326
Adding a database	329
Tutorials and samples	332
Managing applications	334
Application management console	336
Managing application versions	337
Version lifecycle	339
Tagging application versions	340
Create a source bundle	342
Creating a source bundle from the command line	343
Creating a source bundle with Git	343
Zipping files in Mac OS X Finder or Windows explorer	343
Creating a source bundle for a .NET application	347
Testing your source bundle	348
Tagging resources	349
Resources you can tag	349
Tagging applications	349
Managing environments	353
Environment management console	353
Environment overview	355
Environment actions	356
Configuration	358
Logs	358
Health	359
Monitoring	359
Alarms	360
Managed updates	361
Events	361
Tags	362
Creating environments	363
The create new environment wizard	365
Clone an environment	384
Terminate an environment	386
With the AWS CLI	387
With the API	388
Launch Now URL	391
Compose environments	395
Deployments	397
Choosing a deployment policy	398
Deploying a new application version	399
Redeploying a previous version	399
Other ways to deploy your application	400
Deployment options	400
Blue/Green deployments	405
Configuration changes	408
Rolling updates	409
Immutable updates	412
Platform updates	415
Method 1 – Update your environment's platform version	418
Method 2 – Perform a Blue/Green deployment	419

Managed updates	420
Upgrade a legacy environment	425
Upgrade to Amazon Linux 2	426
Cancel an update	432
Rebuild an environment	433
Rebuilding a running environment	433
Rebuilding a terminated environment	434
Environment types	435
Load-balancing, autoscaling environment	435
Single-instance environment	436
Changing environment type	436
Worker environments	437
The worker environment SQS daemon	439
Dead-letter queues	440
Periodic tasks	440
Use Amazon CloudWatch for automatic scaling in worker environment tiers	441
Configuring worker environments	441
Environment links	444
Configuring environments	446
Configuration using the console	447
Configuration overview page	447
Review changes page	450
Amazon EC2 instances	451
Configuring your environment's Amazon EC2 instances	452
The aws:autoscaling:launchconfiguration namespace	456
Auto Scaling group	457
Spot instance support	458
Auto Scaling group configuration using the Elastic Beanstalk console	460
Auto Scaling group configuration using the EB CLI	463
Configuration options	463
Triggers	464
Scheduled actions	466
Health check setting	469
Load balancer	470
Classic Load Balancer	471
Application Load Balancer	479
Network Load Balancer	496
Configuring access logs	505
Database	506
Adding an Amazon RDS DB instance to your environment	506
Connecting to the database	509
Configuring an integrated RDS DB Instance using the console	509
Configuring an integrated RDS DB Instance using configuration files	510
Security	510
Configuring your environment security	511
Environment security configuration namespaces	513
Tagging environments	513
Adding tags during environment creation	514
Managing tags of an existing environment	514
Software settings	516
Configure platform-specific settings	517
Configuring environment properties	518
Software setting namespaces	519
Accessing environment properties	521
Debugging	521
Log viewing	524
Notifications	526

Configuring notifications using the Elastic Beanstalk console	527
Configuring notifications using configuration options	528
Configuring permissions to send notifications	529
Amazon VPC	530
Configuring VPC settings in the Elastic Beanstalk console	530
The <code>aws:ec2:vpc</code> namespace	533
Domain name	534
Configuring environments (advanced)	536
Configuration options	536
Precedence	537
Recommended values	537
Before environment creation	539
During creation	543
After creation	547
General options	555
Platform specific options	592
Custom options	599
.Ebextensions	600
Option settings	601
Linux server	603
Windows server	615
Custom resources	622
Saved configurations	640
Tagging saved configurations	644
<code>env.yaml</code>	645
Custom image	647
Creating a custom AMI	648
Cleaning up a custom AMI	650
Static files	650
Configure static files using the console	650
Configure static files using configuration options	651
HTTPS	652
Create a certificate	653
Upload a certificate	655
Terminate at the load balancer	656
Terminate at the instance	658
End-to-end encryption	679
TCP Passthrough	682
Store keys securely	682
HTTP to HTTPS redirection	683
Monitoring an environment	685
Monitoring console	685
Overview	685
Monitoring graphs	686
Customizing the monitoring console	687
Basic health reporting	688
Health colors	688
Elastic Load Balancing health checks	689
Single instance and worker tier environment health checks	689
Additional checks	690
Amazon CloudWatch metrics	690
Enhanced health reporting and monitoring	691
The Elastic Beanstalk health agent	693
Factors in determining instance and environment health	694
Health check rule customization	696
Enhanced health roles	696
Enhanced health events	696

Enhanced health reporting behavior during updates, deployments, and scaling	697
Enable enhanced health	698
Health console	701
Health colors and statuses	706
Instance metrics	708
Enhanced health rules	710
CloudWatch	714
API users	719
Enhanced health log format	721
Notifications and troubleshooting	723
Manage alarms	725
View events	728
Monitor instances	730
View instance logs	732
Log location on Amazon EC2 instances	734
Log location in Amazon S3	735
Log rotation settings on Linux	735
Extending the default log task configuration	735
Streaming log files to Amazon CloudWatch Logs	737
Integrating AWS services	739
Architectural overview	739
CloudFront	740
Logging Elastic Beanstalk API calls with AWS CloudTrail	741
Elastic Beanstalk information in CloudTrail	741
Understanding Elastic Beanstalk log file entries	741
CloudWatch	742
CloudWatch Logs	743
Prerequisites to instance log streaming to CloudWatch Logs	745
How Elastic Beanstalk sets up CloudWatch Logs	746
Streaming instance logs to CloudWatch Logs	748
Troubleshooting CloudWatch Logs integration	750
Streaming environment health	750
AWS Config	752
Setting up AWS Config	753
Configuring AWS Config to record Elastic Beanstalk resources	753
Viewing Elastic Beanstalk configuration details in the AWS Config console	754
Evaluating Elastic Beanstalk resources using AWS Config rules	756
DynamoDB	756
ElastiCache	757
Amazon EFS	758
Configuration files	758
Encrypted file systems	759
Sample applications	759
Cleaning up file systems	759
IAM	759
Instance profiles	760
Service roles	764
Using service-linked roles	769
User policies	775
ARN format	781
Resources and conditions	782
Tag-based access control	807
Example managed policies	810
Example resource-specific policies	813
Amazon RDS	820
Amazon RDS in default VPC	820
Amazon RDS in EC2 classic	825

Connection string in Amazon S3	829
Cleaning up an external Amazon RDS instance	831
Amazon S3	831
Contents of the Elastic Beanstalk Amazon S3 bucket	831
Deleting objects in the Elastic Beanstalk Amazon S3 bucket	832
Deleting the Elastic Beanstalk Amazon S3 bucket	832
Amazon VPC	834
Public VPC	835
Public/private VPC	836
Private VPC	836
Bastion hosts	837
Amazon RDS	842
VPC endpoints	847
Configuring your development machine	849
Creating a project folder	849
Setting up source control	849
Configuring a remote repository	850
Installing the EB CLI	850
Installing the AWS CLI	850
EB CLI	852
Install the EB CLI	853
Install the EB CLI using setup scripts	853
Manual installation	853
Configure the EB CLI	860
Ignoring files using .ebignore	862
Using named profiles	862
Deploying an artifact instead of the project folder	863
Configuration settings and precedence	863
Instance metadata	864
EB CLI basics	864
Eb create	864
Eb status	865
Eb health	865
Eb events	866
Eb logs	866
Eb open	866
Eb deploy	867
Eb config	867
Eb terminate	868
CodeBuild	868
Creating an application	868
Building and deploying your application code	868
Using the EB CLI with Git	870
Associating Elastic Beanstalk environments with Git branches	870
Deploying changes	870
Using Git submodules	871
Assigning Git tags to your application version	871
CodeCommit	871
Prerequisites	872
Creating a CodeCommit repository with the EB CLI	872
Deploying from your CodeCommit repository	873
Configuring additional branches and environments	874
Using an existing CodeCommit repository	875
Monitoring health	875
Reading the output	878
Interactive health view	879
Interactive health view options	880

Composing environments	881
Troubleshooting	882
Troubleshooting deployments	883
EB CLI commands	884
eb abort	885
eb appversion	886
eb clone	888
eb codesource	890
eb config	891
eb console	893
eb create	894
eb deploy	903
eb events	904
eb health	905
eb init	907
eb labs	910
eb list	910
eb local	911
eb logs	913
eb open	916
eb platform	916
eb printenv	923
eb restore	924
eb scale	924
eb setenv	925
eb ssh	926
eb status	928
eb swap	929
eb tags	930
eb terminate	933
eb upgrade	934
eb use	935
Common options	935
EB CLI 2.6 (retired)	936
Differences from version 3 of EB CLI	936
Migrating to EB CLI 3 and CodeCommit	937
EB API CLI (retired)	937
Converting Elastic Beanstalk API CLI scripts	937
Security	940
Data protection	940
Data encryption	941
Internetwork privacy	942
Identity and access management	942
Logging and monitoring	942
Enhanced health reporting	942
Amazon EC2 instance logs	943
Environment notifications	943
Amazon CloudWatch alarms	943
AWS CloudTrail logs	943
AWS X-Ray debugging	943
Compliance validation	943
Resilience	944
Infrastructure security	944
Shared responsibility model	945
Security best practices	945
Preventive security best practices	945
Detective security best practices	946

Troubleshooting	947
Connectivity	947
Environment creation	948
Deployments	948
Health	948
Configuration	949
Docker	949
FAQ	950
Resources	951
Sample applications	951
Platform history	953

What is AWS Elastic Beanstalk?

Amazon Web Services (AWS) comprises over one hundred services, each of which exposes an area of functionality. While the variety of services offers flexibility for how you want to manage your AWS infrastructure, it can be challenging to figure out which services to use and how to provision them.

With Elastic Beanstalk, you can quickly deploy and manage applications in the AWS Cloud without having to learn about the infrastructure that runs those applications. Elastic Beanstalk reduces management complexity without restricting choice or control. You simply upload your application, and Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.

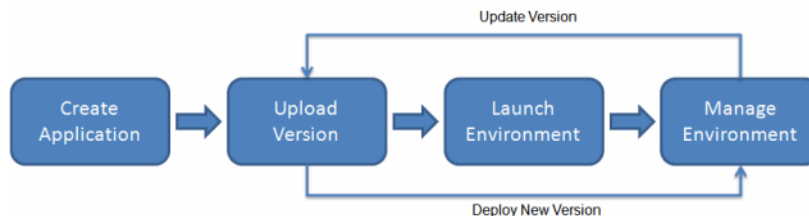
Elastic Beanstalk supports applications developed in Go, Java, .NET, Node.js, PHP, Python, and Ruby. When you deploy your application, Elastic Beanstalk builds the selected supported platform version and provisions one or more AWS resources, such as Amazon EC2 instances, to run your application.

You can interact with Elastic Beanstalk by using the Elastic Beanstalk console, the AWS Command Line Interface (AWS CLI), or **eb**, a high-level CLI designed specifically for Elastic Beanstalk.

To learn more about how to deploy a sample web application using Elastic Beanstalk, see [Getting Started with AWS: Deploying a Web App](#).

You can also perform most deployment tasks, such as changing the size of your fleet of Amazon EC2 instances or monitoring your application, directly from the Elastic Beanstalk web interface (console).

To use Elastic Beanstalk, you create an application, upload an application version in the form of an application source bundle (for example, a Java .war file) to Elastic Beanstalk, and then provide some information about the application. Elastic Beanstalk automatically launches an environment and creates and configures the AWS resources needed to run your code. After your environment is launched, you can then manage your environment and deploy new application versions. The following diagram illustrates the workflow of Elastic Beanstalk.



After you create and deploy your application, information about the application—including metrics, events, and environment status—is available through the Elastic Beanstalk console, APIs, or Command Line Interfaces, including the unified AWS CLI.

Pricing

There is no additional charge for Elastic Beanstalk. You pay only for the underlying AWS resources that your application consumes. For details about pricing, see the [Elastic Beanstalk service detail page](#).

Where to go next

This guide contains conceptual information about the Elastic Beanstalk web service, as well as information about how to use the service to deploy web applications. Separate sections describe how to

use the Elastic Beanstalk console, command line interface (CLI) tools, and API to deploy and manage your Elastic Beanstalk environments. This guide also documents how Elastic Beanstalk is integrated with other services provided by Amazon Web Services.

We recommend that you first read [Getting started using Elastic Beanstalk \(p. 3\)](#) to learn how to start using Elastic Beanstalk. *Getting Started* steps you through creating, viewing, and updating your Elastic Beanstalk application, as well as editing and terminating your Elastic Beanstalk environment. *Getting Started* also describes different ways you can access Elastic Beanstalk.

To learn more about an Elastic Beanstalk application and its components, see the following pages.

- [Elastic Beanstalk concepts \(p. 13\)](#)
- [Elastic Beanstalk platforms glossary \(p. 25\)](#)
- [Shared responsibility model for Elastic Beanstalk platform maintenance \(p. 27\)](#)
- [Elastic Beanstalk platform support policy \(p. 28\)](#)

Getting started using Elastic Beanstalk

To help you understand how AWS Elastic Beanstalk works, this tutorial walks you through creating, exploring, updating, and deleting an Elastic Beanstalk application. It takes less than an hour to complete.

There is no cost for using Elastic Beanstalk, but the AWS resources that it creates for this tutorial are live (and don't run in a sandbox). You incur the standard usage fees for these resources until you terminate them at the end of this tutorial. The total charges are typically less than a dollar. For information about how to minimize charges, see [AWS free tier](#).

Topics

- [Setting up: Create an AWS account](#) (p. 3)
- [Step 1: Create an example application](#) (p. 3)
- [Step 2: Explore your environment](#) (p. 5)
- [Step 3: Deploy a new version of your application](#) (p. 7)
- [Step 4: Configure your environment](#) (p. 8)
- [Step 5: Clean up](#) (p. 10)
- [Next steps](#) (p. 11)

Setting up: Create an AWS account

If you're not already an AWS customer, you need to create an AWS account. Signing up enables you to access Elastic Beanstalk and other AWS services that you need.

To sign up for an AWS account

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. Follow the instructions shown.

Step 1: Create an example application

In this step, you create a new application starting from a preexisting example application. Elastic Beanstalk supports platforms for different programming languages, application servers, and Docker containers. You choose a platform when you create the application.

Create an application and an environment

To create your example application, you'll use the **Create a web app** console wizard. It creates an Elastic Beanstalk application and launches an environment within it. An environment is the collection of AWS resources required to run your application code.

To create an example application

1. Open the Elastic Beanstalk console using this link: <https://console.aws.amazon.com/elasticbeanstalk/home#/gettingStarted?applicationName=getting-started-app>

2. Optionally add [application tags](#) (p. 349).
3. For **Platform**, choose a platform, and then choose **Create application**.

To run the example application on AWS resources, Elastic Beanstalk takes the following actions. They take about five minutes to complete.

1. Creates an Elastic Beanstalk application named **getting-started-app**.
2. Launches an environment named **GettingStartedApp-env** with these AWS resources:
 - An Amazon Elastic Compute Cloud (Amazon EC2) instance (virtual machine)
 - An Amazon EC2 security group
 - An Amazon Simple Storage Service (Amazon S3) bucket
 - Amazon CloudWatch alarms
 - An AWS CloudFormation stack
 - A domain name

For details about these AWS resources, see [the section called “AWS resources created for the example application”](#) (p. 5).

3. Creates a new application version named **Sample Application**. This is the default Elastic Beanstalk example application file.
4. Deploys the code for the example application to the **GettingStartedApp-env** environment.

During the environment creation process, the console tracks progress and displays events.



Creating GettingStarted-env

This will take a few minutes....

```
8:40pm Successfully launched environment: GettingStarted-env
8:39pm Environment health has transitioned from Pending to Ok. Initialization completed 16 seconds ago and took 5 m
8:36pm Added instance [i-045eb69a24818d1d4] to your environment.
8:36pm Waiting for EC2 instances to launch. This may take a few minutes.
8:35pm Created EIP: 34.230.236.246
8:34pm Created security group named:
      eb-dv-e-sbj4gzf2dm-stack-AWSEBSecurityGroup-KATGTRO6V1J9
8:34pm Environment health has transitioned to Pending. Initialization in progress (running for 8 seconds). There are no
8:34pm Using elasticbeanstalk-us-east-1-270205402845 as Amazon S3 storage bucket for environment data.
8:34pm createEnvironment is starting.
```

When all of the resources are launched and the EC2 instances running the application pass health checks, the environment's health changes to Ok. You can now use your web application's website.

AWS resources created for the example application

When you create the example application, Elastic Beanstalk creates the following AWS resources:

- **EC2 instance** – An Amazon EC2 virtual machine configured to run web apps on the platform you choose.

Each platform runs a different set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination thereof. Most platforms use either Apache or nginx as a reverse proxy that processes web traffic in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow incoming traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic is not allowed on other ports.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
- **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

Step 2: Explore your environment

To see an overview of your Elastic Beanstalk application's environment, use the environment page in the Elastic Beanstalk console.

To view the environment overview

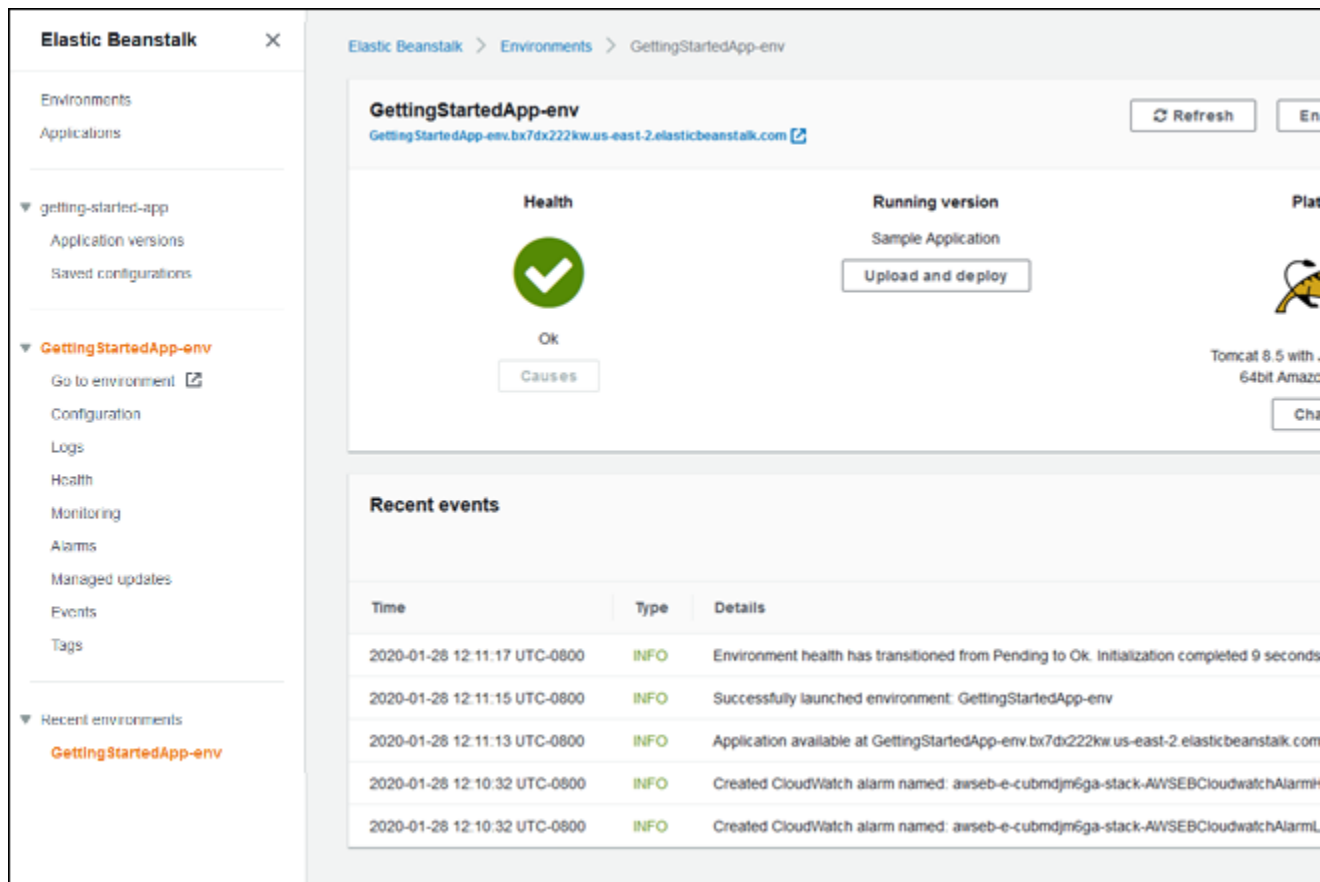
1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

The environment overview pane shows top level information about your environment. This includes its name, its URL, its current health status, the name of the currently deployed application version, and the platform version that the application is running on. Below the overview pane you can see the five most recent environment events.

To learn more about environment tiers, platforms, application versions, and other Elastic Beanstalk concepts, see [Concepts \(p. 13\)](#).



While Elastic Beanstalk creates your AWS resources and launches your application, the environment is in a `pending` state. Status messages about launch events are continuously added to the overview.

The environment's **URL** is located at the top of the overview, below the environment name. This is the URL of the web application that the environment is running. Choose this URL to get to the example application's *Congratulations* page.

The navigation page on the left side of the console links to other pages that contain more detailed information about your environment and provide access to additional features:

- **Configuration** – Shows the resources provisioned for this environment, such as the Amazon Elastic Compute Cloud (Amazon EC2) instances that host your application. You can configure some of the provisioned resources on this page.
- **Health** – Shows the status of and detailed health information about the Amazon EC2 instances running your application.
- **Monitoring** – Shows statistics for the environment, such as average latency and CPU utilization. You can use this page to create alarms for the metrics that you are monitoring.
- **Events** – Shows information or error messages from the Elastic Beanstalk service and from other services whose resources this environment uses.
- **Tags** – Shows environment tags and allows you to manage them. Tags are key-value pairs that are applied to your environment.

Step 3: Deploy a new version of your application

Periodically, you might need to deploy a new version of your application. You can deploy a new version at any time, as long as no other update operations are in progress on your environment.

The application version that you started this tutorial with is called **Sample Application**.

To update your application version

1. Download the sample application that matches your environment's platform. Use one of the following applications.
 - **Single Container Docker** – [docker-singlecontainer-v1.zip](#)
 - **Multicontainer Docker** – [docker-multicontainer-v2.zip](#)
 - **Preconfigured Docker (Glassfish)** – [docker-glassfish-v1.zip](#)
 - **Go** – [go-v1.zip](#)
 - **Java SE** – [java-se-jetty-gradle-v3.zip](#)
 - **Tomcat (default)** – [java-tomcat-v3.zip](#)
 - **Tomcat 7** – [java7-tomcat7.zip](#)
 - **.NET** – [dotnet-asp-v1.zip](#)
 - **Node.js** – [nodejs-v1.zip](#)
 - **PHP** – [php-v1.zip](#)
 - **Python** – [python-v1.zip](#)
 - **Ruby (Passenger Standalone)** – [ruby-passenger-v3.zip](#)
 - **Ruby (Puma)** – [ruby-puma-v3.zip](#)
2. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
3. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note
If you have many environments, use the search bar to filter the environment list.
4. On the environment overview page, choose **Upload and deploy**.
5. Choose **Choose file**, and then upload the sample application source bundle that you downloaded.

Upload and deploy ✕

i To deploy a previous version, go to the [Application Versions](#) page.

Upload application:

📁 Choose file

File name : **java-tomcat-v3.zip** ✔

Version label:

Sample Application-2

▶ **Deployment Preferences**

The application version will be deployed using the **All at once** policy.

Current number of instances: **1**

Cancel **Deploy**

The console automatically fills in the **Version label** with a new unique label. If you type in your own version label, ensure that it's unique.

6. Choose **Deploy**.

While Elastic Beanstalk deploys your file to your Amazon EC2 instances, you can view the deployment status on the environment's overview. While the application version is updated, the **Environment Health** status is gray. When the deployment is complete, Elastic Beanstalk performs an application health check. When the application responds to the health check, it's considered healthy and the status returns to green. The environment overview shows the new **Running Version**—the name you provided as the **Version label**.

Elastic Beanstalk also uploads your new application version and adds it to the table of application versions. To view the table, choose **Application versions** under **getting-started-app** on the navigation pane.

Step 4: Configure your environment

You can configure your environment to better suit your application. For example, if you have a compute-intensive application, you can change the type of Amazon Elastic Compute Cloud (Amazon EC2) instance that is running your application. To apply configuration changes, Elastic Beanstalk performs an environment update.

Some configuration changes are simple and happen quickly. Some changes require deleting and recreating AWS resources, which can take several minutes. When you change configuration settings, Elastic Beanstalk warns you about potential application downtime.

Make a configuration change

In this example of a configuration change, you edit your environment's capacity settings. You configure a load balanced, automatically scaling environment that has between two and four Amazon EC2 instances in its Auto Scaling group, and then you verify that the change occurred. Elastic Beanstalk creates an additional Amazon EC2 instance, adding to the single instance that it created initially. Then, Elastic Beanstalk associates both instances with the environment's load balancer. As a result, your application's responsiveness is improved and its availability is increased.

To change your environment's capacity

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Auto Scaling group** section, change **Environment type** to **Load balanced**.
6. In the **Instances** row, change **Max** to **4**, and then change **Min** to **2**.
7. Choose **Apply**.
8. A warning tells you that this update replaces all of your current instances. Choose **Confirm**.
9. In the navigation pane, choose **Events**.

The environment update can take a few minutes. To find out that it's complete, look for the event **Successfully deployed new configuration to environment** in the event list. This confirms that the Auto Scaling minimum instance count has been set to 2. Elastic Beanstalk automatically launches the second instance.

Verify the configuration change

When the environment update is complete and the environment is ready, verify your change.

To verify the increased capacity

1. In the navigation pane, choose **Health**.
2. Look at the **Enhanced health overview** page.

You can see that two Amazon EC2 instances are listed following the **Overall** line. Your environment capacity has increased to two instances.

Elastic Beanstalk > Environments > GettingStartedApp-env > Health

Enhanced Health Overview

Instances: 2 Total, 2 Ok

[Learn more](#) about enhanced health.

	Instance ID	Status	Running	Deployment ID
●	Overall	Ok	N/A	N/A
○	i-0867c82b5baab7ef1	Ok	10 minutes	1
○	i-0106bcf1fb76efdc4	Ok	10 minutes	1

Step 5: Clean up

Congratulations! You have successfully deployed a sample application to the AWS Cloud, uploaded a new version, and modified its configuration to add a second Auto Scaling instance. To ensure that you're not charged for any services you aren't using, delete all application versions and terminate the environment. This also deletes the AWS resources that the environment created for you.

To delete the application and all associated resources

1. Delete all application versions.
 - a. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
 - b. In the navigation pane, choose **Applications**, and then choose **getting-started-app**.
 - c. In the navigation pane, find your application's name and choose **Application versions**.
 - d. On the **Application versions** page, select all application versions that you want to delete.
 - e. Choose **Actions**, and then choose **Delete**.
 - f. Turn on **Delete versions from Amazon S3**.
 - g. Choose **Delete**, and then choose **Done**.
2. Terminate the environment.
 - a. In the navigation pane, choose **getting-started-app**, and then choose **GettingStartedApp-env** in the environment list.
 - b. Choose **Environment actions**, and then choose **Terminate Environment**.

- c. Confirm that you want to terminate **GettingStartedApp-env** by typing the environment name, and then choose **Terminate**.
3. Delete the getting-started-app application.
 - a. In the navigation pane, choose the **getting-started-app**.
 - b. Choose **Actions**, and then choose **Delete application**.
 - c. Confirm that you want to delete **getting-started-app** by typing the application name, and then choose **Delete**.

Next steps

Now that you know how to create an Elastic Beanstalk application and environment, we recommend that you read [Concepts \(p. 13\)](#). This topic provides information about the Elastic Beanstalk components and architecture, and describes important design considerations for your Elastic Beanstalk application.

In addition to the Elastic Beanstalk console, you can use the following tools to create and manage Elastic Beanstalk environments.

EB CLI

The EB CLI is a command line tool for creating and managing environments. See [Using the Elastic Beanstalk command line interface \(EB CLI\) \(p. 852\)](#) for details.

AWS SDK for Java

The AWS SDK for Java provides a Java API you can use to build applications that use AWS infrastructure services. With the AWS SDK for Java, you can get started in minutes with a single, downloadable package that includes the AWS Java library, code examples, and documentation.

The AWS SDK for Java requires the J2SE Development Kit 5.0 or later. You can download the latest Java software from <http://developers.sun.com/downloads/>. The SDK also requires Apache Commons (Codec, HttpClient, and Logging) and Saxon-HE third-party packages, which are included in the third-party directory of the SDK.

For more information, see [AWS SDK for Java](#).

AWS Toolkit for Eclipse

The AWS Toolkit for Eclipse is an open source plug-in for the Eclipse Java IDE. You can use it to create AWS Java web projects that are preconfigured with the AWS SDK for Java, and then deploy the web applications to Elastic Beanstalk. The Elastic Beanstalk plug-in builds on top of the Eclipse Web Tools Platform (WTP). The toolkit provides a Travel Log sample web application template that demonstrates the use of Amazon S3 and Amazon SNS.

To ensure that you have all the WTP dependencies, we recommend that you start with the Java EE distribution of Eclipse. You can download it from <http://eclipse.org/downloads/>.

For more information about using the Elastic Beanstalk plug-in for Eclipse, see [AWS Toolkit for Eclipse](#). To get started creating your Elastic Beanstalk application using Eclipse, see [Creating and deploying Java applications on Elastic Beanstalk \(p. 94\)](#).

AWS SDK for .NET

The AWS SDK for .NET enables you to build applications that use AWS infrastructure services. With the AWS SDK for .NET, you can get started in minutes with a single, downloadable package that includes the AWS .NET library, code examples, and documentation.

For more information, see [AWS SDK for .NET](#). For supported .NET Framework and Visual Studio versions, see the [AWS SDK for .NET Developer Guide](#).

AWS Toolkit for Visual Studio

With the AWS Toolkit for Visual Studio plug-in, you can deploy an existing .NET application to Elastic Beanstalk. You can also create projects using the AWS templates that are preconfigured with the AWS SDK for .NET.

For prerequisite and installation information, see the [AWS Toolkit for Visual Studio](#). To get started creating your Elastic Beanstalk application using Visual Studio, see [Creating and deploying .NET applications on Elastic Beanstalk \(p. 137\)](#).

AWS SDK for JavaScript in Node.js

The AWS SDK for JavaScript in Node.js enables you to build applications on top of AWS infrastructure services. With the AWS SDK for JavaScript in Node.js, you can get started in minutes with a single, downloadable package that includes the AWS Node.js library, code examples, and documentation.

For more information, see the [AWS SDK for JavaScript in Node.js](#).

AWS SDK for PHP

The AWS SDK for PHP enables you to build applications on top of AWS infrastructure services. With the AWS SDK for PHP, you can get started in minutes with a single, downloadable package that includes the AWS PHP library, code examples, and documentation.

The AWS SDK for PHP requires PHP 5.2 or later. For download details, see <http://php.net/>.

For more information, see the [AWS SDK for PHP](#).

AWS SDK for Python (Boto)

With the AWS SDK for Python (Boto), you can get started in minutes with a single, downloadable package that includes the AWS Python library, code examples, and documentation. You can build Python applications on top of APIs that take the complexity out of coding directly against web service interfaces.

The all-in-one library provides Python developer-friendly APIs that hide many of the lower-level tasks associated with programming for the AWS Cloud, including authentication, request retries, and error handling. The SDK provides practical examples in Python for how to use the libraries to build applications.

For information about Boto, example code, documentation, tools, and additional resources, see the [Python Developer Center](#).

AWS SDK for Ruby

You can get started in minutes with a single, downloadable package complete with the AWS Ruby library, code examples, and documentation. You can build Ruby applications on top of APIs that take the complexity out of coding directly against web services interfaces.

The all-in-one library provides Ruby developer-friendly APIs that hide many of the lower-level tasks associated with programming for the AWS Cloud, including authentication, request retries, and error handling. The SDK provides practical examples in Ruby for how to use the libraries to build applications.

For information about the SDK, example code, documentation, tools, and additional resources, see the [Ruby Developer Center](#).

Elastic Beanstalk concepts

AWS Elastic Beanstalk enables you to manage all of the resources that run your *application* as *environments*. Here are some key Elastic Beanstalk concepts.

Application

An Elastic Beanstalk *application* is a logical collection of Elastic Beanstalk components, including *environments*, *versions*, and *environment configurations*. In Elastic Beanstalk an application is conceptually similar to a folder.

Application version

In Elastic Beanstalk, an *application version* refers to a specific, labeled iteration of deployable code for a web application. An application version points to an Amazon Simple Storage Service (Amazon S3) object that contains the deployable code, such as a Java WAR file. An application version is part of an application. Applications can have many versions and each application version is unique. In a running environment, you can deploy any application version you already uploaded to the application, or you can upload and immediately deploy a new application version. You might upload multiple application versions to test differences between one version of your web application and another.

Environment

An *environment* is a collection of AWS resources running an application version. Each environment runs only one application version at a time, however, you can run the same application version or different application versions in many environments simultaneously. When you create an environment, Elastic Beanstalk provisions the resources needed to run the application version you specified.

Environment tier

When you launch an Elastic Beanstalk environment, you first choose an environment tier. The environment tier designates the type of application that the environment runs, and determines what resources Elastic Beanstalk provisions to support it. An application that serves HTTP requests runs in a [web server environment tier \(p. 14\)](#). An environment that pulls tasks from an Amazon Simple Queue Service (Amazon SQS) queue runs in a [worker environment tier \(p. 15\)](#).

Environment configuration

An *environment configuration* identifies a collection of parameters and settings that define how an environment and its associated resources behave. When you update an environment's configuration settings, Elastic Beanstalk automatically applies the changes to existing resources or deletes and deploys new resources (depending on the type of change).

Saved configuration

A *saved configuration* is a template that you can use as a starting point for creating unique environment configurations. You can create and modify saved configurations, and apply them to environments, using the Elastic Beanstalk console, EB CLI, AWS CLI, or API. The API and the AWS CLI refer to saved configurations as *configuration templates*.

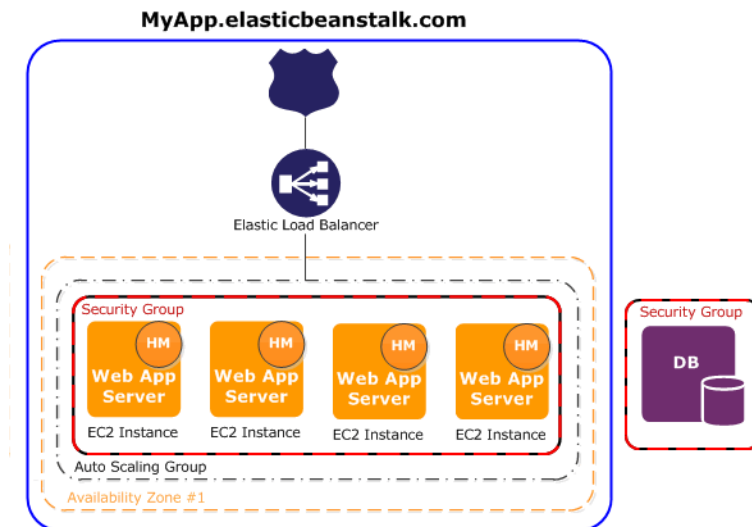
Platform

A *platform* is a combination of an operating system, programming language runtime, web server, application server, and Elastic Beanstalk components. You design and target your web application to a platform. Elastic Beanstalk provides a variety of platforms on which you can build your applications.

For details, see [Elastic Beanstalk platforms \(p. 25\)](#).

Web server environments

The following diagram shows an example Elastic Beanstalk architecture for a web server environment tier, and shows how the components in that type of environment tier work together.



The environment is the heart of the application. In the diagram, the environment is shown within the top-level solid line. When you create an environment, Elastic Beanstalk provisions the resources required to run your application. AWS resources created for an environment include one elastic load balancer (ELB in the diagram), an Auto Scaling group, and one or more Amazon Elastic Compute Cloud (Amazon EC2) instances.

Every environment has a CNAME (URL) that points to a load balancer. The environment has a URL, such as `myapp.us-west-2.elasticbeanstalk.com`. This URL is aliased in [Amazon Route 53](#) to an Elastic Load Balancing URL—something like `abcdef-123456.us-west-2.elb.amazonaws.com`—by using a CNAME record. [Amazon Route 53](#) is a highly available and scalable Domain Name System (DNS) web service. It provides secure and reliable routing to your infrastructure. Your domain name that you registered with your DNS provider will forward requests to the CNAME.

The load balancer sits in front of the Amazon EC2 instances, which are part of an Auto Scaling group. Amazon EC2 Auto Scaling automatically starts additional Amazon EC2 instances to accommodate increasing load on your application. If the load on your application decreases, Amazon EC2 Auto Scaling stops instances, but always leaves at least one instance running.

The software stack running on the Amazon EC2 instances is dependent on the *container type*. A container type defines the infrastructure topology and software stack to be used for that environment. For example, an Elastic Beanstalk environment with an Apache Tomcat container uses the Amazon Linux operating system, Apache web server, and Apache Tomcat software. For a list of supported container types, see [Elastic Beanstalk supported platforms \(p. 29\)](#). Each Amazon EC2 instance that runs your application uses one of these container types. In addition, a software component called the *host manager (HM)* runs on each Amazon EC2 instance. The host manager is responsible for the following:

- Deploying the application
- Aggregating events and metrics for retrieval via the console, the API, or the command line
- Generating instance-level events
- Monitoring the application log files for critical errors
- Monitoring the application server
- Patching instance components
- Rotating your application's log files and publishing them to Amazon S3

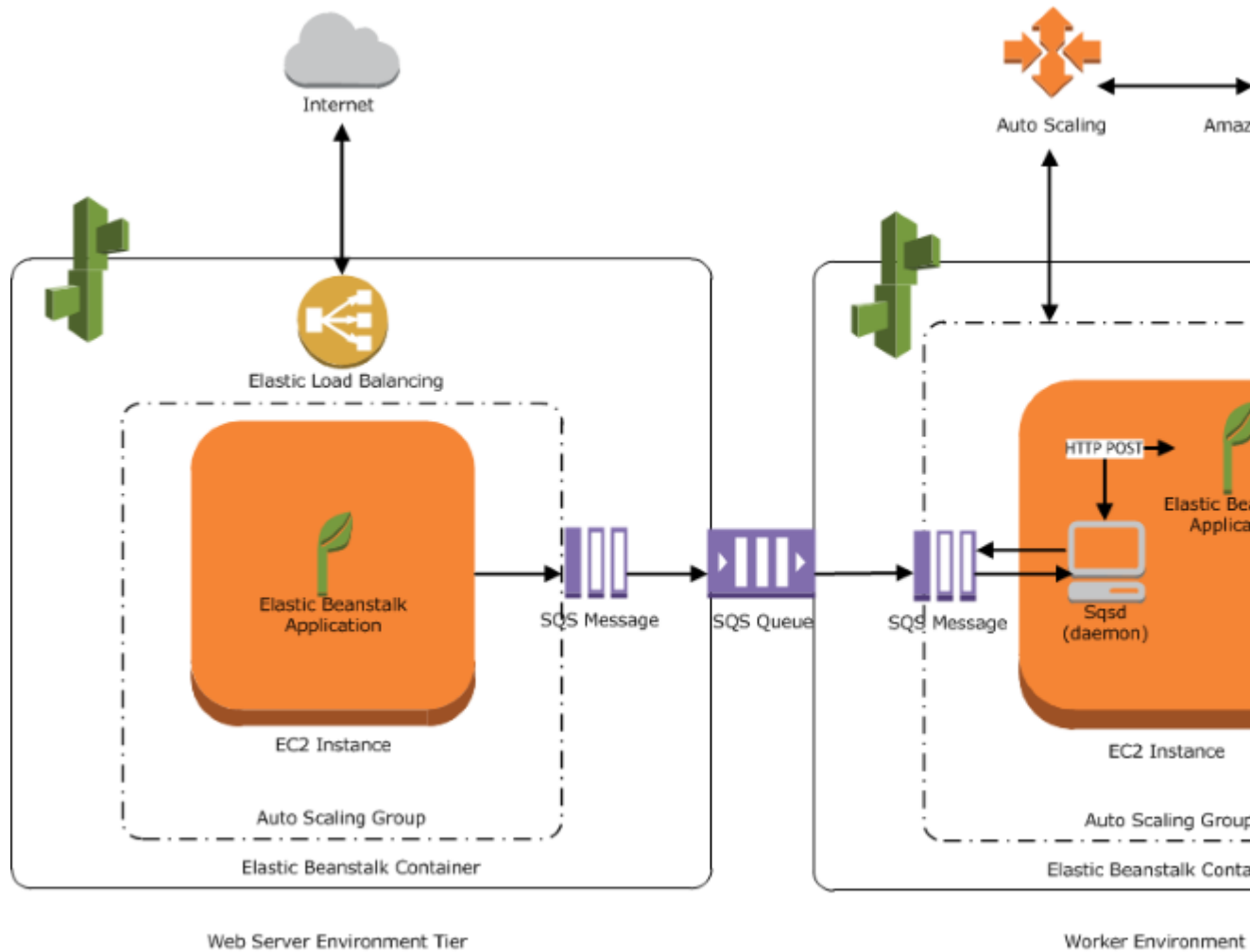
The host manager reports metrics, errors and events, and server instance status, which are available via the Elastic Beanstalk console, APIs, and CLIs.

The Amazon EC2 instances shown in the diagram are part of one *security group*. A security group defines the firewall rules for your instances. By default, Elastic Beanstalk defines a security group, which allows everyone to connect using port 80 (HTTP). You can define more than one security group. For example, you can define a security group for your database server. For more information about Amazon EC2 security groups and how to configure them for your Elastic Beanstalk application, see [Security groups \(p. 455\)](#).

Worker environments

AWS resources created for a worker environment tier include an Auto Scaling group, one or more Amazon EC2 instances, and an IAM role. For the worker environment tier, Elastic Beanstalk also creates and provisions an Amazon SQS queue if you don't already have one. When you launch a worker environment, Elastic Beanstalk installs the necessary support files for your programming language of choice and a daemon on each EC2 instance in the Auto Scaling group. The daemon reads messages from an Amazon SQS queue. The daemon sends data from each message that it reads to the web application running in the worker environment for processing. If you have multiple instances in your worker environment, each instance has its own daemon, but they all read from the same Amazon SQS queue.

The following diagram shows the different components and their interactions across environments and AWS services.



Amazon CloudWatch is used for alarms and health monitoring. For more information, go to [Basic health reporting](#) (p. 688).

For details about how the worker environment tier works, see [Elastic Beanstalk worker environments](#) (p. 437).

Design considerations

Because applications deployed using Elastic Beanstalk run on Amazon cloud resources, you should keep several things in mind when designing your application: *scalability*, *security*, *persistent storage*, *fault tolerance*, *content delivery*, *software updates and patching*, and *connectivity*. For a comprehensive list of technical AWS whitepapers, covering topics such as architecture, security and economics, go to [AWS Cloud Computing Whitepapers](#).

Scalability

When you're operating in a physical hardware environment, as opposed to a cloud environment, you can approach scalability two ways—you can scale up (vertical scaling) or scale out (horizontal scaling). The scale-up approach requires an investment in powerful hardware as the demands on the business

increase, whereas the scale-out approach requires following a distributed model of investment, so hardware and application acquisitions are more targeted, data sets are federated, and design is service-oriented. The scale-up approach could become very expensive, and there's still the risk that demand could outgrow capacity. Although the scale-out approach is usually more effective, it requires predicting the demand at regular intervals and deploying infrastructure in chunks to meet demand. This approach often leads to unused capacity and requires careful monitoring.

By moving to the cloud you can bring the use of your infrastructure into close alignment with demand by leveraging the elasticity of the cloud. Elasticity is the streamlining of resource acquisition and release, so that your infrastructure can rapidly scale in and scale out as demand fluctuates. To implement elasticity, configure your Auto Scaling settings to scale up or down based on metrics from the resources in your environment (utilization of the servers or network I/O, for instance). You can use Auto Scaling to automatically add compute capacity when usage rises and remove it when usage drops. Publish system metrics (CPU, memory, disk I/O, network I/O) to Amazon CloudWatch and configure alarms to trigger Auto Scaling actions or send notifications. For more instructions on configuring Auto Scaling, see [Auto Scaling group for your Elastic Beanstalk environment \(p. 457\)](#).

Elastic Beanstalk applications should also be as *stateless* as possible, using loosely coupled, fault-tolerant components that can be scaled out as needed. For more information about designing scalable application architectures for AWS, read the [Architecting for the Cloud: Best Practices](#) whitepaper.

Security

Security on AWS is a [shared responsibility](#). AWS protects the physical resources in your environment and ensure that the cloud is a safe place for you to run applications. You are responsible for the security of data coming in and out of your Elastic Beanstalk environment and the security of your application.

To protect information flowing between your application and clients, configure SSL. To do this, you need a free certificate from AWS Certificate Manager (ACM). If you already have a certificate from an external certificate authority (CA), you can use ACM to import that certificate programmatically or using the AWS CLI.

If ACM is not [available in your region](#), you can purchase a certificate from an external CA such as VeriSign or Entrust. Then, use the AWS CLI to upload a third-party or self-signed certificate and private key to (AWS Identity and Access Management (IAM). The certificate's public key authenticates your server to the browser. It also serves as the basis for creating the shared session key that encrypts the data in both directions. For instructions on creating, uploading, and assigning an SSL certificate to your environment, see [Configuring HTTPS for your Elastic Beanstalk environment \(p. 652\)](#).

When you configure an SSL certificate for your environment, data is encrypted between the client and your environment's Elastic Load Balancing load balancer. By default, encryption is terminated at the load balancer, and traffic between the load balancer and Amazon EC2 instances is unencrypted.

Persistent storage

Elastic Beanstalk applications run on Amazon EC2 instances that have no persistent local storage. When the Amazon EC2 instances terminate, the local file system is not saved, and new Amazon EC2 instances start with a default file system. You should design your application to store data in a persistent data source. Amazon Web Services offers a number of persistent storage services that you can leverage for your application. The following table lists them.

Storage service	Service documentation	Elastic Beanstalk integration
Amazon S3	Amazon Simple Storage Service Documentation	Using Elastic Beanstalk with Amazon S3 (p. 831)

Storage service	Service documentation	Elastic Beanstalk integration
Amazon Elastic File System	Amazon Elastic File System Documentation	Using Elastic Beanstalk with Amazon Elastic File System (p. 758)
Amazon Elastic Block Store	Amazon Elastic Block Store Feature Guide: Elastic Block Store	
Amazon DynamoDB	Amazon DynamoDB Documentation	Using Elastic Beanstalk with Amazon DynamoDB (p. 756)
Amazon Relational Database Service (RDS)	Amazon Relational Database Service Documentation	Using Elastic Beanstalk with Amazon RDS (p. 820)

Fault tolerance

As a rule of thumb, you should be a pessimist when designing architecture for the cloud. Always design, implement, and deploy for automated recovery from failure. Use multiple Availability Zones for your Amazon EC2 instances and for Amazon RDS. Availability Zones are conceptually like logical data centers. Use Amazon CloudWatch to get more visibility into the health of your Elastic Beanstalk application and take appropriate actions in case of hardware failure or performance degradation. Configure your Auto Scaling settings to maintain your fleet of Amazon EC2 instances at a fixed size so that unhealthy Amazon EC2 instances are replaced by new ones. If you are using Amazon RDS, then set the retention period for backups, so that Amazon RDS can perform automated backups.

Content delivery

When users connect to your website, their requests may be routed through a number of individual networks. As a result users may experience poor performance due to high latency. Amazon CloudFront can help ameliorate latency issues by distributing your web content (such as images, video, and so on) across a network of edge locations around the world. End users are routed to the nearest edge location, so content is delivered with the best possible performance. CloudFront works seamlessly with Amazon S3, which durably stores the original, definitive versions of your files. For more information about Amazon CloudFront, see <https://aws.amazon.com/cloudfront>.

Software updates and patching

Elastic Beanstalk periodically updates its platforms with new software and patches. Elastic Beanstalk doesn't upgrade running environments to new platform versions automatically, but you can initiate a [platform update \(p. 415\)](#) to update your running environment in place. Platform updates use [rolling updates \(p. 409\)](#) to keep your application available by applying changes in batches.

Connectivity

Elastic Beanstalk needs to be able to connect to the instances in your environment to complete deployments. When you deploy an Elastic Beanstalk application inside an Amazon VPC, the configuration required to enable connectivity depends on the type of Amazon VPC environment you create:

- For single-instance environments, no additional configuration is required because Elastic Beanstalk assigns each Amazon EC2 instance a public Elastic IP address that enables the instance to communicate directly with the Internet.
- For load-balancing, autoscaling environments in an Amazon VPC with both public and private subnets, you must do the following:

- Create a load balancer in the public subnet to route inbound traffic from the Internet to the Amazon EC2 instances.
- Create a network address translation (NAT) device to route outbound traffic from the Amazon EC2 instances in private subnets to the Internet.
- Create inbound and outbound routing rules for the Amazon EC2 instances inside the private subnet.
- If using a NAT instance, configure the security groups for the NAT instance and Amazon EC2 instances to enable Internet communication.
- For a load-balancing, autoscaling environment in an Amazon VPC that has one public subnet, no additional configuration is required because the Amazon EC2 instances are configured with a public IP address that enables the instances to communicate with the Internet.

For more information about using Elastic Beanstalk with Amazon VPC, see [Using Elastic Beanstalk with Amazon VPC \(p. 834\)](#).

Service roles, instance profiles, and user policies

When you create an environment, AWS Elastic Beanstalk prompts you to provide two AWS Identity and Access Management (IAM) roles: a service role and an instance profile. The [service role \(p. 20\)](#) is assumed by Elastic Beanstalk to use other AWS services on your behalf. The [instance profile \(p. 21\)](#) is applied to the instances in your environment and allows them to retrieve [application versions \(p. 13\)](#) from Amazon Simple Storage Service (Amazon S3), upload logs to Amazon S3, and perform other tasks that vary depending on the environment type and platform.

The best way to get a properly configured service role and instance profile is to [create an environment running a sample application \(p. 363\)](#) in the Elastic Beanstalk console or by using the Elastic Beanstalk Command Line Interface (EB CLI). When you create an environment, the clients create the required roles and assign them [managed policies \(p. 775\)](#) that include all of the necessary permissions.

In addition to the two roles that you assign to your environment, you can also create [user policies \(p. 23\)](#) and apply them to IAM users and groups in your account to allow users to create and manage Elastic Beanstalk applications and environments. Elastic Beanstalk provides managed policies for full access and read-only access.

You can create your own instance profiles and user policies for advanced scenarios. If your instances need to access services that are not included in the default policies, you can add additional policies to the default or create a new one. You can also create more restrictive user policies if the managed policy is too permissive. See the [AWS Identity and Access Management User Guide](#) for in-depth coverage of AWS permissions.

Topics

- [Elastic Beanstalk service role \(p. 20\)](#)
- [Elastic Beanstalk instance profile \(p. 21\)](#)
- [Elastic Beanstalk user policy \(p. 23\)](#)

Elastic Beanstalk service role

A service role is the IAM role that Elastic Beanstalk assumes when calling other services on your behalf. For example, Elastic Beanstalk uses the service role that you specify when creating an Elastic Beanstalk environment when it calls Amazon Elastic Compute Cloud (Amazon EC2), Elastic Load Balancing, and Amazon EC2 Auto Scaling APIs to gather information about the health of its AWS resources for [enhanced health monitoring \(p. 691\)](#).

The `AWSElasticBeanstalkEnhancedHealth` managed policy contains all of the permissions that Elastic Beanstalk needs to monitor environment health:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:DescribeInstanceHealth",
```

```
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeTargetHealth",
        "ec2:DescribeInstances",
        "ec2:DescribeInstanceStatus",
        "ec2:GetConsoleOutput",
        "ec2:AssociateAddress",
        "ec2:DescribeAddresses",
        "ec2:DescribeSecurityGroups",
        "sqs:GetQueueAttributes",
        "sqs:GetQueueUrl",
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeAutoScalingInstances",
        "autoscaling:DescribeScalingActivities",
        "autoscaling:DescribeNotificationConfigurations",
        "sns:Publish"
    ],
    "Resource": [
        "*"
    ]
}
]
```

This policy also includes Amazon SQS actions to allow Elastic Beanstalk to monitor queue activity for worker environments.

When you create an environment using the Elastic Beanstalk console, Elastic Beanstalk prompts you to create a service role named `aws-elasticbeanstalk-service-role` with the default set of permissions and a trust policy that allows Elastic Beanstalk to assume the service role. If you enable [managed platform updates \(p. 420\)](#), Elastic Beanstalk attaches another policy with permissions that enable that feature.

Similarly, when you create an environment using the [the section called “eb create” \(p. 894\)](#) command of the Elastic Beanstalk Command Line Interface (EB CLI) and don't specify a service role through the `--service-role` option, Elastic Beanstalk creates the default service role `aws-elasticbeanstalk-service-role`. If the default service role already exists, Elastic Beanstalk uses it for the new environment.

When you create an environment by using the `CreateEnvironment` action of the Elastic Beanstalk API, and don't specify a service role, Elastic Beanstalk creates a monitoring service-linked role. This is a unique type of service role that is predefined by Elastic Beanstalk to include all the permissions that the service requires to call other AWS services on your behalf. The service-linked role is associated with your account. Elastic Beanstalk creates it once, then reuses it when creating additional environments. You can also use IAM to create your account's monitoring service-linked role in advance. When your account has a monitoring service-linked role, you can use it to create an environment by using the Elastic Beanstalk API, the Elastic Beanstalk console, or the EB CLI. For details about using service-linked roles with Elastic Beanstalk environments, see [Using service-linked roles for Elastic Beanstalk \(p. 769\)](#).

For more information about service roles, see [Managing Elastic Beanstalk service roles \(p. 764\)](#).

Elastic Beanstalk instance profile

An instance profile is an IAM role that is applied to instances launched in your Elastic Beanstalk environment. When creating an Elastic Beanstalk environment, you specify the instance profile that is used when your instances:

- Retrieve [application versions \(p. 13\)](#) from Amazon Simple Storage Service (Amazon S3)
- Write logs to Amazon S3

- In [AWS X-Ray integrated environments \(p. 521\)](#), upload debugging data to X-Ray
- In multicontainer Docker environments, coordinate container deployments with Amazon Elastic Container Service
- In worker environments, read from an Amazon Simple Queue Service (Amazon SQS) queue
- In worker environments, perform leader election with Amazon DynamoDB
- In worker environments, publish instance health metrics to Amazon CloudWatch

The `AWSElasticBeanstalkWebTier` managed policy contains statements that allow instances in your environment to upload logs to Amazon S3 and send debugging information to X-Ray:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "BucketAccess",
      "Action": [
        "s3:Get*",
        "s3:List*",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::elasticbeanstalk-*",
        "arn:aws:s3:::elasticbeanstalk-*/*"
      ]
    },
    {
      "Sid": "XRayAccess",
      "Action": [
        "xray:PutTraceSegments",
        "xray:PutTelemetryRecords"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchLogsAccess",
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogStream",
        "logs:DescribeLogStreams",
        "logs:DescribeLogGroups"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/elasticbeanstalk*"
      ]
    }
  ]
}
```

Elastic Beanstalk also provides managed policies named `AWSElasticBeanstalkWorkerTier` and `AWSElasticBeanstalkMulticontainerDocker` for the other use cases. Elastic Beanstalk attaches all of these policies to the default instance profile, `aws-elasticbeanstalk-ec2-role`, when you create an environment with the console or EB CLI.

If your web application requires access to any other AWS services, add statements or managed policies to the instance profile that allow access to those services.

For more information about instance profiles, see [Managing Elastic Beanstalk instance profiles \(p. 760\)](#).

Elastic Beanstalk user policy

Create IAM users for each person who uses Elastic Beanstalk to avoid using your root account or sharing credentials. For increased security, only grant these users permission to access services and features that they need.

Elastic Beanstalk requires permissions not only for its own API actions, but for several other AWS services as well. Elastic Beanstalk uses user permissions to launch all of the resources in an environment, including EC2 instances, an Elastic Load Balancing load balancer, and an Auto Scaling group. Elastic Beanstalk also uses user permissions to save logs and templates to Amazon Simple Storage Service (Amazon S3), send notifications to Amazon SNS, assign instance profiles, and publish metrics to CloudWatch. Elastic Beanstalk requires AWS CloudFormation permissions to orchestrate resource deployments and updates. It also requires Amazon RDS permissions to create databases when needed, and Amazon SQS permissions to create queues for worker environments.

The following policy allows access to the actions used to create and manage Elastic Beanstalk environments. This policy is available in the IAM console as a managed policy named `AWSElasticBeanstalkFullAccess`. You can apply the managed policy to an IAM user or group to grant permission to use Elastic Beanstalk, or create your own policy that excludes permissions that are not needed by your users.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticbeanstalk:*",
        "ec2:*",
        "ecs:*",
        "ecr:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
        "dynamodb:*",
        "rds:*",
        "sqs:*",
        "logs:*",
        "iam:GetPolicyVersion",
        "iam:GetRole",
        "iam:PassRole",
        "iam:ListRolePolicies",
        "iam:ListAttachedRolePolicies",
        "iam:ListInstanceProfiles",
        "iam:ListRoles",
        "iam:ListServerCertificates",
        "acm:DescribeCertificate",
        "acm:ListCertificates",
        "codebuild:CreateProject",
        "codebuild>DeleteProject",
        "codebuild:BatchGetBuilds",
        "codebuild:StartBuild"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:AddRoleToInstanceProfile",
```

```

        "iam:CreateInstanceProfile",
        "iam:CreateRole"
    ],
    "Resource": [
        "arn:aws:iam:*:role/aws-elasticbeanstalk*",
        "arn:aws:iam:*:instance-profile/aws-elasticbeanstalk*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": [
        "arn:aws:iam:*:role/aws-service-role/autoscaling.amazonaws.com/
AWSServiceRoleForAutoScaling*"
    ],
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": "autoscaling.amazonaws.com"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": [
        "arn:aws:iam:*:role/aws-service-role/elasticbeanstalk.amazonaws.com/
AWSServiceRoleForElasticBeanstalk*"
    ],
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": "elasticbeanstalk.amazonaws.com"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "iam:AttachRolePolicy"
    ],
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "iam:PolicyArn": [
                "arn:aws:iam::aws:policy/AWSElasticBeanstalk*",
                "arn:aws:iam::aws:policy/service-role/AWSElasticBeanstalk*"
            ]
        }
    }
}
]
}

```

Elastic Beanstalk also provides a read-only managed policy named `AWSElasticBeanstalkReadOnlyAccess`. This policy allows a user to view, but not modify or create, Elastic Beanstalk environments.

For more information about user policies, see [Managing Elastic Beanstalk user policies \(p. 775\)](#).

Elastic Beanstalk platforms

AWS Elastic Beanstalk provides a variety of platforms on which you can build your applications. You design your web application to one of these platforms, and Elastic Beanstalk deploys your code to the platform version you selected to create an active application environment.

Elastic Beanstalk provides platforms for different programming languages, application servers, and Docker containers. Some platforms have multiple concurrently-supported versions.

Elastic Beanstalk also supports [custom platforms \(p. 38\)](#) that can be based on an AMI that you create and can include further customizations.

Topics

- [Elastic Beanstalk platforms glossary \(p. 25\)](#)
- [Shared responsibility model for Elastic Beanstalk platform maintenance \(p. 27\)](#)
- [Elastic Beanstalk platform support policy \(p. 28\)](#)
- [Elastic Beanstalk supported platforms \(p. 29\)](#)
- [Elastic Beanstalk Linux platforms \(p. 30\)](#)
- [Elastic Beanstalk custom platforms \(p. 38\)](#)
- [Deploying Elastic Beanstalk applications from Docker containers \(p. 50\)](#)
- [Creating and deploying Go applications on Elastic Beanstalk \(p. 83\)](#)
- [Creating and deploying Java applications on Elastic Beanstalk \(p. 94\)](#)
- [Creating and deploying .NET applications on Elastic Beanstalk \(p. 137\)](#)
- [Deploying Node.js applications to Elastic Beanstalk \(p. 192\)](#)
- [Creating and deploying PHP applications on Elastic Beanstalk \(p. 227\)](#)
- [Working with Python \(p. 288\)](#)
- [Creating and deploying Ruby applications on Elastic Beanstalk \(p. 314\)](#)

Elastic Beanstalk platforms glossary

Following are key terms related to AWS Elastic Beanstalk platforms and their lifecycle.

Runtime

The programming language-specific runtime software (framework, libraries, interpreter, vm, etc.) required to run your application code.

Elastic Beanstalk Components

Software components that Elastic Beanstalk adds to a platform to enable Elastic Beanstalk functionality. For example, the enhanced health agent is necessary for gathering and reporting health information.

Platform

A combination of an operating system (OS), runtime, web server, application server, and Elastic Beanstalk components. Platforms provide components that are available to run your application.

Platform Version

A combination of specific versions of an operating system (OS), runtime, web server, application server, and Elastic Beanstalk components. You create an Elastic Beanstalk environment based on a platform version and deploy your application to it.

A platform version has a semantic version number of the form *X.Y.Z*, where *X* is the major version, *Y* is the minor version, and *Z* is the patch version.

A platform version can be in one of the following states:

- *Supported* – A platform version that consists entirely of *supported components*. All components have not reached their End of Life (EOL), as designated by their respective suppliers (owners—AWS or third parties—or communities). They receive regular patch or minor updates from their suppliers. Elastic Beanstalk makes supported platform versions available to you for environment creation.
- *Retired* – A platform version with one or more *retired components*, which have reached their End of Life (EOL), as designated by their suppliers. Retired platform versions aren't available for use in Elastic Beanstalk environments for either new or existing customers.

For details about retired components, see [the section called “Platform support policy” \(p. 28\)](#).

Platform Branch

A line of platform versions sharing specific (typically major) versions of some of their components, such as the operating system (OS), runtime, or Elastic Beanstalk components. For example: *Python 3.6 running on 64bit Amazon Linux; IIS 10.0 running on 64bit Windows Server 2016*. Each successive platform version in the branch is an update to the previous one.

The latest platform version in each platform branch is available to you unconditionally for environment creation. Previous platform versions in the branch are still supported—you can create an environment based on a previous platform version if you've used it in an environment in the last 30 days. But these previous platform versions lack the most up-to-date components and aren't recommended for use.

A platform branch can be in one of the following states:

- *Supported* – A current platform branch. It consists entirely of *supported components*. It receives ongoing platform updates, and is recommended for use in production environments. For a list of supported platform branches, see [Elastic Beanstalk supported platforms](#) in the *AWS Elastic Beanstalk Platforms* guide.
- *Beta* – A preview, pre-release platform branch. It's experimental in nature. It may receive ongoing platform updates for a while, but has no long-term support. A beta platform branch isn't recommended for use in production environments. Use it only for evaluation. For a list of beta platform branches, see [Elastic Beanstalk Platform Versions in Public Beta](#) in the *AWS Elastic Beanstalk Platforms* guide.
- *Deprecated* – A platform branch with one or more *deprecated components*. It receives ongoing platform updates, but isn't recommended for use in production environments. For a list of deprecated platform branches, see [Elastic Beanstalk Platform Versions Scheduled for Retirement](#) in the *AWS Elastic Beanstalk Platforms* guide.
- *Retired* – A platform branch with one or more *retired components*. It doesn't receive platform updates anymore, and isn't recommended for use in production environments. Retired platform branches aren't listed in the *AWS Elastic Beanstalk Platforms* guide. Elastic Beanstalk doesn't make platform versions of retired platform branches available to you for environment creation.

A *supported component* has no retirement date scheduled by its supplier (owner or community). The supplier might be AWS or a third party. A *deprecated component* has a retirement date scheduled by its supplier. A *retired component* has reached End of Life (EOL) and is no longer supported by its supplier. For details about retired components, see [the section called “Platform support policy” \(p. 28\)](#).

If your environment uses a deprecated or retired platform branch, we recommend that you update it to a platform version in a supported platform branch. For details, see [the section called “Platform updates” \(p. 415\)](#).

Platform Update

A release of new platform versions that contain updates to some components of the platform—OS, runtime, web server, application server, and Elastic Beanstalk components. Platform updates follow semantic version taxonomy, and can have several levels:

- *Major update* – An update that has changes that are incompatible with existing platform versions. You might need to modify your application to run correctly on a new major version. A major update has a new major platform version number.
- *Minor update* – An update that adds functionality that is backward compatible with an existing platform version. You don't need to modify your application to run correctly on a new minor version. A minor update has a new minor platform version number.
- *Patch update* – An update that consists of maintenance releases (bug fixes, security updates, and performance improvements) that are backward compatible with an existing platform version. A patch update has a new patch platform version number.

Managed Updates

An Elastic Beanstalk feature that automatically applies patch and minor updates to the operating system (OS), runtime, web server, application server, and Elastic Beanstalk components for an Elastic Beanstalk supported platform version. A managed update applies a newer platform version in the same platform branch to your environment. You can configure managed updates to apply only patch updates, or minor and patch updates. You can also disable managed updates completely.

For more information, see [Managed platform updates \(p. 420\)](#).

Shared responsibility model for Elastic Beanstalk platform maintenance

AWS and our customers share responsibility for achieving a high level of software component security and compliance. This shared model reduces your operational burden.

For details, see the AWS [Shared Responsibility Model](#).

AWS Elastic Beanstalk helps you perform your side of the shared responsibility model by providing a *managed updates* feature. This feature automatically applies patch and minor updates for an Elastic Beanstalk supported platform version. If a managed update fails, Elastic Beanstalk notifies you of the failure to ensure that you are aware of it and can take immediate action.

For more information, see [Managed platform updates \(p. 420\)](#).

In addition, Elastic Beanstalk does the following:

- Publishes its [platform support policy \(p. 28\)](#) and retirement schedule for the coming 12 months.
- Releases patch, minor, and major updates of operating system (OS), runtime, application server, and web server components typically within 30 days of their availability. Elastic Beanstalk is responsible for creating updates to Elastic Beanstalk components that are present on its supported platform versions. All other updates come directly from their suppliers (owners or community).

You are responsible to do the following:

- Update all the components that you control (identified as **Customer** in the AWS [Shared Responsibility Model](#)). This includes ensuring the security of your application, your data, and any components that your application requires and that you downloaded.

- Ensure that your Elastic Beanstalk environments are running on a supported platform version, and migrate any environment running on a retired platform version to a supported version.
- Resolve all issues that come up in failed managed update attempts and retry the update.
- Patch the OS, runtime, application server, and web server yourself if you opted out of Elastic Beanstalk managed updates. You can do this by [applying platform updates manually \(p. 415\)](#) or directly patching the components on all relevant environment resources.
- Manage the security and compliance of any AWS services that you use outside of Elastic Beanstalk according to the AWS [Shared Responsibility Model](#).

Elastic Beanstalk platform support policy

AWS Elastic Beanstalk provides a variety of platforms for running applications on AWS. Elastic Beanstalk supports platform branches that still receive ongoing minor and patch updates from their suppliers (owners or community). For a complete definition of related terms, see [Elastic Beanstalk platforms glossary \(p. 25\)](#).

When a component (operating system [OS], runtime, application server, or web server) of a supported platform branch is marked End of Life (EOL) by its supplier, Elastic Beanstalk marks the platform branch as retired. When a platform branch is marked as retired, Elastic Beanstalk no longer makes it available to both existing and new Elastic Beanstalk customers for deployments to new environments. Retired platform branches are available to existing customer environments for a period of 90 days from the published retirement date.

Elastic Beanstalk isn't able to provide security updates, technical support, or hotfixes for retired platform branches due to the supplier marking their component EOL. For existing customers running an Elastic Beanstalk environment on a retired platform version beyond the 90 day period, Elastic Beanstalk may need to automatically remove the Elastic Beanstalk components and transfer ongoing management and support responsibility of the running application and associated AWS resources to the customer. To continue to benefit from important security, performance, and functionality enhancements offered by component suppliers in more recent releases, we strongly encourage you to update all your Elastic Beanstalk environments to a supported platform version.

Retired platform branch schedule

The following tables list existing platform components that are either marked as retired or have retirement dates scheduled in the next 12 months. The tables provide the availability end date for Elastic Beanstalk platform branches that contain these components.

Web server versions

Web server version	Availability end date		
Apache HTTP Server 2.2	October 31, 2020		
Nginx 1.12.2	October 31, 2020		

Runtime versions

Runtime version	Availability end date		
Go 1.3–1.10	October 31, 2020		

Runtime version	Availability end date		
Java 6	October 31, 2020		
Node.js 4.x–8.x	October 31, 2020		
PHP 5.4–5.6	October 31, 2020		
PHP 7.0–7.1	October 31, 2020		
Python 2.6, 2.7, 3.4	October 31, 2020		
Ruby 1.9.3	October 31, 2020		
Ruby 2.0–2.3	October 31, 2020		

Application server versions

Application server version	Availability end date		
Tomcat 6	October 31, 2020		
Tomcat 8	October 31, 2020		

Retired platform branches

The following tables list platform components that were marked as retired in the past. The tables provide the date on which Elastic Beanstalk retired platform branches that contained these components.

Operating System (OS) versions

OS version	Platform retirement date		
Windows Server 2008 R2	October 28, 2019		

Elastic Beanstalk supported platforms

AWS Elastic Beanstalk provides a variety of platforms on which you can build your applications. You design your web application to one of these platforms, and Elastic Beanstalk deploys your code to the platform version you selected to create an active application environment.

Elastic Beanstalk provides platforms for programming languages (Go, Java, Node.js, PHP, Python, Ruby), application servers (Tomcat, Passenger, Puma), and Docker containers. Some platforms have multiple concurrently-supported versions.

Elastic Beanstalk provisions the resources needed to run your application, including one or more Amazon EC2 instances. The software stack running on the Amazon EC2 instances depends on the specific platform version you've selected for your environment.

You can use the solution stack name listed under the platform version name to launch an environment with the [EB CLI \(p. 852\)](#), [Elastic Beanstalk API](#), or [AWS CLI](#). You can also retrieve solution stack names

from the service with the `ListAvailableSolutionStacks` API (`aws elasticbeanstalk list-available-solution-stacks` in the AWS CLI). This operation returns all of the solution stacks that you can use to create an environment.

Note

Each platform has supported and retired platform versions. You can always create an environment based on a supported platform version. Retired platform versions are available only to existing customer environments for a period of 90 days from the published retirement date. For a list of published platform version retirement dates, see [Retired platform branch schedule \(p. 28\)](#).

When Elastic Beanstalk updates a platform, previous platform versions are still supported, but they lack the most up-to-date components and aren't recommended for use. We recommend that you transition to the latest platform version. You can still create an environment based on a previous platform version if you've used it in an environment in the last 30 days (using the same account, in the same region).

You can customize and configure the software that your application depends on in your platform. Learn more at [Customizing software on Linux servers \(p. 603\)](#) and [Customizing software on Windows servers \(p. 615\)](#). Detailed release notes are available for recent releases at [AWS Elastic Beanstalk Release Notes](#).

Supported platform versions

All current platform versions are listed in [Elastic Beanstalk Supported Platforms](#). Each platform-specific section also points to the *platform history*, a list of previous platform versions. For direct access to the version list of a specific platform, use one of the following links.

- [Single Container Docker](#)
- [Multicontainer Docker](#)
- [Preconfigured Docker](#)
- [Go](#)
- [Java SE](#)
- [Java with Tomcat](#)
- [.NET on Windows Server with IIS](#)
- [Node.js](#)
- [PHP](#)
- [Python](#)
- [Ruby](#)

Elastic Beanstalk Linux platforms

AWS Elastic Beanstalk provides a variety of platforms on which you can build your applications. You design your web application to one of these platforms, and Elastic Beanstalk deploys your code to the platform version you selected to create an active application environment.

Elastic Beanstalk provides platforms for different programming languages, application servers, and Docker containers. Some platforms have multiple concurrently-supported versions.

For full coverage of Elastic Beanstalk platforms, see [Elastic Beanstalk platforms \(p. 25\)](#).

Many of the platforms that Elastic Beanstalk supports are based on the Linux operating system (OS). Specifically, these platforms are based on Amazon Linux, a Linux distribution provided by AWS. Elastic Beanstalk Linux platforms use Amazon Elastic Compute Cloud (Amazon EC2) instances, and these

instances run Amazon Linux. To learn more, see [Amazon Linux](#) in the *Amazon EC2 User Guide for Linux Instances*.

The Elastic Beanstalk Linux platforms provide a lot of functionality out of the box. You can extend the platforms in several ways to support your application. For details, see [the section called “Extending Linux platforms” \(p. 31\)](#).

Topics

- [Linux platform versions \(p. 31\)](#)
- [List of Elastic Beanstalk Linux platforms \(p. 31\)](#)
- [Extending Elastic Beanstalk Linux platforms \(p. 31\)](#)

Linux platform versions

AWS provides two versions of Amazon Linux: [Amazon Linux 2](#) and [Amazon Linux AMI](#). Some key improvements in Amazon Linux 2 compared to Amazon Linux AMI are:

- Amazon Linux 2 offers long-term support.
- Amazon Linux 2 is available as virtual machine images for on-premises development and testing.
- Amazon Linux 2 comes with updated components: the Linux kernel, C library, compiler, and tools. It also uses the systemd service and systems manager as opposed to System V init system in Amazon Linux AMI.

Elastic Beanstalk maintains platform versions with both Amazon Linux versions. For details about supported platform versions, see [Elastic Beanstalk supported platforms \(p. 29\)](#).

Note

Amazon Linux 2 platform versions are incompatible with previous Amazon Linux AMI platform versions. If you're migrating your Elastic Beanstalk application to Amazon Linux 2, read [the section called “Upgrade to Amazon Linux 2” \(p. 426\)](#).

List of Elastic Beanstalk Linux platforms

The following list mentions the Linux platforms that Elastic Beanstalk supports for different programming languages, as well as for Docker containers, and links to chapters about them in this developer guide.

- [Docker \(p. 50\)](#)
- [Go \(p. 83\)](#)
- [Java \(p. 94\)](#)
- [Node.js \(p. 192\)](#)
- [PHP \(p. 227\)](#)
- [Python \(p. 288\)](#)
- [Ruby \(p. 314\)](#)

Extending Elastic Beanstalk Linux platforms

The [AWS Elastic Beanstalk Linux platforms \(p. 30\)](#) provide a lot of functionality out of the box to support developing and running your application. When necessary, you can extend the platforms in several ways to configure options, install software, add files and start-up commands, provide build and runtime instructions, and add initialization scripts that run in various provisioning stages of your environment's Amazon Elastic Compute Cloud (Amazon EC2) instances.

Buildfile and Procfile

Some platforms allow you to customize how you build or prepare your application, and to specify the processes that run your application. Each individual platform topic specifically mentions *Buildfile* and/or *Procfile* if the platform supports them. Look for your specific platform under [Platforms \(p. 25\)](#).

For all supporting platforms, syntax and semantics are identical, and are as described on this page. Individual platform topics mention specific usage of these files for building and running applications in their respective languages.

Buildfile

To specify a custom build and configuration command for your application, place a file named `Buildfile` in the root directory of your application source. The file name is case sensitive. Use the following syntax for your `Buildfile`.

```
<process_name>: <command>
```

The command in your `Buildfile` must match the following regular expression: `^[A-Za-z0-9_-]+:\s*[\s].*$`

Elastic Beanstalk doesn't monitor the application that is run with a `Buildfile`. Use a `Buildfile` for commands that run for short periods and terminate after completing their tasks. For long-running application processes that should not exit, use a [Procfile \(p. 32\)](#).

All paths in the `Buildfile` are relative to the root of the source bundle. In the following example of a `Buildfile`, `build.sh` is a shell script located at the root of the source bundle.

Example Buildfile

```
make: ./build.sh
```

If you want to provide custom build steps, we recommend that you use `predeploy` platform hooks for anything but the simplest commands, instead of a `Buildfile`. Platform hooks allow richer scripts and better error handling. Platform hooks are described in the previous section.

Procfile

To specify custom commands to start and run your application, place a file named `Procfile` in the root directory of your application source. The file name is case sensitive. Use the following syntax for your `Procfile`. You can specify one or more commands.

```
<process_name1>: <command1>
<process_name2>: <command2>
...
```

Each line in your `Procfile` must match the following regular expression: `^[A-Za-z0-9_-]+:\s*[\s].*$`

Use a `Procfile` for long-running application processes that shouldn't exit. Elastic Beanstalk expects processes run from the `Procfile` to run continuously. Elastic Beanstalk monitors these processes and restarts any process that terminates. For short-running processes, use a [Buildfile \(p. 32\)](#).

All paths in the `Procfile` are relative to the root of the source bundle. The following example `Procfile` defines three processes. The first one, called `web` in the example, is the *main web application*.

Example Procfile

```
web: bin/myserver
```

```
cache: bin/mycache  
foo: bin/fooapp
```

Elastic Beanstalk configures the proxy server to forward requests to your main web application on port 5000, and you can configure this port number. A common use for a `Procfile` is to pass this port number to your application as a command argument. For details about proxy configuration, expand the *Reverse proxy configuration* section on this page.

Elastic Beanstalk captures standard output and error streams from `Procfile` processes in log files. Elastic Beanstalk names the log files after the process and stores them in `/var/log`. For example, the web process in the preceding example generates logs named `web-1.log` and `web-1.error.log` for `stdout` and `stderr`, respectively.

Platform hooks

Platform hooks are specifically designed to extend your environment's platform. These are executable files that you deploy as part of your application's source code, and Elastic Beanstalk runs during various instance provisioning stages.

Note

Platform hooks aren't supported on Amazon Linux AMI platform versions (preceding Amazon Linux 2).

Place your custom scripts and other executable files under the `.platform/hooks` directory in your source bundle, in one of the following subdirectories.

- `prebuild` – Files here run after the Elastic Beanstalk platform engine downloads and extracts the application source bundle, and before it sets up and configures the application and web server.

The `prebuild` files run after running commands found in the [commands \(p. 608\)](#) section of any configuration file and before running `Buildfile` commands.

- `predeploy` – Files here run after the Elastic Beanstalk platform engine sets up and configures the application and web server, and before it deploys them to their final runtime location.

The `predeploy` files run after running commands found in the [container_commands \(p. 611\)](#) section of any configuration file and before running `Procfile` commands.

- `postdeploy` – Files here run after the Elastic Beanstalk platform engine deploys the application and proxy server.

This is the last deployment workflow step.

Executable files can be binary files, or script files starting with a `#!` line containing their interpreter path, such as `#!/bin/bash`. All files have to have execute permission. Use `chmod +x` to set execute permission on your hook files.

Elastic Beanstalk executes files in each one of these directories in lexicographical order of file names. All files run as the `root` user. The current working directory (`cwd`) for platform hooks is the application's root directory. For `prebuild` and `predeploy` files it's the application staging directory, and for `postdeploy` files it's the current application directory. If one of the files fails (exits with a non-zero exit code), the deployment aborts and fails.

Executed files have access to all environment properties that you've defined in application options, and to the system environment variables `HOME`, `PATH`, and `PORT`.

Configuration files

You can add [configuration files \(p. 600\)](#) to the `.ebextensions` directory of your application's source code to configure various aspects of your Elastic Beanstalk environment. Among other things,

configuration files let you customize software and other files on your environment's instances and run initialization commands on the instances. For more information, see [the section called "Linux server" \(p. 603\)](#).

You can also set [configuration options \(p. 536\)](#) using configuration files. Many of the options control platform behavior, and some of these options are [platform specific \(p. 592\)](#).

On Amazon Linux 2 platforms, we recommend using *Buildfile*, *Procfile*, and *platform hooks* to configure and run custom code on your environment instances during instance provisioning. These mechanisms are described in the previous sections on this page. You can still use commands and container commands in `.ebextensions` configuration files, but they aren't as easy to work with. For example, writing command scripts inside a YAML file can be challenging from a syntax standpoint. You still need to use `.ebextensions` configuration files for any script that needs a reference to a AWS CloudFormation resource.

Reverse proxy configuration

Some platforms allow you to configure the nginx reverse proxy on your environment instances. Specifically, all Amazon Linux 2 platform versions use nginx as their reverse proxy server and support nginx configuration as described here.

Note

On Amazon Linux AMI platform versions (preceding Amazon Linux 2) you might have to configure nginx differently. You can find these legacy details under the [respective platform topics \(p. 25\)](#) in this guide.

Elastic Beanstalk uses nginx as the reverse proxy to map your application to your Elastic Load Balancing load balancer on port 80. Elastic Beanstalk provides a default nginx configuration that you can extend or override completely with your own configuration.

To extend the Elastic Beanstalk default nginx configuration, add `.conf` configuration files to a folder named `.platform/nginx/conf.d/` in your application source bundle. The Elastic Beanstalk nginx configuration includes `.conf` files in this folder automatically.

```
~/workspace/my-app/  
|-- .platform  
|   |-- nginx  
|       |-- conf.d  
|           |-- myconf.conf  
|-- other source files
```

To override the Elastic Beanstalk default nginx configuration completely, include a configuration in your source bundle at `.platform/nginx/nginx.conf`:

```
~/workspace/my-app/  
|-- .platform  
|   |-- nginx  
|       |-- nginx.conf  
|-- other source files
```

If you override the Elastic Beanstalk nginx configuration, add the following line to your `nginx.conf` to pull in the Elastic Beanstalk configurations for [Enhanced health reporting and monitoring \(p. 691\)](#), automatic application mappings, and static files.

```
include conf.d/elasticbeanstalk/*.conf;
```

Elastic Beanstalk configures the proxy server on your environment's instances to forward web traffic to the main web application on the root URL of the environment; for example, `http://my-env.elasticbeanstalk.com`.

By default, Elastic Beanstalk configures the proxy to forward requests to your main web application on port 5000. You can configure this port number by setting the `PORT` environment property using the [aws:elasticbeanstalk:application:environment](#) (p. 570) namespace in a configuration file, as shown in the following example.

```
option_settings:
- namespace: aws:elasticbeanstalk:application:environment
  option_name: PORT
  value: <main_port_number>
```

For more information about setting environment variables for your application, see [the section called "Option settings"](#) (p. 601).

Your application should listen on the port that is configured for it in the proxy. If you change the default port using the `PORT` environment property, your code can access it by reading the value of the `PORT` environment variable. For example, call `os.Getenv("PORT")` in Go, or `System.getenv("PORT")` in Java. If you configure your proxy to send traffic to multiple application processes, you can configure several environment properties, and use their values in both proxy configuration and your application code. Another option is to pass the port value to the process as a command argument in the `Procfile`. For details on that, expand the *Buildfile and Procfile* section on this page.

If you're migrating your Elastic Beanstalk application to an Amazon Linux 2 platform, be sure to also read the information in [the section called "Upgrade to Amazon Linux 2"](#) (p. 426).

Application example with extensions

The following example demonstrates an application source bundle with several extensibility features that Elastic Beanstalk Amazon Linux 2 platforms support: a `Procfile`, `.ebextensions` configuration files, custom hooks, and proxy configuration files.

```
~/my-app/
|-- web.jar
|-- Procfile
|-- readme.md
|-- .ebextensions/
|   |-- options.config          # Option settings
|   |-- cloudwatch.config      # Other .ebextensions sections, for example files and
|   |                           container commands
|-- .platform/
|   |-- nginx/                 # Proxy configuration
|   |   |-- nginx.conf
|   |   |-- conf.d/
|   |   |-- custom.conf
|   |-- hooks/                 # Platform hooks
|   |   |-- prebuild/
|   |   |   |-- 01_set_secrets.sh
|   |   |   |-- 12_update_permissions.sh
|   |   |-- predeploy/
|   |   |   |-- 01_some_service_stop.sh
|   |   |-- postdeploy/
|   |   |   |-- 01_set_tmp_file_permissions.sh
|   |   |   |-- 50_run_something_after_deployment.sh
|   |   |   |-- 99_some_service_start.sh
```

Note

Some of these extensions aren't supported on Amazon Linux AMI platform versions (preceding Amazon Linux 2).

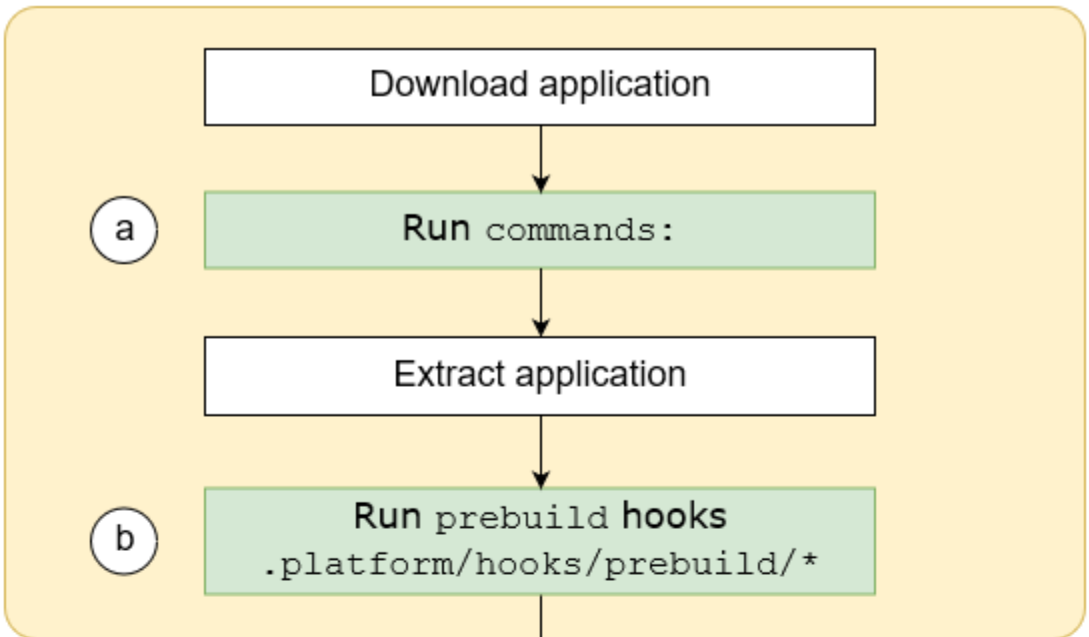
Instance deployment workflow

With many ways to extend your environment's platform, it's useful to know what happens whenever Elastic Beanstalk provisions an instance or runs a deployment to an instance. The following diagram shows this entire deployment workflow. It depicts the different phases in a deployment, and the steps that Elastic Beanstalk takes in each phase.

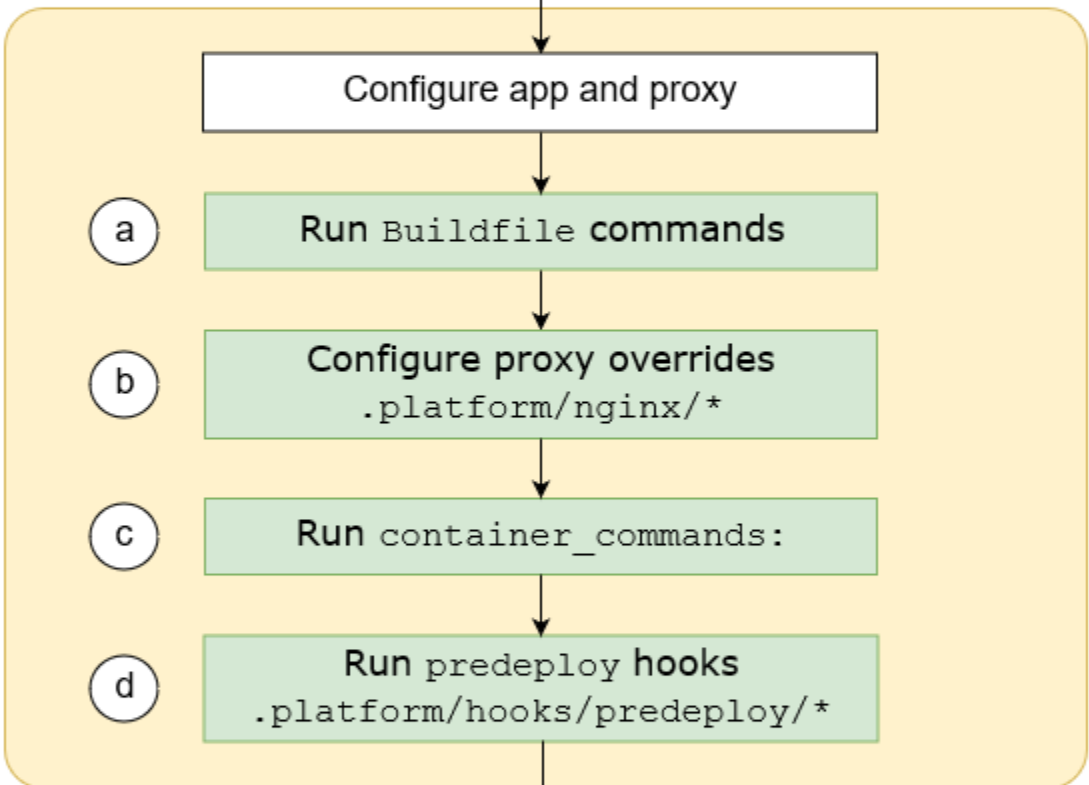
Note

The diagram doesn't represent the complete set of steps that Elastic Beanstalk executes on environment instances during deployment. We provide this diagram for illustration, to provide you with the order and context for the execution of your customizations.

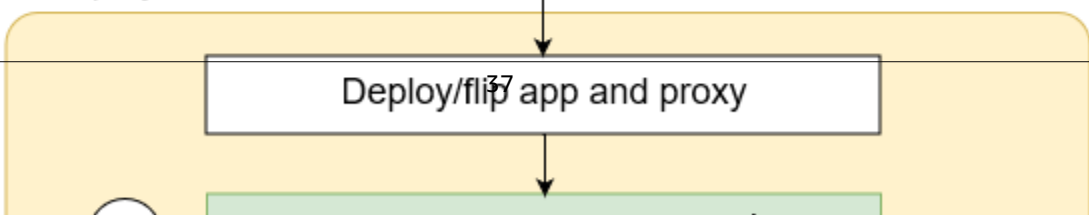
1. Initial steps



2. Configure



3. Deploy



The following list details the deployment phases and steps.

1. Initial steps

Elastic Beanstalk downloads and extracts your application. After each one of these steps, Elastic Beanstalk executes one of the extensibility steps.

- a. Runs commands found in the [commands: \(p. 608\)](#) section of any configuration file.
- b. Runs any executable files found in the `.platform/hooks/prebuild` directory of your source bundle.

2. Configure

Elastic Beanstalk configures your application and the proxy server.

- a. Runs the commands found in the `Buildfile` in your source bundle.
- b. Copies your custom proxy configuration files, if you have any in the `.platform/nginx` directory of your source bundle, to their runtime location.
- c. Runs commands found in the [container_commands: \(p. 611\)](#) section of any configuration file.
- d. Runs any executable files found in the `.platform/hooks/predeploy` directory of your source bundle.

3. Deploy

Elastic Beanstalk deploys and runs your application and the proxy server.

- a. Runs the command found in the `Profile` file in your source bundle.
- b. Runs or reruns the proxy server with your custom proxy configuration files, if you have any.
- c. Runs any executable files found in the `.platform/hooks/postdeploy` directory of your source bundle.

Elastic Beanstalk custom platforms

AWS Elastic Beanstalk supports custom platforms. A custom platform is a more advanced customization than a [custom image \(p. 647\)](#) in several ways. A custom platform lets you develop an entire new platform from scratch, customizing the operating system, additional software, and scripts that Elastic Beanstalk runs on platform instances. This flexibility enables you to build a platform for an application that uses a language or other infrastructure software, for which Elastic Beanstalk doesn't provide a managed platform. Compare that to custom images, where you modify an Amazon Machine Image (AMI) for use with an existing Elastic Beanstalk platform, and Elastic Beanstalk still provides the platform scripts and controls the platform's software stack. In addition, with custom platforms you use an automated, scripted way to create and maintain your customization, whereas with custom images you make the changes manually over a running instance.

Note

Elastic Beanstalk doesn't support custom platforms based on an Amazon Linux 2 AMI.

To create a custom platform, you build an AMI from one of the supported operating systems—Ubuntu, RHEL, or Amazon Linux (see the `flavor` entry in [Platform.yaml file format \(p. 46\)](#) for the exact version numbers)—and add further customizations. You create your own Elastic Beanstalk platform using [Packer](#), which is an open-source tool for creating machine images for many platforms, including AMIs for use with Amazon Elastic Compute Cloud (Amazon EC2). An Elastic Beanstalk platform comprises an AMI configured to run a set of software that supports an application, and metadata that can include custom configuration options and default configuration option settings.

Elastic Beanstalk manages Packer as a separate built-in platform, and you don't need to worry about Packer configuration and versions.

You create a platform by providing Elastic Beanstalk with a Packer template, and the scripts and files that the template invokes to build an AMI. These components are packaged with a [platform definition](#)

[file \(p. 39\)](#), which specifies the template and metadata, into a ZIP archive, known as a [platform definition archive \(p. 43\)](#).

When you create a custom platform, you launch a single instance environment without an Elastic IP that runs Packer. Packer then launches another instance to build an image. You can reuse this environment for multiple platforms and multiple versions of each platform.

Note

Custom platforms are AWS Region specific. If you use Elastic Beanstalk in multiple Regions, you must create your platforms separately in each Region.

In certain circumstances, instances launched by Packer are not cleaned up and have to be manually terminated. To learn how to manually clean up these instances, see [Packer instance cleanup \(p. 46\)](#).

Users in your account can use your custom platforms by specifying a [platform ARN \(p. 781\)](#) during environment creation. These ARNs are returned by the **eb platform create** command that you used to create the custom platform.

Each time you build your custom platform, Elastic Beanstalk creates a new platform version. Users can specify a platform by name to get only the latest version of the platform, or include a version number to get a specific version.

For example, to deploy the latest version of the custom platform with the ARN **MyCustomPlatformARN**, which could be version 3.0, your EB CLI command line would look like this:

```
eb create -p MyCustomPlatformARN
```

To deploy version 2.1 your EB CLI command line would look like this:

```
eb create -p MyCustomPlatformARN --version 2.1
```

You can apply tags to a custom platform version when you create it, and edit tags of existing custom platform versions. For details, see [Tagging custom platform versions \(p. 48\)](#).

Creating a custom platform

To create a custom platform, the root of your application must include a platform definition file, `platform.yaml`, which defines the type of builder used to create the custom platform. The format of this file is described in [Platform.yaml file format \(p. 46\)](#). You can create your custom platform from scratch, or use one of the [sample custom platforms \(p. 39\)](#) as a starting point.

Using a sample custom platform

One alternative to creating your own custom platform is to use one of the platform definition archive samples to bootstrap your custom platform. The only items you have to configure in the samples before you can use them are a source AMI and a Region.

Note

Do not use an unmodified sample custom platform in production. The goal of the samples is to show some of the functionality available for a custom platform, but they have not been hardened for production use.

[NodePlatform_Ubuntu.zip](#)

This custom platform is based on **Ubuntu 16.04** and supports **Node.js 4.4.4**. We use this custom platform for the examples in this section.

[NodePlatform_RHEL.zip](#)

This custom platform is based on **RHEL 7.2** and supports **Node.js 4.4.4**.

[NodePlatform_AmazonLinux.zip](#)

This custom platform is based on **Amazon Linux 2016.09.1** and supports **Node.js 4.4.4**.

[TomcatPlatform_Ubuntu.zip](#)

This custom platform is based on **Ubuntu 16.04** and supports **Tomcat 7/Java 8**.

[CustomPlatform_NodeSampleApp.zip](#)

A Node.js sample that uses **express** and **ejs** to display a static webpage.

[CustomPlatform_TomcatSampleApp.zip](#)

A Tomcat sample that displays a static webpage when deployed.

Download the sample platform definition archive: `NodePlatform_Ubuntu.zip`. This file contains a platform definition file, Packer template, scripts that Packer runs during image creation, and scripts and configuration files that Packer copies onto the builder instance during platform creation.

Example `NodePlatform_Ubuntu.zip`

-- builder	Contains files used by Packer to create the custom platform
-- custom_platform.json	Packer template
-- platform.yaml	Platform definition file
-- ReadMe.txt	Briefly describes the sample

The platform definition file, `platform.yaml`, tells Elastic Beanstalk the name of the Packer template, `custom_platform.json`.

```
version: "1.0"

provisioner:
  type: packer
  template: custom_platform.json
  flavor: ubuntu1604
```

The Packer template tells Packer how to build the AMIs for the platform, using an [Ubuntu AMI](#) as a base for the platform image for HVM instance types. The `provisioners` section tells Packer to copy all files in the `builder` folder within the archive to the instance, and to run the `builder.sh` script on the instance. When the scripts complete, Packer creates an image from the modified instance.

Elastic Beanstalk creates three environment variables that can be used to tag AMIs in Packer:

`AWS_EB_PLATFORM_ARN`

The ARN of the custom platform.

`AWS_EB_PLATFORM_NAME`

The name of the custom platform.

`AWS_EB_PLATFORM_VERSION`

The version of the custom platform.

The sample `custom_platform.json` file uses these variables to define the following values that it uses in the scripts:

- `platform_name`, which is set by `platform.yaml`
- `platform_version`, which is set by `platform.yaml`
- `platform_arn`, which is set by the main build script, `builder.sh`, which is shown at the end of the sample `custom_platform.json` file.

The `custom_platform.json` file contains two properties that you have to provide values for: `source_ami` and `region`. For details about choosing the right AMI and Region values, see [Updating Packer template](#) in the `eb-custom-platforms-samples` GitHub repository.

Example `custom_platform.json`

```
{
  "variables": {
    "platform_name": "{{env `AWS_EB_PLATFORM_NAME`}}",
    "platform_version": "{{env `AWS_EB_PLATFORM_VERSION`}}",
    "platform_arn": "{{env `AWS_EB_PLATFORM_ARN`}}"
  },
  "builders": [
    {
      ...
      "region": "",
      "source_ami": "",
      ...
    }
  ],
  "provisioners": [
    {...},
    {
      "type": "shell",
      "execute_command": "chmod +x {{ .Path }}; {{ .Vars }} sudo {{ .Path }}",
      "scripts": [
        "builder/builder.sh"
      ]
    }
  ]
}
```

The scripts and other files that you include in your platform definition archive will vary greatly depending on the modifications that you want to make to the instance. The sample platform includes the following scripts:

- `00-sync-apt.sh` – Commented out: `apt -y update`. We commented out the command because it prompts the user for input, which breaks the automated package update. This might be an Ubuntu issue. However, running `apt -y update` is still recommended as a best practice. For this reason, we left the command in the sample script for reference.
- `01-install-nginx.sh` – Installs `nginx`.
- `02-setup-platform.sh` – Installs `wget`, `tree`, and `git`. Copies hooks and [logging configurations \(p. 732\)](#) to the instance, and creates the following directories:
 - `/etc/SampleNodePlatform` – Where the container configuration file is uploaded during deployment.
 - `/opt/elasticbeanstalk/deploy/appsource/` – Where the `00-unzip.sh` script uploads application source code during deployment (see the [Platform scripts \(p. 45\)](#) section for information about this script).
 - `/var/app/staging/` – Where application source code is processed during deployment.
 - `/var/app/current/` – Where application source code runs after processing.
 - `/var/log/nginx/healthd/` – Where the [enhanced health agent \(p. 693\)](#) writes logs.
 - `/var/nodejs` – Where the Node.js files are uploaded during deployment.

Use the EB CLI to create your first custom platform with the sample platform definition archive.

To create a custom platform

1. [Install the EB CLI \(p. 853\)](#).
2. Create a directory in which you will extract the sample custom platform.

```
~$ mkdir ~/custom-platform
```

3. Extract `NodePlatform_Ubuntu.zip` to the directory, and then change to the extracted directory.

```
~$ cd ~/custom-platform
~/custom-platform$ unzip ~/NodePlatform_Ubuntu.zip
~/custom-platform$ cd NodePlatform_Ubuntu
```

4. Edit the `custom_platform.json` file, and provide values for the `source_ami` and `region` properties. For details, see [Updating Packer template](#).
5. Run [eb platform init \(p. 919\)](#) and follow the prompts to initialize a platform repository.

You can shorten `eb platform` to `ebp`.

Note

Windows PowerShell uses `ebp` as a command alias. If you're running the EB CLI in Windows PowerShell, use the long form of this command: `eb platform`.

```
~/custom-platform$ eb platform init
```

This command also creates the directory `.elasticbeanstalk` in the current directory and adds the configuration file `config.yml` to the directory. Don't change or delete this file, because Elastic Beanstalk relies on it when creating the custom platform.

By default, `eb platform init` uses the name of the current folder as the name of the custom platform, which would be `custom-platform` in this example.

6. Run [eb platform create \(p. 918\)](#) to launch a Packer environment and get the ARN of the custom platform. You'll need this value later when you create an environment from the custom platform.

```
~/custom-platform$ eb platform create
...
```

By default, Elastic Beanstalk creates the instance profile `aws-elasticbeanstalk-custom-platform-ec2-role` for custom platforms. If, instead, you want to use an existing instance profile, add the option `-ip INSTANCE_PROFILE` to the [eb platform create \(p. 918\)](#) command.

Note

Packer will fail to create a custom platform if you use the Elastic Beanstalk default instance profile `aws-elasticbeanstalk-ec2-role`.

The EB CLI shows event output of the Packer environment until the build is complete. You can exit the event view by pressing `Ctrl+C`.

7. You can check the logs for errors using the [eb platform logs \(p. 921\)](#) command.

```
~/custom-platform$ eb platform logs
...
```

8. You can check on the process later with [eb platform events \(p. 919\)](#).

```
~/custom-platform$ eb platform events
```

```
...
```

9. Check the status of your platform with [eb platform status](#) (p. 921).

```
~/custom-platform$ eb platform status  
...
```

When the operation completes, you have a platform that you can use to launch an Elastic Beanstalk environment.

You can use the custom platform when creating an environment from the console. See [The create new environment wizard](#) (p. 365).

To launch an environment on your custom platform

1. Create a directory for your application.

```
~$ mkdir custom-platform-app  
~$ cd ~/custom-platform-app
```

2. Initialize an application repository.

```
~/custom-platform-app$ eb init  
...
```

3. Download the sample application [NodeSampleApp.zip](#).
4. Extract the sample application.

```
~/custom-platform-app$ unzip ~/NodeSampleApp.zip
```

5. Run **eb create -p *CUSTOM-PLATFORM-ARN***, where *CUSTOM-PLATFORM-ARN* is the ARN returned by an **eb platform create** command, to launch an environment running your custom platform.

```
~/custom-platform-app$ eb create -p CUSTOM-PLATFORM-ARN  
...
```

Platform definition archive contents

A platform definition archive is the platform equivalent of an [application source bundle](#) (p. 342). The platform definition archive is a ZIP file that contains a platform definition file, a Packer template, and the scripts and files used by the Packer template to create your platform.

Note

When you use the EB CLI to create a custom platform, the EB CLI creates a platform definition archive from the files and folders in your platform repository, so you don't need to create the archive manually.

The platform definition file is a YAML-formatted file that must be named `platform.yaml` and be in the root of your platform definition archive. See [Creating a custom platform](#) (p. 39) for a list of required and optional keys supported in a platform definition file.

You don't need to name the Packer template in a specific way, but the name of the file must match the provisioner template specified in the platform definition file. See the official [Packer documentation](#) for instructions on creating Packer templates.

The other files in your platform definition archive are scripts and files used by the template to customize an instance before creating an AMI.

Custom platform hooks

Elastic Beanstalk uses a standardized directory structure for hooks on custom platforms. These are scripts that are run during lifecycle events and in response to management operations: when instances in your environment are launched, or when a user initiates a deployment or uses the restart application server feature.

Place scripts that you want hooks to trigger in one of the subfolders of the `/opt/elasticbeanstalk/hooks/` folder.

Warning

Using custom platform hooks on managed platforms isn't supported. Custom platform hooks are designed for custom platforms. On Elastic Beanstalk managed platforms they might work differently or have some issues, and behavior might differ across platforms. On Amazon Linux AMI platforms (preceding Amazon Linux 2), they might still work in useful ways in some cases; use them with caution.

Custom platform hooks are a legacy feature that exists on Amazon Linux AMI platforms. On Amazon Linux 2 platforms, custom platform hooks in the `/opt/elasticbeanstalk/hooks/` folder are entirely discontinued. Elastic Beanstalk doesn't read or execute them. Amazon Linux 2 platforms support a new kind of platform hooks, specifically designed to extend Elastic Beanstalk managed platforms. You can add custom scripts and programs directly to a hooks directory in your application source bundle. Elastic Beanstalk runs them during various instance provisioning stages. For more information, expand the *Platform Hooks* section in [the section called "Extending Linux platforms" \(p. 31\)](#).

Hooks are organized into the following folders:

- `appdeploy` — Scripts run during an application deployment. Elastic Beanstalk performs an application deployment when new instances are launched and when a client initiates a new version deployment.
- `configdeploy` — Scripts run when a client performs a configuration update that affects the software configuration on instance, for example, by setting environment properties or enabling log rotation to Amazon S3.
- `restartappserver` — Scripts run when a client performs a restart app server operation.
- `preinit` — Scripts run during instance bootstrapping.
- `postinit` — Scripts run after instance bootstrapping.

The `appdeploy`, `configdeploy`, and `restartappserver` folders contain `pre`, `enact`, and `post` subfolders. In each phase of an operation, all scripts in the `pre` folder are run in alphabetical order, then those in the `enact` folder, and then those in the `post` folder.

When an instance is launched, Elastic Beanstalk runs `preinit`, `appdeploy`, and `postinit`, in this order. On subsequent deployments to running instances, Elastic Beanstalk runs `appdeploy` hooks. `configdeploy` hooks are run when a user updates instance software configuration settings. `restartappserver` hooks are run only when the user initiates an application server restart.

When your scripts encounter errors, they can exit with a non-zero status and write to `stderr` to fail the operation. The message that you write to `stderr` will appear in the event that is output when the operation fails. Elastic Beanstalk also captures this information in the log file `/var/log/eb-activity.log`. If you don't want to fail the operation, return 0 (zero). Messages that you write to `stderr` or `stdout` appear in the [deployment logs \(p. 732\)](#), but won't appear in the event stream unless the operation fails.

Platform scripts

Elastic Beanstalk installs the shell script `get-config` that you can use to get environment variables and other information in hooks that run on-instance in environments launched with your custom platform.

This tool is available at `/opt/elasticbeanstalk/bin/get-config`.

Note

The `get-config` tool is only available on platform versions based on Amazon Linux AMI (preceding Amazon Linux 2).

You can use `get-config` in the following ways:

- `get-config optionsettings` – Returns a JSON object listing the configuration options set on the environment, organized by namespace.

```
$ /opt/elasticbeanstalk/bin/get-config optionsettings
{"aws:elasticbeanstalk:container:php:phpini":
{"memory_limit":"256M","max_execution_time":"60","display_errors":"Off","composer_options":"","allow_
{"LogPublicationControl":"false"},"aws:elasticbeanstalk:application:environment":
{"TESTPROPERTY":"testvalue"}}
```

To return a specific configuration option, use the `-n` option to specify a namespace, and the `-o` option to specify an option name.

```
$ /opt/elasticbeanstalk/bin/get-config optionsettings -
n aws:elasticbeanstalk:container:php:phpini -o memory_limit
256M
```

- `get-config environment` – Returns a JSON object containing a list of environment properties, including both user-configured properties and those provided by Elastic Beanstalk.

```
$ /opt/elasticbeanstalk/bin/get-config environment
{"TESTPROPERTY":"testvalue","RDS_PORT":"3306","RDS_HOSTNAME":"anj9aw1b0tbj6b.cijbpanmxz5u.us-
west-2.rds.amazonaws.com","RDS_USERNAME":"testusername","RDS_DB_NAME":"ebdb","RDS_PASSWORD":"testpass
```

For example, Elastic Beanstalk provides environment properties for connecting to an integrated RDS DB instance (`RDS_HOSTNAME`, etc.). These properties appear in the output of `get-config environment` but not in the output of `get-config optionsettings`, because they are not set by the user.

To return a specific environment property, use the `-k` option to specify a property key.

```
$ /opt/elasticbeanstalk/bin/get-config environment -k TESTPROPERTY
testvalue
```

You can test the previous commands by using SSH to connect to an instance in an Elastic Beanstalk environment running a Linux-based platform.

See the following files in the [sample platform definition archive \(p. 39\)](#) for an example of `get-config` usage:

- `builder/platform-uploads/opt/elasticbeanstalk/hooks/configdeploy/enact/02-gen-envvars.sh` – Gets environment properties.
- `builder/platform-uploads/opt/SampleNodePlatform/bin/createPM2ProcessFile.js` – Parses the output.

Elastic Beanstalk installs the shell script `download-source-bundle` that you can use to download your application source code during the deployment of your custom platform. This tool is available at `/opt/elasticbeanstalk/bin/download-source-bundle`. See the sample script `00-unzip.sh`, which is in the `appdeploy/pre` folder, for an example of how to use `download-source-bundle` to download application source code to the `/opt/elasticbeanstalk/deploy/appsource` folder during deployment.

Packer instance cleanup

In certain circumstances, such as killing the Packer builder process before it is finished, instances launched by Packer are not cleaned up. These instances are not part of the Elastic Beanstalk environment and can be viewed and terminated only by using the Amazon EC2 service.

To manually clean up these instances

1. Open the [Amazon EC2 console](#).
2. Make sure you are in the same AWS Region in which you created the instance with Packer.
3. Under **Resources**, choose ***N* Running Instances**, where ***N*** indicates the number of running instances.
4. Click in the query text box.
5. Select the **Name** tag.
6. Enter **packer**.

The query should look like: **tag:Name: packer**

7. Select any instances that match the query.
8. If the **Instance State** is **running**, choose **Actions, Instance State, Stop**, and then **Actions, Instance State, Terminate**.

Platform.yaml file format

The `platform.yaml` file has the following format.

```
version: "version-number"

provisioner:
  type: provisioner-type
  template: provisioner-template
  flavor: provisioner-flavor

metadata:
  maintainer: metadata-maintainer
  description: metadata-description
  operating_system_name: metadata-operating_system_name
  operating_system_version: metadata-operating_system_version
  programming_language_name: metadata-programming_language_name
  programming_language_version: metadata-programming_language_version
  framework_name: metadata-framework_name
  framework_version: metadata-framework_version

option_definitions:
- namespace: option-def-namespace
  option_name: option-def-option_name
  description: option-def-description
  default_value: option-def-default_value

option_settings:
- namespace: "option-setting-namespace"
  option_name: "option-setting-option_name"
```

```
value: "option-setting-value"
```

Replace the placeholders with these values:

version-number

Required. The version of the YAML definition. Must be **1.0**.

provisioner-type

Required. The type of builder used to create the custom platform. Must be **packer**.

provisioner-template

Required. The JSON file containing the settings for *provisioner-type*.

provisioner-flavor

Optional. The base operating system used for the AMI. One of the following:

amazon (default)

Amazon Linux. If not specified, the latest version of Amazon Linux when the platform is created.

Amazon Linux 2 isn't a supported operating system flavor.

ubuntu1604

Ubuntu 16.04 LTS

rhel7

RHEL 7

rhel6

RHEL 6

metadata-maintainer

Optional. Contact information for the person who owns the platform (100 characters).

metadata-description

Optional. Description of the platform (2,000 characters).

metadata-operating_system_name

Optional. Name of the platform's operating system (50 characters). This value is available when filtering the output for the [ListPlatformVersions](#) API.

metadata-operating_system_version

Optional. Version of the platform's operating system (20 characters).

metadata-programming_language_name

Optional. Programming language supported by the platform (50 characters)

metadata-programming_language_version

Optional. Version of the platform's language (20 characters).

metadata-framework_name

Optional. Name of the web framework used by the platform (50 characters).

metadata-framework_version

Optional. Version of the platform's web framework (20 characters).

option-def-namespace

Optional. A namespace under `aws:elasticbeanstalk:container:custom` (100 characters).

option-def-option_name

Optional. The option's name (100 characters). You can define up to 50 custom configuration options that the platform provides to users.

option-def-description

Optional. Description of the option (1,024 characters).

option-def-default_value

Optional. Default value used when the user doesn't specify one.

The following example creates the option **NPM_START**.

```
options_definitions:
- namespace: "aws:elasticbeanstalk:container:custom:application"
  option_name: "NPM_START"
  description: "Default application startup command"
  default_value: "node application.js"
```

option-setting-namespace

Optional. Namespace of the option.

option-setting-option_name

Optional. Name of the option. You can specify up to 50 [options provided by Elastic Beanstalk \(p. 555\)](#).

option-setting-value

Optional. Value used when the user doesn't specify one.

The following example creates the option **TEST**.

```
option_settings:
- namespace: "aws:elasticbeanstalk:application:environment"
  option_name: "TEST"
  value: "This is a test"
```

Tagging custom platform versions

You can apply tags to your AWS Elastic Beanstalk custom platform versions. Tags are key-value pairs associated with AWS resources. For information about Elastic Beanstalk resource tagging, use cases, tag key and value constraints, and supported resource types, see [Tagging Elastic Beanstalk application resources \(p. 349\)](#).

You can specify tags when you create a custom platform version. In an existing custom platform version, you can add or remove tags, and update the values of existing tags. You can add up to 50 tags to each custom platform version.

Adding tags during custom platform version creation

If you use the EB CLI to create your custom platform version, use the `--tags` option with [eb platform create \(p. 918\)](#) to add tags.

```
~/workspace/my-app$ eb platform create --tags mytag1=value1,mytag2=value2
```

With the AWS CLI or other API-based clients, add tags by using the `--tags` parameter on the [create-platform-version](#) command.

```
$ aws elasticbeanstalk create-platform-version \  
  --tags Key=mytag1,Value=value1 Key=mytag2,Value=value2 \  
  --platform-name my-platform --platform-version 1.0.0 --platform-definition-bundle \  
  S3Bucket=my-bucket,S3Key=sample.zip
```

Managing tags of an existing custom platform version

You can add, update, and delete tags in an existing Elastic Beanstalk custom platform version.

If you use the EB CLI to update your custom platform version, use [eb tags \(p. 930\)](#) to add, update, delete, or list tags.

For example, the following command lists the tags in a custom platform version.

```
~/workspace/my-app$ eb tags --list --resource "arn:aws:elasticbeanstalk:us-east-2:my-  
account-id:platform/my-platform/1.0.0"
```

The following command updates the tag `mytag1` and deletes the tag `mytag2`.

```
~/workspace/my-app$ eb tags --update mytag1=newvalue --delete mytag2 \  
  --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:platform/my-  
platform/1.0.0"
```

For a complete list of options and more examples, see [eb tags \(p. 930\)](#).

With the AWS CLI or other API-based clients, use the [list-tags-for-resource](#) command to list the tags of a custom platform version.

```
$ aws elasticbeanstalk list-tags-for-resource --resource-arn "arn:aws:elasticbeanstalk:us-  
east-2:my-account-id:platform/my-platform/1.0.0"
```

Use the [update-tags-for-resource](#) command to add, update, or delete tags in a custom platform version.

```
$ aws elasticbeanstalk update-tags-for-resource \  
  --tags-to-add Key=mytag1,Value=newvalue --tags-to-remove mytag2 \  
  --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-id:platform/my-  
platform/1.0.0"
```

Specify both tags to add and tags to update in the `--tags-to-add` parameter of [update-tags-for-resource](#). A nonexisting tag is added, and an existing tag's value is updated.

Note

To use some of the EB CLI and AWS CLI commands with an Elastic Beanstalk custom platform version, you need the custom platform version's ARN. You can retrieve the ARN by using the following command.

```
$ aws elasticbeanstalk list-platform-versions
```

Use the `--filters` option to filter the output down to your custom platform's name.

Deploying Elastic Beanstalk applications from Docker containers

Elastic Beanstalk supports the deployment of web applications from Docker containers. With Docker containers, you can define your own runtime environment. You can choose your own platform, programming language, and any application dependencies (such as package managers or tools), that aren't supported by other platforms. Docker containers are self-contained and include all the configuration information and software your web application requires to run. All environment variables defined in the Elastic Beanstalk console are passed to the containers.

By using Docker with Elastic Beanstalk, you have an infrastructure that automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring. You can manage your web application in an environment that supports the range of services that are integrated with Elastic Beanstalk, including but not limited to [VPC](#), [RDS](#), and [IAM](#). For more information about Docker, including how to install it, what software it requires, and how to use Docker images to launch Docker containers, go to [Docker: the Linux container engine](#).

Note

If a Docker container running in an Elastic Beanstalk environment crashes or is killed for any reason, Elastic Beanstalk restarts it automatically.

The topics in this chapter assume some knowledge of Elastic Beanstalk environments. If you haven't used Elastic Beanstalk before, try the [getting started tutorial \(p. 3\)](#) to learn the basics.

Docker platforms

The Docker platform family for Elastic Beanstalk has two generic platforms (single container and multicontainer), and several preconfigured containers.

See the [Supported Platforms](#) page in the *AWS Elastic Beanstalk Platforms* document for details on the currently supported platform version of each Docker platform.

Single Container Docker

The single container platform can be used to deploy a Docker image (described in a Dockerfile or `Dockerrun.aws.json` definition) and source code to EC2 instances running in an Elastic Beanstalk environment. Use the single container platform when you only need to run one container per instance.

For samples and help getting started with a single container Docker environment, see [Single Container Docker environments \(p. 51\)](#). For detailed information on the container definition formats and their use, see [Single Container Docker configuration \(p. 54\)](#).

Multicontainer Docker

The other basic platform, Multicontainer Docker, uses the Amazon Elastic Container Service to coordinate a deployment of multiple Docker containers to an Amazon ECS cluster in an Elastic Beanstalk environment. The instances in the environment each run the same set of containers, which are defined in a `Dockerrun.aws.json` file. Use the multicontainer platform when you need to deploy multiple Docker containers to each instance.

For more details on the Multicontainer Docker platform and its use, see [Multicontainer Docker environments \(p. 58\)](#). The [Multicontainer Docker configuration \(p. 62\)](#) topic details version 2 of the `Dockerrun.aws.json` format, which is similar to but not compatible with the version used with the single container platform. There is also a [tutorial \(p. 66\)](#) available that guides you through a from scratch deployment of a multicontainer environment running a PHP website with an nginx proxy running in front of it in a separate container.

Preconfigured Docker containers

In addition to the two generic Docker platforms, there are several *preconfigured* Docker platform versions that you can use to run your application in a popular software stack such as *Java with Glassfish* or *Python with uWSGI*. Use a preconfigured container if it matches the software used by your application.

For more information, see [Preconfigured Docker containers \(p. 71\)](#).

Single Container Docker environments

Important

Amazon Linux 2 platform versions are fundamentally different than Amazon Linux AMI platform versions (preceding Amazon Linux 2). These different platform generations are incompatible in several ways. If you are migrating to an Amazon Linux 2 platform version, be sure to read the information in [the section called "Upgrade to Amazon Linux 2" \(p. 426\)](#).

AWS Elastic Beanstalk can launch single container Docker environments by building an image described in a `Dockerfile` or pulling a remote Docker image. If you're deploying a remote Docker image, you don't need to include a `Dockerfile`. Instead, use a `Dockerrun.aws.json` file, which specifies an image to use and additional configuration options.

Topics

- [Prerequisites \(p. 51\)](#)
- [Containerize an Elastic Beanstalk application \(p. 51\)](#)
- [Test a container locally \(p. 52\)](#)
- [Deploy a container with a Dockerfile \(p. 53\)](#)
- [Test a remote Docker image \(p. 53\)](#)
- [Deploy a remote Docker image to Elastic Beanstalk \(p. 54\)](#)
- [Clean up \(p. 54\)](#)
- [Single Container Docker configuration \(p. 54\)](#)

Prerequisites

This tutorial assumes that you have some knowledge of basic Elastic Beanstalk operations, [Using the Elastic Beanstalk command line interface \(EB CLI\) \(p. 852\)](#), and Docker. To follow this tutorial, you need a working local installation of Docker. For more information about installing Docker, see the [Docker installation guide](#).

If you haven't already, follow the instructions in [Getting started using Elastic Beanstalk \(p. 3\)](#) to launch your first Elastic Beanstalk environment. This tutorial uses the EB CLI, but you can also create environments and upload applications by using the Elastic Beanstalk console. To learn more about configuring single container Docker environments, see [Single Container Docker configuration \(p. 54\)](#).

Containerize an Elastic Beanstalk application

For this example, we create a Docker image of the sample Flask application from [Deploying a flask application to Elastic Beanstalk \(p. 295\)](#). The application consists of one main file, `application.py`. We also need a `Dockerfile`. Put both files at the root of a directory.

```
~/eb-docker-flask/  
|-- Dockerfile  
|-- application.py
```

Example `~/eb-docker-flask/application.py`

```
from flask import Flask

# Print a nice greeting
def say_hello(username = "World"):
    return '<p>Hello %s!</p>\n' % username

# Some bits of text for the page
header_text = '''
<html>\n<head> <title>EB Flask Test</title> </head>\n<body>'''
instructions = '''
<p><em>Hint</em>: This is a RESTful web service! Append a username
to the URL (for example: <code>/Thelonious</code>) to say hello to
someone specific.</p>\n'''
home_link = '<p><a href="/">Back</a></p>\n'
footer_text = '</body>\n</html>'

# Elastic Beanstalk looks for an 'application' that is callable by default
application = Flask(__name__)

# Add a rule for the index page
application.add_url_rule('/', 'index', (lambda: header_text +
    say_hello() + instructions + footer_text))

# Add a rule when the page is accessed with a name appended to the site
# URL
application.add_url_rule('/<username>', 'hello', (lambda username:
    header_text + say_hello(username) + home_link + footer_text))

# Run the application
if __name__ == "__main__":
    # Setting debug to True enables debug output. This line should be
    # removed before deploying a production application.
    application.debug = True
    application.run(host="0.0.0.0")
```

Example `~/eb-docker-flask/Dockerfile`

```
FROM python:3.6
COPY . /app
WORKDIR /app
RUN pip install Flask==1.0.2
EXPOSE 5000
CMD ["python", "application.py"]
```

Test a container locally

Use the Elastic Beanstalk CLI (EB CLI) to configure your local repository for deployment to Elastic Beanstalk. Set your application's Dockerfile at the root of the directory.

```
~/eb-docker-flask$ eb init -p docker application-name
```

(Optional) Use **eb local run** to build and run your container locally. To learn more about the **eb local** command, see [eb local](#) (p. 911). The **eb local** command isn't supported on Windows. Alternatively, you can build and run your container with the **docker build** and **docker run** commands. For more information, see the [Docker documentation](#).

```
~/eb-docker-flask$ eb local run --port 5000
```

(Optional) While your container is running, use the **eb local open** command to view your application in a web browser. Alternatively, open <http://localhost:5000/> in a web browser.

```
~/eb-docker-flask$ eb local open
```

Deploy a container with a Dockerfile

After testing your application locally, deploy it to an Elastic Beanstalk environment. Elastic Beanstalk uses the instructions in your `Dockerfile` to build and run the image.

Use the EB CLI to create an environment and deploy your application.

```
~/eb-docker-flask$ eb create environment-name
```

Once your environment has launched, use **eb open** to view it in a web browser.

```
~/eb-docker-flask$ eb open
```

Test a remote Docker image

Next, we build a Docker image of the Flask application from the previous section and push it to Docker Hub.

Note

The following steps create a publicly available Docker image.

Once we've built and pushed our image, we can deploy it to Elastic Beanstalk with a `Dockerrun.aws.json` file. To build a Docker image of the Flask application and push it to Docker Hub, run the following commands. We're using the same directory from the previous example, but you can use any directory with your application's code.

```
~/eb-docker-flask$ docker build -t docker-username/beanstalk-flask:latest .  
~/eb-docker-flask$ docker push docker-username/beanstalk-flask:latest
```

Note

Before pushing your image, you might need to run **docker login**.

Now you can deploy your application using only a `Dockerrun.aws.json` file. To learn more about `Dockerrun.aws.json` files, see [Single Container Docker configuration \(p. 54\)](#).

Make a new directory and create a `Dockerrun.aws.json` file.

Example `~/remote-docker/Dockerrun.aws.json`

```
{  
  "AWSEBDockerrunVersion": "1",  
  "Image": {  
    "Name": "username/beanstalk-flask",  
    "Update": "true"  
  },  
  "Ports": [  
    {  
      "ContainerPort": "5000"  
    }  
  ]  
}
```

Use the EB CLI to configure your local repository for deployment to Elastic Beanstalk.

```
~/remote-docker$ eb init -p docker application-name
```

(Optional) Use **eb local run** to build and run your container locally. To learn more about the **eb local** command, see [eb local](#) (p. 911).

```
~/remote-docker$ eb local run --port 5000
```

(Optional) While your container is running, use the **eb local open** command to view your application in a web browser. Alternatively, open <http://localhost:5000/> in a web browser.

```
~/remote-docker$ eb local open
```

Deploy a remote Docker image to Elastic Beanstalk

After testing your container locally, deploy it to an Elastic Beanstalk environment. Elastic Beanstalk uses the `Dockerrun.aws.json` file to pull and run your image.

Use the EB CLI to create an environment and deploy your image.

```
~/remote-docker$ eb create environment-name
```

Once your environment is launched, use **eb open** to view it in a web browser.

```
~/remote-docker$ eb open
```

Clean up

When you finish working with Elastic Beanstalk, you can terminate your environment. Elastic Beanstalk terminates all AWS resources associated with your environment, such as [Amazon EC2 instances](#) (p. 451), [database instances](#) (p. 506), [load balancers](#) (p. 470), security groups, and [alarms](#) (p.).

To terminate your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

With Elastic Beanstalk, you can easily create a new environment for your application at any time.

Or, with the EB CLI:

```
~/remote-docker$ eb terminate environment-name
```

Single Container Docker configuration

This section describes how to prepare your Docker image and container for deployment to Elastic Beanstalk. Any web application that you deploy to Elastic Beanstalk in a single container Docker

environment must include a `Dockerfile` or a `Dockerrun.aws.json` file. You can deploy your web application from a Docker container to Elastic Beanstalk by doing one of the following:

- Create a `Dockerfile` to have Elastic Beanstalk build and run a custom image.
- Create a `Dockerrun.aws.json` file to deploy a Docker image from a hosted repository to Elastic Beanstalk.
- Create a `.zip` file containing your application files, any application file dependencies, the `Dockerfile`, and the `Dockerrun.aws.json` file. If you use the EB CLI to deploy your application, it will create a `.zip` file for you.

If you use only a `Dockerfile` or only a `Dockerrun.aws.json` file to deploy your application, you don't need to create a `.zip` file.

This topic is a syntax reference. For detailed procedures on launching single container Docker environments, see [Single Container Docker environments \(p. 51\)](#).

Sections

- [Dockerrun.aws.json v1 \(p. 55\)](#)
- [Using images from a private repository \(p. 57\)](#)
- [Building custom images with a Dockerfile \(p. 58\)](#)

Dockerrun.aws.json v1

A `Dockerrun.aws.json` file describes how to deploy a remote Docker image as an Elastic Beanstalk application. This JSON file is specific to Elastic Beanstalk. If your application runs on an image that is available in a hosted repository, you can specify the image in a `Dockerrun.aws.json` file and omit the `Dockerfile`.

Valid keys and values for the `Dockerrun.aws.json` file include the following:

AWSEBDockerrunVersion

(Required) Specifies the version number as the value 1 for single container Docker environments.

Authentication

(Required only for private repositories) Specifies the Amazon S3 object storing the `.dockercfg` file.

See [Using images from a private repository \(p. 57\)](#).

Image

Specifies the Docker base image on an existing Docker repository from which you're building a Docker container. Specify the value of the **Name** key in the format `<organization>/<image name>` for images on Docker Hub, or `<site>/<organization name>/<image name>` for other sites.

When you specify an image in the `Dockerrun.aws.json` file, each instance in your Elastic Beanstalk environment will run `docker pull` on that image and run it. Optionally, include the **Update** key. The default value is `true` and instructs Elastic Beanstalk to check the repository, pull any updates to the image, and overwrite any cached images.

When using a `Dockerfile`, do not specify the **Image** key in the `Dockerrun.aws.json` file. Elastic Beanstalk always builds and uses the image described in the `Dockerfile` when one is present.

Ports

(Required when you specify the **Image** key) Lists the ports to expose on the Docker container. Elastic Beanstalk uses the **ContainerPort** value to connect the Docker container to the reverse proxy running on the host.

You can specify multiple container ports, but Elastic Beanstalk uses only the first one to connect your container to the host's reverse proxy and route requests from the public internet. If you're using a `Dockerfile`, the first **ContainerPort** value should match the first entry in the `Dockerfile`'s **EXPOSE** list.

Optionally, you can specify a list of ports in **HostPort**. **HostPort** entries specify the host ports that **ContainerPort** values are mapped to. If you don't specify a **HostPort** value, it defaults to the **ContainerPort** value.

```
{
  "Image": {
    "Name": "image-name"
  },
  "Ports": [
    {
      "ContainerPort": 8080,
      "HostPort": 8000
    }
  ]
}
```

Volumes

Map volumes from an EC2 instance to your Docker container. Specify one or more arrays of volumes to map.

```
{
  "Volumes": [
    {
      "HostDirectory": "/path/inside/host",
      "ContainerDirectory": "/path/inside/container"
    }
  ]
  ...
}
```

Logging

Specify the directory inside the container to which your application writes logs. Elastic Beanstalk uploads any logs in this directory to Amazon S3 when you request tail or bundle logs. If you rotate logs to a folder named `rotated` within this directory, you can also configure Elastic Beanstalk to upload rotated logs to Amazon S3 for permanent storage. For more information, see [Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment \(p. 732\)](#).

Command

Specify a command to execute in the container. If you specify an **Entrypoint**, then **Command** is added as an argument to **Entrypoint**. For more information, see [CMD](#) in the Docker documentation.

Entrypoint

Specify a default command to run when the container starts. For more information, see [ENTRYPOINT](#) in the Docker documentation.

The following snippet is an example that illustrates the syntax of the `Dockerrun.aws.json` file for a single container.

```
{
  "AWSEBDockerrunVersion": "1",
  "Image": {
```

```
"Name": "janedoe/image",
"Update": "true"
},
"Ports": [
  {
    "ContainerPort": "1234"
  }
],
"Volumes": [
  {
    "HostDirectory": "/var/app/mydb",
    "ContainerDirectory": "/etc/mysql"
  }
],
"Logging": "/var/log/nginx",
"Entrypoint": "/app/bin/myapp",
"Command": "--argument"
}
```

You can provide Elastic Beanstalk with only the `Dockerrun.aws.json` file, or with a `.zip` archive containing both the `Dockerrun.aws.json` and `Dockerfile` files. When you provide both files, the `Dockerfile` describes the Docker image and the `Dockerrun.aws.json` file provides additional information for deployment, as described later in this section.

Note

The two files must be at the root, or top level, of the `.zip` archive. Do not build the archive from a directory containing the files. Navigate into that directory and build the archive there. When you provide both files, do not specify an image in the `Dockerrun.aws.json` file. Elastic Beanstalk builds and uses the image described in the `Dockerfile` and ignores the image specified in the `Dockerrun.aws.json` file.

Using images from a private repository

Add the information about the Amazon S3 bucket that contains the authentication file in the `Authentication` parameter of the `Dockerrun.aws.json` file. Make sure that the `Authentication` parameter contains a valid Amazon S3 bucket and key. The Amazon S3 bucket must be hosted in the same AWS Region as the environment that is using it. Elastic Beanstalk will not download files from Amazon S3 buckets hosted in other Regions.

For information about generating and uploading the authentication file, see [Using images from a private repository \(p. 76\)](#).

The following example shows the use of an authentication file named `mydockercfg` in a bucket named `my-bucket` to use a private image in a third-party registry.

```
{
  "AWSEBDockerrunVersion": "1",
  "Authentication": {
    "Bucket": "my-bucket",
    "Key": "mydockercfg"
  },
  "Image": {
    "Name": "quay.io/johndoe/private-image",
    "Update": "true"
  },
  "Ports": [
    {
      "ContainerPort": "1234"
    }
  ],
  "Volumes": [
    {
```



```
    "HostDirectory": "/var/app/mydb",  
    "ContainerDirectory": "/etc/mysql"  
  }  
],  
"Logging": "/var/log/nginx"  
}
```

Building custom images with a Dockerfile

Create a `Dockerfile` when you don't already have an existing image hosted in a repository.

The following snippet is an example of the `Dockerfile`. When you follow the instructions in [Single Container Docker environments \(p. 51\)](#), you can upload this `Dockerfile` as written. Elastic Beanstalk runs the game 2048 when you use this `Dockerfile`.

```
FROM ubuntu:12.04  
  
RUN apt-get update  
RUN apt-get install -y nginx zip curl  
  
RUN echo "daemon off;" >> /etc/nginx/nginx.conf  
RUN curl -o /usr/share/nginx/www/master.zip -L https://codeload.github.com/  
gabrielecirulli/2048/zip/master  
RUN cd /usr/share/nginx/www/ && unzip master.zip && mv 2048-master/* . && rm -rf 2048-  
master master.zip  
  
EXPOSE 80  
  
CMD ["/usr/sbin/nginx", "-c", "/etc/nginx/nginx.conf"]
```

For more information about instructions you can include in the `Dockerfile`, see [Dockerfile reference](#) on the Docker website.

Multicontainer Docker environments

You can create docker environments that support multiple containers per Amazon EC2 instance with multicontainer Docker platform for Elastic Beanstalk.

Elastic Beanstalk uses Amazon Elastic Container Service (Amazon ECS) to coordinate container deployments to multicontainer Docker environments. Amazon ECS provides tools to manage a cluster of instances running Docker containers. Elastic Beanstalk takes care of Amazon ECS tasks including cluster creation, task definition and execution.

Note

Some regions don't offer Amazon ECS. Multicontainer Docker environments aren't supported in these regions.

For information about the AWS services offered in each region, see [Region Table](#).

Topics

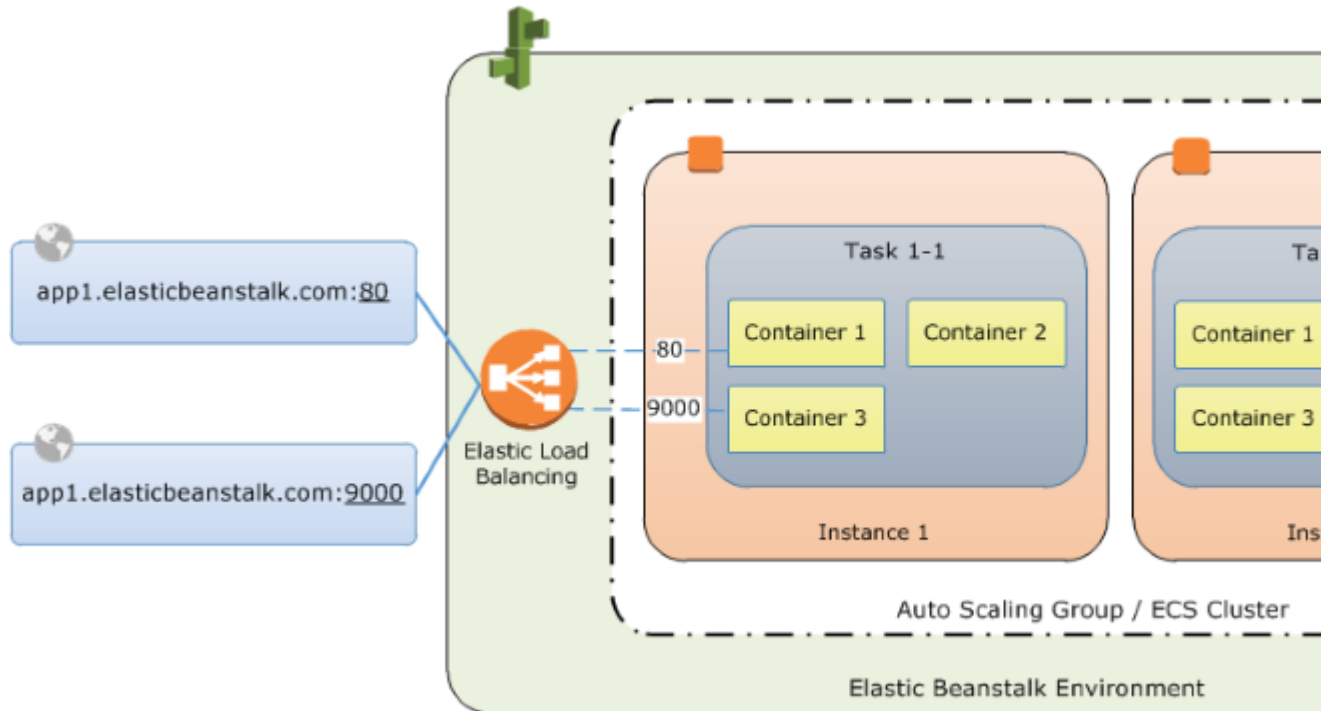
- [Multicontainer Docker platform \(p. 59\)](#)
- [Dockerrun.aws.json file \(p. 59\)](#)
- [Docker images \(p. 59\)](#)
- [Container instance role \(p. 60\)](#)
- [Amazon ECS resources created by Elastic Beanstalk \(p. 60\)](#)
- [Using multiple Elastic Load Balancing listeners \(p. 61\)](#)
- [Failed container deployments \(p. 62\)](#)
- [Multicontainer Docker configuration \(p. 62\)](#)

- [Multicontainer Docker environments with the Elastic Beanstalk console \(p. 66\)](#)

Multicontainer Docker platform

Standard generic and preconfigured Docker platforms on Elastic Beanstalk support only a single Docker container per Elastic Beanstalk environment. In order to get the most out of Docker, Elastic Beanstalk lets you create an environment where your Amazon EC2 instances run multiple Docker containers side by side.

The following diagram shows an example Elastic Beanstalk environment configured with three Docker containers running on each Amazon EC2 instance in an Auto Scaling group:



Dockerrun.aws.json file

Container instances—Amazon EC2 instances running Multicontainer Docker in an Elastic Beanstalk environment—require a configuration file named `Dockerrun.aws.json`. This file is specific to Elastic Beanstalk and can be used alone or combined with source code and content in a [source bundle \(p. 342\)](#) to create an environment on a Docker platform.

Note

Version 1 of the `Dockerrun.aws.json` format is used to launch a single Docker container to an Elastic Beanstalk environment. Version 2 adds support for multiple containers per Amazon EC2 instance and can only be used with the multicontainer Docker platform. The format differs significantly from the previous version which is detailed under [Single Container Docker configuration \(p. 54\)](#)

See [Dockerrun.aws.json v2 \(p. 62\)](#) for details on the updated format and an example file.

Docker images

The Multicontainer Docker platform for Elastic Beanstalk requires images to be prebuilt and stored in a public or private online image repository.

Note

Building custom images during deployment with a `Dockerfile` is not supported by the multicontainer Docker platform on Elastic Beanstalk. Build your images and deploy them to an online repository before creating an Elastic Beanstalk environment.

Specify images by name in `Dockerrun.aws.json`. Note these conventions:

- Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).
- Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`).
- Images in other online registries are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu`).

To configure Elastic Beanstalk to authenticate to a private repository, include the `authentication` parameter in your `Dockerrun.aws.json` file.

Container instance role

Elastic Beanstalk uses an Amazon ECS-optimized AMI with an Amazon ECS container agent that runs in a Docker container. The agent communicates with Amazon ECS to coordinate container deployments. In order to communicate with Amazon ECS, each Amazon EC2 instance must have the corresponding permissions in IAM. These permissions are attached to the default [instance profile \(p. 20\)](#) when you create an environment in the Elastic Beanstalk Management Console:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ECSAccess",
      "Effect": "Allow",
      "Action": [
        "ecs:Poll",
        "ecs:StartTask",
        "ecs:StopTask",
        "ecs:DiscoverPollEndpoint",
        "ecs:StartTelemetrySession",
        "ecs:RegisterContainerInstance",
        "ecs:DeregisterContainerInstance",
        "ecs:DescribeContainerInstances",
        "ecs:Submit*"
      ],
      "Resource": "*"
    }
  ]
}
```

If you create your own instance profile, you can attach the `AWSElasticBeanstalkMulticontainerDocker` managed policy to make sure the permissions stay up-to-date. For instructions on creating policies and roles in IAM, see [Creating IAM Roles](#) in the IAM User Guide.

Amazon ECS resources created by Elastic Beanstalk

When you create an environment using the multicontainer Docker platform, Elastic Beanstalk automatically creates and configures several Amazon Elastic Container Service resources while building the environment in order to create the necessary containers on each Amazon EC2 instance.

- **Amazon ECS Cluster** – Container instances in Amazon ECS are organized into clusters. When used with Elastic Beanstalk, one cluster is always created for each multicontainer Docker environment.

- **Amazon ECS Task Definition** – Elastic Beanstalk uses the `Dockerrun.aws.json` file in your project to generate the Amazon ECS task definition that is used to configure container instances in the environment.
- **Amazon ECS Task** – Elastic Beanstalk communicates with Amazon ECS to run a task on every instance in the environment to coordinate container deployment. In an autoscaling environment, Elastic Beanstalk initiates a new task whenever an instance is added to the cluster. In rare cases you may have to increase the amount of space reserved for containers and images. Learn more in the [Configuring Docker environments \(p. 74\)](#) section.
- **Amazon ECS Container Agent** – The agent runs in a Docker container on the instances in your environment. The agent polls the Amazon ECS service and waits for a task to run.
- **Amazon ECS Data Volumes** – Elastic Beanstalk inserts volume definitions (in addition to the volumes that you define in `Dockerrun.aws.json`) into the task definition to facilitate log collection.

Elastic Beanstalk creates log volumes on the container instance, one for each container, at `/var/log/containers/containername`. These volumes are named `awseb-logs-containername` and are provided for containers to mount. See [Container definition format \(p. 64\)](#) for details on how to mount them.

Using multiple Elastic Load Balancing listeners

You can configure multiple Elastic Load Balancing listeners on a multicontainer Docker environment in order to support inbound traffic for proxies or other services that don't run on the default HTTP port.

Create a `.ebextensions` folder in your source bundle and add a file with a `.config` file extension. The following example shows a configuration file that creates an Elastic Load Balancing listener on port 8080.

`.ebextensions/elb-listener.config`

```
option_settings:
  aws:elb:listener:8080:
    ListenerProtocol: HTTP
    InstanceProtocol: HTTP
    InstancePort: 8080
```

If your environment is running in a custom [Amazon Virtual Private Cloud \(Amazon VPC\)](#) that you created, Elastic Beanstalk takes care of the rest. In a default VPC, you need to configure your instance's security group to allow ingress from the load balancer. Add a second configuration file that adds an ingress rule to the security group:

`.ebextensions/elb-ingress.config`

```
Resources:
  port8080SecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 8080
      FromPort: 8080
      SourceSecurityGroupName: { "Fn::GetAtt": ["AWSEBLoadBalancer",
"SourceSecurityGroup.GroupName"] }
```

For more information on the configuration file format, see [Adding and customizing Elastic Beanstalk environment resources \(p. 622\)](#) and [Option settings \(p. 601\)](#).

In addition to adding a listener to the Elastic Load Balancing configuration and opening a port in the security group, you need to map the port on the host instance to a port on the Docker container in the

containerDefinitions section of the `Dockerrun.aws.json` file. The following excerpt shows an example:

```
"portMappings": [  
  {  
    "hostPort": 8080,  
    "containerPort": 8080  
  }  
]
```

See [Dockerrun.aws.json v2 \(p. 62\)](#) for details on the `Dockerrun.aws.json` file format.

Failed container deployments

If an Amazon ECS task fails, one or more containers in your Elastic Beanstalk environment will not start. Elastic Beanstalk does not roll back multicontainer environments due to a failed Amazon ECS task. If a container fails to start in your environment, redeploy the current version or a previous working version from the Elastic Beanstalk console.

To deploy an existing version

1. Open the Elastic Beanstalk console in your environment's region.
2. Click **Actions** to the right of your application name and then click **View application versions**.
3. Select a version of your application and click **Deploy**.

Multicontainer Docker configuration

A `Dockerrun.aws.json` file is an Elastic Beanstalk-specific JSON file that describes how to deploy a set of Docker containers as an Elastic Beanstalk application. You can use a `Dockerrun.aws.json` file for a multicontainer Docker environment.

`Dockerrun.aws.json` describes the containers to deploy to each container instance (Amazon EC2 instance that hosts Docker containers) in the environment as well as the data volumes to create on the host instance for the containers to mount.

A `Dockerrun.aws.json` file can be used on its own or zipped up with additional source code in a single archive. Source code that is archived with a `Dockerrun.aws.json` is deployed to Amazon EC2 container instances and accessible in the `/var/app/current/` directory. Use the `volumes` section of the config to provide file volumes for the Docker containers running on the host instance. Use the `mountPoints` section of the embedded container definitions to map these volumes to mount points that applications on the Docker containers can use.

Topics

- [Dockerrun.aws.json v2 \(p. 62\)](#)
- [Using images from a private repository \(p. 64\)](#)
- [Container definition format \(p. 64\)](#)

Dockerrun.aws.json v2

The `Dockerrun.aws.json` file includes three sections:

AWSEBDockerrunVersion

Specifies the version number as the value 2 for multicontainer Docker environments.

containerDefinitions

An array of container definitions, detailed below.

volumes

Creates volumes from folders in the Amazon EC2 container instance, or from your source bundle (deployed to `/var/app/current`). Mount these volumes to paths within your Docker containers using `mountPoints` in the [container definition \(p. 64\)](#).

Note

Elastic Beanstalk configures additional volumes for logs, one for each container. These should be mounted by your Docker containers in order to write logs to the host instance.

See [Container definition format \(p. 64\)](#) for details.

Volumes are specified in the following format:

```
"volumes": [  
  {  
    "name": "volumename",  
    "host": {  
      "sourcePath": "/path/on/host/instance"  
    }  
  }  
],
```

authentication

(optional) The location in Amazon S3 of a `.dockercfg` file that contains authentication data for a private repository. Uses the following format:

```
"authentication": {  
  "bucket": "my-bucket",  
  "key": "mydockercfg"  
},
```

See [Using images from a private repository \(p. 64\)](#) for details.

The following snippet is an example that illustrates the syntax of the `DockerRun.aws.json` file for an instance with two containers.

```
{  
  "AWSEBDockerrunVersion": 2,  
  "volumes": [  
    {  
      "name": "php-app",  
      "host": {  
        "sourcePath": "/var/app/current/php-app"  
      }  
    },  
    {  
      "name": "nginx-proxy-conf",  
      "host": {  
        "sourcePath": "/var/app/current/proxy/conf.d"  
      }  
    }  
  ],  
  "containerDefinitions": [  
    {  
      "name": "php-app",  
      "image": "php:fpm",  
      "environment": [  

```

```
    {
      "name": "Container",
      "value": "PHP"
    }
  ],
  "essential": true,
  "memory": 128,
  "mountPoints": [
    {
      "sourceVolume": "php-app",
      "containerPath": "/var/www/html",
      "readOnly": true
    }
  ]
},
{
  "name": "nginx-proxy",
  "image": "nginx",
  "essential": true,
  "memory": 128,
  "portMappings": [
    {
      "hostPort": 80,
      "containerPort": 80
    }
  ],
  "links": [
    "php-app"
  ],
  "mountPoints": [
    {
      "sourceVolume": "php-app",
      "containerPath": "/var/www/html",
      "readOnly": true
    },
    {
      "sourceVolume": "nginx-proxy-conf",
      "containerPath": "/etc/nginx/conf.d",
      "readOnly": true
    },
    {
      "sourceVolume": "awseb-logs-nginx-proxy",
      "containerPath": "/var/log/nginx"
    }
  ]
}
]
```

Using images from a private repository

Add the information about the Amazon S3 bucket that contains the authentication file in the authentication parameter of the `Dockerrun.aws.json` file. Make sure that the authentication parameter contains a valid Amazon S3 bucket and key. The Amazon S3 bucket must be hosted in the same region as the environment that is using it. Elastic Beanstalk will not download files from Amazon S3 buckets hosted in other regions.

For information about generating and uploading the authentication file, see [Using images from a private repository \(p. 76\)](#).

Container definition format

The container definition and volumes sections of `Dockerrun.aws.json` use the same formatting as the corresponding sections of an Amazon ECS task definition file.

The following examples show a subset of parameters that are commonly used. More optional parameters are available. For more information on the task definition format and a full list of task definition parameters, see [Amazon ECS Task Definitions](#) in the *Amazon Elastic Container Service Developer Guide*.

A `DockerRun.aws.json` file contains an array of one or more container definition objects with the following fields:

name

The name of the container. See [Standard Container Definition Parameters](#) for information about the maximum length and allowed characters.

image

The name of a Docker image in an online Docker repository from which you're building a Docker container. Note these conventions:

- Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).
- Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`).
- Images in other online repositories are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu`).

environment

An array of environment variables to pass to the container.

For example, the following entry defines an environment variable with the name `Container` and the value `PHP`:

```
"environment": [  
  {  
    "name": "Container",  
    "value": "PHP"  
  }  
],
```

essential

True if the task should stop if the container fails. Nonessential containers can finish or crash without affecting the rest of the containers on the instance.

memory

Amount of memory on the container instance to reserve for the container. Specify a non-zero integer for one or both of the `memory` or `memoryReservation` parameters in container definitions.

memoryReservation

The soft limit (in MiB) of memory to reserve for the container. Specify a non-zero integer for one or both of the `memory` or `memoryReservation` parameters in container definitions.

mountPoints

Volumes from the Amazon EC2 container instance to mount, and the location on the Docker container file system at which to mount them. When you mount volumes that contain application content, your container can read the data you upload in your source bundle. When you mount log volumes for writing log data, Elastic Beanstalk can gather log data from these volumes.

Elastic Beanstalk creates log volumes on the container instance, one for each Docker container, at `/var/log/containers/containername`. These volumes are named `awseb-logs-containername` and should be mounted to the location within the container file structure where logs are written.

For example, the following mount point maps the nginx log location in the container to the Elastic Beanstalk-generated volume for the `nginx-proxy` container.

```
{
  "sourceVolume": "awseb-logs-nginx-proxy",
  "containerPath": "/var/log/nginx"
}
```

portMappings

Maps network ports on the container to ports on the host.

links

List of containers to link to. Linked containers can discover each other and communicate securely.

volumesFrom

Mount all of the volumes from a different container. For example, to mount volumes from a container named `web`:

```
"volumesFrom": [
  {
    "sourceContainer": "web"
  }
],
```

Multicontainer Docker environments with the Elastic Beanstalk console

You can launch a cluster of multicontainer instances in a single-instance or autoscaling Elastic Beanstalk environment using the Elastic Beanstalk console. This tutorial details container configuration and source code preparation for an environment that uses two containers.

The containers, a PHP application and an nginx proxy, run side by side on each of the Amazon Elastic Compute Cloud (Amazon EC2) instances in an Elastic Beanstalk environment. After creating the environment and verifying that the applications are running, you'll connect to a container instance to see how it all fits together.

Sections

- [Define Docker containers \(p. 66\)](#)
- [Add content \(p. 68\)](#)
- [Deploy to Elastic Beanstalk \(p. 69\)](#)
- [Connect to a container instance \(p. 70\)](#)
- [Inspect the Amazon ECS container agent \(p. 70\)](#)

Define Docker containers

The first step in creating a new Docker environment is to create a directory for your application data. This folder can be located anywhere on your local machine and have any name you choose. In addition to a container configuration file, this folder will contain the content that you will upload to Elastic Beanstalk and deploy to your environment.

Note

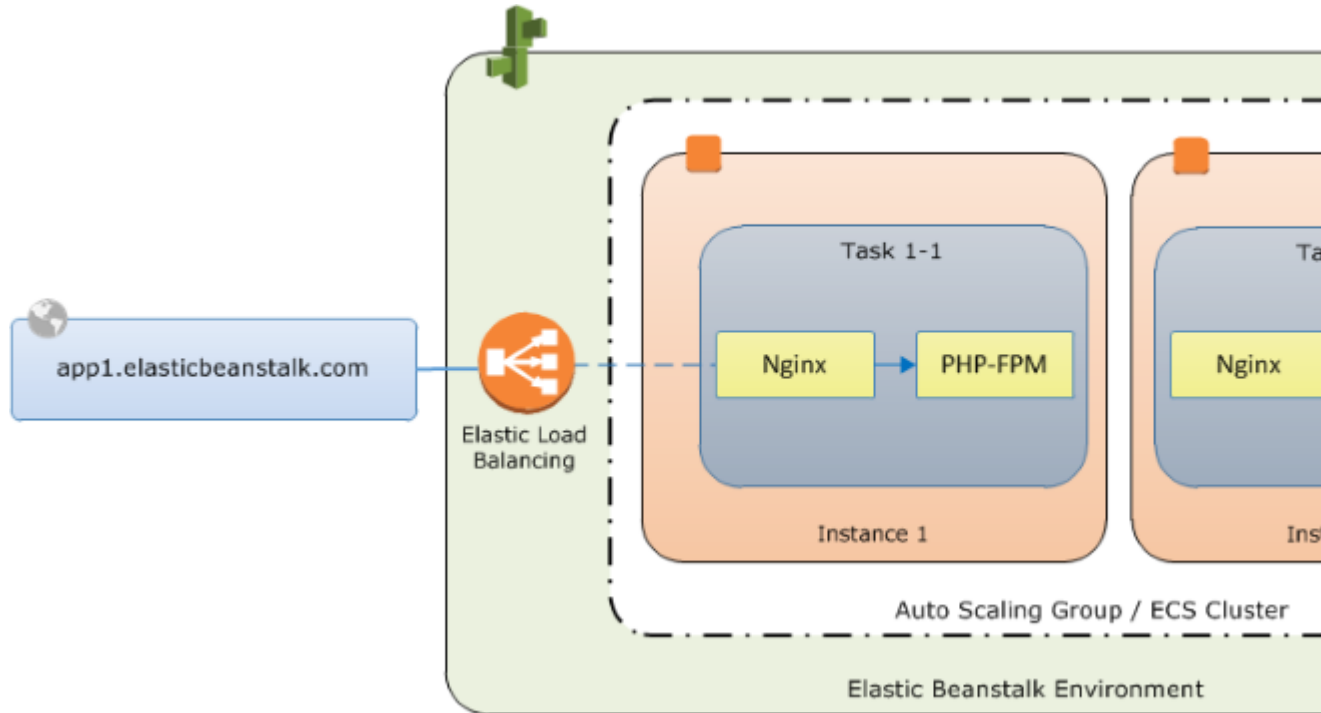
All of the code for this tutorial is available in the `awslabs` repository on GitHub at <https://github.com/awslabs/eb-docker-nginx-proxy>.

The file that Elastic Beanstalk uses to configure the containers on an Amazon EC2 instance is a JSON-formatted text file named `Dockerrun.aws.json`. Create a text file with this name at the root of your application and add the following text:

```
{
  "AWSEBDockerrunVersion": 2,
  "volumes": [
    {
      "name": "php-app",
      "host": {
        "sourcePath": "/var/app/current/php-app"
      }
    },
    {
      "name": "nginx-proxy-conf",
      "host": {
        "sourcePath": "/var/app/current/proxy/conf.d"
      }
    }
  ],
  "containerDefinitions": [
    {
      "name": "php-app",
      "image": "php:fpm",
      "essential": true,
      "memory": 128,
      "mountPoints": [
        {
          "sourceVolume": "php-app",
          "containerPath": "/var/www/html",
          "readOnly": true
        }
      ]
    },
    {
      "name": "nginx-proxy",
      "image": "nginx",
      "essential": true,
      "memory": 128,
      "portMappings": [
        {
          "hostPort": 80,
          "containerPort": 80
        }
      ],
      "links": [
        "php-app"
      ],
      "mountPoints": [
        {
          "sourceVolume": "php-app",
          "containerPath": "/var/www/html",
          "readOnly": true
        },
        {
          "sourceVolume": "nginx-proxy-conf",
          "containerPath": "/etc/nginx/conf.d",
          "readOnly": true
        },
        {
          "sourceVolume": "awseb-logs-nginx-proxy",
          "containerPath": "/var/log/nginx"
        }
      ]
    }
  ]
}
```

```
]
}
```

This example configuration defines two containers, a PHP web site with an nginx proxy in front of it. These two containers will run side by side in Docker containers on each instance in your Elastic Beanstalk environment, accessing shared content (the content of the website) from volumes on the host instance, which are also defined in this file. The containers themselves are created from images hosted in official repositories on Docker Hub. The resulting environment looks like the following:



The volumes defined in the configuration correspond to the content that you will create next and upload as part of your application source bundle. The containers access content on the host by mounting volumes in the `mountPoints` section of the container definitions.

For more information on the format of `Dockerrun.aws.json` and its parameters, see [Container definition format \(p. 64\)](#).

Add content

Next you will add some content for your PHP site to display to visitors, and a configuration file for the nginx proxy.

php-app/index.php

```
<h1>Hello World!!!</h1>
<h3>PHP Version <pre><?= phpversion()?</pre></h3>
```

php-app/static.html

```
<h1>Hello World!</h1>
<h3>This is a static HTML page.</h3>
```

proxy/conf.d/default.conf

```
server {
    listen 80;
    server_name localhost;
    root /var/www/html;

    index index.php;

    location ~ [^/]\.php(/|$) {
        fastcgi_split_path_info ^(.+?\.php)(/.*)$;
        if (!-f $document_root$fastcgi_script_name) {
            return 404;
        }

        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
        fastcgi_param PATH_TRANSLATED $document_root$fastcgi_path_info;

        fastcgi_pass php-app:9000;
        fastcgi_index index.php;
    }
}
```

Deploy to Elastic Beanstalk

Your application folder now contains the following files:

```
### Dockerrun.aws.json
### php-app
#   ### index.php
#   ### static.html
### proxy
###   conf.d
###     default.conf
```

This is all you need to create the Elastic Beanstalk environment. Create a `.zip` archive of the above files and folders (not including the top-level project folder). To create the archive in Windows explorer, select the contents of the project folder, right-click, select **Send To**, and then click **Compressed (zipped) Folder**.

Note

For information on the required file structure and instructions for creating archives in other environments, see [Create an application source bundle \(p. 342\)](#).

Next, upload the source bundle to Elastic Beanstalk and create your environment. For **Platform**, select **Docker**. For **Platform branch**, select **Multi-container Docker running on 64bit Amazon Linux**.

To launch an environment (console)

1. Open the Elastic Beanstalk console with this preconfigured link:
console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Upload your code**.
4. Choose **Local file**, choose **Choose file**, and then open the source bundle.
5. Choose **Review and launch**.
6. Review the available settings, and then choose **Create app**.

The Elastic Beanstalk console redirects you to the management dashboard for your new environment. This screen shows the health status of the environment and events output by the Elastic Beanstalk service. When the status is Green, click the URL next to the environment name to see your new website.

Connect to a container instance

Next you will connect to an Amazon EC2 instance in your Elastic Beanstalk environment to see some of the moving parts in action.

The easiest way to connect to an instance in your environment is by using the EB CLI. To use it, [install the EB CLI \(p. 853\)](#), if you haven't done so already. You'll also need to configure your environment with an Amazon EC2 SSH key pair. Use either the console's [security configuration page \(p. 510\)](#) or the EB CLI [eb init \(p. 907\)](#) command to do that. To connect to an environment instance, use the EB CLI [eb ssh \(p. 926\)](#) command.

Now that you're connected to an Amazon EC2 instance hosting your Docker containers, you can see how things are set up. Run `ls` on `/var/app/current`:

```
[ec2-user@ip-10-0-0-117 ~]$ ls /var/app/current
Dockerrun.aws.json  php-app  proxy
```

This directory contains the files from the source bundle that you uploaded to Elastic Beanstalk during environment creation.

```
[ec2-user@ip-10-0-0-117 ~]$ ls /var/log/containers
nginx                nginx-proxy-ffffd873ada5-stdouterr.log  rotated
nginx-66a4fd37eb63-stdouterr.log        php-app
nginx-proxy          php-app-b894601a1364-stdouterr.log
```

This is where logs are created on the container instance and collected by Elastic Beanstalk. Elastic Beanstalk creates a volume in this directory for each container, which you mount to the container location where logs are written.

You can also take a look at Docker to see the running containers with `docker ps`.

```
[ec2-user@ip-10-0-0-117 ~]$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
ffffd873ada5	nginx:1.7	"nginx -g 'daemon of	About an
hour ago	Up About an hour	443/tcp, 0.0.0.0:80->80/tcp	ecs-eb-dv-example-env-
ycmk5geqrm-2-nginx-proxy-90fce996cc8cbeeb2800			
b894601a1364	php:5-fpm	"php-fpm"	About an
hour ago	Up About an hour	9000/tcp	ecs-eb-dv-example-env-
ycmk5geqrm-2-php-app-cec0918ed1a3a49a8001			
09fb19828e38	amazon/amazon-ecs-agent:latest	"/agent"	About an hour
ago	Up About an hour	127.0.0.1:51678->51678/tcp	ecs-agent

This shows the two running containers that you deployed, as well as the Amazon ECS container agent that coordinated the deployment.

Inspect the Amazon ECS container agent

Amazon EC2 instances in a Multicontainer Docker environment on Elastic Beanstalk run an agent process in a Docker container. This agent connects to the Amazon ECS service in order to coordinate container deployments. These deployments run as tasks in Amazon ECS, which are configured in task definition files. Elastic Beanstalk creates these task definition files based on the `Dockerrun.aws.json` that you upload in a source bundle.

Check the status of the container agent with an HTTP get request to `http://localhost:51678/v1/metadata`:

```
[ec2-user@ip-10-0-0-117 ~]$ curl http://localhost:51678/v1/metadata
{
  "Cluster": "eb-dv-example-env-qpoziguye24",
  "ContainerInstanceArn": "arn:aws:ecs:us-east-2:123456789012:container-
instance/6a72af64-2838-400d-be09-3ab2d836ebcd"
}
```

This structure shows the name of the Amazon ECS cluster, and the ARN ([Amazon Resource Name](#)) of the cluster instance (the Amazon EC2 instance that you are connected to).

For more information, make an HTTP get request to information is available at `http://localhost:51678/v1/tasks`:

```
[ec2-user@ip-10-0-0-117 ~]$ curl http://localhost:51678/v1/tasks
{
  "Tasks": [
    {
      "Arn": "arn:aws:ecs:us-east-2:123456789012:task/3ff2bf0f-790d-4f6d-affb-5b127b3b6e4a",
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Family": "eb-dv-example-env-qpoziguye24",
      "Version": "2",
      "Containers": [
        {
          "DockerId": "b894601a1364a438156a239813c77cdef17040785bc4d5e49349470dc1556b15",
          "DockerName": "ecs-eb-dv-example-env-qpoziguye24-2-php-app-cec0918ed1a3a49a8001",
          "Name": "php-app"
        },
        {
          "DockerId": "ffffd873ada5f537c88862cce4e1de7ec3edf962645982fb236961c833a5d0fe",
          "DockerName": "ecs-eb-dv-example-env-qpoziguye24-2-nginx-
proxy-90fce996cc8cbech2800",
          "Name": "nginx-proxy"
        }
      ]
    }
  ]
}
```

This structure describes the task that is run to deploy the two docker containers from this tutorial's example project. The following information is displayed:

- **KnownStatus** – The `RUNNING` status indicates that the containers are still active.
- **Family** – The name of the task definition that Elastic Beanstalk created from `Dockerrun.aws.json`.
- **Version** – The version of the task definition. This is incremented each time the task definition file is updated.
- **Containers** – Information about the containers running on the instance.

Even more information is available from the Amazon ECS service itself, which you can call using the AWS Command Line Interface. For instructions on using the AWS CLI with Amazon ECS, and information about Amazon ECS in general, see the [Amazon ECS User Guide](#).

Preconfigured Docker containers

Elastic Beanstalk has a platform version running a Docker container that is preconfigured with the Java EE Glassfish application server software stack. You can use the preconfigured Docker container

to develop and test your application locally and then deploy the application in an Elastic Beanstalk environment that is identical to your local environment.

Note

Elastic Beanstalk also supports platform versions with preconfigured Docker containers for Go and Python. These platform versions are scheduled for retirement.

The following section provides a detailed procedure for deploying an application to Elastic Beanstalk using a preconfigured Docker container.

For details about currently supported preconfigured Docker platform versions, see [Preconfigured Docker](#) in the *AWS Elastic Beanstalk Platforms* document.

Getting started with preconfigured Docker containers

This section shows you how to develop an example application locally and then deploy your application to Elastic Beanstalk with a preconfigured Docker container.

Set up your local development environment

For this walk-through we use a Glassfish example application.

To set up your environment

1. Create a new folder for the example application.

```
~$ mkdir eb-preconf-example  
~$ cd eb-preconf-example
```

2. Download the example application code into the new folder.

```
~$ wget https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/samples/docker-glassfish-v1.zip  
~$ unzip docker-glassfish-v1.zip  
~$ rm docker-glassfish-v1.zip
```

Develop and test locally

To develop an example glassfish application

1. Add a `Dockerfile` to your application's root folder. In the file, specify the AWS Elastic Beanstalk Docker base image to be used to run your local preconfigured Docker container. You'll later deploy your application to an Elastic Beanstalk Preconfigured Docker Glassfish platform version. Choose the Docker base image that this platform version uses. To find out the current Docker image of the platform version, see the [Preconfigured Docker](#) section of the *AWS Elastic Beanstalk Supported Platforms* page in the *AWS Elastic Beanstalk Platforms* guide.

Example `~/Eb-preconf-example/Dockerfile`

```
# For Glassfish 5.0 Java 8  
FROM amazon/aws-eb-glassfish:5.0-al-onbuild-2.11.1
```

For more information about using a `Dockerfile`, see [Single Container Docker configuration \(p. 54\)](#).

2. Build the Docker image.

```
~/eb-preconf-example$ docker build -t my-app-image .
```

3. Run the Docker container from the image.

Note

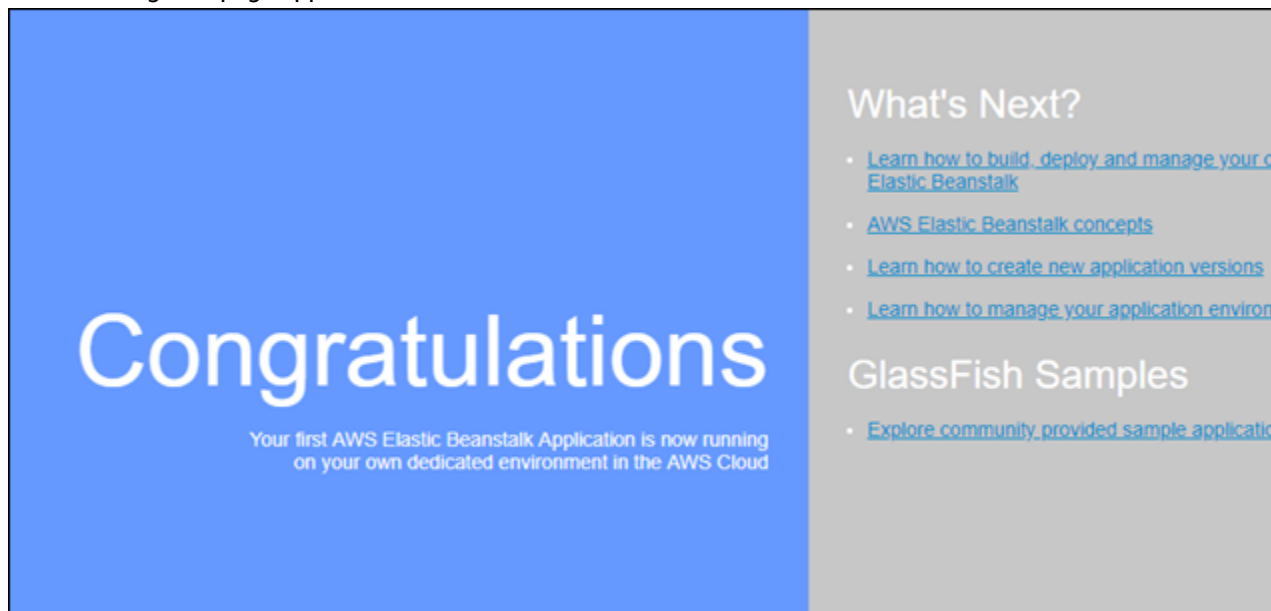
You must include the `-p` flag to map port 8080 on the container to the localhost port 3000. Elastic Beanstalk Docker containers always expose the application on port 8080 on the container. The `-it` flags run the image as an interactive process. The `--rm` flag cleans up the container file system when the container exits. You can optionally include the `-d` flag to run the image as a daemon.

```
$ docker run -it --rm -p 3000:8080 my-app-image
```

4. To view the example application, type the following URL into your web browser.

```
http://localhost:3000
```

The following web page appears.



Deploy to Elastic Beanstalk

After testing your application, you are ready to deploy it to Elastic Beanstalk.

To deploy your application to Elastic Beanstalk

1. In your application's root folder, rename the `Dockerfile` to `Dockerfile.local`. This step is required for Elastic Beanstalk to use the `Dockerfile` that contains the correct instructions for Elastic Beanstalk to build a customized Docker image on each Amazon EC2 instance in your Elastic Beanstalk environment.

Note

You do not need to perform this step if your `Dockerfile` includes instructions that modify the platform version's base Docker image. You do not need to use a `Dockerfile` at all if your `Dockerfile` includes only a `FROM` line to specify the base image from which to build the container. In that situation, the `Dockerfile` is redundant.

2. Create an application source bundle.

```
~/eb-preconf-example$ zip myapp.zip -r *
```

3. Open the Elastic Beanstalk console with this preconfigured link:
console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
4. For **Platform**, under **Preconfigured – Docker**, choose **Glassfish**.
5. For **Application code**, choose **Upload your code**, and then choose **Upload**.
6. Choose **Local file**, choose **Browse**, and then open the application source bundle you just created.
7. Choose **Upload**.
8. Choose **Review and launch**.
9. Review the available settings, and then choose **Create app**.
10. When the environment is created, you can view the deployed application. Choose the environment URL that is displayed at the top of the console dashboard.

Configuring Docker environments

There are several ways to configure the behavior of your Elastic Beanstalk Docker environment.

Note

If your Elastic Beanstalk environment uses an Amazon Linux AMI Docker platform version (preceding Amazon Linux 2), be sure to read the additional information in [the section called “Docker configuration on Amazon Linux AMI \(preceding Amazon Linux 2\)”](#) (p. 78).

Sections

- [Configuring software in Docker environments](#) (p. 74)
- [Docker images](#) (p. 75)
- [Reclaiming Docker storage space](#) (p. 77)
- [Configuring managed updates for Docker environments](#) (p. 77)
- [Docker configuration on Amazon Linux AMI \(preceding Amazon Linux 2\)](#) (p. 78)
- [Docker configuration namespaces](#) (p. 79)

Configuring software in Docker environments

You can use the Elastic Beanstalk console to configure the software running on your environment's instances.

To configure your Docker environment in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. Make necessary configuration changes.

6. Choose **Apply**.

For information about configuring software settings in any environment, see [the section called “Software settings” \(p. 516\)](#). The following sections cover Docker specific information.

Container options

The **Container options** section has platform-specific options. For Docker environments, it lets you choose whether or not your environment includes the Nginx proxy server.

Environment properties

The **Environment properties** section lets you specify environment configuration settings on the Amazon Elastic Compute Cloud (Amazon EC2) instances that are running your application. Environment properties are passed in as key-value pairs to the application.

In a Docker environment, Elastic Beanstalk passes environment properties to Docker containers as environment variables. Your application code running in a container can refer to an environment variable by name and read its value. The code to do so depends on the programming language you're using. You can find instructions for reading environment variable values in the programming languages that Elastic Beanstalk managed platforms support in the respective platform topic. For a list of links to these topics, see [the section called “Software settings” \(p. 516\)](#).

Docker images

The single container and multicontainer Docker platforms for Elastic Beanstalk support the use of Docker images stored in a public or private online image repository.

Specify images by name in `DockerRun.aws.json`. Note these conventions:

- Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).
- Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`).
- Images in other online repositories are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu` or `account-id.dkr.ecr.us-east-2.amazonaws.com/ubuntu:trusty`).

For single container environments only, you can also build your own image during environment creation with a Dockerfile. See [Building custom images with a Dockerfile \(p. 58\)](#) for details.

Using images from an Amazon ECR repository

You can store your custom Docker images in AWS with [Amazon Elastic Container Registry \(Amazon ECR\)](#). When you store your Docker images in Amazon ECR, Elastic Beanstalk automatically authenticates to the Amazon ECR registry with your environment's [instance profile \(p. 21\)](#), so you don't need to [generate an authentication file \(p. 76\)](#) and upload it to Amazon Simple Storage Service (Amazon S3).

You do, however, need to provide your instances with permission to access the images in your Amazon ECR repository by adding permissions to your environment's instance profile. You can attach the [AmazonEC2ContainerRegistryReadOnly](#) managed policy to the instance profile to provide read-only access to all Amazon ECR repositories in your account, or grant access to single repository by using the following template to create a custom policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
{
  "Sid": "AllowEbAuth",
  "Effect": "Allow",
  "Action": [
    "ecr:GetAuthorizationToken"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Sid": "AllowPull",
  "Effect": "Allow",
  "Resource": [
    "arn:aws:ecr:us-east-2:account-id:repository/repository-name"
  ],
  "Action": [
    "ecr:GetAuthorizationToken",
    "ecr:BatchCheckLayerAvailability",
    "ecr:GetDownloadUrlForLayer",
    "ecr:GetRepositoryPolicy",
    "ecr:DescribeRepositories",
    "ecr:ListImages",
    "ecr:BatchGetImage"
  ]
}
]
```

Replace the Amazon Resource Name (ARN) in the above policy with the ARN of your repository.

In your `Dockerrun.aws.json` file, refer to the image by URL. For the [single container platform \(p. 54\)](#), the URL goes in the `Image` definition:

```
"Image": {
  "Name": "account-id.dkr.ecr.us-east-2.amazonaws.com/repository-name:latest",
  "Update": "true"
},
```

For the [multicontainer platform \(p. 62\)](#), use the `image` key in a container definition object:

```
"containerDefinitions": [
  {
    "name": "my-image",
    "image": "account-id.dkr.ecr.us-east-2.amazonaws.com/repository-name:latest",
```

Using images from a private repository

To use a Docker image in a private repository hosted by an online registry, you must provide an authentication file that contains information required to authenticate with the registry.

Generate an authentication file with the **docker login** command. For repositories on Docker Hub, run `docker login`:

```
$ docker login
```

For other registries, include the URL of the registry server:

```
$ docker login registry-server-url
```

Note

If your Elastic Beanstalk environment uses an Amazon Linux AMI Docker platform version (preceding Amazon Linux 2), read the additional information in [the section called “Docker configuration on Amazon Linux AMI \(preceding Amazon Linux 2\)”](#) (p. 78).

Upload a copy named `.dockercfg` of the authentication file to a secure Amazon S3 bucket. The Amazon S3 bucket must be hosted in the same AWS Region as the environment that is using it. Elastic Beanstalk cannot download files from an Amazon S3 bucket hosted in other Regions. Grant permissions for the `s3:GetObject` operation to the IAM role in the instance profile. For details, see [Managing Elastic Beanstalk instance profiles](#) (p. 760).

Include the Amazon S3 bucket information in the `Authentication (v1)` or `authentication (v2)` parameter in your `Dockerrun.aws.json` file.

For more information about the `Dockerrun.aws.json` format for single container environments, see [Single Container Docker configuration](#) (p. 54). For multicontainer environments, see [Multicontainer Docker configuration](#) (p. 62).

For more information about the authentication file, see [Store images on Docker Hub](#) and [docker login](#) on the Docker website.

Reclaiming Docker storage space

Docker does not clean up (delete) the space used when a file is created and then deleted from within a running container; the space is only returned to the pool once the container is deleted. This becomes an issue if a container process creates and deletes many files, such as regularly dumping database backups, filling up the application storage space.

One solution is to increase the size of the application storage space, as described in the previous section. The other option is less-performant: run `fstrim` on the host OS periodically, such as using `cron`, against container free space to reclaim the unused container data blocks.

```
docker ps -q | xargs docker inspect --format='{{ .State.Pid }}' | xargs -IZ sudo fstrim /proc/Z/root/
```

Configuring managed updates for Docker environments

With [managed platform updates](#) (p. 420), you can configure your environment to automatically update to the latest version of a platform on a schedule.

In the case of Docker environments, you might want to decide if an automatic platform update should happen across Docker versions—when the new platform version includes a new Docker version. Elastic Beanstalk supports managed platform updates across Docker versions when updating from an environment running a Docker platform version newer than 2.9.0. When a new platform version includes a new version of Docker, Elastic Beanstalk increments the minor update version number. Therefore, to allow managed platform updates across Docker versions, enable managed platform updates for both minor and patch version updates. To prevent managed platform updates across Docker versions, enable managed platform updates to apply patch version updates only.

For example, the following [configuration file](#) (p. 600) enables managed platform updates at 9:00 AM UTC each Tuesday for both minor and patch version updates, thereby allowing for managed updates across Docker versions:

Example `.ebextensions/managed-platform-update.config`

```
option_settings:
```

```
aws:elasticbeanstalk:managedactions:  
  ManagedActionsEnabled: true  
  PreferredStartTime: "Tue:09:00"  
aws:elasticbeanstalk:managedactions:platformupdate:  
  UpdateLevel: minor
```

For environments running Docker platform versions 2.9.0 or earlier, Elastic Beanstalk never performs managed platform updates if the new platform version includes a new Docker version.

Docker configuration on Amazon Linux AMI (preceding Amazon Linux 2)

If your Elastic Beanstalk Docker environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Using an authentication file for a private repository

This information is relevant to you if you are [using images from a private repository \(p. 76\)](#). Beginning with Docker version 1.7, the **docker login** command changed the name of the authentication file, and the format of the file. Amazon Linux AMI Docker platform versions (preceding Amazon Linux 2) require the older `~/.dockercfg` format configuration file.

With Docker version 1.7 and later, the **docker login** command creates the authentication file in `~/.docker/config.json` in the following format.

```
{  
  "auths": {  
    "server": {  
      "auth": "key"  
    }  
  }  
}
```

With Docker version 1.6.2 and earlier, the **docker login** command creates the authentication file in `~/.dockercfg` in the following format.

```
{  
  "server" :  
  {  
    "auth" : "auth_token",  
    "email" : "email"  
  }  
}
```

To convert a `config.json` file, remove the outer `auths` key, add an `email` key, and flatten the JSON document to match the old format.

On Amazon Linux 2 Docker platform versions, Elastic Beanstalk uses the newer authentication file name and format. If you're using an Amazon Linux 2 Docker platform version, you can use the authentication file that the **docker login** command creates without any conversion.

Configuring additional storage volumes

For improved performance on Amazon Linux AMI, Elastic Beanstalk configures two Amazon EBS storage volumes for your Docker environment's Amazon EC2 instances. In addition to the root volume provisioned for all Elastic Beanstalk environments, a second 12GB volume named `xvdcz` is provisioned for image storage on Docker environments.

If you need more storage space or increased IOPS for Docker images, you can customize the image storage volume by using the `BlockDeviceMapping` configuration option in the [aws:autoscaling:launchconfiguration](#) (p. 556) namespace.

For example, the following [configuration file](#) (p. 600) increases the storage volume's size to 100 GB with 500 provisioned IOPS:

Example `.ebextensions/blockdevice-xvdcz.config`

```
option_settings:
  aws:autoscaling:launchconfiguration:
    BlockDeviceMappings: /dev/xvdcz=:100::io1:500
```

If you use the `BlockDeviceMappings` option to configure additional volumes for your application, you should include a mapping for `xvdcz` to ensure that it is created. The following example configures two volumes, the image storage volume `xvdcz` with default settings and an additional 24 GB application volume named `sdh`:

Example `.ebextensions/blockdevice-sdh.config`

```
option_settings:
  aws:autoscaling:launchconfiguration:
    BlockDeviceMappings: /dev/xvdcz=:12:true:gp2,/dev/sdh=:24
```

Note

When you change settings in this namespace, Elastic Beanstalk replaces all instances in your environment with instances running the new configuration. See [Configuration changes](#) (p. 408) for details.

Docker configuration namespaces

You can use a [configuration file](#) (p. 600) to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be defined by the Elastic Beanstalk service or the platform that you use and are organized into *namespaces*.

The Docker platform supports options in the following namespaces, in addition to the [options supported for all Elastic Beanstalk environments](#) (p. 555):

- `aws:elasticbeanstalk:environment:proxy` – Choose the proxy server for your environment. Docker supports either running Nginx or no proxy server.

The following example configuration file configures a Docker environment to run no proxy server.

Example `.ebextensions/docker-settings.config`

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: none
```

Running a Docker environment locally with the EB CLI

You can use the Elastic Beanstalk Command Line Interface (EB CLI) to run the Docker containers configured in your AWS Elastic Beanstalk application locally. The EB CLI uses the Docker configuration

file (Dockerfile or Dockerrun.aws.json) and source code in your project directory to run your application locally in Docker.

The EB CLI supports running single container, multicontainer, and preconfigured container applications locally.

Topics

- [Prerequisites for running Docker applications locally \(p. 80\)](#)
- [Preparing a Docker application for use with the EB CLI \(p. 81\)](#)
- [Running a Docker application locally \(p. 81\)](#)
- [Cleaning up after running a Docker application locally \(p. 82\)](#)

Prerequisites for running Docker applications locally

- Linux OS or Mac OS X
- [EB CLI version 3.3 or greater \(p. 853\)](#)

Run **eb init** in your project directory to initialize an EB CLI repository. If you haven't used the EB CLI before, see [Managing Elastic Beanstalk environments with the EB CLI \(p. 864\)](#).

- [Docker version 1.6 or greater](#)

Add yourself to the `docker` group, log out, and then log back in to ensure that you can run Docker commands without `sudo`:

```
$ sudo usermod -a -G docker $USER
```

Run `docker ps` to verify that the Docker daemon is up and running:

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS              NAMES
```

- A Docker application

If you don't have a Docker application in a project folder on your local machine, see [Deploying Elastic Beanstalk applications from Docker containers \(p. 50\)](#) for an introduction to using Docker with AWS Elastic Beanstalk.

- Docker profile (optional)

If your application uses Docker images that are in a private repository, run `docker login` and follow the prompts to create an authentication profile.

- `w3m` (optional)

`W3m` is a web browser that you can use to view your running web application within a command line terminal with **eb local run**. If you are using the command line in a desktop environment, you don't need `w3m`.

Docker containers run locally without emulating AWS resources that are provisioned when you deploy an application to Elastic Beanstalk, including security groups and data or worker tiers.

You can configure your local containers to connect to a database by passing the necessary connection string or other variables with the `envvars` option, but you must ensure that any resources in AWS are accessible from your local machine by [opening the appropriate ports](#) in their assigned security groups or attaching a [default gateway](#) or [elastic IP address](#).

Preparing a Docker application for use with the EB CLI

Prepare your Docker configuration file and source data as though you were deploying them to Elastic Beanstalk. This topic uses the PHP and nginx proxy example from the [Multicontainer Docker tutorial \(p. 66\)](#) earlier in this guide as an example, but you can use the same commands with any single container, multicontainer, or preconfigured Docker application.

Running a Docker application locally

Run your Docker application locally with the **eb local run** command from within the project directory:

```
~/project$ eb local run
Creating elasticbeanstalk_phpapp_1...
Creating elasticbeanstalk_nginxproxy_1...
Attaching to elasticbeanstalk_phpapp_1, elasticbeanstalk_nginxproxy_1
phpapp_1      | [23-Apr-2015 23:24:25] NOTICE: fpm is running, pid 1
phpapp_1      | [23-Apr-2015 23:24:25] NOTICE: ready to handle connections
```

The EB CLI reads the Docker configuration and executes the Docker commands necessary to run your application. The first time you run a project locally, Docker downloads images from a remote repository and stores them on your local machine. This process can take several minutes.

Note

The **eb local run** command takes two optional parameters, `port` and `envvars`. To override the default port for a single container application, use the `port` option:

```
$ eb local run --port 8080
```

This command tells the EB CLI to use port 8080 on the host and map it to the exposed port on the container. If you don't specify a port, the EB CLI uses the container's port for the host. This option only works with single container applications

To pass environment variables to the application containers, use the `envvars` option:

```
$ eb local run --envvars RDS_HOST=#RDS_HOST,RDS_DB=#RDS_DB,RDS_USER=#RDS_USER,RDS_PASS=#RDS_PASS
```

Use environment variables to configure a database connection, set debug options, or pass secrets securely to your application. For more information on the options supported by the **eb local** subcommands, see [eb local \(p. 911\)](#).

After the containers are up and running in Docker, they are ready to take requests from clients. The **eb local** process stays open as long as the containers are running. If you need to stop the process and containers, press **Ctrl+C**.

Open a second terminal to run additional commands while the **eb local** process is running. Use **eb local status** to view your application's status:

```
~/project$ eb local status
Platform: 64bit Amazon Linux 2014.09 v1.2.1 running Multi-container Docker 1.3.3 (Generic)
Container name: elasticbeanstalk_nginxproxy_1
Container ip: 127.0.0.1
Container running: True
Exposed host port(s): 80
Full local URL(s): 127.0.0.1:80

Container name: elasticbeanstalk_phpapp_1
Container ip: 127.0.0.1
```



```
Container running: True  
Exposed host port(s): None  
Full local URL(s): None
```

You can use `docker ps` to see the status of the containers from Docker's point of view:

```
~/project$ docker ps  
CONTAINER ID   PORTS          IMAGE           COMMAND                                     CREATED        STATUS  
6a8e71274fed   0.0.0.0:80->80/tcp, 443/tcp  nginx:latest    "nginx -g 'daemon of 9 minutes ago    Up 9  
82cbf620bdc1   9000/tcp       php:fpm         "php-fpm"                               9 minutes ago    Up 9  
elasticbeanstalk_nginxproxy_1  
elasticbeanstalk_phpapp_1
```

Next, view your application in action with `eb local open`:

```
~/project$ eb local open
```

This command opens your application in the default web browser. If you are running a terminal in a desktop environment, this may be Firefox, Safari, or Google Chrome. If you are running a terminal in a headless environment or over an SSH connection, a command line browser, such as `w3m`, will be used if one is available.

Switch back to the terminal running the application process for a moment and note the additional output:

```
phpapp_1 | 172.17.0.36 - 21/Apr/2015:23:46:17 +0000 "GET /index.php" 200
```

This shows that the web application in the Docker container received an HTTP GET request for `index.php` that was returned successfully with a 200 (non error) status.

Run `eb local logs` to see where the EB CLI writes the logs.

```
~/project$ eb local logs  
Elastic Beanstalk will write logs locally to /home/user/project/.elasticbeanstalk/logs/local.  
Logs were most recently created 3 minutes ago and written to /home/user/project/.elasticbeanstalk/logs/local/150420_234011665784.
```

Cleaning up after running a Docker application locally

When you are done testing your application locally, you can stop the applications and remove the images downloaded by Docker when you use `eb local run`. Removing the images is optional. You may want to keep them for future use.

Return to the terminal running the `eb local` process and press **Ctrl+C** to stop the application:

```
^CGracefully stopping... (press Ctrl+C again to force)  
Stopping elasticbeanstalk_nginxproxy_1...  
Stopping elasticbeanstalk_phpapp_1...  
  
Aborting.  
[1]+  Exit 5                  eb local run
```

The EB CLI attempts to stop each running container gracefully with Docker commands. If you need to stop a process immediately, press **Ctrl+C** again.

After you stop the applications, the Docker containers should also stop running. Verify this with `docker ps`:

```
$ docker ps --all
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
73d515d99d2a       nginx:latest       "nginx -g 'daemon of 21 minutes ago      Exited
(0) 11 minutes ago elasticbeanstalk_nginxproxy_1
7061c76220de       php:fpm            "php-fpm"          21 minutes ago     Exited
(0) 11 minutes ago elasticbeanstalk_phpapp_1
```

The `all` option shows stopped containers (if you omitted this option, the output will be blank). In the above example, Docker shows that both containers exited with a 0 (non-error) status.

If you are done using Docker and EB CLI local commands, you can remove the Docker images from your local machine to save space.

To remove Docker images from your local machine

1. View the images that you downloaded using `docker images`:

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL
SIZE
php                 fpm                68bc5150cffc       1 hour ago         414.1
MB
nginx              latest             637d3b2f5fb5       1 hour ago         93.44
MB
```

2. Remove the two Docker containers with `docker rm`:

```
$ docker rm 73d515d99d2a 7061c76220de
73d515d99d2a
7061c76220de
```

3. Remove the images with `docker rmi`:

```
$ docker rmi 68bc5150cffc 637d3b2f5fb5
Untagged: php:fpm
Deleted: 68bc5150cffc0526c66b92265c3ed8f2ea50f3c71d266aa655b7a4d20c3587b0
Untagged: nginx:latest
Deleted: 637d3b2f5fb5c4f70895b77a9e76751a6e7670f4ef27a159dad49235f4fe61e0
```

Creating and deploying Go applications on Elastic Beanstalk

Topics

- [Getting started with Go on Elastic Beanstalk \(p. 84\)](#)
- [Setting up your Go development environment \(p. 86\)](#)
- [Using the Elastic Beanstalk Go platform \(p. 86\)](#)
- [Deploying a Go application to Elastic Beanstalk \(p. 91\)](#)

AWS Elastic Beanstalk for Go makes it easy to deploy, manage, and scale your Go web applications using Amazon Web Services. Elastic Beanstalk for Go is available to anyone developing or hosting a web

application using Go. This chapter provides step-by-step instructions for deploying your web application to Elastic Beanstalk.

After you deploy your Elastic Beanstalk application, you can continue to use the EB CLI to manage your application and environment, or you can use the Elastic Beanstalk console, AWS CLI, or the APIs.

The topics in this chapter assume some knowledge of Elastic Beanstalk environments. If you haven't used Elastic Beanstalk before, try the [getting started tutorial \(p. 3\)](#) to learn the basics.

Getting started with Go on Elastic Beanstalk

To get started with Go applications on AWS Elastic Beanstalk, all you need is an application [source bundle \(p. 342\)](#) to upload as your first application version, and deploy it to an environment. When you create an environment, Elastic Beanstalk allocates all of the AWS resources needed to run a highly scalable web application.

Launching an environment with a sample Go application

Elastic Beanstalk provides single-page sample applications for each platform. Elastic Beanstalk also provides more complex examples that show the use of additional AWS resources, such as Amazon RDS, and language or platform-specific features and APIs.

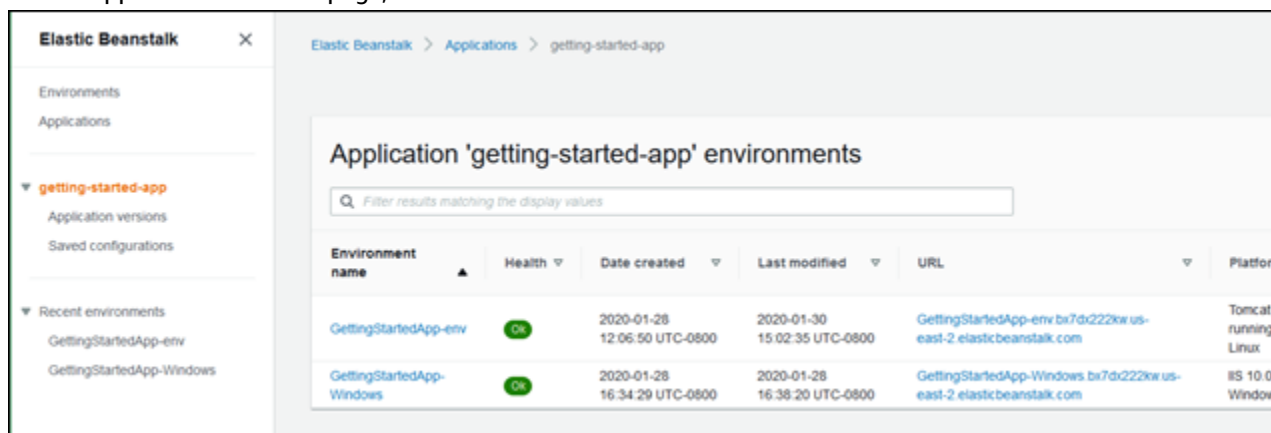
Samples

Supported configurations	Environment type	Source bundle	Description
Go	Web server	go-v1.zip	Single page application.

Download the sample application and deploy it to Elastic Beanstalk by following these steps.

To launch an environment with a sample application (console)

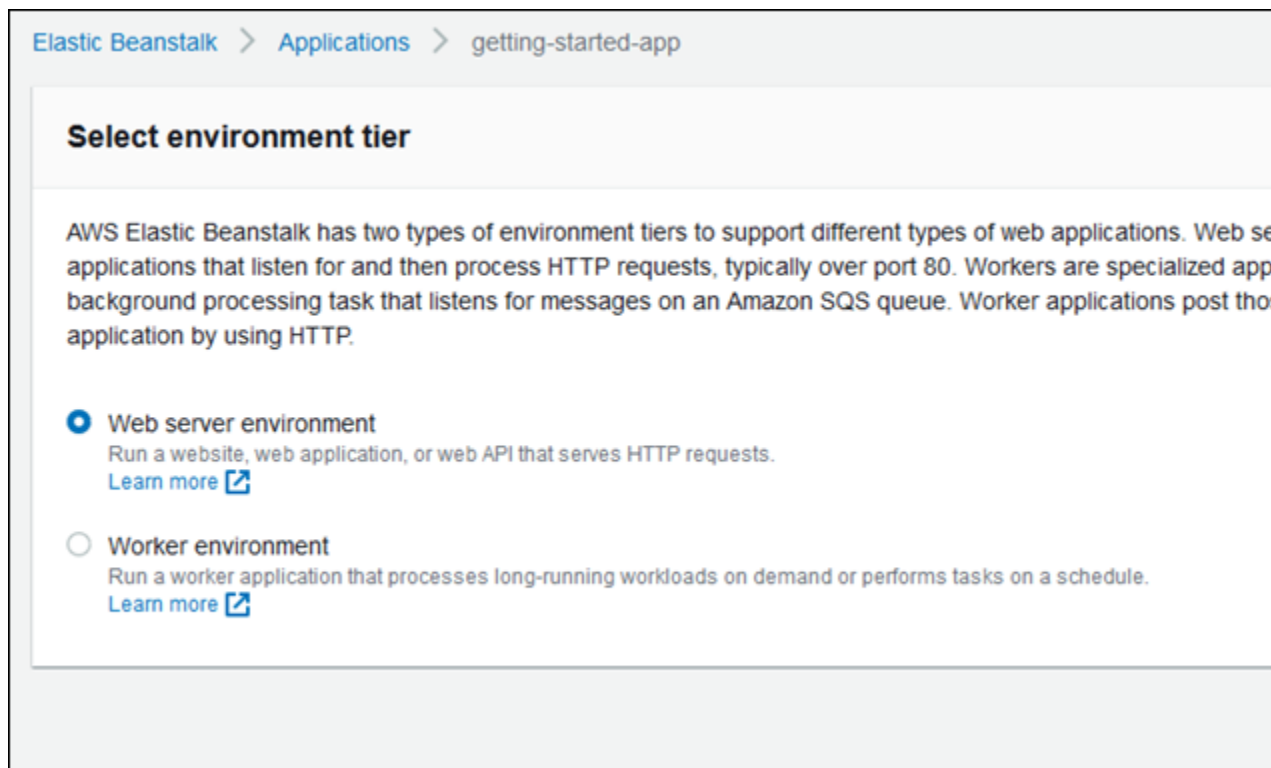
1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose an existing application's name on the list or [create one \(p. 334\)](#).
3. On the application overview page, choose **Create a new environment**.



4. Choose the **Web server environment** or **Worker environment** [environment tier \(p. 13\)](#). You can't change an environment's tier after creation.

Note

The [.NET on Windows Server platform \(p. 137\)](#) doesn't support the worker environment tier.



5. For **Platform**, select the platform and platform branch that match the language used by your application.

Note

Elastic Beanstalk supports multiple [versions \(p. 29\)](#) for most of the platforms that are listed. By default, the console selects the recommended version for the platform and platform branch you choose. If your application requires different version, you can select it here, or by choosing **Configure more options**, as described in step 7. For information about supported platform versions, see [the section called "Supported platforms" \(p. 29\)](#).

6. For **Application code**, choose **Sample application**.
7. To further customize your environment, choose **Configure more options**. You can set the following options only during environment creation:
 - Environment name
 - Domain name
 - Platform version
 - VPC
 - Tier

You can change the following settings after environment creation, but they require new instances or other resources to be provisioned and can take a long time to apply:

- Instance type, root volume, key pair, and AWS Identity and Access Management (IAM) role
- Internal Amazon RDS database
- Load balancer

For details on all available settings, see [The create new environment wizard \(p. 365\)](#).

8. Choose **Create environment**.

Next steps

After you have an environment running an application, you can deploy a new version of the application or a different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances.

After you deploy a sample application or two and are ready to start developing and running Go applications locally, see [Setting up your Go development environment \(p. 86\)](#).

Setting up your Go development environment

Set up a Go development environment to test your application locally before you deploy it to AWS Elastic Beanstalk. This topic describes the setup steps for your development environment and provides links to installation pages for useful tools.

For common setup steps and tools that apply to all languages, see [Configuring your development machine for use with Elastic Beanstalk \(p. 849\)](#).

Installing Go

To run Go applications locally, install Go. If you don't need a specific version, get the latest version that Elastic Beanstalk supports. For a list of supported versions, see [Go](#) in the *AWS Elastic Beanstalk Platforms* document.

Download Go at <https://golang.org/doc/install>.

Installing the AWS SDK for Go

If you need to manage AWS resources from within your application, install the AWS SDK for Go by using the following command.

```
$ go get github.com/aws/aws-sdk-go
```

For more information, see [AWS SDK for Go](#).

Using the Elastic Beanstalk Go platform

Important

Amazon Linux 2 platform versions are fundamentally different than Amazon Linux AMI platform versions (preceding Amazon Linux 2). These different platform generations are incompatible in several ways. If you are migrating to an Amazon Linux 2 platform version, be sure to read the information in [the section called "Upgrade to Amazon Linux 2" \(p. 426\)](#).

You can use AWS Elastic Beanstalk to run, build, and configure Go-based applications. For simple Go applications, there are two ways to deploy your application:

- Provide a source bundle with a source file at the root called `application.go` that contains the main package for your application. Elastic Beanstalk builds the binary using the following command:

```
go build -o bin/application application.go
```

After the application is built, Elastic Beanstalk starts it on port 5000.

- Provide a source bundle with a binary file called `application`. The binary file can be located either at the root of the source bundle or in the `bin/` directory of the source bundle. If you place the `application` binary file in both locations, Elastic Beanstalk uses the file in the `bin/` directory.

Elastic Beanstalk launches this application on port 5000.

In both cases, with Go 1.11 or later, you can also provide module requirements in a file called `go.mod`. For more information, see [Migrating to Go Modules](#) in the Go blog.

For more complex Go applications, there are two ways to deploy your application:

- Provide a source bundle that includes your application source files, along with a [Buildfile \(p. 90\)](#) and a [Procfile \(p. 89\)](#). The Buildfile includes a command to build the application, and the Procfile includes instructions to run the application.
- Provide a source bundle that includes your application binary files, along with a Procfile. The Procfile includes instructions to run the application.

The Go platform includes a proxy server to serve static assets and forward traffic to your application. You can [extend or override the default proxy configuration \(p. 90\)](#) for advanced scenarios.

For details about the various ways you can extend an Elastic Beanstalk Linux-based platform, see [the section called “Extending Linux platforms” \(p. 31\)](#).

Configuring your Go environment

The Go platform settings let you fine-tune the behavior of your Amazon EC2 instances. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration using the Elastic Beanstalk console.

Use the Elastic Beanstalk console to enable log rotation to Amazon S3 and configure variables that your application can read from the environment.

To configure your Go environment in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.

Log options

The Log Options section has two settings:

- **Instance profile** – Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3** – Specifies whether log files for your application's Amazon EC2 instances should be copied to the Amazon S3 bucket associated with your application.

Static files

To improve performance, the **Static files** section lets you configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. For each directory, you set the virtual path to directory mapping. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

For details about configuring static files using the Elastic Beanstalk console, see [the section called “Static files” \(p. 650\)](#).

Environment properties

The **Environment Properties** section lets you specify environment configuration settings on the Amazon EC2 instances that are running your application. Environment properties are passed in as key-value pairs to the application.

Inside the Go environment running in Elastic Beanstalk, environment variables are accessible using the `os.Getenv` function. For example, you could read a property named `API_ENDPOINT` to a variable with the following code:

```
endpoint := os.Getenv("API_ENDPOINT")
```

See [Environment properties and other software settings \(p. 516\)](#) for more information.

Go configuration namespace

You can use a [configuration file \(p. 600\)](#) to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be defined by the Elastic Beanstalk service or the platform that you use and are organized into *namespaces*.

The Go platform doesn't define any platform-specific namespaces. You can configure the proxy to serve static files by using the `aws:elasticbeanstalk:environment:proxy:staticfiles` namespace. For details and an example, see [the section called “Static files” \(p. 650\)](#).

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See [Configuration options \(p. 536\)](#) for more information.

The Amazon Linux AMI (preceding Amazon Linux 2) Go platform

If your Elastic Beanstalk Go environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Go configuration namespaces

You can use a [configuration file \(p. 600\)](#) to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be defined by the Elastic Beanstalk service or the platform that you use and are organized into *namespaces*.

The Amazon Linux AMI Go platform supports one platform-specific configuration namespace in addition to the [namespaces supported by all platforms \(p. 555\)](#). The `aws:elasticbeanstalk:container:golang:staticfiles` namespace lets you define options that map paths on your web application to folders in your application source bundle that contain static content.

For example, this [configuration file \(p. 600\)](#) tells the proxy server to serve files in the `staticimages` folder at the path `/images`:

Example `.ebextensions/go-settings.config`

```
option_settings:  
  aws:elasticbeanstalk:container:golang:staticfiles:  
    /html: statichtml  
    /images: staticimages
```

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See [Configuration options \(p. 536\)](#) for more information.

Configuring the application process with a Procfile

To specify custom commands to start a Go application, include a file called `Procfile` at the root of your source bundle.

For details about writing and using a `Procfile`, expand the *Buildfile and Procfile* section in [the section called "Extending Linux platforms" \(p. 31\)](#).

Example Procfile

```
web: bin/server  
queue_process: bin/queue_processor  
foo: bin/foapp
```

You must call the main application `web`, and list it as the first command in your `Procfile`. Elastic Beanstalk exposes the main `web` application on the root URL of the environment; for example, `http://my-go-env.elasticbeanstalk.com`.

Elastic Beanstalk also runs any application whose name does not have the `web_` prefix, but these applications are not available from outside of your instance.

Elastic Beanstalk expects processes run from the `Procfile` to run continuously. Elastic Beanstalk monitors these applications and restarts any process that terminates. For short-running processes, use a [Buildfile \(p. 90\)](#) command.

Using a Procfile on Amazon Linux AMI (preceding Amazon Linux 2)

If your Elastic Beanstalk Go environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Port passing

Elastic Beanstalk configures the `nginx` proxy to forward requests to your application on the port number specified in the `PORT` [environment property \(p. 87\)](#) for your application. Your application should always listen on that port. You can access this variable within your application by calling the `os.Getenv("PORT")` method.

Elastic Beanstalk uses the port number specified in the `PORT` environment property for the port for the first application in the `Procfile`, and then increments the port number for each subsequent application in the `Procfile` by 100. If the `PORT` environment property is not set, Elastic Beanstalk uses 5000 for the initial port.

In the preceding example, the `PORT` environment property for the `web` application is 5000, the `queue_process` application is 5100, and the `foo` application is 5200.

You can specify the initial port by setting the `PORT` option with the [aws:elasticbeanstalk:application:environment \(p. 570\)](#) namespace, as shown in the following example.

```
option_settings:
```



```
- namespace:  aws:elasticbeanstalk:application:environment
  option_name:  PORT
  value:  <first_port_number>
```

For more information about setting environment properties for your application, see [Option settings \(p. 601\)](#).

Building executable on-server with a Buildfile

To specify a custom build and configuration command for your Go application, include a file called `Buildfile` at the root of your source bundle. The file name is case sensitive. Use the following format for the `Buildfile`:

```
<process_name>: <command>
```

The command in your `Buildfile` must match the following regular expression: `^[A-Za-z0-9_]+:\s*\.+$.`

Elastic Beanstalk doesn't monitor the application that is run with a `Buildfile`. Use a `Buildfile` for commands that run for short periods and terminate after completing their tasks. For long-running application processes that should not exit, use the [Procfile \(p. 89\)](#) instead.

In the following example of a `Buildfile`, `build.sh` is a shell script that is located at the root of the source bundle:

```
make: ./build.sh
```

All paths in the `Buildfile` are relative to the root of the source bundle. If you know in advance where the files reside on the instance, you can include absolute paths in the `Buildfile`.

Configuring the reverse proxy

Elastic Beanstalk uses `nginx` as the reverse proxy to map your application to your Elastic Load Balancing load balancer on port 80. Elastic Beanstalk provides a default `nginx` configuration that you can either extend or override completely with your own configuration.

By default, Elastic Beanstalk configures the `nginx` proxy to forward requests to your application on port 5000. You can override the default port by setting the `PORT` [environment property \(p. 87\)](#) to the port on which your main application listens.

Note

The port that your application listens on doesn't affect the port that the `nginx` server listens to receive requests from the load balancer.

All Amazon Linux 2 platforms support a uniform proxy configuration feature. For details about configuring the proxy server on the new Amazon Corretto platform versions running Amazon Linux 2, expand the *Reverse Proxy Configuration* section in [the section called "Extending Linux platforms" \(p. 31\)](#).

Configuring the proxy on Amazon Linux AMI (preceding Amazon Linux 2)

If your Elastic Beanstalk Go environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the information in this section.

Extending and overriding the default proxy configuration

Elastic Beanstalk uses `nginx` as the reverse proxy to map your application to your load balancer on port 80. If you want to provide your own `nginx` configuration, you can override the default configuration

provided by Elastic Beanstalk by including the `.ebextensions/nginx/nginx.conf` file in your source bundle. If this file is present, Elastic Beanstalk uses it in place of the default nginx configuration file.

If you want to include directives in addition to those in the `nginx.conf http` block, you can also provide additional configuration files in the `.ebextensions/nginx/conf.d/` directory of your source bundle. All files in this directory must have the `.conf` extension.

To take advantage of functionality provided by Elastic Beanstalk, such as [Enhanced health reporting and monitoring \(p. 691\)](#), automatic application mappings, and static files, you must include the following line in the `server` block of your nginx configuration file:

```
include conf.d/elasticbeanstalk/*.conf;
```

Deploying a Go application to Elastic Beanstalk

This tutorial walks you through the process of creating a Go application and deploying it to an AWS Elastic Beanstalk environment.

Sections

- [Prerequisites \(p. 91\)](#)
- [Create a Go application \(p. 91\)](#)
- [Deploy your Go application with the EB CLI \(p. 92\)](#)
- [Clean up \(p. 93\)](#)

Prerequisites

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in [Getting started using Elastic Beanstalk \(p. 3\)](#) to launch your first Elastic Beanstalk environment.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command  
this is output
```

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

This tutorial uses the Elastic Beanstalk Command Line Interface (EB CLI). For details on installing and configuring the EB CLI, see [Install the EB CLI \(p. 853\)](#) and [Configure the EB CLI \(p. 860\)](#).

Create a Go application

Create a project directory.

```
~$ mkdir eb-go  
~$ cd eb-go
```

Next, create an application that you'll deploy using Elastic Beanstalk. We'll create a "Hello World" RESTful web service.

This example prints a customized greeting that varies based on the path used to access the service.

Create a text file in this directory named `application.go` with the following contents.

Example `~/eb-go/application.go`

```
package main

import (
    "fmt"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    if r.URL.Path == "/" {
        fmt.Fprintf(w, "Hello World! Append a name to the URL to say hello. For example, use %s/
Mary to say hello to Mary.", r.Host)
    } else {
        fmt.Fprintf(w, "Hello, %s!", r.URL.Path[1:])
    }
}

func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":5000", nil)
}
```

Deploy your Go application with the EB CLI

Next, you create your application environment and deploy your configured application with Elastic Beanstalk.

To create an environment and deploy your Go application

1. Initialize your EB CLI repository with the **eb init** command.

```
~/eb-go$ eb init -p go go-tutorial --region us-east-2
Application go-tutorial has been created.
```

This command creates an application named `go-tutorial`, and configures your local repository to create environments with the latest Go platform version.

2. (Optional) Run **eb init** again to configure a default key pair so that you can use SSH to connect to the EC2 instance running your application.

```
~/eb-go$ eb init
Do you want to set up SSH for your instances?
(y/n): y
Select a keypair.
1) my-keypair
2) [ Create new KeyPair ]
```

Select a key pair if you have one already, or follow the prompts to create one. If you don't see the prompt or need to change your settings later, run **eb init -i**.

3. Create an environment and deploy your application to it with **eb create**. Elastic Beanstalk automatically builds a binary file for your application and starts it on port 5000.

```
~/eb-go$ eb create go-env
```

Environment creation takes about five minutes and creates the following resources:

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or nginx as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
- **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

Elastic Beanstalk manages all of these resources. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see [Using Elastic Beanstalk with Amazon S3 \(p. 831\)](#).

When the environment creation process completes, open your website with **eb open**.

```
~/eb-go$ eb open
```

This opens a browser window using the domain name created for your application.

If you don't see your application running, or get an error message, see [Troubleshooting Deployments \(p. 948\)](#) for help with how to determine the cause of the error.

If you *do* see your application running, then congratulations, you've deployed a Go application with Elastic Beanstalk!

Clean up

When you finish working with Elastic Beanstalk, you can terminate your environment. Elastic Beanstalk terminates all AWS resources associated with your environment, such as [Amazon EC2 instances \(p. 451\)](#), [database instances \(p. 506\)](#), [load balancers \(p. 470\)](#), security groups, and [alarms \(p. 470\)](#).

To terminate your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

With Elastic Beanstalk, you can easily create a new environment for your application at any time.

Or, with the EB CLI, do the following.

```
~/eb-go$ eb terminate
```

Creating and deploying Java applications on Elastic Beanstalk

AWS Elastic Beanstalk supports several [platform versions \(p. 29\)](#) for Java applications, including multiple versions of Java with the Tomcat application server and Java-only platform versions for applications that do not use Tomcat.

[Apache Tomcat \(p. 101\)](#) is an open source web container for applications that use Java servlets and JavaServer Pages (JSPs) to serve HTTP requests. Tomcat facilitates web application development by providing multithreading, declarative security configuration, and extensive customization. Platform versions are available for each of Tomcat's current major versions.

[Java SE platform versions \(p. 112\)](#) (without Tomcat) are also provided for applications that don't use a web container, or use one other than Tomcat, such as Jetty or GlassFish. You can include any library Java Archives (JARs) used by your application in the source bundle that you deploy to Elastic Beanstalk.

AWS provides several tools for working with Java and Elastic Beanstalk. Regardless of the platform version that you choose, you can use the [AWS SDK for Java \(p. 101\)](#) to use other AWS services from within your Java application. The AWS SDK for Java is a set of libraries that allow you to use AWS APIs from your application code without writing the raw HTTP calls from scratch.

If you use the Eclipse integrated development environment (IDE) to develop your Java application, you can also get the [AWS Toolkit for Eclipse \(p. 123\)](#). The AWS Toolkit for Eclipse is an open source plug-in that lets you manage AWS resources, including Elastic Beanstalk applications and environments, from within the Eclipse IDE.

If the command line is more your style, install the [Elastic Beanstalk Command Line Interface \(p. 852\)](#) (EB CLI) and use it to create, monitor, and manage your Elastic Beanstalk environments from the command line. If you run multiple environments for your application, the EB CLI integrates with Git to let you associate each of your environments with a different Git branch.

The topics in this chapter assume some knowledge of Elastic Beanstalk environments. If you haven't used Elastic Beanstalk before, try the [getting started tutorial \(p. 3\)](#) to learn the basics.

Topics

- [Getting started with Java on Elastic Beanstalk \(p. 95\)](#)
- [Setting up your Java development environment \(p. 100\)](#)

- [Using the Elastic Beanstalk Tomcat platform \(p. 101\)](#)
- [Using the Elastic Beanstalk Java SE platform \(p. 112\)](#)
- [Adding an Amazon RDS DB instance to your Java application environment \(p. 118\)](#)
- [Using the AWS Toolkit for Eclipse \(p. 123\)](#)
- [Resources \(p. 137\)](#)

Getting started with Java on Elastic Beanstalk

To get started with Java applications on AWS Elastic Beanstalk, all you need is an application [source bundle \(p. 342\)](#) to upload as your first application version and to deploy to an environment. When you create an environment, Elastic Beanstalk allocates all of the AWS resources needed to run a scalable web application.

Launching an environment with a sample Java application

Elastic Beanstalk provides single page sample applications for each platform as well as more complex examples that show the use of additional AWS resources such as Amazon RDS and language or platform-specific features and APIs.

The single page samples are the same code that you get when you create an environment without supplying your own source code. The more complex examples are hosted on GitHub and may need to be compiled or built prior to deploying to an Elastic Beanstalk environment.

Samples

Name	Supported versions	Environment type	Source	Description
Tomcat Default	Tomcat 8 with Java 8	Web Server Worker	java-tomcat-v3.zip	<p>Tomcat web application with a single page (<code>index.jsp</code>) configured to be displayed at the website root.</p> <p>For worker environments (p. 437), this sample includes a <code>cron.yaml</code> file that configures a scheduled task that calls <code>scheduled.jsp</code> once per minute. When <code>scheduled.jsp</code> is called, it writes to a log file at <code>/tmp/sample-app.log</code>. Finally, a configuration file is included in <code>.ebextensions</code> that copies the logs from <code>/tmp/</code> to the locations read by Elastic Beanstalk when you request environment logs.</p> <p>If you enable X-Ray integration (p. 521) on an environment running this sample, the application shows additional content regarding X-Ray and provides an option to generate debug information that you can view in the X-Ray console.</p>

Name	Supported versions	Environment type	Source	Description
Tomcat 7	Tomcat 7 with Java 7 Tomcat 7 with Java 6	Web Service Worker	java7-tomcat7.zip	A version of the Tomcat Default application that targets Java 7 or earlier with Tomcat 7.
Java SE Default	Java 8 Java 7	Web Service	java-se-jetty-gradle-v3.zip	<p>Jetty SE application with Buildfile and Procfile configuration files. The Buildfile in this sample runs a Gradle command to build the application source on-instance.</p> <p>If you enable X-Ray integration (p. 521) on an environment running this sample, the application shows additional content regarding X-Ray and provides an option to generate debug information that you can view in the X-Ray console.</p>

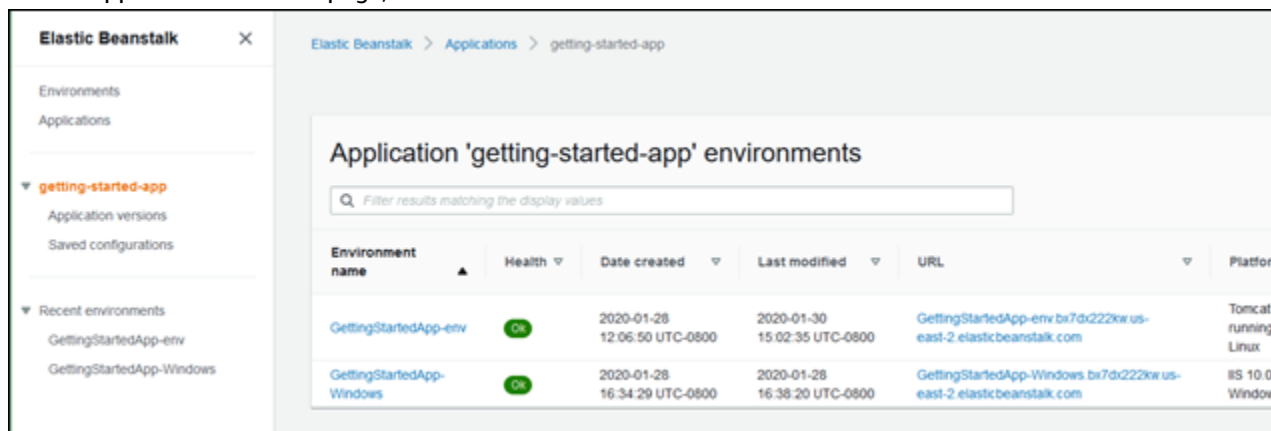
Name	Supported versions	Environment type	Source	Description
Scorekeep	Java 8	Web Service	Clone the repo at GitHub.com	<p><i>Scorekeep</i> is a RESTful web API that uses the Spring framework to provide an interface for creating and managing users, sessions, and games. The API is bundles with an Angular 1.5 web app that consumes the API over HTTP.</p> <p>The application uses features of the Java SE platform to download dependencies and build on-instance, minimizing the size of the source bundle. The application also includes nginx configuration files that override the default configuration to serve the frontend web app statically on port 80 through the proxy, and route requests to paths under <code>/api</code> to the API running on <code>localhost:5000</code>.</p> <p>Scorekeep also includes an <code>xray</code> branch that shows how to instrument a Java application for use with AWS X-Ray. It shows instrumentation of incoming HTTP requests with a servlet filter, automatic and manual AWS SDK client instrumentation, recorder configuration, and instrumentation of outgoing HTTP requests and SQL clients.</p> <p>See the readme for instructions or use the AWS X-Ray getting started tutorial to try the application with X-Ray.</p>

Name	Supported versions	Environment type	Source	Description
Does it Have Snakes?	Tomcat 8 with Java 8	Web Server	Clone the repo at GitHub.com	<p><i>Does it Have Snakes?</i> is a Tomcat web application that shows the use of Elastic Beanstalk configuration files, Amazon RDS, JDBC, PostgreSQL, Servlets, JSPs, Simple Tag Support, Tag Files, Log4J, Bootstrap, and Jackson.</p> <p>The source code for this project includes a minimal build script that compiles the servlets and models into class files and packages the required files into a Web Archive that you can deploy to an Elastic Beanstalk environment. See the readme file in the project repository for full instructions.</p>
Locust Load Generator	Java 8	Web Server	Clone the repo at GitHub.com	<p>Web application that you can use to load test another web application running in a different Elastic Beanstalk environment. Shows the use of <code>Buildfile</code> and <code>Procfile</code> files, DynamoDB, and <code>Locust</code>, an open source load testing tool.</p>

Download any of the sample applications and deploy it to Elastic Beanstalk by following these steps:

To launch an environment with a sample application (console)

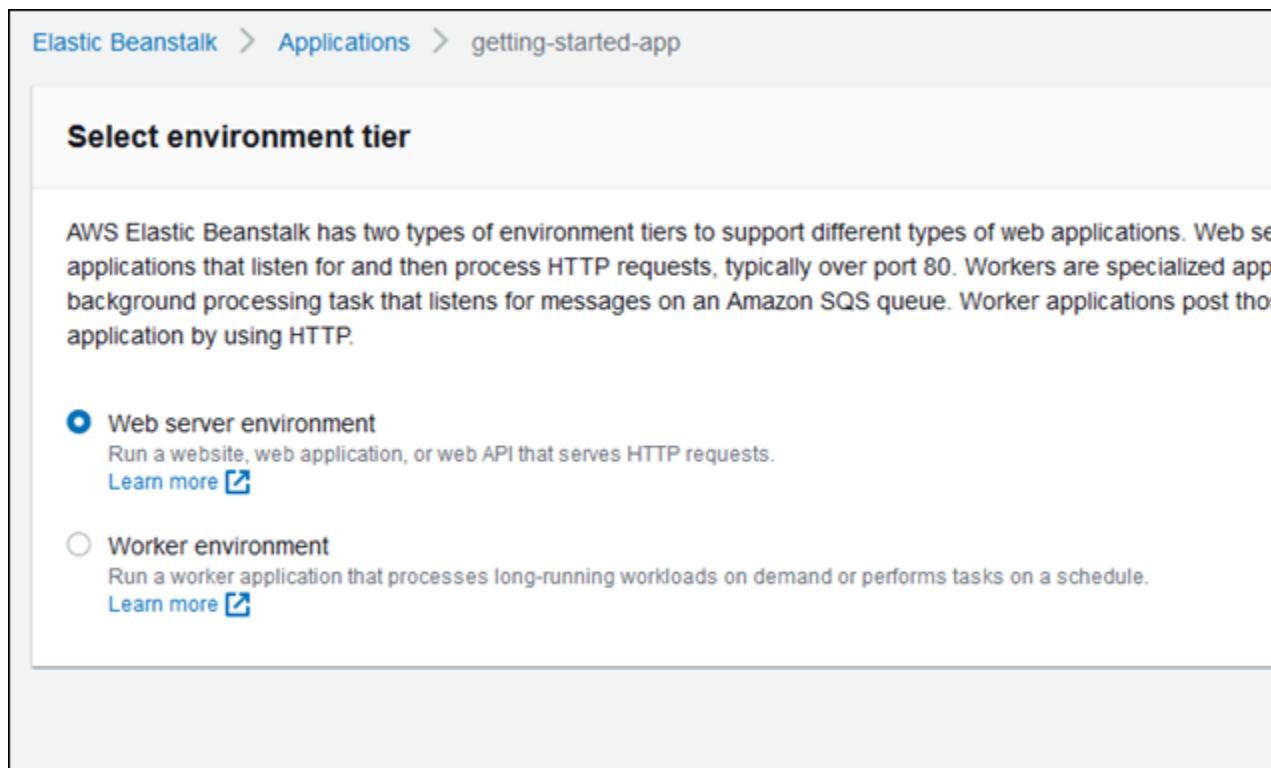
1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose an existing application's name on the list or [create one](#) (p. 334).
3. On the application overview page, choose **Create a new environment**.



4. Choose the **Web server environment** or **Worker environment** [environment tier](#) (p. 13). You can't change an environment's tier after creation.

Note

The [.NET on Windows Server platform \(p. 137\)](#) doesn't support the worker environment tier.



5. For **Platform**, select the platform and platform branch that match the language used by your application.

Note

Elastic Beanstalk supports multiple [versions \(p. 29\)](#) for most of the platforms that are listed. By default, the console selects the recommended version for the platform and platform branch you choose. If your application requires different version, you can select it here, or by choosing **Configure more options**, as described in step 7. For information about supported platform versions, see [the section called "Supported platforms" \(p. 29\)](#).

6. For **Application code**, choose **Sample application**.
7. To further customize your environment, choose **Configure more options**. You can set the following options only during environment creation:
 - Environment name
 - Domain name
 - Platform version
 - VPC
 - Tier

You can change the following settings after environment creation, but they require new instances or other resources to be provisioned and can take a long time to apply:

- Instance type, root volume, key pair, and AWS Identity and Access Management (IAM) role
- Internal Amazon RDS database
- Load balancer

For details on all available settings, see [The create new environment wizard \(p. 365\)](#).

8. Choose **Create environment**.

Next steps

After you have an environment running an application, you can [deploy a new version \(p. 397\)](#) of the application or a completely different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances.

After you've deployed a sample application or two and are ready to start developing and running Java applications locally, see [the next section \(p. 100\)](#) to set up a Java development environment with all of the tools and libraries that you will need.

Setting up your Java development environment

Set up a Java development environment to test your application locally prior to deploying it to AWS Elastic Beanstalk. This topic outlines development environment setup steps and links to installation pages for useful tools.

For common setup steps and tools that apply to all languages, see [Configuring your development machine \(p. 849\)](#).

Sections

- [Installing the Java development kit \(p. 100\)](#)
- [Installing a web container \(p. 100\)](#)
- [Downloading libraries \(p. 100\)](#)
- [Installing the AWS SDK for Java \(p. 101\)](#)
- [Installing an IDE or text editor \(p. 101\)](#)
- [Installing the AWS toolkit for Eclipse \(p. 101\)](#)

Installing the Java development kit

Install the Java Development Kit (JDK). If you don't have a preference, get the latest version. Download the JDK at oracle.com

The JDK includes the Java compiler, which you can use to build your source files into class files that can be executed on an Elastic Beanstalk web server.

Installing a web container

If you don't already have another web container or framework, install the appropriate version of Tomcat:

- [Download Tomcat 8 \(requires Java 7 or later\)](#)
- [Download Tomcat 7 \(requires Java 6 or later\)](#)

Downloading libraries

Elastic Beanstalk platforms include few libraries by default. Download libraries that your application will use and save them in your project folder to deploy in your application source bundle.

If you've installed Tomcat locally, you can copy the servlet API and JavaServer Pages (JSP) API libraries from the installation folder. If you deploy to a Tomcat platform version, you don't need to include these files in your source bundle, but you do need to have them in your `classpath` to compile any classes that use them.

JUnit, Google Guava, and Apache Commons provide several useful libraries. Visit their home pages to learn more:

- [Download JUnit](#)
- [Download Google Guava](#)
- [Download Apache Commons](#)

Installing the AWS SDK for Java

If you need to manage AWS resources from within your application, install the AWS SDK for Java. For example, with the AWS SDK for Java, you can use Amazon DynamoDB (DynamoDB) to share session states of Apache Tomcat applications across multiple web servers. For more information, see [Manage Tomcat Session State with Amazon DynamoDB](#) in the AWS SDK for Java documentation.

Visit the [AWS SDK for Java home page](#) for more information and installation instructions.

Installing an IDE or text editor

Integrated development environments (IDEs) provide a wide range of features that facilitate application development. If you haven't used an IDE for Java development, try Eclipse and IntelliJ and see which works best for you.

- [Install Eclipse IDE for Java EE Developers](#)
- [Install IntelliJ](#)

Note

An IDE might add files to your project folder that you might not want to commit to source control. To prevent committing these files to source control, use `.gitignore` or your source control tool's equivalent.

If you just want to begin coding and don't need all of the features of an IDE, consider [installing Sublime Text](#).

Installing the AWS toolkit for Eclipse

The [AWS Toolkit for Eclipse \(p. 123\)](#) is an open source plug-in for the Eclipse Java IDE that makes it easier for developers to develop, debug, and deploy Java applications using AWS. Visit the [AWS Toolkit for Eclipse home page](#) for installation instructions.

Using the Elastic Beanstalk Tomcat platform

Important

Amazon Linux 2 platform versions are fundamentally different than Amazon Linux AMI platform versions (preceding Amazon Linux 2). These different platform generations are incompatible in several ways. If you are migrating to an Amazon Linux 2 platform version, be sure to read the information in [the section called "Upgrade to Amazon Linux 2" \(p. 426\)](#).

The AWS Elastic Beanstalk Tomcat platform is a set of [platform versions](#) for Java web applications that can run in a Tomcat web container. Tomcat runs behind an nginx proxy server. Each platform branch corresponds to a major version of Tomcat, like *Java 8 with Tomcat 8*.

Configuration options are available in the Elastic Beanstalk console for [modifying the configuration of a running environment](#) (p. 547). To avoid losing your environment's configuration when you terminate it, you can use [saved configurations](#) (p. 640) to save your settings and later apply them to another environment.

To save settings in your source code, you can include [configuration files](#) (p. 600). Settings in configuration files are applied every time you create an environment or deploy your application. You can also use configuration files to install packages, run scripts, and perform other instance customization operations during deployments.

The Elastic Beanstalk Tomcat platform includes a reverse proxy that forwards requests to your application. You can use [configuration options](#) (p. 104) to configure the proxy server to serve static assets from a folder in your source code to reduce the load on your application. For advanced scenarios, you can [include your own .conf files](#) (p. 108) in your source bundle to extend the Elastic Beanstalk proxy configuration or overwrite it completely.

Note

Elastic Beanstalk uses [nginx](#) as the proxy server on the Tomcat platform. If your Elastic Beanstalk Tomcat environment uses an Amazon Linux AMI platform branch (preceding Amazon Linux 2), you also have the option of using [Apache 2.4](#) or [Apache HTTP Server Version 2.2](#). Apache 2.4 is the default on these older platform branches.

You must package Java applications in a web application archive (WAR) file with a specific structure. For information on the required structure and how it relates to the structure of your project directory, see [Structuring your project folder](#) (p. 106).

To run multiple applications on the same web server, you can [bundle multiple WAR files](#) (p. 105) into a single source bundle. Each application in a multiple WAR source bundle runs at the root path (ROOT.war runs at `myapp.elasticbeanstalk.com/`) or at a path directly beneath it (app2.war runs at `myapp.elasticbeanstalk.com/app2/`), as determined by the name of the WAR. In a single WAR source bundle, the application always runs at the root path.

Settings applied in the Elastic Beanstalk console override the same settings in configuration files, if they exist. This lets you have default settings in configuration files, and override them with environment-specific settings in the console. For more information about precedence, and other methods of changing settings, see [Configuration options](#) (p. 536).

For details about the various ways you can extend an Elastic Beanstalk Linux-based platform, see [the section called "Extending Linux platforms"](#) (p. 31).

Topics

- [Configuring your Tomcat environment](#) (p. 102)
- [Tomcat configuration namespaces](#) (p. 104)
- [The Amazon Linux AMI \(preceding Amazon Linux 2\) Tomcat platform](#) (p. 105)
- [Bundling multiple WAR files for Tomcat environments](#) (p. 105)
- [Structuring your project folder](#) (p. 106)
- [Configuring your Tomcat environment's proxy server](#) (p. 108)

Configuring your Tomcat environment

The Elastic Beanstalk Tomcat platform provides a few platform-specific options in addition to the standard options that all platforms have. These options enable you to configure the Java virtual machine (JVM) that runs on your environment's web servers, and define system properties that provide information configuration strings to your application.

You can use the Elastic Beanstalk console to enable log rotation to Amazon S3 and configure variables that your application can read from the environment.

To configure your Tomcat environment in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.

JVM container options

The heap size in the Java virtual machine (JVM) determines how many objects your application can create in memory before *garbage collection* occurs. You can modify the **Initial JVM Heap Size (-Xms argument)** and a **Maximum JVM Heap Size (-Xmx argument)**. A larger initial heap size allows more objects to be created before garbage collection occurs, but it also means that the garbage collector will take longer to compact the heap. The maximum heap size specifies the maximum amount of memory the JVM can allocate when expanding the heap during heavy activity.

Note

The available memory depends on the Amazon EC2 instance type. For more information about the EC2 instance types available for your Elastic Beanstalk environment, see [Instance Types](#) in the *Amazon Elastic Compute Cloud User Guide for Linux Instances*.

The *permanent generation* is a section of the JVM heap that stores class definitions and associated metadata. To modify the size of the permanent generation, type the new size in the **Maximum JVM PermGen Size (-XX:MaxPermSize argument)** field. This setting applies only to Java 7 and earlier.

Log options

The **Log Options** section has two settings:

- **Instance profile** – Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3** – Specifies whether log files for your application's Amazon EC2 instances should be copied to the Amazon S3 bucket associated with your application.

Static files

To improve performance, the **Static files** section lets you configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. For each directory, you set the virtual path to directory mapping. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

For details about configuring static files using the Elastic Beanstalk console, see [the section called “Static files”](#) (p. 650).

Environment properties

In the **Environment Properties** section, you can specify environment configuration settings on the Amazon EC2 instances that are running your application. Environment properties are passed in as key-value pairs to the application.

The Tomcat platform defines a placeholder property named `JDBC_CONNECTION_STRING` for Tomcat environments for passing a connection string to an external database.

Note

If you attach an RDS DB instance to your environment, construct the JDBC connection string dynamically from the Amazon Relational Database Service (Amazon RDS) environment properties provided by Elastic Beanstalk. Use `JDBC_CONNECTION_STRING` only for database instances that are not provisioned using Elastic Beanstalk.

For more information about using Amazon RDS with your Java application, see [Adding an Amazon RDS DB instance to your Java application environment \(p. 118\)](#).

Inside the Tomcat environment running in Elastic Beanstalk, environment variables are accessible using the `System.getProperty()`. For example, you could read a property named `API_ENDPOINT` to a variable with the following code.

```
String endpoint = System.getProperty("API_ENDPOINT");
```

See [Environment properties and other software settings \(p. 516\)](#) for more information.

Tomcat configuration namespaces

You can use a [configuration file \(p. 600\)](#) to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be defined by the Elastic Beanstalk service or the platform that you use and are organized into *namespaces*.

The Tomcat platform supports options in the following namespaces, in addition to the [options supported for all Elastic Beanstalk environments \(p. 555\)](#):

- `aws:elasticbeanstalk:container:tomcat:jvmoptions` – Modify JVM settings. Options in this namespace correspond to options in the management console, as follows:
 - `Xms` – **JVM command line options**
 - `Xmx` – **JVM command line options**
 - `JVM Options` – **JVM command line options**
- `aws:elasticbeanstalk:environment:proxy:staticfiles` – Configure the proxy to serve static assets from a path in your source bundle.

The following example configuration file shows the use of the Tomcat-specific configuration options.

Example `.ebextensions/tomcat-settings.config`

```
option_settings:  
  aws:elasticbeanstalk:container:tomcat:jvmoptions:  
    Xms: 512m  
    Xmx: 512m  
    JVM Options: '-Xmn128m'  
  aws:elasticbeanstalk:application:environment:  
    API_ENDPOINT: mywebapi.zkpexsjtmd.us-west-2.elasticbeanstalk.com  
  aws:elasticbeanstalk:environment:proxy:staticfiles:  
    /html: statichtml  
    /images: staticimages
```

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See [Configuration options \(p. 536\)](#) for more information.

The Amazon Linux AMI (preceding Amazon Linux 2) Tomcat platform

If your Elastic Beanstalk Tomcat environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Tomcat configuration namespaces

The Tomcat Amazon Linux AMI platform supports additional options in the following namespaces:

- `aws:elasticbeanstalk:container:tomcat:jvmoptions` – In addition to the options mentioned earlier on this page for this namespace, older Amazon Linux AMI platform versions also support:
 - `XX:MaxPermSize` – **Maximum JVM permanent generation size**
- `aws:elasticbeanstalk:environment:proxy` – Choose the proxy server and configure response compression.

The following example configuration file shows the use of the proxy namespace configuration options.

Example `.ebextensions/tomcat-settings.config`

```
option_settings:  
  aws:elasticbeanstalk:environment:proxy:  
    GzipCompression: 'true'  
    ProxyServer: nginx
```

Include Elastic Beanstalk configurations files

To deploy `.ebextensions` configuration files, include them in your application source. For a single application, add your `.ebextensions` to a compressed WAR file by running the following command:

Example

```
zip -ur your_application.war .ebextensions
```

For an application requiring multiple WAR files, see [Bundling multiple WAR files for Tomcat environments](#) (p. 105) for further instructions.

Bundling multiple WAR files for Tomcat environments

If your web app comprises multiple web application components, you can simplify deployments and reduce operating costs by running components in a single environment, instead of running a separate environment for each component. This strategy is effective for lightweight applications that don't require a lot of resources, and for development and test environments.

To deploy multiple web applications to your environment, combine each component's web application archive (WAR) files into a single [source bundle](#) (p. 342).

To create an application source bundle that contains multiple WAR files, organize the WAR files using the following structure.

```
MyApplication.zip  
### .ebextensions  
### .platform  
### foo.war
```



```
### bar.war  
### ROOT.war
```

When you deploy a source bundle containing multiple WAR files to an AWS Elastic Beanstalk environment, each application is accessible from a different path off of the root domain name. The preceding example includes three applications: `foo`, `bar`, and `ROOT`. `ROOT.war` is a special file name that tells Elastic Beanstalk to run that application at the root domain, so that the three applications are available at `http://MyApplication.elasticbeanstalk.com/foo`, `http://MyApplication.elasticbeanstalk.com/bar`, and `http://MyApplication.elasticbeanstalk.com`.

The source bundle can include WAR files, an optional `.ebextensions` folder, and an optional `.platform` folder. For details about these optional configuration folders, see [the section called “Extending Linux platforms” \(p. 31\)](#).

To launch an environment (console)

1. Open the Elastic Beanstalk console with this preconfigured link:
console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Upload your code**.
4. Choose **Local file**, choose **Choose file**, and then open the source bundle.
5. Choose **Review and launch**.
6. Review the available settings, and then choose **Create app**.

For information about creating source bundles, see [Create an application source bundle \(p. 342\)](#).

Structuring your project folder

To work when deployed to a Tomcat server, compiled Java Platform Enterprise Edition (*Java EE*) web application archives (WAR files) must be structured according to certain [guidelines](#). Your project directory doesn't have to meet the same standards, but it's a good idea to structure it in the same way to simplify compiling and packaging. Structuring your project folder like the WAR file contents also helps you understand how files are related and how they behave on a web server.

In the following recommended hierarchy, the source code for the web application is placed in a `src` directory, to isolate it from the build script and the WAR file it generates.

```
~/workspace/my-app/  
|-- build.sh           - Build script that compiles classes and creates a WAR  
|-- README.MD         - Readme file with information about your project, notes  
|-- ROOT.war          - Source bundle artifact created by build.sh  
`-- src                - Source code folder  
    |-- WEB-INF        - Folder for private supporting files  
    |   |-- classes    - Compiled classes  
    |   |-- lib         - JAR libraries  
    |   |-- tags       - Tag files  
    |   |-- tlds       - Tag Library Descriptor files  
    |   |-- web.xml    - Deployment Descriptor  
    |-- com            - Uncompiled classes  
    |-- css            - Style sheets  
    |-- images         - Image files  
    |-- js             - JavaScript files  
    |-- default.jsp    - JSP (JavaServer Pages) webpage
```

The `src` folder contents match what you will package and deploy to the server, with the exception of the `com` folder. The `com` folder contains your uncompiled classes (`.java` files). These need to be compiled and placed in the `WEB-INF/classes` directory to be accessible from your application code.

The `WEB-INF` directory contains code and configurations that are not served publicly on the web server. The other folders at the root of the source directory (`css`, `images`, and `js`) are publicly available at the corresponding path on the web server.

The following example is identical to the preceding project directory, except that it contains more files and subdirectories. This example project includes simple tags, model and support classes, and a Java Server Pages (JSP) file for a `record` resource. It also includes a style sheet and JavaScript for [Bootstrap](#), a default JSP file, and an error page for 404 errors.

`WEB-INF/lib` includes a Java Archive (JAR) file containing the Java Database Connectivity (JDBC) driver for PostgreSQL. `WEB-INF/classes` is empty because class files have not been compiled yet.

```
~/workspace/my-app/  
|-- build.sh  
|-- README.MD  
|-- ROOT.war  
`-- src  
    |-- WEB-INF  
    |   |-- classes  
    |   |-- lib  
    |   |   |-- postgresql-9.4-1201.jdbc4.jar  
    |   |-- tags  
    |   |   |-- header.tag  
    |   |-- tlds  
    |   |   |-- records.tld  
    |   |-- web.xml  
    |-- com  
    |   |-- myapp  
    |   |   |-- model  
    |   |   |   |-- Record.java  
    |   |   |-- web  
    |   |   |   |-- ListRecords.java  
    |-- css  
    |   |-- bootstrap.min.css  
    |   |-- myapp.css  
    |-- images  
    |   |-- myapp.png  
    |-- js  
    |   |-- bootstrap.min.js  
    |-- 404.jsp  
    |-- default.jsp  
    |-- records.jsp
```

Building a WAR file with a shell script

`build.sh` is a very simple shell script that compiles Java classes, constructs a WAR file, and copies it to the Tomcat `webapps` directory for local testing.

```
cd src  
javac -d WEB-INF/classes com/myapp/model/Record.java  
javac -classpath WEB-INF/lib/*:WEB-INF/classes -d WEB-INF/classes com/myapp/model/  
Record.java  
javac -classpath WEB-INF/lib/*:WEB-INF/classes -d WEB-INF/classes com/myapp/web/  
ListRecords.java  
  
jar -cvf ROOT.war *.jsp images css js WEB-INF  
cp ROOT.war /Library/Tomcat/webapps  
mv ROOT.war ../
```

Inside the WAR file, you find the same structure that exists in the `src` directory in the preceding example, excluding the `src/com` folder. The `jar` command automatically creates the `META-INF/MANIFEST.MF` file.

```
~/workspace/my-app/ROOT.war
|-- META-INF
|   |-- MANIFEST.MF
|-- WEB-INF
|   |-- classes
|   |   |-- com
|   |   |   |-- myapp
|   |   |   |   |-- model
|   |   |   |   |   |-- Records.class
|   |   |   |   |-- web
|   |   |   |   |   |-- ListRecords.class
|   |-- lib
|   |   |-- postgresql-9.4-1201.jdbc4.jar
|   |-- tags
|   |   |-- header.tag
|   |-- tlds
|   |   |-- records.tld
|   |-- web.xml
|-- css
|   |-- bootstrap.min.css
|   |-- myapp.css
|-- images
|   |-- myapp.png
|-- js
|   |-- bootstrap.min.js
|-- 404.jsp
|-- default.jsp
|-- records.jsp
```

Using .gitignore

To avoid committing compiled class files and WAR files to your Git repository, or seeing messages about them appear when you run Git commands, add the relevant file types to a file named `.gitignore` in your project folder.

`~/workspace/myapp/.gitignore`

```
*.zip
*.class
```

Configuring your Tomcat environment's proxy server

The Tomcat platform uses [nginx](#) as the reverse proxy to relay requests from port 80 on the instance to your Tomcat web container listening on port 8080. Elastic Beanstalk provides a default proxy configuration that you can extend or override completely with your own configuration.

All Amazon Linux 2 platforms support a uniform proxy configuration feature. For details about configuring the proxy server on Tomcat platform versions running Amazon Linux 2, expand the *Reverse Proxy Configuration* section in [the section called "Extending Linux platforms"](#) (p. 31).

Configuring the proxy on the Amazon Linux AMI (preceding Amazon Linux 2) Tomcat platform

If your Elastic Beanstalk Tomcat environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Choosing a proxy server for your Tomcat environment

Tomcat platform versions based on Amazon Linux AMI (preceding Amazon Linux 2) use [Apache 2.4](#) for the proxy by default. You can choose to use [Apache 2.2](#) or [nginx](#) by including a [configuration file \(p. 600\)](#) in your source code. The following example configures Elastic Beanstalk to use nginx.

Example `.ebextensions/nginx-proxy.config`

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: nginx
```

Migrating from Apache 2.2 to Apache 2.4

If your application was developed for [Apache 2.2](#), read this section to learn about migrating to [Apache 2.4](#).

Starting with Tomcat platform version 3.0.0 configurations, which were released with the [Java with Tomcat platform update on May 24, 2018](#), Apache 2.4 is the default proxy of the Tomcat platform. The Apache 2.4 `.conf` files are mostly, but not entirely, backward compatible with those of Apache 2.2. Elastic Beanstalk includes default `.conf` files that work correctly with each Apache version. If your application doesn't customize Apache's configuration, as explained in [Extending and overriding the default Apache configuration \(p. 110\)](#), it should migrate to Apache 2.4 without any issues.

If your application extends or overrides Apache's configuration, you might have to make some changes to migrate to Apache 2.4. For more information, see [Upgrading to 2.4 from 2.2 on The Apache Software Foundation's site](#). As a temporary measure, until you successfully migrate to Apache 2.4, you can choose to use Apache 2.2 with your application by including the following [configuration file \(p. 600\)](#) in your source code.

Example `.ebextensions/apache-legacy-proxy.config`

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: apache/2.2
```

For a quick fix, you can also select the proxy server in the Elastic Beanstalk console.

To select the proxy in your Tomcat environment in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

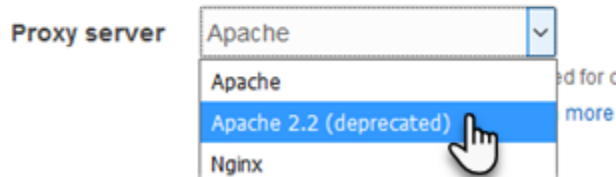
If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. For **Proxy server**, choose `Apache 2.2 (deprecated)`.
6. Choose **Apply**.

Modify software

Container Options

The following settings control container behavior and let you pass key-value pairs in as OS environment variables. [Learn](#)



Extending and overriding the default Apache configuration

You can extend the Elastic Beanstalk default Apache configuration with your additional configuration files. Alternatively, you can override the Elastic Beanstalk default Apache configuration completely.

To extend the Elastic Beanstalk default Apache configuration, add `.conf` configuration files to a folder named `.ebextensions/httpd/conf.d` in your application source bundle. The Elastic Beanstalk Apache configuration includes `.conf` files in this folder automatically.

```
~/workspace/my-app/  
|-- .ebextensions  
|   -- httpd  
|       -- conf.d  
|           -- myconf.conf  
|           -- ssl.conf  
|-- index.jsp
```

For example, the following Apache 2.4 configuration adds a listener on port 5000.

Example `.ebextensions/httpd/conf.d/port5000.conf`

```
listen 5000  
<VirtualHost *:5000>  
  <Proxy *>  
    Require all granted  
  </Proxy>  
  ProxyPass / http://localhost:8080/ retry=0  
  ProxyPassReverse / http://localhost:8080/  
  ProxyPreserveHost on  
  
  ErrorLog /var/log/httpd/elasticbeanstalk-error_log  
</VirtualHost>
```

To override the Elastic Beanstalk default Apache configuration completely, include a configuration in your source bundle at `.ebextensions/httpd/conf/httpd.conf`.

```
~/workspace/my-app/  
|-- .ebextensions  
|   `-- httpd  
|       `-- conf  
|           `-- httpd.conf  
|-- index.jsp
```

If you override the Elastic Beanstalk Apache configuration, add the following lines to your `httpd.conf` to pull in the Elastic Beanstalk configurations for [Enhanced health reporting and monitoring \(p. 691\)](#), response compression, and static files.

```
IncludeOptional conf.d/*.conf
IncludeOptional conf.d/elasticbeanstalk/*.conf
```

If your environment uses Apache 2.2 as its proxy, replace the `IncludeOptional` directives with `Include`. For details about the behavior of these two directives in the two Apache versions, see [Include in Apache 2.4](#), [IncludeOptional in Apache 2.4](#), and [Include in Apache 2.2](#).

Note

To override the default listener on port 80, include a file named `00_application.conf` at `.ebextensions/httpd/conf.d/elasticbeanstalk/` to overwrite the Elastic Beanstalk configuration.

For a working example, take a look at the Elastic Beanstalk default configuration file at `/etc/httpd/conf/httpd.conf` on an instance in your environment. All files in the `.ebextensions/httpd` folder in your source bundle are copied to `/etc/httpd` during deployments.

Extending the default nginx configuration

To extend Elastic Beanstalk's default nginx configuration, add `.conf` configuration files to a folder named `.ebextensions/nginx/conf.d/` in your application source bundle. The Elastic Beanstalk nginx configuration includes `.conf` files in this folder automatically.

```
~/workspace/my-app/
|-- .ebextensions
|   |-- nginx
|       |-- conf.d
|           |-- elasticbeanstalk
|               |-- my-server-conf.conf
|               |-- my-http-conf.conf
|-- index.jsp
```

Files with the `.conf` extension in the `conf.d` folder are included in the `http` block of the default configuration. Files in the `conf.d/elasticbeanstalk` folder are included in the `server` block within the `http` block.

To override the Elastic Beanstalk default nginx configuration completely, include a configuration in your source bundle at `.ebextensions/nginx/nginx.conf`.

```
~/workspace/my-app/
|-- .ebextensions
|   |-- nginx
|       |-- nginx.conf
|-- index.jsp
```

If you override the Elastic Beanstalk nginx configuration, add the following line to your configuration's `server` block to pull in the Elastic Beanstalk configurations for the port 80 listener, response compression, and static files.

```
include conf.d/elasticbeanstalk/*.conf;
```

Note

To override the default listener on port 80, include a file named `00_application.conf` at `.ebextensions/nginx/conf.d/elasticbeanstalk/` to overwrite the Elastic Beanstalk configuration.

Also include the following line in your configuration's `http` block to pull in the Elastic Beanstalk configurations for [Enhanced health reporting and monitoring](#) (p. 691) and logging.

```
include    conf.d/*.conf;
```

For a working example, take a look at the Elastic Beanstalk default configuration file at `/etc/nginx/nginx.conf` on an instance in your environment. All files in the `.ebextensions/nginx` folder in your source bundle are copied to `/etc/nginx` during deployments.

Using the Elastic Beanstalk Java SE platform

Important

Amazon Linux 2 platform versions are fundamentally different than Amazon Linux AMI platform versions (preceding Amazon Linux 2). These different platform generations are incompatible in several ways. If you are migrating to an Amazon Linux 2 platform version, be sure to read the information in [the section called "Upgrade to Amazon Linux 2"](#) (p. 426).

The AWS Elastic Beanstalk Java SE platform is a set of [platform versions](#) for Java web applications that can run on their own from a compiled JAR file. You can compile your application locally or upload the source code with a build script to compile it on-instance. Java SE platform versions are grouped into platform branches, each of which corresponds to a major version of Java, for example *Java 8* and *Java 7*.

Note

Elastic Beanstalk doesn't parse your application's JAR file. Keep files that Elastic Beanstalk needs outside of the JAR file. For example, include the `cron.yaml` file of a [worker environment](#) (p. 437) at the root of your application's source bundle, next to the JAR file.

Configuration options are available in the Elastic Beanstalk console for [modifying the configuration of a running environment](#) (p. 547). To avoid losing your environment's configuration when you terminate it, you can use [saved configurations](#) (p. 640) to save your settings and later apply them to another environment.

To save settings in your source code, you can include [configuration files](#) (p. 600). Settings in configuration files are applied every time you create an environment or deploy your application. You can also use configuration files to install packages, run scripts, and perform other instance customization operations during deployments.

The Elastic Beanstalk Java SE platform includes an [nginx](#) server that acts as a reverse proxy, serving cached static content and passing requests to your application. The platform provides configuration options to configure the proxy server to serve static assets from a folder in your source code to reduce the load on your application. For advanced scenarios, you can [include your own .conf files](#) (p. 116) in your source bundle to extend Elastic Beanstalk's proxy configuration or overwrite it completely.

If you only provide a single JAR file for your application source (on its own, not within a source bundle), Elastic Beanstalk renames your JAR file to `application.jar`, and then runs it using `java -jar application.jar`. To configure the processes that run on the server instances in your environment, include an optional [Procfile](#) (p. 115) in your source bundle. A `Procfile` is required if you have more than one JAR in your source bundle root, or if you want to customize the `java` command to set JVM options.

We recommend that you always provide a `Procfile` in the source bundle alongside your application. This way you precisely control which processes Elastic Beanstalk runs for your application and which arguments these processes receive.

To compile Java classes and run other build commands on the EC2 instances in your environment at deploy time, include a [Buildfile](#) (p. 114) in your application source bundle. A `Buildfile` lets you deploy your source code as-is and build on the server instead of compiling JARs locally. The Java SE platform includes common build tools to let you build on-server.

For details about the various ways you can extend an Elastic Beanstalk Linux-based platform, see [the section called “Extending Linux platforms” \(p. 31\)](#).

Configuring your Java SE environment

The Java SE platform settings let you fine-tune the behavior of your Amazon EC2 instances. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration using the Elastic Beanstalk console.

Use the Elastic Beanstalk console to enable log rotation to Amazon S3 and configure variables that your application can read from the environment.

To configure your Java SE environment in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.

Log options

The Log Options section has two settings:

- **Instance profile** – Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3** – Specifies whether log files for your application's Amazon EC2 instances should be copied to the Amazon S3 bucket associated with your application.

Static files

To improve performance, the **Static files** section lets you configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. For each directory, you set the virtual path to directory mapping. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

For details about configuring static files using the Elastic Beanstalk console, see [the section called “Static files” \(p. 650\)](#).

Environment properties

The **Environment Properties** section lets you specify environment configuration settings on the Amazon EC2 instances that are running your application. Environment properties are passed in as key-value pairs to the application.

Inside the Java SE environment running in Elastic Beanstalk, environment variables are accessible using the `System.getenv()`. For example, you could read a property named `API_ENDPOINT` to a variable with the following code:

```
String endpoint = System.getenv("API_ENDPOINT");
```

See [Environment properties and other software settings \(p. 516\)](#) for more information.

Java SE configuration namespace

You can use a [configuration file \(p. 600\)](#) to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be defined by the Elastic Beanstalk service or the platform that you use and are organized into *namespaces*.

The Java SE platform doesn't define any platform-specific namespaces. You can configure the proxy to serve static files by using the `aws:elasticbeanstalk:environment:proxy:staticfiles` namespace. For details and an example, see [the section called "Static files" \(p. 650\)](#).

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See [Configuration options \(p. 536\)](#) for more information.

The Amazon Linux AMI (preceding Amazon Linux 2) Java SE platform

If your Elastic Beanstalk Java SE environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Java SE configuration namespaces

You can use a [configuration file \(p. 600\)](#) to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be defined by the Elastic Beanstalk service or the platform that you use and are organized into *namespaces*.

The Java SE platform supports one platform-specific configuration namespace in addition to the [namespaces supported by all platforms \(p. 555\)](#). The `aws:elasticbeanstalk:container:java:staticfiles` namespace lets you define options that map paths on your web application to folders in your application source bundle that contain static content.

For example, this [option_settings \(p. 601\)](#) snippet defines two options in the static files namespace. The first maps the path `/public` to a folder named `public`, and the second maps the path `/images` to a folder named `img`:

```
option_settings:
  aws:elasticbeanstalk:container:java:staticfiles:
    /html: statichtml
    /images: staticimages
```

The folders that you map using this namespace must be actual folders in the root of your source bundle. You cannot map a path to a folder in a JAR file.

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See [Configuration options \(p. 536\)](#) for more information.

Building JARs on-server with a Buildfile

You can build your application's class files and JAR(s) on the EC2 instances in your environment by invoking a build command from a `Buildfile` file in your source bundle.

Commands in a `Buildfile` are only run once and must terminate upon completion, whereas commands in a [Procfile \(p. 115\)](#) are expected to run for the life of the application and will be restarted if they terminate. To run the JARs in your application, use a `Procfile`.

For details about the placement and syntax of a `Buildfile`, expand the *Buildfile and Procfile* section in [the section called “Extending Linux platforms” \(p. 31\)](#).

The following `Buildfile` example runs Apache Maven to build a web application from source code. For a sample application that uses this feature, see [Java web application samples \(p. 95\)](#).

Example Buildfile

```
build: mvn assembly:assembly -DdescriptorId=jar-with-dependencies
```

The Java SE platform includes the following build tools, which you can invoke from your build script:

- `javac` – Java compiler
- `ant` – Apache Ant
- `mvn` – Apache Maven
- `gradle` – Gradle

Configuring the application process with a Procfile

If you have more than one JAR file in the root of your application source bundle, you must include a `Procfile` file that tells Elastic Beanstalk which JAR(s) to run. You can also include a `Procfile` file for a single JAR application to configure the Java virtual machine (JVM) that runs your application.

We recommend that you always provide a `Procfile` in the source bundle alongside your application. This way you precisely control which processes Elastic Beanstalk runs for your application and which arguments these processes receive.

For details about writing and using a `Procfile`, expand the *Buildfile and Procfile* section in [the section called “Extending Linux platforms” \(p. 31\)](#).

Example Procfile

```
web: java -jar server.jar -Xms256m
cache: java -jar mycache.jar
web_foo: java -jar other.jar
```

The command that runs the main JAR in your application must be called `web`, and it must be the first command listed in your `Procfile`. The nginx server forwards all HTTP requests that it receives from your environment's load balancer to this application.

Elastic Beanstalk assumes that all entries in the `Procfile` should run at all times and automatically restarts any application defined in the `Procfile` that terminates. To run commands that will terminate and should not be restarted, use a [Buildfile \(p. 114\)](#).

Using a Procfile on Amazon Linux AMI (preceding Amazon Linux 2)

If your Elastic Beanstalk Java SE environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Port passing

By default, Elastic Beanstalk configures the nginx proxy to forward requests to your application on port 5000. You can override the default port by setting the `PORT` [environment property \(p. 113\)](#) to the port on which your main application listens.

If you use a `Procfile` to run multiple applications, Elastic Beanstalk on Amazon Linux AMI platform versions expects each additional application to listen on a port 100 higher than the previous one.

Elastic Beanstalk sets the `PORT` variable accessible from within each application to the port that it expects the application to run on. You can access this variable within your application code by calling `System.getenv("PORT")`.

In the preceding `Procfile` example, the `web` application listens on port 5000, `cache` listens on port 5100, and `web_foo` listens on port 5200. `web` configures its listening port by reading the `PORT` variable, and adds 100 to that number to determine which port `cache` is listening on so that it can send it requests.

Configuring the reverse proxy

Elastic Beanstalk uses [nginx](#) as the reverse proxy to map your application to your Elastic Load Balancing load balancer on port 80. Elastic Beanstalk provides a default nginx configuration that you can either extend or override completely with your own configuration.

By default, Elastic Beanstalk configures the nginx proxy to forward requests to your application on port 5000. You can override the default port by setting the `PORT` [environment property \(p. 113\)](#) to the port on which your main application listens.

Note

The port that your application listens on doesn't affect the port that the nginx server listens to receive requests from the load balancer.

All Amazon Linux 2 platforms support a uniform proxy configuration feature. For details about configuring the proxy server on the new Amazon Corretto platform versions running Amazon Linux 2, expand the *Reverse Proxy Configuration* section in [the section called "Extending Linux platforms" \(p. 31\)](#).

Configuring the proxy on Amazon Linux AMI (preceding Amazon Linux 2)

If your Elastic Beanstalk Java SE environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Extending and overriding the default proxy configuration

To extend Elastic Beanstalk's default nginx configuration, add `.conf` configuration files to a folder named `.ebextensions/nginx/conf.d/` in your application source bundle. Elastic Beanstalk's nginx configuration includes `.conf` files in this folder automatically.

```
~/workspace/my-app/
|-- .ebextensions
|   |-- nginx
|       |-- conf.d
|           |-- myconf.conf
|-- web.jar
```

To override Elastic Beanstalk's default nginx configuration completely, include a configuration in your source bundle at `.ebextensions/nginx/nginx.conf`:

```
~/workspace/my-app/
|-- .ebextensions
|   |-- nginx
|       |-- nginx.conf
|-- web.jar
```

If you override Elastic Beanstalk's nginx configuration, add the following line to your `nginx.conf` to pull in Elastic Beanstalk's configurations for [Enhanced health reporting and monitoring \(p. 691\)](#), automatic application mappings, and static files.

```
include conf.d/elasticbeanstalk/*.conf;
```

The following example configuration from the [Scorekeep sample application](#) overrides Elastic Beanstalk's default configuration to serve a static web application from the `public` subdirectory of `/var/app/current`, where the Java SE platform copies the application source code. The `/api` location forwards traffic to routes under `/api/` to the Spring application listening on port 5000. All other traffic is served by the web app at the root path.

Example `.ebextensions/nginx/nginx.conf`

```
user          nginx;
error_log     /var/log/nginx/error.log warn;
pid           /var/run/nginx.pid;
worker_processes auto;
worker_rlimit_nofile 33282;

events {
    worker_connections 1024;
}

http {
    include     /etc/nginx/mime.types;
    default_type application/octet-stream;

    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
                   '$status $body_bytes_sent "$http_referer" '
                   '"$http_user_agent" "$http_x_forwarded_for"';

    include     conf.d/*.conf;

    map $http_upgrade $connection_upgrade {
        default  "upgrade";
    }

    server {
        listen      80 default_server;
        root        /var/app/current/public;

        location / {
        }

        location /api {
            proxy_pass      http://127.0.0.1:5000;
            proxy_http_version 1.1;

            proxy_set_header    Connection      $connection_upgrade;
            proxy_set_header    Upgrade        $http_upgrade;
            proxy_set_header    Host          $host;
            proxy_set_header    X-Real-IP     $remote_addr;
            proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        }

        access_log    /var/log/nginx/access.log main;

        client_header_timeout 60;
        client_body_timeout 60;
        keepalive_timeout 60;
        gzip off;
        gzip_comp_level 4;

        # Include the Elastic Beanstalk generated locations
        include conf.d/elasticbeanstalk/01_static.conf;
        include conf.d/elasticbeanstalk/healthd.conf;
    }
}
```

```
}  
}
```

Adding an Amazon RDS DB instance to your Java application environment

You can use an Amazon Relational Database Service (Amazon RDS) DB instance to store data that your application gathers and modifies. The database can be attached to your environment and managed by Elastic Beanstalk, or created and managed externally.

If you are using Amazon RDS for the first time, add a DB instance to a test environment by using the Elastic Beanstalk console and verify that your application can connect to it.

To add a DB instance to your environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Choose a DB engine, and enter a user name and password.
6. Choose **Apply**.

Adding a DB instance takes about 10 minutes. When the environment update is complete, the DB instance's hostname and other connection information are available to your application through the following environment properties:

Property name	Description	Property value
RDS_HOSTNAME	The hostname of the DB instance.	On the Connectivity & security tab on the Amazon RDS console: Endpoint .
RDS_PORT	The port on which the DB instance accepts connections. The default value varies among DB engines.	On the Connectivity & security tab on the Amazon RDS console: Port .
RDS_DB_NAME	The database name, ebdb .	On the Configuration tab on the Amazon RDS console: DB Name .
RDS_USERNAME	The username that you configured for your database.	On the Configuration tab on the Amazon RDS console: Master username .
RDS_PASSWORD	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

For more information about configuring an internal DB instance, see [Adding a database to your Elastic Beanstalk environment \(p. 506\)](#). For instructions on configuring an external database for use with Elastic Beanstalk, see [Using Elastic Beanstalk with Amazon RDS \(p. 820\)](#).

To connect to the database, add the appropriate driver JAR file to your application, load the driver class in your code, and create a connection object with the environment properties provided by Elastic Beanstalk.

Sections

- [Downloading the JDBC driver \(p. 119\)](#)
- [Connecting to a database \(Java SE platforms\) \(p. 119\)](#)
- [Connecting to a database \(Tomcat platforms\) \(p. 120\)](#)
- [Troubleshooting database connections \(p. 122\)](#)

Downloading the JDBC driver

You will need the JAR file of the JDBC driver for the DB engine that you choose. Save the JAR file in your source code and include it in your classpath when you compile the class that creates connections to the database.

You can find the latest driver for your DB engine in the following locations:

- **MySQL** – [MySQL Connector/J](#)
- **Oracle SE-1** – [Oracle JDBC Driver](#)
- **Postgres** – [PostgreSQL JDBC Driver](#)
- **SQL Server** – [Microsoft JDBC Driver](#)

To use the JDBC driver, call `Class.forName()` to load it before creating the connection with `DriverManager.getConnection()` in your code.

JDBC uses a connection string in the following format:

```
jdbc:driver://hostname:port/dbName?user=username&password=password
```

You can retrieve the hostname, port, database name, user name, and password from the environment variables that Elastic Beanstalk provides to your application. The driver name is specific to your database type and driver version. The following are example driver names:

- `mysql` for MySQL
- `postgresql` for PostgreSQL
- `oracle:thin` for Oracle Thin
- `oracle:oci` for Oracle OCI
- `oracle:oci8` for Oracle OCI 8
- `oracle:kprb` for Oracle KPRB
- `sqlserver` for SQL Server

Connecting to a database (Java SE platforms)

In a Java SE environment, use `System.getenv()` to read the connection variables from the environment. The following example code shows a class that creates a connection to a PostgreSQL database.

```
private static Connection getRemoteConnection() {  
    if (System.getenv("RDS_HOSTNAME") != null) {  
        try {
```

```
Class.forName("org.postgresql.Driver");
String dbName = System.getenv("RDS_DB_NAME");
String userName = System.getenv("RDS_USERNAME");
String password = System.getenv("RDS_PASSWORD");
String hostname = System.getenv("RDS_HOSTNAME");
String port = System.getenv("RDS_PORT");
String jdbcUrl = "jdbc:postgresql://" + hostname + ":" + port + "/" + dbName + "?
user=" + userName + "&password=" + password;
logger.trace("Getting remote connection with connection string from environment
variables.");
Connection con = DriverManager.getConnection(jdbcUrl);
logger.info("Remote connection successful.");
return con;
}
catch (ClassNotFoundException e) { logger.warn(e.toString());}
catch (SQLException e) { logger.warn(e.toString());}
}
return null;
}
```

Connecting to a database (Tomcat platforms)

In a Tomcat environment, environment properties are provided as system properties that are accessible with `System.getProperty()`.

The following example code shows a class that creates a connection to a PostgreSQL database.

```
private static Connection getRemoteConnection() {
    if (System.getProperty("RDS_HOSTNAME") != null) {
        try {
            Class.forName("org.postgresql.Driver");
            String dbName = System.getProperty("RDS_DB_NAME");
            String userName = System.getProperty("RDS_USERNAME");
            String password = System.getProperty("RDS_PASSWORD");
            String hostname = System.getProperty("RDS_HOSTNAME");
            String port = System.getProperty("RDS_PORT");
            String jdbcUrl = "jdbc:postgresql://" + hostname + ":" + port + "/" + dbName + "?
user=" + userName + "&password=" + password;
            logger.trace("Getting remote connection with connection string from environment
variables.");
            Connection con = DriverManager.getConnection(jdbcUrl);
            logger.info("Remote connection successful.");
            return con;
        }
        catch (ClassNotFoundException e) { logger.warn(e.toString());}
        catch (SQLException e) { logger.warn(e.toString());}
    }
    return null;
}
```

If you have trouble getting a connection or running SQL statements, try placing the following code in a JSP file. This code connects to a DB instance, creates a table, and writes to it.

```
<%@ page import="java.sql.*" %>
<%
    // Read RDS connection information from the environment
    String dbName = System.getProperty("RDS_DB_NAME");
    String userName = System.getProperty("RDS_USERNAME");
    String password = System.getProperty("RDS_PASSWORD");
    String hostname = System.getProperty("RDS_HOSTNAME");
    String port = System.getProperty("RDS_PORT");
    String jdbcUrl = "jdbc:mysql://" + hostname + ":" +
```

```
port + "/" + dbName + "?user=" + userName + "&password=" + password;

// Load the JDBC driver
try {
    System.out.println("Loading driver...");
    Class.forName("com.mysql.jdbc.Driver");
    System.out.println("Driver loaded!");
} catch (ClassNotFoundException e) {
    throw new RuntimeException("Cannot find the driver in the classpath!", e);
}

Connection conn = null;
Statement setupStatement = null;
Statement readStatement = null;
ResultSet resultSet = null;
String results = "";
int numresults = 0;
String statement = null;

try {
    // Create connection to RDS DB instance
    conn = DriverManager.getConnection(jdbcUrl);

    // Create a table and write two rows
    setupStatement = conn.createStatement();
    String createTable = "CREATE TABLE Beanstalk (Resource char(50));";
    String insertRow1 = "INSERT INTO Beanstalk (Resource) VALUES ('EC2 Instance');";
    String insertRow2 = "INSERT INTO Beanstalk (Resource) VALUES ('RDS Instance');";

    setupStatement.addBatch(createTable);
    setupStatement.addBatch(insertRow1);
    setupStatement.addBatch(insertRow2);
    setupStatement.executeBatch();
    setupStatement.close();

} catch (SQLException ex) {
    // Handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
} finally {
    System.out.println("Closing the connection.");
    if (conn != null) try { conn.close(); } catch (SQLException ignore) {}
}

try {
    conn = DriverManager.getConnection(jdbcUrl);

    readStatement = conn.createStatement();
    resultSet = readStatement.executeQuery("SELECT Resource FROM Beanstalk;");

    resultSet.first();
    results = resultSet.getString("Resource");
    resultSet.next();
    results += ", " + resultSet.getString("Resource");

    resultSet.close();
    readStatement.close();
    conn.close();

} catch (SQLException ex) {
    // Handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
} finally {
```



```
        System.out.println("Closing the connection.");
        if (conn != null) try { conn.close(); } catch (SQLException ignore) {}
    }
    %>
```

To display the results, place the following code in the body of the HTML portion of the JSP file.

```
<p>Established connection to RDS. Read first two rows: <%= results %></p>
```

Troubleshooting database connections

If you run into issues connecting to a database from within your application, review the web container log and database.

Reviewing logs

You can view all the logs from your Elastic Beanstalk environment from within Eclipse. If you don't have the AWS Explorer view open, choose the arrow next to the orange AWS icon in the toolbar, and then choose **Show AWS Explorer View**. Expand **AWS Elastic Beanstalk** and your environment name, and then open the context (right-click) menu for the server. Choose **Open in WTP Server Editor**.

Choose the **Log** tab of the **Server** view to see the aggregate logs from your environment. To open the latest logs, choose the **Refresh** button at the upper right corner of the page.

Scroll down to locate the Tomcat logs in `/var/log/tomcat7/catalina.out`. If you loaded the webpage from our earlier example several times, you might see the following.

```
-----
/var/log/tomcat7/catalina.out
-----
INFO: Server startup in 9285 ms
Loading driver...
Driver loaded!
SQLException: Table 'Beanstalk' already exists
SQLState: 42S01
VendorError: 1050
Closing the connection.
Closing the connection.
```

All information that the web application sends to standard output appears in the web container log. In the previous example, the application tries to create the table every time the page loads. This results in catching a SQL exception on every page load after the first one.

As an example, the preceding is acceptable. But in actual applications, keep your database definitions in schema objects, perform transactions from within model classes, and coordinate requests with controller servlets.

Connecting to an RDS DB Instance

You can connect directly to the RDS DB instance in your Elastic Beanstalk environment by using the MySQL client application.

First, open the security group to your RDS DB instance to allow traffic from your computer.

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Next to **Endpoint**, choose the Amazon RDS console link.
6. On the **RDS Dashboard** instance details page, under **Security and Network**, choose the security group starting with *rds-* next to **Security Groups**.

Note

The database might have multiple entries labeled **Security Groups**. Use the first, which starts with *awseb*, only if you have an older account that doesn't have a default [Amazon Virtual Private Cloud](#) (Amazon VPC).

7. In **Security group details**, choose the **Inbound** tab, and then choose **Edit**.
8. Add a rule for MySQL (port 3306) that allows traffic from your IP address, specified in CIDR format.
9. Choose **Save**. The changes take effect immediately.

Return to the Elastic Beanstalk configuration details for your environment and note the endpoint. You will use the domain name to connect to the RDS DB instance.

Install the MySQL client and initiate a connection to the database on port 3306. On Windows, install MySQL Workbench from the MySQL home page and follow the prompts.

On Linux, install the MySQL client using the package manager for your distribution. The following example works on Ubuntu and other Debian derivatives.

```
// Install MySQL client
$ sudo apt-get install mysql-client-5.5
...
// Connect to database
$ mysql -h aas839jo2vwhwb.cnubrrfwfka8.us-west-2.rds.amazonaws.com -u username -ppassword
ebdb
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 117
Server version: 5.5.40-log Source distribution
...
```

After you have connected, you can run SQL commands to see the status of the database, whether your tables and rows were created, and other information.

```
mysql> SELECT Resource from Beanstalk;
+-----+
| Resource      |
+-----+
| EC2 Instance |
| RDS Instance |
+-----+
2 rows in set (0.01 sec)
```

Using the AWS Toolkit for Eclipse

The AWS Toolkit for Eclipse integrates AWS Elastic Beanstalk management features with your Tomcat development environment to facilitate environment creation, configuration, and code deployment. The

toolkit includes support for multiple AWS accounts, managing existing environments, and connecting directly to instances in your environment for troubleshooting.

Note

The AWS Toolkit for Eclipse only supports projects that use the Java with Tomcat platform, not the Java SE platform.

For more information about prerequisites and installing the AWS Toolkit for Eclipse, go to <https://aws.amazon.com/eclipse>. You can also check out the [Using AWS Elastic Beanstalk with the AWS Toolkit for Eclipse](#) video. This topic also provides useful information covering tools, how-to topics, and additional resources for Java developers.

Importing existing environments into Eclipse

You can import existing environments that you created in the AWS Management Console into Eclipse.

To import existing environments, expand the **AWS Elastic Beanstalk** node and double-click on an environment in the **AWS Explorer** inside Eclipse. You can now deploy your Elastic Beanstalk applications to this environment.

Managing Elastic Beanstalk application environments

Topics

- [Changing environment configuration settings \(p. 124\)](#)
- [Changing environment type \(p. 125\)](#)
- [Configuring EC2 server instances using AWS Toolkit for Eclipse \(p. 125\)](#)
- [Configuring Elastic Load Balancing using AWS Toolkit for Eclipse \(p. 128\)](#)
- [Configuring Auto Scaling using AWS Toolkit for Eclipse \(p. 131\)](#)
- [Configuring notifications using AWS Toolkit for Eclipse \(p. 133\)](#)
- [Configuring Java containers using AWS Toolkit for Eclipse \(p. 133\)](#)
- [Setting system properties with AWS Toolkit for Eclipse \(p. 134\)](#)

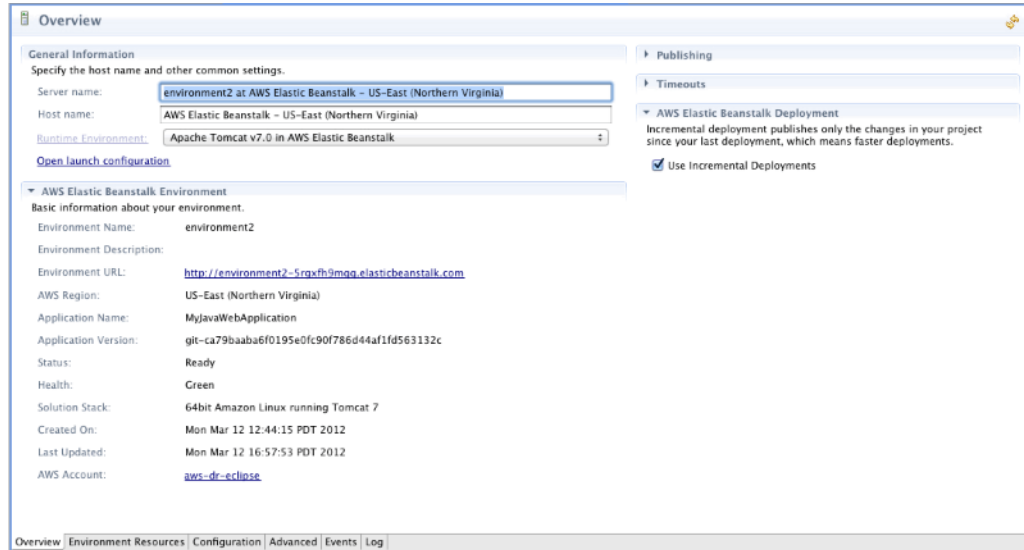
With the AWS Toolkit for Eclipse, you can change the provisioning and configuration of the AWS resources that are used by your application environments. For information on how to manage your application environments using the AWS Management Console, see [Managing environments \(p. 353\)](#). This section discusses the specific service settings you can edit in the AWS Toolkit for Eclipse as part of your application environment configuration. For more about AWS Toolkit for Eclipse, see [AWS Toolkit for Eclipse Getting Started Guide](#).

Changing environment configuration settings

When you deploy your application, Elastic Beanstalk configures a number of AWS cloud computing services. You can control how these individual services are configured using the AWS Toolkit for Eclipse.

To edit an application's environment settings

1. If Eclipse isn't displaying the **AWS Explorer** view, in the menu choose **Window, Show View, AWS Explorer**. Expand the Elastic Beanstalk node and your application node.
2. In **AWS Explorer**, double-click your Elastic Beanstalk environment.
3. At the bottom of the pane, click the **Configuration** tab.

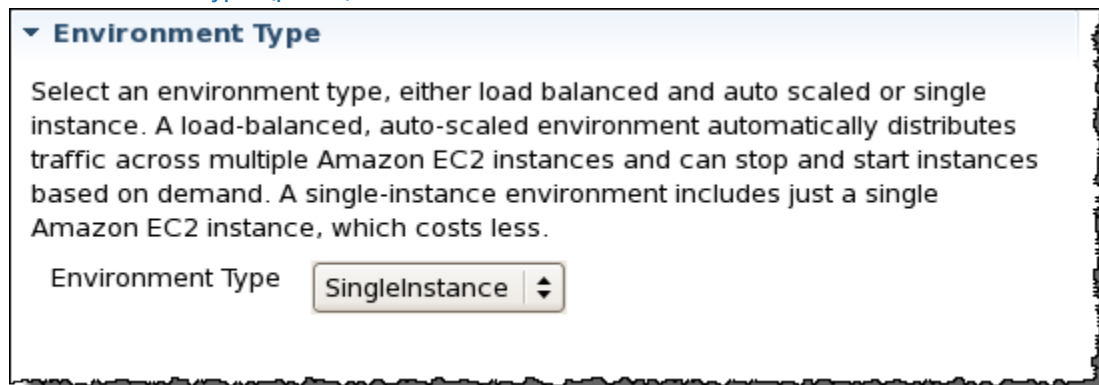


You can now configure settings for the following:

- EC2 server instances
- Load balancer
- Autoscaling
- Notifications
- Environment types
- Environment properties

Changing environment type

In AWS Toolkit for Eclipse, the **Environment Type** section of your environment's **Configuration** tab lets you select either **Load balanced, auto scaled** or a **Single instance** environment, depending on the requirements of the application that you deploy. For an application that requires scalability, select **Load balanced, auto scaled**. For a simple, low traffic application, select **Single instance**. For more information, see [Environment types \(p. 435\)](#).

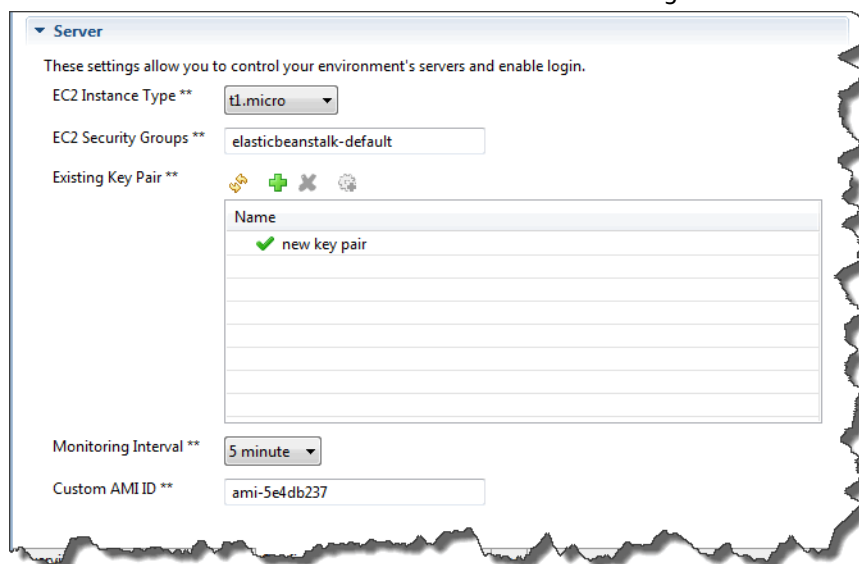


Configuring EC2 server instances using AWS Toolkit for Eclipse

Amazon Elastic Compute Cloud (EC2) is a web service for launching and managing server instances in Amazon's data centers. You can use Amazon EC2 server instances at any time, for as long as you

need, and for any legal purpose. Instances are available in different sizes and configurations. For more information, go to the [Amazon EC2 product page](#).

Under **Server**, on your environment's **Configuration** tab inside the Toolkit for Eclipse, you can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration.



Amazon EC2 instance types

Instance type displays the instance types available to your Elastic Beanstalk application. Change the instance type to select a server with the characteristics (including memory size and CPU power) that are most appropriate to your application. For example, applications with intensive and long-running operations can require more CPU or memory.

For more information about the Amazon EC2 instance types available for your Elastic Beanstalk application, see [Instance Types](#) in the *Amazon Elastic Compute Cloud User Guide*.

Amazon EC2 security groups

You can control access to your Elastic Beanstalk application using an *Amazon EC2 Security Group*. A security group defines firewall rules for your instances. These rules specify which ingress (i.e., incoming) network traffic should be delivered to your instance. All other ingress traffic will be discarded. You can modify rules for a group at any time. The new rules are automatically enforced for all running instances and instances launched in the future.

You can set up your Amazon EC2 security groups using the AWS Management Console or by using the AWS Toolkit for Eclipse. You can specify which Amazon EC2 security groups control access to your Elastic Beanstalk application by entering the names of one or more Amazon EC2 security group names (delimited by commas) into the **EC2 Security Groups** box.

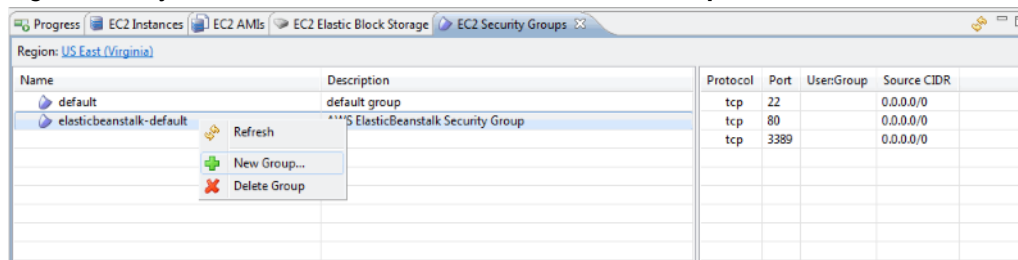
Note

If you are running your application using a legacy container type, make sure port 80 (HTTP) is accessible from 0.0.0.0/0 as the source CIDR range if you want to enable health checks for your application. For more information about health checks, see [Health checks \(p. 129\)](#). To check if you are using a legacy container type, see [the section called "Why are some platform versions marked legacy?" \(p. 426\)](#)

To create a security group using the AWS Toolkit for Eclipse

1. In the AWS Toolkit for Eclipse, click **AWS Explorer** tab. Expand the **Amazon EC2** node, and then double-click **Security Groups**.

2. Right-click anywhere in the left table, and then click **New Group**.



3. In the **Security Group** dialog box, type the security group name and description and then click **OK**.

For more information on Amazon EC2 Security Groups, see [Using Security Groups](#) in the *Amazon Elastic Compute Cloud User Guide*.

Amazon EC2 key pairs

You can securely log in to the Amazon EC2 instances provisioned for your Elastic Beanstalk application with an Amazon EC2 key pair.

Important

You must create an Amazon EC2 key pair and configure your Elastic Beanstalk-provisioned Amazon EC2 instances to use the Amazon EC2 key pair before you can access your Elastic Beanstalk-provisioned Amazon EC2 instances. You can create your key pair using the **Publish to Beanstalk Wizard** inside AWS Toolkit for Eclipse when you deploy your application to Elastic Beanstalk. Alternatively, you can set up your Amazon EC2 key pairs using the [AWS Management Console](#). For instructions on creating a key pair for Amazon EC2, see the [Amazon Elastic Compute Cloud Getting Started Guide](#).

For more information on Amazon EC2 key pairs, go to [Using Amazon EC2 Credentials](#) in the *Amazon Elastic Compute Cloud User Guide*. For more information on connecting to Amazon EC2 instances, go to [Connecting to Instances](#) and [Connecting to a Linux/UNIX Instance from Windows using PuTTY](#) in the *Amazon Elastic Compute Cloud User Guide*.

CloudWatch metrics

By default, only basic Amazon CloudWatch metrics are enabled. They return data in five-minute periods. You can enable more granular one-minute CloudWatch metrics by selecting **1 minute** for the **Monitoring Interval** in the **Server** section of the **Configuration** tab for your environment in the AWS Toolkit for Eclipse.

Note

Amazon CloudWatch service charges can apply for one-minute interval metrics. See [Amazon CloudWatch](#) for more information.

Custom AMI ID

You can override the default AMI used for your Amazon EC2 instances with your own custom AMI by entering the identifier of your custom AMI into the **Custom AMI ID** box in the **Server** section of the **Configuration** tab for your environment in the AWS Toolkit for Eclipse.

Important

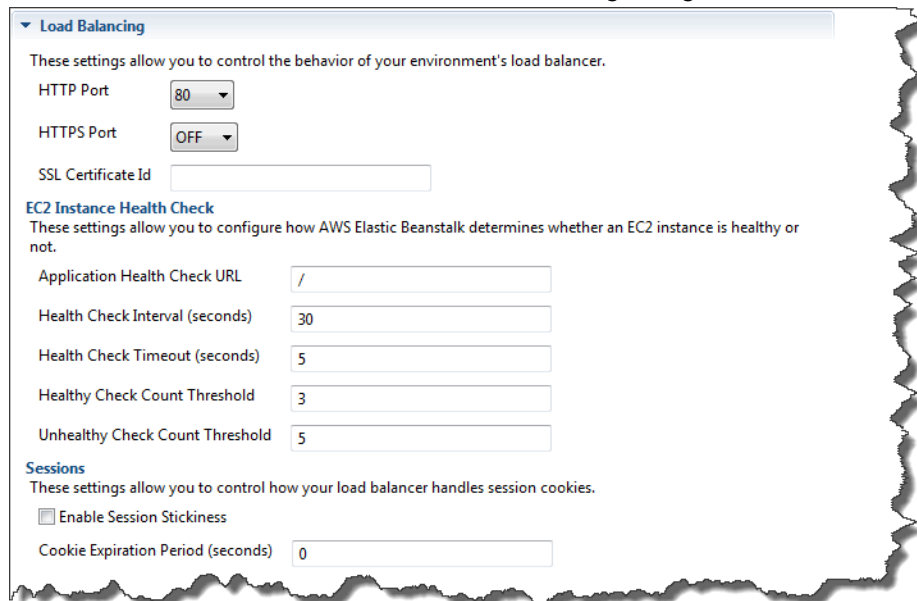
Using your own AMI is an advanced task that you should do with care. If you need a custom AMI, we recommend you start with the default Elastic Beanstalk AMI and then modify it. To be considered healthy, Elastic Beanstalk expects Amazon EC2 instances to meet a set of requirements, including having a running host manager. If these requirements are not met, your environment might not work properly.

Configuring Elastic Load Balancing using AWS Toolkit for Eclipse

Elastic Load Balancing is an Amazon web service that improves the availability and scalability of your application. With Elastic Load Balancing, you can distribute application loads between two or more Amazon EC2 instances. Elastic Load Balancing improves availability through redundancy, and it supports traffic growth for your application.

Elastic Load Balancing automatically distributes and balances incoming application traffic among all the EC2 server instances you are running. The service also makes it easy to add new instances when you need to increase the capacity of your application.

Elastic Beanstalk automatically provisions Elastic Load Balancing when you deploy an application. Under **Load Balancing**, on the **Configuration** tab for your environment inside the Toolkit for Eclipse, you can edit the Elastic Beanstalk environment's load balancing configuration.



The screenshot shows the 'Load Balancing' configuration panel in the AWS Toolkit for Eclipse. It includes sections for 'Load Balancing' (with HTTP and HTTPS ports, and SSL Certificate Id), 'EC2 Instance Health Check' (with Application Health Check URL, Health Check Interval, Health Check Timeout, Healthy Check Count Threshold, and Unhealthy Check Count Threshold), and 'Sessions' (with an 'Enable Session Stickiness' checkbox and a Cookie Expiration Period).

The following sections describe the Elastic Load Balancing parameters you can configure for your application.

Ports

The load balancer provisioned to handle requests for your Elastic Beanstalk application sends requests to the Amazon EC2 instances that are running your application. The provisioned load balancer can listen for requests on HTTP and HTTPS ports and route requests to the Amazon EC2 instances in your AWS Elastic Beanstalk application. By default, the load balancer handles requests on the HTTP port. At least one of the ports (either HTTP or HTTPS) must be turned on.



This screenshot shows a portion of the configuration window, specifically the 'Load Balancing' section. The HTTP Port is set to 80 and the HTTPS Port is set to 443. The SSL Certificate Id is set to 'arn:aws:iam::123456789012:/server-ce'.

Important

Make sure that the port you specified is not locked down; otherwise, users will not be able to connect to your Elastic Beanstalk application.

Controlling the HTTP port

To turn off the HTTP port, you select OFF for **HTTP Listener Port**. To turn on the HTTP port, you select an HTTP port (for example, **80**).

Note

To access your environment using a port other than the default port 80, such as port 8080, add a listener to the existing load balancer and configure the new listener to listen on that port. For example, using the [AWS CLI for Classic load balancers](#), type the following command, replacing `LOAD_BALANCER_NAME` with the name of your load balancer for Elastic Beanstalk.

```
aws elb create-load-balancer-listeners --load-balancer-name LOAD_BALANCER_NAME
--listeners "Protocol=HTTP, LoadBalancerPort=8080, InstanceProtocol=HTTP,
InstancePort=80"
```

For example, using the [AWS CLI for Application Load Balancers](#), type the following command, replacing `LOAD_BALANCER_ARN` with the ARN of your load balancer for Elastic Beanstalk.

```
aws elbv2 create-listener --load-balancer-arn LOAD_BALANCER_ARN --protocol HTTP --
port 8080
```

If you want Elastic Beanstalk to monitor your environment, do not remove the listener on port 80.

Controlling the HTTPS port

Elastic Load Balancing supports the HTTPS/TLS protocol to enable traffic encryption for client connections to the load balancer. Connections from the load balancer to the EC2 instances are done using plain text. By default, the HTTPS port is turned off.

To turn on the HTTPS port

1. Create a new certificate using AWS Certificate Manager (ACM) or upload a certificate and key to AWS Identity and Access Management (IAM). For more information about requesting an ACM certificate, see [Request a Certificate](#) in the *AWS Certificate Manager User Guide*. For more information about importing third-party certificates into ACM, see [Importing Certificates](#) in the *AWS Certificate Manager User Guide*. If ACM isn't [available in your AWS Region](#), use AWS Identity and Access Management (IAM) to upload a third-party certificate. The ACM and IAM services store the certificate and provide an Amazon Resource Name (ARN) for the SSL certificate. For more information about creating and uploading certificates to IAM, see [Working with Server Certificates](#) in *IAM User Guide*.
2. Specify the HTTPS port by selecting a port from the **HTTPS Listener Port** drop-down list.
3. In the **SSL Certificate ID** text box, enter the Amazon Resources Name (ARN) of your SSL certificate. For example, `arn:aws:iam::123456789012:server-certificate/abc/certs/build` or `arn:aws:acm:us-east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678`. Use the SSL certificate that you created and uploaded in step 1.

To turn off the HTTPS port, select **OFF** for **HTTPS Listener Port**.

Health checks

You can control the settings for the health check using the **EC2 Instance Health Check** section of the **Load Balancing** panel.

EC2 Instance Health Check
These settings allow you to configure how AWS Elastic Beanstalk determines whether an EC2 instance is healthy or not.

Application Health Check URL	/
Health Check Interval (seconds)	30
Health Check Timeout (seconds)	5
Healthy Check Count Threshold	3
Unhealthy Check Count Threshold	5

The following list describes the health check parameters you can set for your application.

- To determine instance health, Elastic Beanstalk looks for a 200 response code on a URL it queries. By default, Elastic Beanstalk checks TCP:80 for nonlegacy containers and HTTP:80 for legacy containers. You can override to match an existing resource in your application (e.g., `/myapp/index.jsp`) by entering it in the **Application Health Check URL** box. If you override the default URL, Elastic Beanstalk uses HTTP to query the resource. To check if you are using a legacy container type, see [the section called "Why are some platform versions marked legacy?"](#) (p. 426)
- For **Health Check Interval (seconds)**, enter the number of seconds between your application's Amazon EC2 instances health checks.
- For **Health Check Timeout**, specify the number of seconds for Elastic Load Balancing to wait for a response before it considers an instance unresponsive.
- Use the **Healthy Check Count Threshold** and **Unhealthy Check Count Threshold** boxes, specify the number of consecutive successful or unsuccessful URL probes before Elastic Load Balancing changes the instance health status. For example, specifying 5 in the **Unhealthy Check Count Threshold** text box means that the URL would have to return an error message or timeout five consecutive times before Elastic Load Balancing considers the health check "failed."

Sessions

By default, a load balancer routes each request independently to the server instance with the smallest load. By comparison, a sticky session binds a user's session to a specific server instance so that all requests coming from the user during the session are sent to the same server instance.

Elastic Beanstalk uses load balancer-generated HTTP cookies when sticky sessions are enabled for an application. The load balancer uses a special load balancer-generated cookie to track the application instance for each request. When the load balancer receives a request, it first checks to see if this cookie is present in the request. If so, the request is sent to the application instance specified in the cookie. If it finds no cookie, the load balancer chooses an application instance based on the existing load balancing algorithm. A cookie is inserted into the response for binding subsequent requests from the same user to that application instance. The policy configuration defines a cookie expiry, which establishes the duration of validity for each cookie.

Under **Load Balancer** in the **Sessions** section, specify whether or not the load balancer for your application allows session stickiness and the duration for each cookie.

Sessions
These settings allow you to control how your load balancer handles session cookies.

<input type="checkbox"/> Enable Session Stickiness
Cookie Expiration Period (seconds) 0

For more information on Elastic Load Balancing, see the [Elastic Load Balancing Developer Guide](#).

Configuring Auto Scaling using AWS Toolkit for Eclipse

Amazon EC2 Auto Scaling is an Amazon web service designed to automatically launch or terminate Amazon EC2 instances based on user-defined triggers. Users can set up *Auto Scaling groups* and associate *triggers* with these groups to automatically scale computing resources based on metrics such as bandwidth usage or CPU utilization. Amazon EC2 Auto Scaling works with Amazon CloudWatch to retrieve metrics for the server instances running your application.

Amazon EC2 Auto Scaling lets you take a group of Amazon EC2 instances and set various parameters to have this group automatically increase or decrease in number. Amazon EC2 Auto Scaling can add or remove Amazon EC2 instances from that group to help you seamlessly deal with traffic changes to your application.

Amazon EC2 Auto Scaling also monitors the health of each Amazon EC2 instance that it launches. If any instance terminates unexpectedly, Amazon EC2 Auto Scaling detects the termination and launches a replacement instance. This capability enables you to maintain a fixed, desired number of Amazon EC2 instances automatically.

Elastic Beanstalk provisions Amazon EC2 Auto Scaling for your application. Under **Auto Scaling**, on your environment's **Configuration** tab inside the Toolkit for Eclipse, you can edit the Elastic Beanstalk environment's Auto Scaling configuration.

Auto Scaling

Auto-scaling automatically launches or terminates EC2 instances based on defined metrics and thresholds called triggers. Auto-scaling will also launch a new EC2 instance in the event of a failure. These settings allow you to control auto-scaling behavior.

Minimum Instance Count:

Maximum Instance Count:

Availability Zones:

Scaling Cooldown Time (seconds):

Scaling Trigger

Trigger Measurement:

Trigger Statistic:

Unit of Measurement:

Measurement Period (seconds):

Breach Duration (seconds):

Upper Threshold:

Scale-up Increment:

Lower Threshold:

Scale-down Increment:

The following sections discuss how to configure Auto Scaling parameters for your application.

Launch configuration

You can edit the launch configuration to control how your Elastic Beanstalk application provisions Amazon EC2 Auto Scaling resources.

Use the **Minimum Instance Count** and **Maximum Instance Count** settings to specify the minimum and maximum size of the Auto Scaling group that your Elastic Beanstalk application uses.

A screenshot of the AWS Elastic Beanstalk configuration interface. It shows four fields: 'Minimum Instance Count' with a value of 1, 'Maximum Instance Count' with a value of 4, 'Availability Zones' with a dropdown menu set to 'Any 1', and 'Scaling Cooldown Time (seconds)' with a value of 360.

Note

To maintain a fixed number of Amazon EC2 instances, set the **Minimum Instance Count** and **Maximum Instance Count** text boxes to the same value.

For **Availability Zones**, specify the number of Availability Zones you want your Amazon EC2 instances to be in. It is important to set this number if you want to build fault-tolerant applications: If one Availability Zone goes down, your instances will still be running in your other Availability Zones.

Note

Currently, it is not possible to specify which Availability Zone your instance will be in.

Triggers

A *trigger* is an Amazon EC2 Auto Scaling mechanism that you set to tell the system when to increase (*scale out*) and decrease (*scale in*) the number of instances. You can configure triggers to *fire* on any metric published to Amazon CloudWatch, such as CPU utilization, and determine whether the specified conditions have been met. When your upper or lower thresholds for the metric have been breached for the specified period of time, the trigger launches a long-running process called a *scaling activity*.

You can define a scaling trigger for your Elastic Beanstalk application using the AWS Toolkit for Eclipse.

A screenshot of the 'Scaling Trigger' configuration section in the AWS Elastic Beanstalk interface. It includes the following fields: 'Trigger Measurement' (NetworkOut), 'Trigger Statistic' (Average), 'Unit of Measurement' (Bytes), 'Measurement Period (seconds)' (5), 'Breach Duration (seconds)' (5), 'Upper Threshold' (6000000), 'Scale-up Increment' (1), 'Lower Threshold' (2000000), and 'Scale-down Increment' (-1).

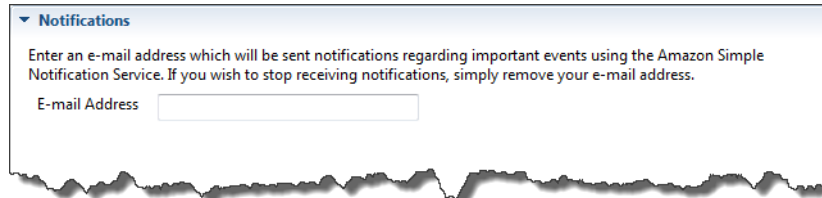
You can configure the following list of trigger parameters in the **Scaling Trigger** section of the **Configuration** tab for your environment inside the Toolkit for Eclipse.

- For **Trigger Measurement**, specify the metric for your trigger.
- For **Trigger Statistic**, specify which statistic the trigger will use—**Minimum**, **Maximum**, **Sum**, or **Average**.
- For **Unit of Measurement**, specify the units for the trigger measurement.
- For **Measurement Period**, specify how frequently Amazon CloudWatch measures the metrics for your trigger. For **Breach Duration**, specify the amount of time a metric can be beyond its defined limit (as specified for **Upper Threshold** and **Lower Threshold**) before the trigger fires.
- For **Scale-up Increment** and **Scale-down Increment**, specify how many Amazon EC2 instances to add or remove when performing a scaling activity.

For more information on Amazon EC2 Auto Scaling, see the *Amazon EC2 Auto Scaling* section on [Amazon Elastic Compute Cloud Documentation](#).

Configuring notifications using AWS Toolkit for Eclipse

Elastic Beanstalk uses the Amazon Simple Notification Service (Amazon SNS) to notify you of important events affecting your application. To enable Amazon SNS notifications, simply enter your email address in the **Email Address** text box under **Notifications** on the **Configuration** tab for your environment inside the Toolkit for Eclipse. To disable Amazon SNS notifications, remove your email address from the text box.



Configuring Java containers using AWS Toolkit for Eclipse

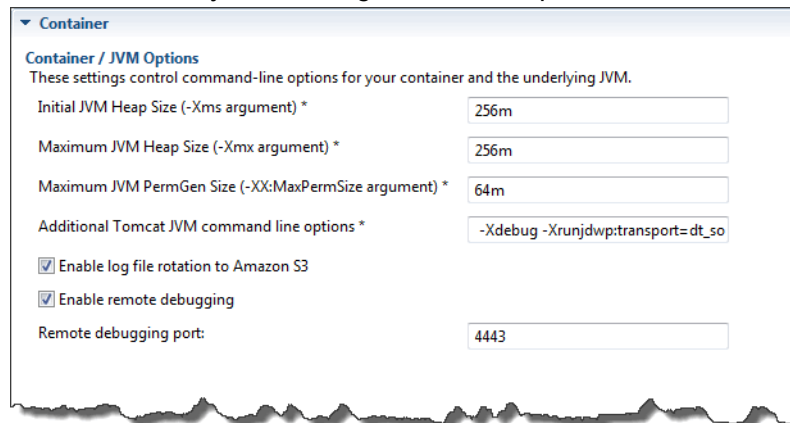
The **Container/JVM Options** panel lets you fine-tune the behavior of the Java Virtual Machine on your Amazon EC2 instances and enable or disable Amazon S3 log rotation. You can use the AWS Toolkit for Eclipse to configure your container information. For more information on the options available for Tomcat environments, see [the section called "Configuring your Tomcat environment" \(p. 102\)](#).

Note

You can modify your configuration settings with zero downtime by swapping the CNAME for your environments. For more information, see [Blue/Green deployments with Elastic Beanstalk \(p. 405\)](#).

To access the Container/JVM options panel for your Elastic Beanstalk application

1. If Eclipse isn't displaying the **AWS Explorer** view, in the menu choose **Window, Show View, AWS Explorer**. Expand the Elastic Beanstalk node and your application node.
2. In the **AWS Explorer**, double-click your Elastic Beanstalk environment.
3. At the bottom of the pane, click the **Configuration** tab.
4. Under **Container**, you can configure container options.



Remote debugging

To test your application remotely, you can run your application in debug mode.

To enable remote debugging

1. Select **Enable remote debugging**.
2. For **Remote debugging port**, specify the port number to use for remote debugging.

The **Additional Tomcat JVM command line options** setting is filled automatically.

To start remote debugging

1. In the AWS Toolkit for Eclipse menu, choose **Window, Show View, Other**.
2. Expand the **Server** folder, and then choose **Servers**. Choose **OK**.
3. In the **Servers** pane, right-click the server your application is running on, and then click **Restart in Debug**.

Setting system properties with AWS Toolkit for Eclipse

The following example sets the `JDBC_CONNECTION_STRING` system property in the AWS Toolkit for Eclipse. After you set this properties, it becomes available to your Elastic Beanstalk application as system properties called `JDBC_CONNECTION_STRING`.

Note

The AWS Toolkit for Eclipse does not yet support modifying environment configuration, including system properties, for environments in a VPC. Unless you have an older account using EC2 Classic, you must use the AWS Management Console (described in the next section) or the [EB CLI \(p. 852\)](#).

Note

Environment configuration settings can contain any printable ASCII character except the grave accent (` , ASCII 96) and cannot exceed 200 characters in length.

To set system properties for your Elastic Beanstalk application

1. If Eclipse isn't displaying the **AWS Explorer** view, choose **Window, Show View, Other**. Expand **AWS Toolkit** and then choose **AWS Explorer**.
2. In the **AWS Explorer** pane, expand **Elastic Beanstalk**, expand the node for your application, and then double-click your Elastic Beanstalk environment.
3. At the bottom of the pane for your environment, click the **Advanced** tab.
4. Under **aws:elasticbeanstalk:application:environment**, click **JDBC_CONNECTION_STRING** and then type a connection string. For example, the following JDBC connection string would connect to a MySQL database instance on port 3306 of localhost, with a user name of `me` and a password of `mypassword`:

```
jdbc:mysql://localhost:3306/mydatabase?user=me&password=mypassword
```

This will be accessible to your Elastic Beanstalk application as a system property called `JDBC_CONNECTION_STRING`.

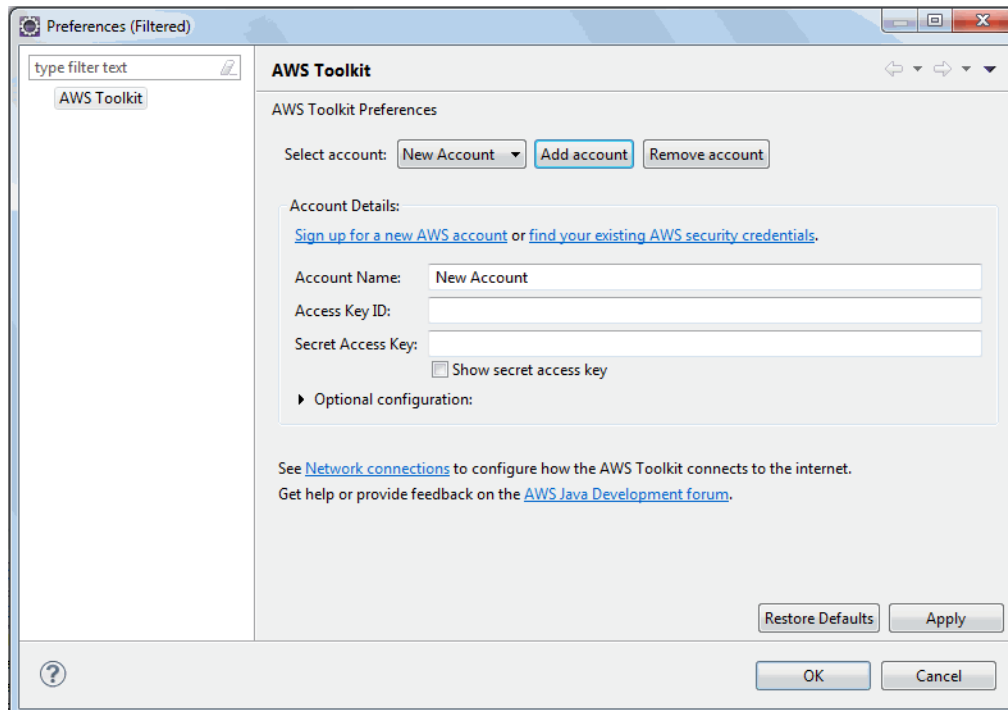
5. Press **Ctrl+C** on the keyboard or choose **File, Save** to save your changes to the environment configuration. Changes are reflected in about one minute.

Managing multiple AWS accounts

You might want to set up different AWS accounts to perform different tasks, such as testing, staging, and production. You can use the AWS Toolkit for Eclipse to add, edit, and delete accounts easily.

To add an AWS account with the AWS Toolkit for Eclipse

1. In Eclipse, make sure the toolbar is visible. On the toolbar, click the arrow next to the AWS icon and select **Preferences**.
2. Click **Add account**.



3. In the **Account Name** text box, type the display name for the account.
4. In the **Access Key ID** text box, type your AWS access key ID.
5. In the **Secret Access Key** text box, type your AWS secret key.

For API access, you need an access key ID and secret access key. Use IAM user access keys instead of AWS account root user access keys. For more information about creating access keys, see [Managing Access Keys for IAM Users](#) in the *IAM User Guide*.

6. Click **OK**.

To use a different account to deploy an application to Elastic Beanstalk

1. In the Eclipse toolbar, click the arrow next to the AWS icon and select **Preferences**.
2. For **Default Account**, select the account you want to use to deploy applications to Elastic Beanstalk.
3. Click **OK**.
4. In the **Project Explorer** pane, right-click the application you want to deploy, and then select **Amazon Web Services > Deploy to Elastic Beanstalk**.

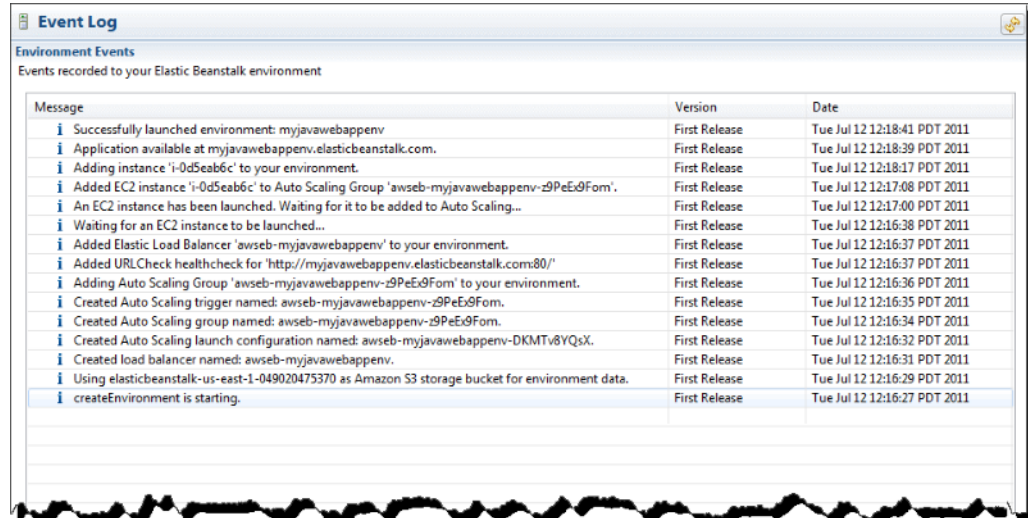
Viewing events

You can use the AWS Toolkit for Eclipse to access events and notifications associated with your application.

To view application events

1. If Eclipse isn't displaying the **AWS Explorer** view, in the menu click **Window > Show View > AWS Explorer**. Expand the Elastic Beanstalk node and your application node.
2. In the AWS Explorer, double-click your Elastic Beanstalk environment.
3. At the bottom of the pane, click the **Events** tab.

A list of the events for all environments for your application is displayed.

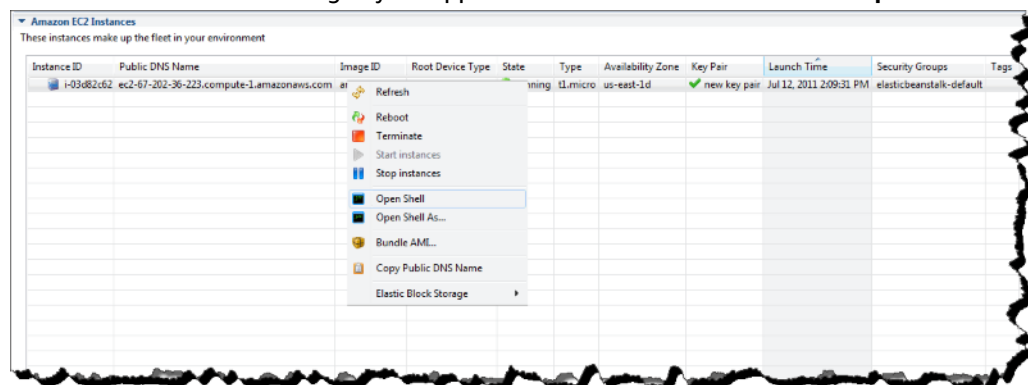


Listing and connecting to server instances

You can view a list of Amazon EC2 instances running your Elastic Beanstalk application environment through the AWS Toolkit for Eclipse or from the AWS Management Console. You can connect to these instances using Secure Shell (SSH). For information about listing and connecting to your server instances using the AWS Management Console, see [Listing and connecting to server instances \(p. 730\)](#). The following section steps you through viewing and connecting you to your server instances using the AWS Toolkit for Eclipse.

To view and connect to Amazon EC2 instances for an environment

1. In the AWS Toolkit for Eclipse, click **AWS Explorer**. Expand the **Amazon EC2** node, and then double-click **Instances**.
2. In the Amazon EC2 Instances window, in the **Instance ID** column, right-click the **Instance ID** for the Amazon EC2 instance running in your application's load balancer. Then click **Open Shell**.



Eclipse automatically opens the SSH client and makes the connection to the EC2 instance.

For more information on connecting to an Amazon EC2 instance, see the [Amazon Elastic Compute Cloud Getting Started Guide](#).

Terminating an environment

To avoid incurring charges for unused AWS resources, you can use the AWS Toolkit for Eclipse to terminate a running environment. For details about environment termination, see [Terminate an Elastic Beanstalk environment](#) (p. 386).

To terminate an environment

1. In the AWS Toolkit for Eclipse, click the **AWS Explorer** pane. Expand the **Elastic Beanstalk** node.
2. Expand the Elastic Beanstalk application and right-click on the Elastic Beanstalk environment.
3. Click **Terminate Environment**. It will take a few minutes for Elastic Beanstalk to terminate the AWS resources running in the environment.

Resources

There are several places you can go to get additional help when developing your Java applications.

Resource	Description
The AWS Java Development Forum	Post your questions and get feedback.
Java Developer Center	One-stop shop for sample code, documentation, tools, and additional resources.

Creating and deploying .NET applications on Elastic Beanstalk

AWS Elastic Beanstalk for .NET makes it easier to deploy, manage, and scale your ASP.NET web applications that use Amazon Web Services. Elastic Beanstalk for .NET is available to anyone who is developing or hosting a web application that uses IIS.

Get started now: To get started with a tutorial, you can go directly to [Tutorial: How to deploy a .NET sample application using Elastic Beanstalk](#) (p. 149). In this tutorial, you will deploy a sample ASP.NET Web Application to an AWS Elastic Beanstalk application container.

The rest of this section presents instructions for creating, testing, deploying, and redeploying your ASP.NET web application to Elastic Beanstalk. Some examples demonstrate using the AWS Toolkit for Visual Studio, and [the section called “The AWS Toolkit for Visual Studio”](#) (p. 166) subsection explains how to manage and configure your applications and environments using the toolkit. For more information about prerequisites, installation instructions, and running code samples, go to the [AWS Toolkit for Microsoft Visual Studio](#). This site also provides useful information about tools, how-to topics, and additional resources for ASP.NET developers.

Note

This platform doesn't support the following Elastic Beanstalk features:

- Worker environments. For details, see [Elastic Beanstalk worker environments](#) (p. 437).
- Bundle logs. For details, see [View instance logs](#) (p. 732).

In addition, platform versions earlier than v2.0.0 don't support enhanced health reporting, managed platform updates, immutable updates, immutable deployments, and rolling deployments with an additional batch.

The topics in this chapter assume some knowledge of Elastic Beanstalk environments. If you haven't used Elastic Beanstalk before, try the [getting started tutorial \(p. 3\)](#) to learn the basics.

Topics

- [Getting started with .NET on Elastic Beanstalk \(p. 138\)](#)
- [Setting up your .NET development environment \(p. 140\)](#)
- [Using the Elastic Beanstalk .NET platform \(p. 141\)](#)
- [Tutorial: How to deploy a .NET sample application using Elastic Beanstalk \(p. 149\)](#)
- [Deploying an ASP.NET core application with Elastic Beanstalk \(p. 155\)](#)
- [Adding an Amazon RDS DB instance to your .NET application environment \(p. 163\)](#)
- [The AWS Toolkit for Visual Studio \(p. 166\)](#)
- [Migrating your on-premises .NET application to Elastic Beanstalk \(p. 191\)](#)
- [Resources \(p. 192\)](#)

Getting started with .NET on Elastic Beanstalk

To get started with .NET applications on AWS Elastic Beanstalk, all you need is an application [source bundle \(p. 342\)](#) to upload as your first application version and to deploy to an environment. When you create an environment, Elastic Beanstalk allocates all of the AWS resources needed to run a highly scalable web application.

Launching an environment with a sample .NET application

Elastic Beanstalk provides single page sample applications for each platform as well as more complex examples that show the use of additional AWS resources such as Amazon RDS and language or platform-specific features and APIs.

Samples

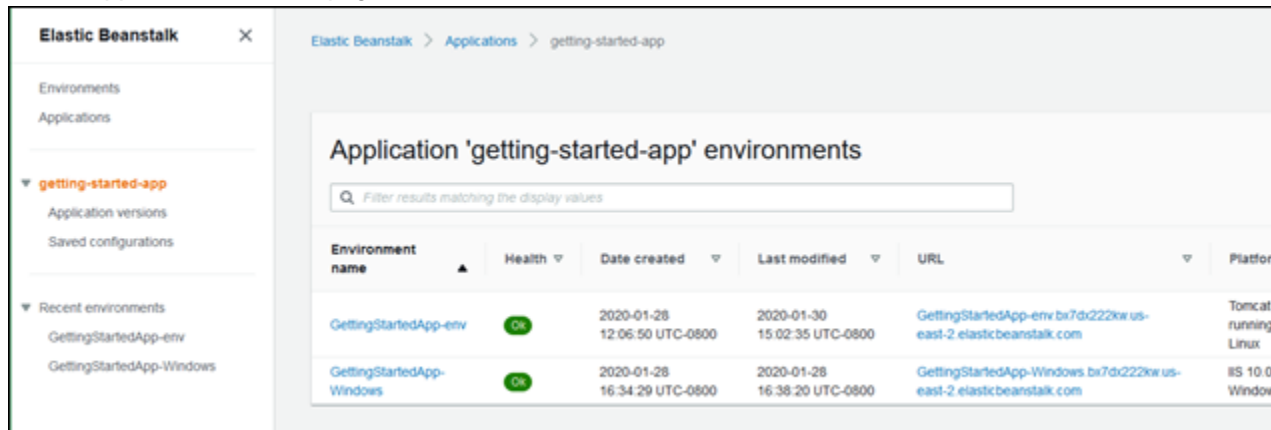
Platform	Supported configurations	Environment type	Source	Description
.NET	WS 2012 R2 Default WS 2012 R2 Server Core WS 2012 WS 2008 R2	Web Service	dotnet-asp-v1.zip	ASP.NET web application with a single page configured to be displayed at the website root.
ASP.NET MVC5	WS 2012 R2	Web Service	dotnet-aspmvc5-v1.zip	ASP.NET web application with a classic model-view-control architecture.

Download any of the sample applications and deploy it to Elastic Beanstalk by using the following procedure.

To launch an environment with a sample application (console)

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.

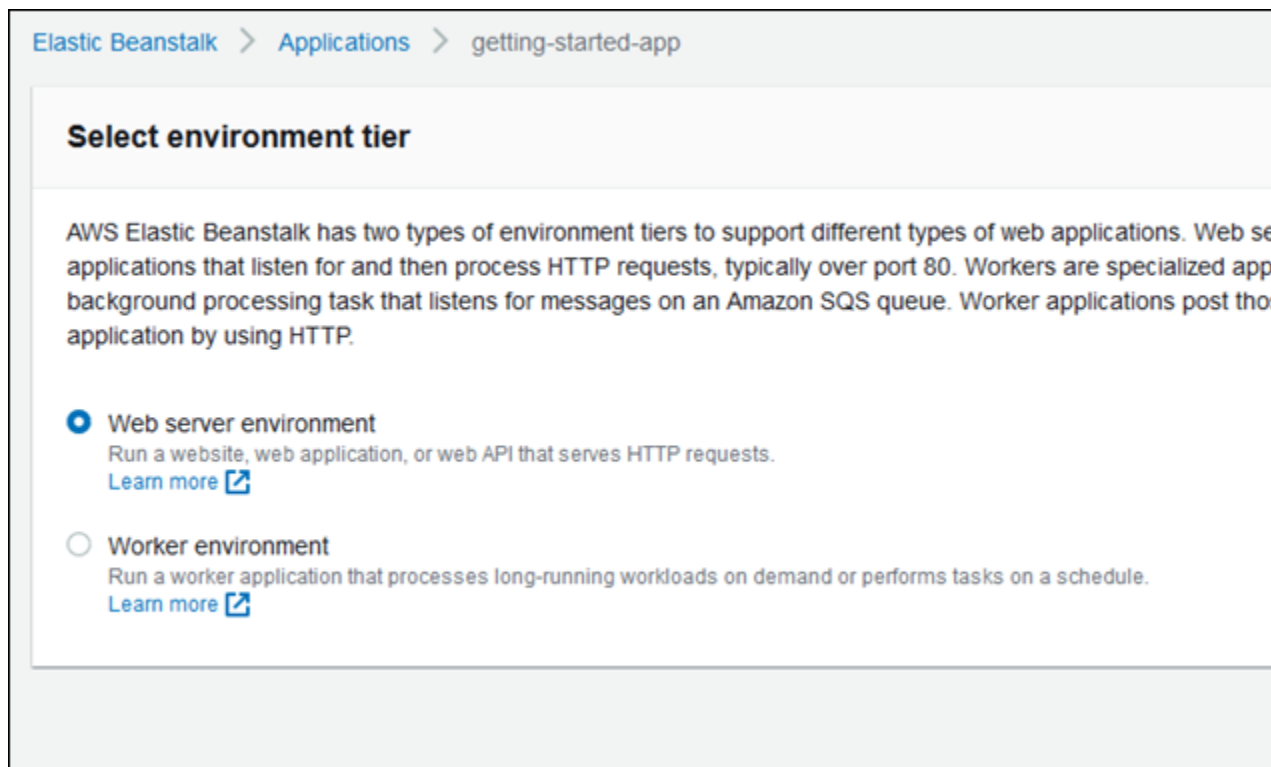
2. In the navigation pane, choose **Applications**, and then choose an existing application's name on the list or [create one](#) (p. 334).
3. On the application overview page, choose **Create a new environment**.



4. Choose the **Web server environment** or **Worker environment** [environment tier](#) (p. 13). You can't change an environment's tier after creation.

Note

The [.NET on Windows Server platform](#) (p. 137) doesn't support the worker environment tier.



5. For **Platform**, select the platform and platform branch that match the language used by your application.

Note

Elastic Beanstalk supports multiple [versions](#) (p. 29) for most of the platforms that are listed. By default, the console selects the recommended version for the platform and platform branch you choose. If your application requires different version, you can select it

here, or by choosing **Configure more options**, as described in step 7. For information about supported platform versions, see [the section called "Supported platforms" \(p. 29\)](#).

6. For **Application code**, choose **Sample application**.
7. To further customize your environment, choose **Configure more options**. You can set the following options only during environment creation:
 - Environment name
 - Domain name
 - Platform version
 - VPC
 - Tier

You can change the following settings after environment creation, but they require new instances or other resources to be provisioned and can take a long time to apply:

- Instance type, root volume, key pair, and AWS Identity and Access Management (IAM) role
- Internal Amazon RDS database
- Load balancer

For details on all available settings, see [The create new environment wizard \(p. 365\)](#).

8. Choose **Create environment**.

Next steps

After you have an environment running an application, you can [deploy a new version \(p. 397\)](#) of the application or a completely different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances.

After you've deployed a sample application or two and are ready to start developing locally, see [the next section \(p. 140\)](#) to set up a .NET development environment.

Setting up your .NET development environment

Set up a .NET development environment to test your application locally prior to deploying it to AWS Elastic Beanstalk. This topic outlines development environment setup steps and links to installation pages for useful tools.

For common setup steps and tools that apply to all languages, see [Configuring your development machine for use with Elastic Beanstalk \(p. 849\)](#).

Sections

- [Installing an IDE \(p. 141\)](#)
- [Installing the AWS Toolkit for Visual Studio \(p. 141\)](#)

If you need to manage AWS resources from within your application, install the AWS SDK for .NET. For example, you can use Amazon S3 to store and retrieve data.

With the AWS SDK for .NET, you can get started in minutes with a single, downloadable package complete with Visual Studio project templates, the AWS .NET library, C# code samples, and documentation. Practical examples are provided in C# for how to use the libraries to build applications. Online video tutorials and reference documentation are provided to help you learn how to use the libraries and code samples.

Visit the [AWS SDK for .NET homepage](#) for more information and installation instructions.

Installing an IDE

Integrated development environments (IDEs) provide a wide range of features that facilitate application development. If you haven't used an IDE for .NET development, try Visual Studio Community to get started.

Visit the [Visual Studio Community](#) page to download and install Visual Studio Community.

Installing the AWS Toolkit for Visual Studio

The [AWS Toolkit for Visual Studio \(p. 166\)](#) is an open source plug-in for the Visual Studio IDE that makes it easier for developers to develop, debug, and deploy .NET applications using AWS. Visit the [Toolkit for Visual Studio homepage](#) for installation instructions.

Using the Elastic Beanstalk .NET platform

AWS Elastic Beanstalk supports a number of platforms for different versions of the .NET programming framework and Windows Server. See [.NET on Windows Server with IIS](#) in the *AWS Elastic Beanstalk Platforms* document for a full list.

Elastic Beanstalk provides [configuration options \(p. 536\)](#) that you can use to customize the software that runs on the EC2 instances in your Elastic Beanstalk environment. You can configure environment variables needed by your application, enable log rotation to Amazon S3, and set .NET framework settings.

Configuration options are available in the Elastic Beanstalk console for [modifying the configuration of a running environment \(p. 547\)](#). To avoid losing your environment's configuration when you terminate it, you can use [saved configurations \(p. 640\)](#) to save your settings and later apply them to another environment.

To save settings in your source code, you can include [configuration files \(p. 600\)](#). Settings in configuration files are applied every time you create an environment or deploy your application. You can also use configuration files to install packages, run scripts, and perform other instance customization operations during deployments.

Settings applied in the Elastic Beanstalk console override the same settings in configuration files, if they exist. This lets you have default settings in configuration files, and override them with environment-specific settings in the console. For more information about precedence, and other methods of changing settings, see [Configuration options \(p. 536\)](#).

Configuring your .NET environment in the Elastic Beanstalk console

You can use the Elastic Beanstalk console to enable log rotation to Amazon S3, configure variables that your application can read from the environment, and change .NET framework settings.

To configure your .NET environment in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.

4. In the **Software** configuration category, choose **Edit**.

Container options

- **Target .NET runtime** – Set to 2.0 to run CLR v2.
- **Enable 32-bit applications** – Set to `True` to run 32-bit applications.

Log options

The Log Options section has two settings:

- **Instance profile** – Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3** – Specifies whether log files for your application's Amazon EC2 instances should be copied to the Amazon S3 bucket associated with your application.

Environment properties

The **Environment Properties** section lets you specify environment configuration settings on the Amazon EC2 instances that are running your application. These settings are passed in as key-value pairs to the application. Use `System.EnvironmentVariable` to read them. Identical keys can exist in both `web.config` and as environment properties. Use the `System.Configuration` namespace to read values from `web.config`.

```
NameValueCollection appConfig = ConfigurationManager.AppSettings;  
string endpoint = appConfig["API_ENDPOINT"];
```

Note

Elastic Beanstalk doesn't support passing environment variables to .NET Core applications and multiple-application IIS deployments that use a [deployment manifest \(p. 145\)](#).

See [Environment properties and other software settings \(p. 516\)](#) for more information.

The `aws:elasticbeanstalk:container:dotnet:apppool` namespace

You can use a [configuration file \(p. 600\)](#) to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be defined by the Elastic Beanstalk service or the platform that you use and are organized into *namespaces*.

The .NET platform defines options in the `aws:elasticbeanstalk:container:dotnet:apppool` namespace that you can use to configure the .NET runtime.

The following example configuration file shows settings for each of the options available in this namespace:

Example `.ebextensions/dotnet-settings.config`

```
option_settings:  
  aws:elasticbeanstalk:container:dotnet:apppool:  
    Target Runtime: 2.0  
    Enable 32-bit Applications: True
```

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See [Configuration options \(p. 536\)](#) for more information.

Migrating across major versions of the Elastic Beanstalk Windows server platform

AWS Elastic Beanstalk has had several major versions of its Windows Server platform. This page covers the main improvements for each major version, and what to consider before you migrate to a later version.

The Windows Server platform is currently at version 2 (v2). If your application uses any Windows Server platform version earlier than v2, we recommend that you migrate to v2.

What's new in major versions of the Windows server platform

Windows server platform V2

Version 2 (v2) of the Elastic Beanstalk Windows Server platform was [released in February 2019](#). V2 brings the behavior of the Windows Server platform closer to that of the Elastic Beanstalk Linux-based platforms in several important ways. V2 is fully backward compatible with v1, making migration from v1 easy.

The Windows Server platform now supports the following:

- *Versioning* – Each release gets a new version number, and you can refer to past versions (that are still available to you) when creating and managing environments.
- *Enhanced health* – For details, see [Enhanced health reporting and monitoring \(p. 691\)](#).
- *Immutable and Rolling with an Additional Batch* deployments – For details about deployment policies, see [Deploying applications to Elastic Beanstalk environments \(p. 397\)](#).
- *Immutable updates* – For details about update types, see [Configuration changes \(p. 408\)](#).
- *Managed platform updates* – For details, see [Managed platform updates \(p. 420\)](#).

Note

The new deployment and update features depend on enhanced health. Enable enhanced health to use them. For details, see [Enabling Elastic Beanstalk enhanced health reporting \(p. 698\)](#).

Windows server platform V1

Version 1.0.0 (v1) of the Elastic Beanstalk Windows Server platform was released in October 2015. This version changes the order in which Elastic Beanstalk processes commands in [configuration files \(p. 600\)](#) during environment creation and updates.

Previous platform versions don't have a version number in the solution stack name:

- 64bit Windows Server 2012 R2 running IIS 8.5
- 64bit Windows Server Core 2012 R2 running IIS 8.5
- 64bit Windows Server 2012 running IIS 8
- 64bit Windows Server 2008 R2 running IIS 7.5

In earlier versions, the processing order for configuration files is inconsistent. During environment creation, `Container Commands` run after the application source is deployed to IIS. During a deployment to a running environment, container commands run before the new version is deployed. During a scale up, configuration files are not processed at all.

In addition to this, IIS starts up before container commands run. This behavior has led some customers to implement workarounds in container commands, pausing the IIS server before commands run, and starting it again after they complete.

Version 1 fixes the inconsistency and brings the behavior of the Windows Server platform closer to that of the Elastic Beanstalk Linux-based platforms. In the v1 platform, Elastic Beanstalk always runs container commands before starting the IIS server.

The v1 platform solution stacks have a v1 after the Windows Server version:

- 64bit Windows Server 2012 R2 v1.1.0 running IIS 8.5
- 64bit Windows Server Core 2012 R2 v1.1.0 running IIS 8.5
- 64bit Windows Server 2012 v1.1.0 running IIS 8
- 64bit Windows Server 2008 R2 v1.1.0 running IIS 7.5

Additionally, the v1 platform extracts the contents of your application source bundle to `C:\staging\` before running container commands. After container commands complete, the contents of this folder are compressed into a .zip file and deployed to IIS. This workflow allows you to modify the contents of your application source bundle with commands or a script before deployment.

Migrating from earlier major versions of the Windows server platform

Read this section for migration considerations before you update your environment. To update your environment's platform to a newer version, see [Updating your Elastic Beanstalk environment's platform version](#) (p. 415).

From V1 to V2

The Windows Server platform v2 doesn't support .NET Core 1.x and 2.0. If you're migrating your application from Windows Server v1 to v2, and your application uses one of these .NET Core versions, update your application to a .NET Core version that v2 supports. For a list of supported versions, see [.NET on Windows Server with IIS](#) in the *AWS Elastic Beanstalk Platforms*.

If your application uses a custom Amazon Machine Image (AMI), create a new custom AMI based on a Windows Server platform v2 AMI. To learn more, see [Using a custom Amazon machine image \(AMI\)](#) (p. 647).

Note

The deployment and update features that are new to Windows Server v2 depend on enhanced health. When you migrate an environment to v2, enhanced health is disabled. Enable it to use these features. For details, see [Enabling Elastic Beanstalk enhanced health reporting](#) (p. 698).

From pre-V1

In addition to considerations for migrating from v1, if you're migrating your application from a Windows Server solution stack that's earlier than v1, and you currently use container commands, remove any commands that you added to work around the processing inconsistencies when you migrate to a newer version. Starting with v1, container commands are guaranteed to run completely before the application source that is deployed and before IIS starts. This enables you to make any changes to the source in `C:\staging` and modify IIS configuration files during this step without issue.

For example, you can use the AWS CLI to download a DLL file to your application source from Amazon S3:

```
.ebextensions\copy-dll.config
```

```
container_commands:
  copy-dll:
    command: aws s3 cp s3://my-bucket/dlls/large-dll.dll .\lib\
```

For more information on using configuration files, see [Advanced environment customization with configuration files \(.ebextensions\)](#) (p. 600).

Running multiple applications and ASP.NET core applications with a deployment manifest

You can use a deployment manifest to tell Elastic Beanstalk how to deploy your application. For example, instead of using `MSDeploy` to generate a source bundle for a single ASP.NET application that runs at the root path of your website, you can use a manifest file to run multiple applications at different paths, or tell Elastic Beanstalk to deploy and run the app with ASP.NET Core. You can also use a deployment manifest to configure an application pool in which to run your applications.

Deployment manifests add support for [.NET Core applications \(p. 145\)](#) to Elastic Beanstalk. You can deploy a .NET Framework application without a deployment manifest, but .NET Core applications require a deployment manifest to run on Elastic Beanstalk. When you use a deployment manifest, you create a site archive for each application and then bundle the site archives in a second ZIP archive that contains the deployment manifest.

Deployment manifests also add the ability to [run multiple applications at different paths \(p. 147\)](#). A deployment manifest defines an array of deployment targets, each with a site archive and a path at which IIS should run it. For example, you could run a web API at the `/api` path to serve asynchronous requests, and a web app at the root path that consumes the API.

You can also use a deployment manifest to [create application pools in IIS \(p. 147\)](#) in which to run one or more applications. You can configure an application pool to restart your applications periodically, run 32-bit applications, or use a specific version of the .NET Framework runtime.

For full customization, you can [write your own deployment scripts \(p. 148\)](#) in Windows PowerShell and tell Elastic Beanstalk which scripts to run to install, uninstall, and restart your application.

Deployment manifests and related features require a Windows Server platform [version 1.2.0 or newer \(p. 143\)](#).

Sections

- [.NET core apps \(p. 145\)](#)
- [Run multiple applications \(p. 147\)](#)
- [Configure application pools \(p. 147\)](#)
- [Define custom deployments \(p. 148\)](#)

.NET core apps

You can use a deployment manifest to run .NET Core applications on Elastic Beanstalk. .NET Core is a cross-platform version of .NET that comes with a command line tool (`dotnet`) that you can use to generate an application, run it locally, and prepare it for publishing.

Note

See [Deploying an ASP.NET core application with Elastic Beanstalk \(p. 155\)](#) for a tutorial and sample application that use a deployment manifest to run a .NET Core application on Elastic Beanstalk.

To run a .NET Core application on Elastic Beanstalk, run `dotnet publish` and package the output in a ZIP archive, not including any containing directories. Place the site archive in a source bundle with a deployment manifest with a deployment target of type `aspNetCoreWeb`.

The following deployment manifest runs a .NET Core application from a site archive named `dotnet-core-app.zip` at the root path.

Example `aws-windows-deployment-manifest.json` - .NET core

```
{
```



```
"manifestVersion": 1,
"deployments": {
  "aspNetCoreWeb": [
    {
      "name": "my-dotnet-core-app",
      "parameters": {
        "archive": "dotnet-core-app.zip",
        "iisPath": "/"
      }
    }
  ]
}
```

Bundle the manifest and site archive in a ZIP archive to create a source bundle.

Example dotnet-core-bundle.zip

```
.
|-- aws-windows-deployment-manifest.json
^-- dotnet-core-app.zip
```

The site archive contains the compiled application code, dependencies, and `web.config` file.

Example dotnet-core-app.zip

```
.
|-- Microsoft.AspNetCore.Hosting.Abstractions.dll
|-- Microsoft.AspNetCore.Hosting.Server.Abstractions.dll
|-- Microsoft.AspNetCore.Hosting.dll
|-- Microsoft.AspNetCore.Http.Abstractions.dll
|-- Microsoft.AspNetCore.Http.Extensions.dll
|-- Microsoft.AspNetCore.Http.Features.dll
|-- Microsoft.AspNetCore.Http.dll
|-- Microsoft.AspNetCore.HttpOverrides.dll
|-- Microsoft.AspNetCore.Server.IISIntegration.dll
|-- Microsoft.AspNetCore.Server.Kestrel.dll
|-- Microsoft.AspNetCore.WebUtilities.dll
|-- Microsoft.Extensions.Configuration.Abstractions.dll
|-- Microsoft.Extensions.Configuration.EnvironmentVariables.dll
|-- Microsoft.Extensions.Configuration.dll
|-- Microsoft.Extensions.DependencyInjection.Abstractions.dll
|-- Microsoft.Extensions.DependencyInjection.dll
|-- Microsoft.Extensions.FileProviders.Abstractions.dll
|-- Microsoft.Extensions.FileProviders.Physical.dll
|-- Microsoft.Extensions.FileSystemGlobbing.dll
|-- Microsoft.Extensions.Logging.Abstractions.dll
|-- Microsoft.Extensions.Logging.dll
|-- Microsoft.Extensions.ObjectPool.dll
|-- Microsoft.Extensions.Options.dll
|-- Microsoft.Extensions.PlatformAbstractions.dll
|-- Microsoft.Extensions.Primitives.dll
|-- Microsoft.Net.Http.Headers.dll
|-- System.Diagnostics.Contracts.dll
|-- System.Net.WebSockets.dll
|-- System.Text.Encodings.Web.dll
|-- dotnet-core-app.deps.json
|-- dotnet-core-app.dll
|-- dotnet-core-app.pdb
|-- dotnet-core-app.runtimeconfig.json
^-- web.config
```

See [the tutorial \(p. 155\)](#) for a full example.

Run multiple applications

You can run multiple applications with a deployment manifest by defining multiple deployment targets.

The following deployment manifest configures two .NET Core applications. The `WebAPITest` application implements a few web APIs and serves asynchronous requests at the `/api` path. The `ASPNetTest` application is a web application that serves requests at the root path.

Example `aws-windows-deployment-manifest.json` - multiple apps

```
{
  "manifestVersion": 1,
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "WebAPITest",
        "parameters": {
          "appBundle": "webapi.zip",
          "iisPath": "/api"
        }
      },
      {
        "name": "ASPNetTest",
        "parameters": {
          "appBundle": "aspnet.zip",
          "iisPath": "/"
        }
      }
    ]
  }
}
```

A sample application with multiple applications is available here:

- **Deployable source bundle** - [dotnet-multiapp-sample-bundle-v2.zip](#)
- **Source code** - [dotnet-multiapp-sample-source-v2.zip](#)

Configure application pools

You can use a deployment manifest to configure an application pool in IIS and use it to run one or more applications.

The following deployment manifest configures an application pool that restarts its applications every 10 minutes, and attaches it to a .NET Framework web application that runs at the root path.

Example `aws-windows-deployment-manifest.json` - app pool

```
{
  "manifestVersion": 1,
  "iisConfig": {
    "appPools": [
      {
        "name": "App pool",
        "recycling": {
          "regularTimeInterval": 10
        }
      }
    ]
  },
  "deployments": {
    "msDeploy": [
```

```
    {
      "name": "Web app",
      "parameters": {
        "archive": "site.zip",
        "iisPath": "/",
        "appPool": "MyPool"
      }
    }
  ]
}
```

The `appPools` block under `iisConfig` defines the application pool.

Each deployment in the `deployments` block specifies an archive, a path to run it at, and an `appPool` in which to run it.

Define custom deployments

For even more control, you can completely customize an application deployment by defining a *custom deployment*.

The following deployment manifest tells Elastic Beanstalk to run an `install` script named `siteInstall.ps1` to install the website during instance launch and deployments, run an `uninstall` script prior to installing a new version during a deployment, and a `restart` script to restart the application when you choose [Restart App Server \(p. 356\)](#) in the management console.

Example `aws-windows-deployment-manifest.json` - custom deployment

```
{
  "manifestVersion": 1,
  "deployments": {
    "custom": [
      {
        "name": "Custom site",
        "scripts": {
          "install": {
            "file": "siteInstall.ps1"
          },
          "restart": {
            "file": "siteRestart.ps1"
          },
          "uninstall": {
            "file": "siteUninstall.ps1"
          }
        }
      }
    ]
  }
}
```

Include any artifacts required to run the application in your source bundle with the manifest and scripts.

Example `Custom-site-bundle.zip`

```
.
|-- aws-windows-deployment-manifest.json
|-- siteInstall.ps1
|-- siteRestart.ps1
|-- siteUninstall.ps1
^-- site-content.zip
```

Tutorial: How to deploy a .NET sample application using Elastic Beanstalk

In this tutorial, you will learn how to deploy a .NET sample application to AWS Elastic Beanstalk using the AWS Toolkit for Visual Studio.

Note

This tutorial uses a sample ASP.NET Web application that you can download [here](#). It also uses the [Toolkit for Visual Studio](#) and was tested using Visual Studio Professional 2012.

Create the environment

First, use the Create New Application wizard in the Elastic Beanstalk console to create the application environment. For **Platform**, choose **.NET**.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

When the environment is up and running, add an Amazon RDS database instance that the application uses to store data. For **DB engine**, choose **sqlserver-ex**.

To add a DB instance to your environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Choose a DB engine, and enter a user name and password.
6. Choose **Apply**.

Publish your application to Elastic Beanstalk

Use the AWS Toolkit for Visual Studio to publish your application to Elastic Beanstalk.

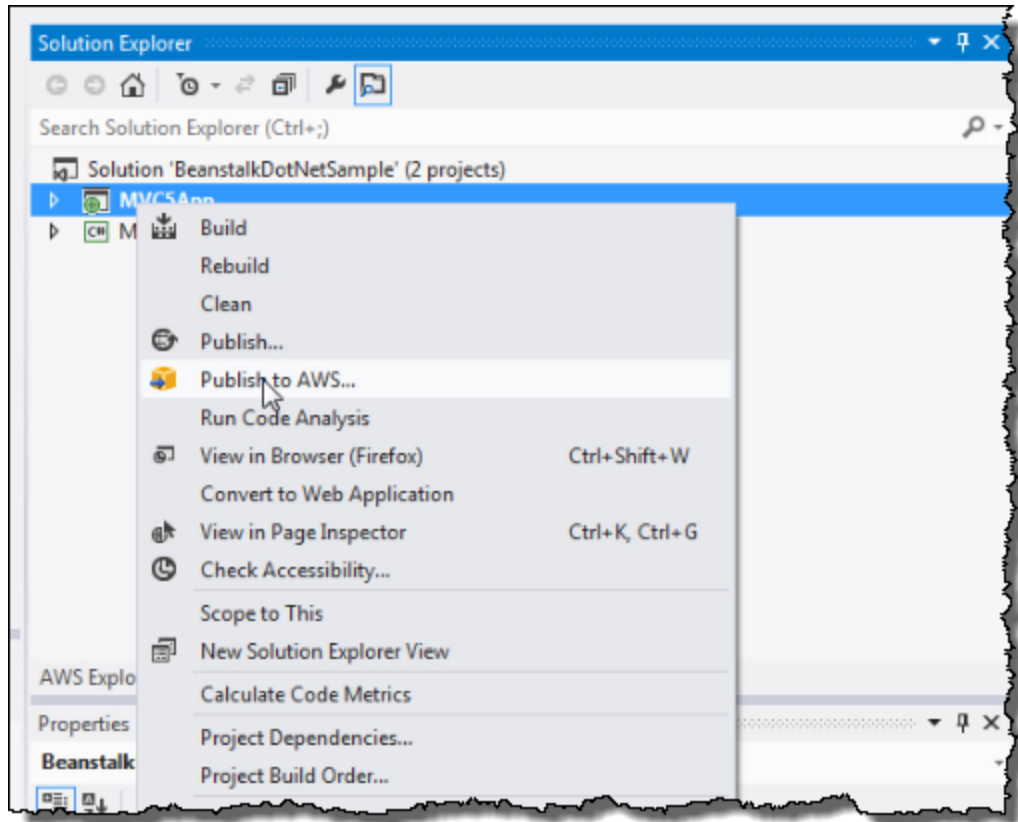
To publish your application to Elastic Beanstalk

1. Ensure that your environment launched successfully by checking the **Health** status in the Elastic Beanstalk console. It should be **Ok** (green).
2. In Visual Studio, open **BeanstalkDotNetSample.sln**.

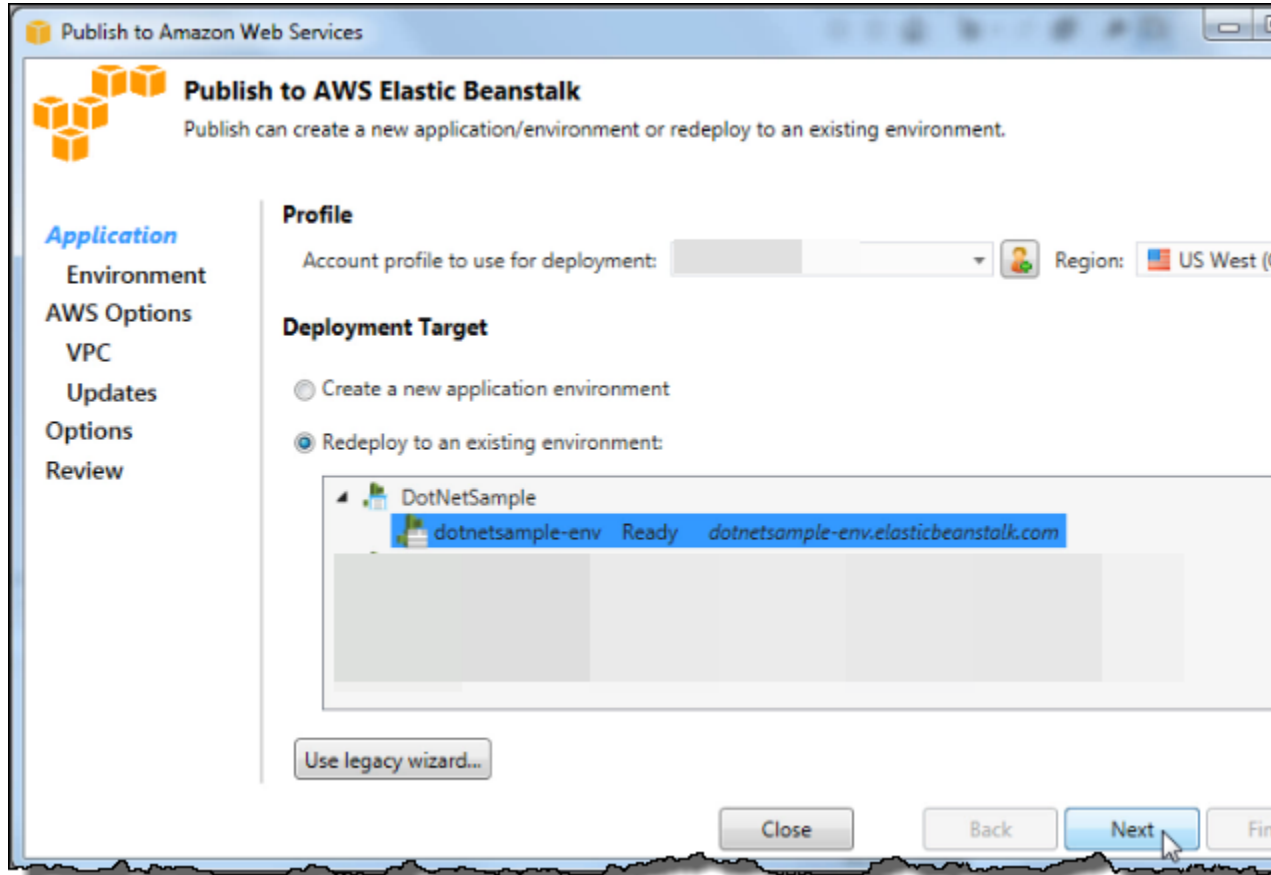
Note

If you haven't done so already, you can get the sample [here](#).

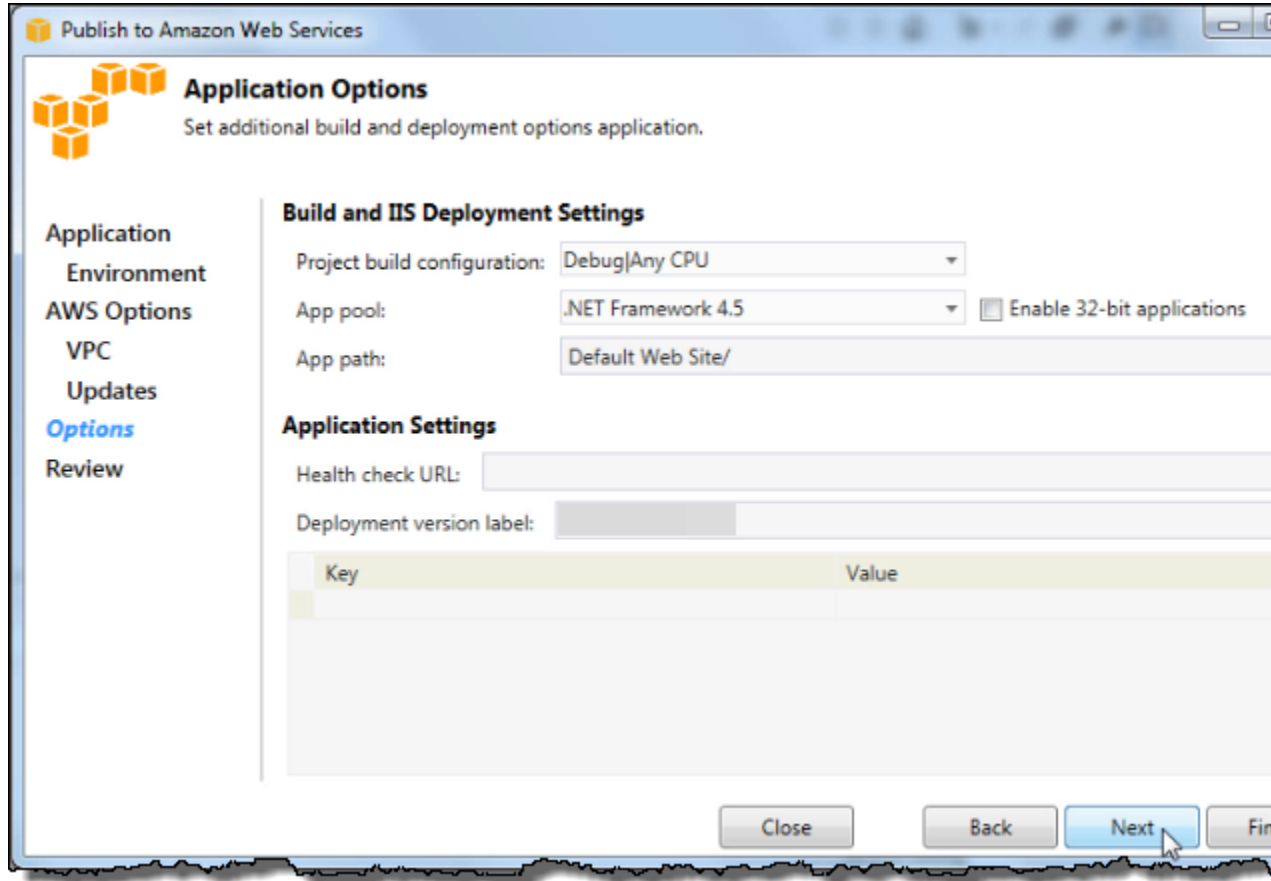
3. On the **View** menu, choose **Solution Explorer**.
4. Expand **Solution 'BeanstalkDotNetSample' (2 projects)**.
5. Open the context (right-click) menu for **MVC5App**, and then choose **Publish to AWS**.



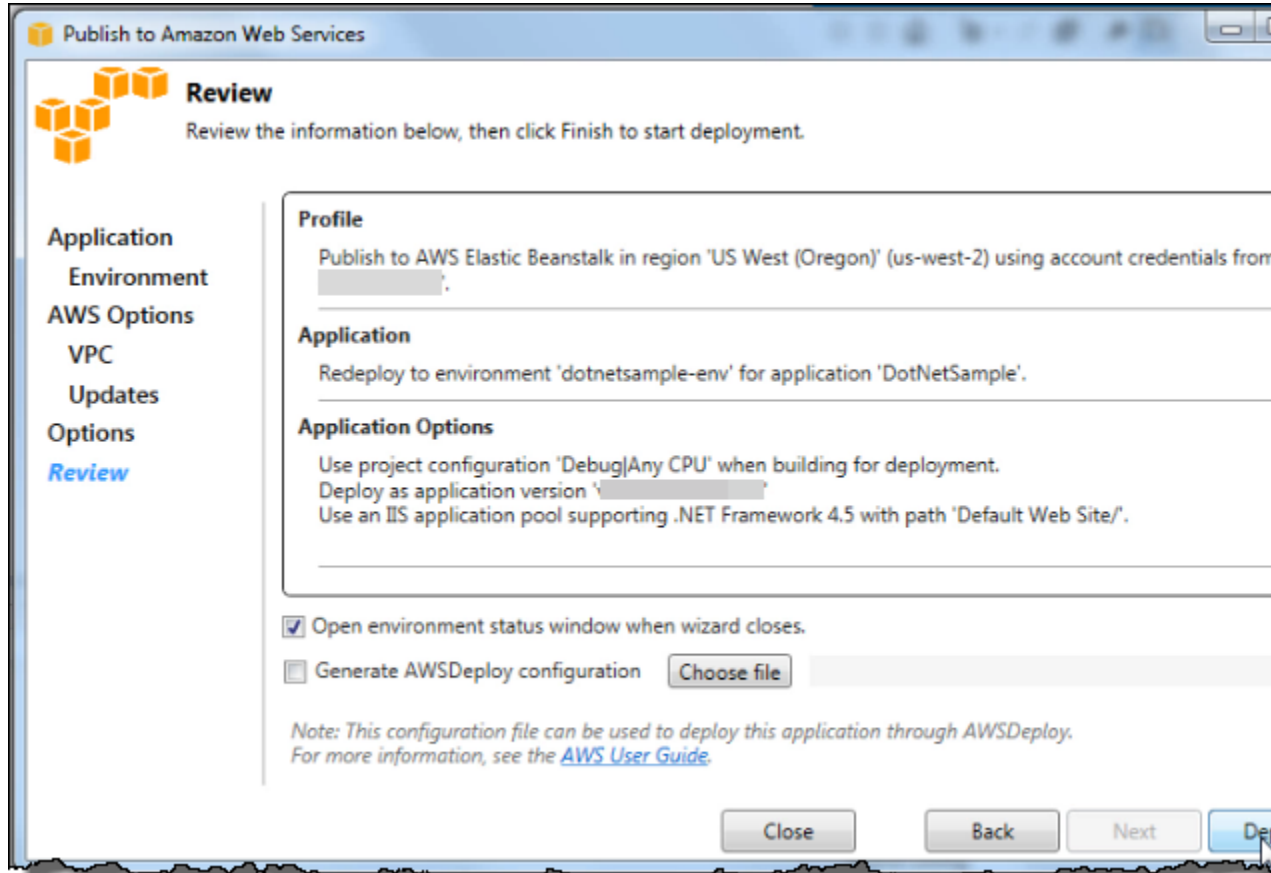
6. On the **Publish to AWS Elastic Beanstalk** page, for **Deployment Target**, choose the environment that you just created, and then choose **Next**.



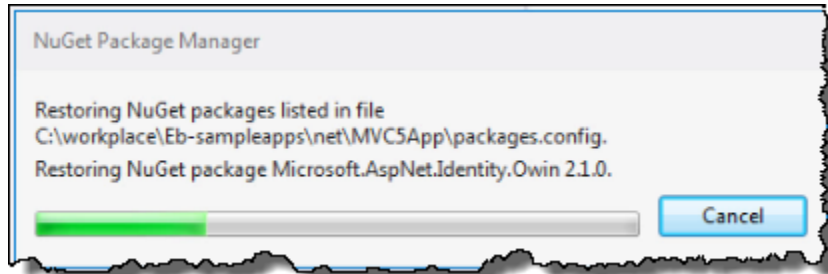
7. On the **Application Options** page, accept all of the defaults, and then choose **Next**.



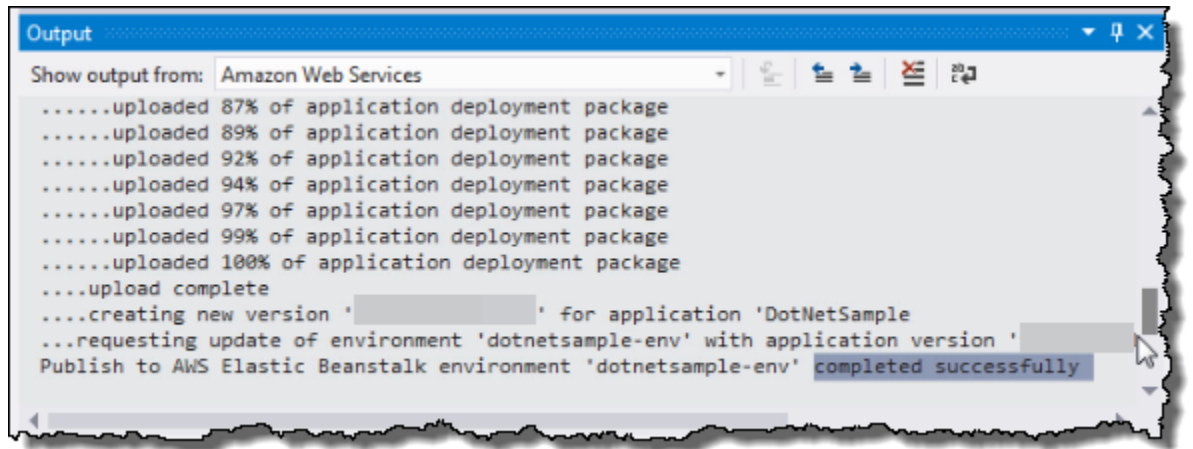
8. On the **Review** page, choose **Deploy**.



9. If you want to monitor deployment status, use the **NuGet Package Manager** in Visual Studio.



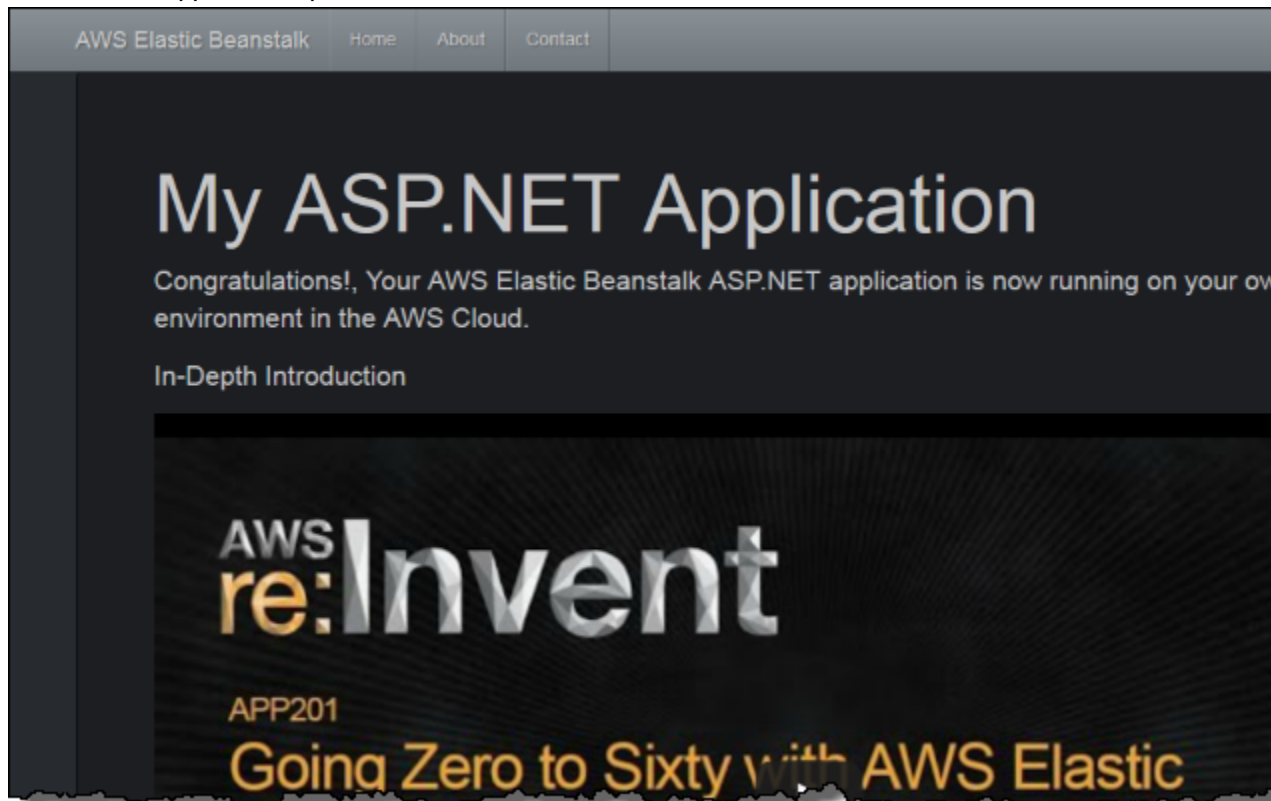
When the application has successfully been deployed, the **Output** box displays **completed successfully**.



```
Output
Show output from: Amazon Web Services
.....uploaded 87% of application deployment package
.....uploaded 89% of application deployment package
.....uploaded 92% of application deployment package
.....uploaded 94% of application deployment package
.....uploaded 97% of application deployment package
.....uploaded 99% of application deployment package
.....uploaded 100% of application deployment package
....upload complete
...creating new version '...' for application 'DotNetSample'
...requesting update of environment 'dotnetsample-env' with application version '...'
Publish to AWS Elastic Beanstalk environment 'dotnetsample-env' completed successfully
```

10. Return to the Elastic Beanstalk console. In the navigation pane, choose **Go to environment**.

Your ASP.NET application opens in a new tab.



Clean up your AWS resources

After your application has deployed successfully, learn more about Elastic Beanstalk by [watching the video](#) in the application.

If you are done working with Elastic Beanstalk for now, you can terminate your .NET environment.

To terminate your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions** and then choose **Terminate environment**.

Elastic Beanstalk cleans up all AWS resources associated with your environment, including EC2 instances, DB instance, load balancer, security groups, CloudWatch alarms, etc.

For more information, see [Creating and deploying .NET applications on Elastic Beanstalk \(p. 137\)](#), the [AWS .NET Development Blog](#), or the [AWS Application Management Blog](#).

Deploying an ASP.NET core application with Elastic Beanstalk

In this tutorial, you will walk through the process of building a new ASP.NET Core application and deploying it to AWS Elastic Beanstalk.

First, you will use the .NET Core SDK's `dotnet` command line tool to generate a basic .NET Core command line application, install dependencies, compile code, and run applications locally. Next, you will create the default `Program.cs` class, and add an ASP.NET `Startup.cs` class and configuration files to make an application that serves HTTP requests with ASP.NET and IIS.

Finally, Elastic Beanstalk uses a [deployment manifest \(p. 145\)](#) to configure deployments for .NET Core applications, custom applications, and multiple .NET Core or MSBuild applications on a single server. To deploy a .NET Core application to a Windows Server environment, you add a site archive to an application source bundle with a deployment manifest. The `dotnet publish` command generates compiled classes and dependencies that you can bundle with a `web.config` file to create a site archive. The deployment manifest tells Elastic Beanstalk the path at which the site should run and can be used to configure application pools and run multiple applications at different paths.

The application source code is available here: [dotnet-core-tutorial-source.zip](#)

The deployable source bundle is available here: [dotnet-core-tutorial-bundle.zip](#)

Sections

- [Prerequisites \(p. 155\)](#)
- [Generate a .NET core project \(p. 156\)](#)
- [Launch an Elastic Beanstalk environment \(p. 157\)](#)
- [Update the source code \(p. 157\)](#)
- [Deploy your application \(p. 162\)](#)
- [Cleanup \(p. 163\)](#)
- [Next steps \(p. 163\)](#)

Prerequisites

This tutorial uses the .NET Core SDK to generate a basic .NET Core application, run it locally, and build a deployable package.

Requirements

- .NET Core (x64) 1.0.1, 2.0.0, or later

To install the .NET core SDK

1. Download the installer from microsoft.com/net/core. Choose **Windows**. Choose **Download .NET SDK**.
2. Run the installer and follow the instructions.

This tutorial uses a command line ZIP utility to create a source bundle that you can deploy to Elastic Beanstalk. To use the `zip` command in Windows, you can install `UnxUtils`, a lightweight collection of useful command line utilities like `zip` and `ls`. Alternatively, you can [use Windows Explorer \(p. 343\)](#) or any other ZIP utility to create source bundle archives.

To install UnxUtils

1. Download [UnxUtils](#).
2. Extract the archive to a local directory. For example, `C:\Program Files (x86)`.
3. Add the path to the binaries to your Windows PATH user variable. For example, `C:\Program Files (x86)\UnxUtils\usr\local\wbin`.
 - a. Press the Windows key, and then enter **environment variables**.
 - b. Choose **Edit environment variables for your account**.
 - c. Choose **PATH**, and then choose **Edit**.
 - d. Add paths to the **Variable value** field, separated by semicolons. For example: `C:\item1\path;C:\item2\path`
 - e. Choose **OK** twice to apply the new settings.
 - f. Close any running Command Prompt windows, and then reopen a Command Prompt window.
4. Open a new command prompt window and run the `zip` command to verify that it works.

```
> zip -h
Copyright (C) 1990-1999 Info-ZIP
Type 'zip -L' for software license.
...
```

Generate a .NET core project

Use the `dotnet` command line tool to generate a new C# .NET Core project and run it locally. The default .NET Core application is a command line utility that prints `Hello World!` and then exits.

To generate a new .NET core project

1. Open a new command prompt window and navigate to your user folder.

```
> cd %USERPROFILE%
```

2. Use the `dotnet new` command to generate a new .NET Core project.

```
C:\Users\username> dotnet new console -o dotnet-core-tutorial
Content generation time: 65.0152 ms
The template "Console Application" created successfully.
```

```
C:\Users\username> cd dotnet-core-tutorial
```

3. Use the `dotnet restore` command to install dependencies.

```
C:\Users\username\dotnet-core-tutorial> dotnet restore
Restoring packages for C:\Users\username\dotnet-core-tutorial\dotnet-core-
tutorial.csproj...
Generating MSBuild file C:\Users\username\dotnet-core-tutorial\obj\dotnet-core-
tutorial.csproj.nuget.g.props.
Generating MSBuild file C:\Users\username\dotnet-core-tutorial\obj\dotnet-core-
tutorial.csproj.nuget.g.targets.
Writing lock file to disk. Path: C:\Users\username\dotnet-core-tutorial\obj
\project.assets.json
Restore completed in 1.25 sec for C:\Users\username\dotnet-core-tutorial\dotnet-core-
tutorial.csproj.

NuGet Config files used:
  C:\Users\username\AppData\Roaming\NuGet\NuGet.Config
  C:\Program Files (x86)\NuGet\Config\Microsoft.VisualStudio.Offline.config
Feeds used:
  https://api.nuget.org/v3/index.json
  C:\Program Files (x86)\Microsoft SDKs\NuGetPackages\
```

4. Use the `dotnet run` command to build and run the application locally.

```
C:\Users\username\dotnet-core-tutorial> dotnet run
Hello World!
```

Launch an Elastic Beanstalk environment

Use the Elastic Beanstalk console to launch an Elastic Beanstalk environment. For this example, you will launch with a .NET platform. After you launch and configure your environment, you can deploy new source code at any time.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link:
console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 10 minutes. During this time you can update your source code.

Update the source code

Modify the default application into a web application that uses ASP.NET and IIS.

- ASP.NET is the website framework for .NET.
- IIS is the web server that runs the application on the Amazon EC2 instances in your Elastic Beanstalk environment.

The source code examples to follow are available here: [dotnet-core-tutorial-source.zip](#)

Note

The following procedure shows how to convert the project code into a web application. To simplify the process, you can generate the project as a web application right from the start. In the previous section [Generate a .NET core project \(p. 156\)](#), modify the `dotnet new` step's command with the following command.

```
C:\Users\username> dotnet new web -o dotnet-core-tutorial
```

To add ASP.NET and IIS support to your code

1. Copy `Program.cs` to your application directory to run as a web host builder.

Example `c:\users\username\dotnet-core-tutorial\Program.cs`

```
using System;
using Microsoft.AspNetCore.Hosting;
using System.IO;

namespace aspnetcoreapp
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = new WebHostBuilder()
                .UseKestrel()
                .UseContentRoot(Directory.GetCurrentDirectory())
                .UseIISIntegration()
                .UseStartup<Startup>()
                .Build();

            host.Run();
        }
    }
}
```

2. Add `Startup.cs` to run an ASP.NET website.

Example `c:\users\username\dotnet-core-tutorial\Startup.cs`

```
using System;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;

namespace aspnetcoreapp
{
    public class Startup
    {
        public void Configure(IApplicationBuilder app)
        {
            app.Run(context =>
            {
                return context.Response.WriteAsync("Hello from ASP.NET Core!");
            });
        }
    }
}
```

3. Add the `web.config` file to configure the IIS server.

Example c:\users\username\dotnet-core-tutorial\web.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.webServer>
    <handlers>
      <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModule"
resourceType="Unspecified" />
    </handlers>
    <aspNetCore processPath="dotnet" arguments=".\\dotnet-core-
tutorial.dll" stdoutLogEnabled="false" stdoutLogFile=".\logs\stdout"
forwardWindowsAuthToken="false" />
  </system.webServer>
</configuration>
```

4. Add `dotnet-core-tutorial.csproj`, which includes IIS middleware and includes the `web.config` file from the output of `dotnet publish`.

Note

The following example was developed using .NET Core Runtime 2.2.1. You might need to modify the `TargetFramework` or the `Version` attribute values in the `PackageReference` elements to match the version of .NET Core Runtime that you are using in your custom projects.

Example c:\users\username\dotnet-core-tutorial\dotnet-core-tutorial.csproj

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp2.2</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Server.Kestrel"
Version="2.2.0" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Server.IISIntegration"
Version="2.2.0" />
  </ItemGroup>

  <ItemGroup>
    <None Include="web.config" CopyToPublishDirectory="Always" />
  </ItemGroup>

</Project>
```

Next, install the new dependencies and run the ASP.NET website locally.

To run the website locally

1. Use the `dotnet restore` command to install dependencies.
2. Use the `dotnet run` command to build and run the app locally.
3. Open localhost:5000 to view the site.

To run the application on a web server, you need to bundle the compiled source code with a `web.config` configuration file and runtime dependencies. The `dotnet` tool provides a `publish` command that gathers these files in a directory based on the configuration in `dotnet-core-tutorial.csproj`.

To build your website

- Use the `dotnet publish` command to output compiled code and dependencies to a folder named `site`.

```
C:\users\username\dotnet-core-tutorial> dotnet publish -o site
```

To deploy the application to Elastic Beanstalk, bundle the site archive with a [deployment manifest](#) (p. 145). This tells Elastic Beanstalk how to run it.

To create a source bundle

1. Add the files in the `site` folder to a ZIP archive.

Note

If you use a different ZIP utility, be sure to add all files to the root folder of the resulting ZIP archive. This is required for a successful deployment of the application to your Elastic Beanstalk environment.

```
C:\users\username\dotnet-core-tutorial> cd site
C:\users\username\dotnet-core-tutorial\site> zip ../site.zip *
  adding: dotnet-core-tutorial.deps.json (164 bytes security) (deflated 84%)
  adding: dotnet-core-tutorial.dll (164 bytes security) (deflated 59%)
  adding: dotnet-core-tutorial.pdb (164 bytes security) (deflated 28%)
  adding: dotnet-core-tutorial.runtimeconfig.json (164 bytes security) (deflated 26%)
  adding: Microsoft.AspNetCore.Authentication.Abstractions.dll (164 bytes security)
(deflated 49%)
  adding: Microsoft.AspNetCore.Authentication.Core.dll (164 bytes security) (deflated
57%)
  adding: Microsoft.AspNetCore.Connections.Abstractions.dll (164 bytes security)
(deflated 51%)
  adding: Microsoft.AspNetCore.Hosting.Abstractions.dll (164 bytes security) (deflated
49%)
  adding: Microsoft.AspNetCore.Hosting.dll (164 bytes security) (deflated 60%)
  adding: Microsoft.AspNetCore.Hosting.Server.Abstractions.dll (164 bytes security)
(deflated 44%)
  adding: Microsoft.AspNetCore.Http.Abstractions.dll (164 bytes security) (deflated
54%)
  adding: Microsoft.AspNetCore.Http.dll (164 bytes security) (deflated 55%)
  adding: Microsoft.AspNetCore.Http.Extensions.dll (164 bytes security) (deflated 50%)
  adding: Microsoft.AspNetCore.Http.Features.dll (164 bytes security) (deflated 50%)
  adding: Microsoft.AspNetCore.HttpOverrides.dll (164 bytes security) (deflated 49%)
  adding: Microsoft.AspNetCore.Server.IISIntegration.dll (164 bytes security) (deflated
46%)
  adding: Microsoft.AspNetCore.Server.Kestrel.Core.dll (164 bytes security) (deflated
63%)
  adding: Microsoft.AspNetCore.Server.Kestrel.dll (164 bytes security) (deflated 46%)
  adding: Microsoft.AspNetCore.Server.Kestrel.Https.dll (164 bytes security) (deflated
44%)
  adding: Microsoft.AspNetCore.Server.Kestrel.Transport.Abstractions.dll (164 bytes
security) (deflated 56%)
  adding: Microsoft.AspNetCore.Server.Kestrel.Transport.Sockets.dll (164 bytes
security) (deflated 51%)
  adding: Microsoft.AspNetCore.WebUtilities.dll (164 bytes security) (deflated 55%)
  adding: Microsoft.Extensions.Configuration.Abstractions.dll (164 bytes security)
(deflated 48%)
```

```
adding: Microsoft.Extensions.Configuration.Binder.dll (164 bytes security) (deflated 47%)
adding: Microsoft.Extensions.Configuration.dll (164 bytes security) (deflated 46%)
adding: Microsoft.Extensions.Configuration.EnvironmentVariables.dll (164 bytes security) (deflated 46%)
adding: Microsoft.Extensions.Configuration.FileExtensions.dll (164 bytes security) (deflated 47%)
adding: Microsoft.Extensions.DependencyInjection.Abstractions.dll (164 bytes security) (deflated 54%)
adding: Microsoft.Extensions.DependencyInjection.dll (164 bytes security) (deflated 53%)
adding: Microsoft.Extensions.FileProviders.Abstractions.dll (164 bytes security) (deflated 46%)
adding: Microsoft.Extensions.FileProviders.Physical.dll (164 bytes security) (deflated 47%)
adding: Microsoft.Extensions.FileSystemGlobbing.dll (164 bytes security) (deflated 49%)
adding: Microsoft.Extensions.Hosting.Abstractions.dll (164 bytes security) (deflated 47%)
adding: Microsoft.Extensions.Logging.Abstractions.dll (164 bytes security) (deflated 54%)
adding: Microsoft.Extensions.Logging.dll (164 bytes security) (deflated 48%)
adding: Microsoft.Extensions.ObjectPool.dll (164 bytes security) (deflated 45%)
adding: Microsoft.Extensions.Options.dll (164 bytes security) (deflated 53%)
adding: Microsoft.Extensions.Primitives.dll (164 bytes security) (deflated 50%)
adding: Microsoft.Net.Http.Headers.dll (164 bytes security) (deflated 53%)
adding: System.IO.Pipelines.dll (164 bytes security) (deflated 50%)
adding: System.Runtime.CompilerServices.Unsafe.dll (164 bytes security) (deflated 43%)
adding: System.Text.Encodings.Web.dll (164 bytes security) (deflated 57%)
adding: web.config (164 bytes security) (deflated 39%)
C:\users\username\dotnet-core-tutorial\site> cd ../
```

2. Add a deployment manifest that points to the site archive.

Example c:\users\username\dotnet-core-tutorial\aws-windows-deployment-manifest.json

```
{
  "manifestVersion": 1,
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "test-dotnet-core",
        "parameters": {
          "appBundle": "site.zip",
          "iisPath": "/",
          "iisWebSite": "Default Web Site"
        }
      }
    ]
  }
}
```

3. Use the zip command to create a source bundle named dotnet-core-tutorial.zip.

```
C:\users\username\dotnet-core-tutorial> zip dotnet-core-tutorial.zip site.zip aws-
windows-deployment-manifest.json
adding: site.zip (164 bytes security) (stored 0%)
adding: aws-windows-deployment-manifest.json (164 bytes security) (deflated 50%)
```


Deploy your application

Deploy the source bundle to the Elastic Beanstalk environment that you created.

You can download the source bundle here: [dotnet-core-tutorial-bundle.zip](#)

To deploy a source bundle

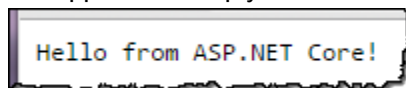
1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

The application simply writes `Hello from ASP.NET Core!` to the response and returns.



Launching an environment creates the following resources:

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or nginx as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
- **Domain name** – A domain name that routes to your web app in the form `subdomain.region.elasticbeanstalk.com`.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and isn't deleted during environment termination. For more information, see [Using Elastic Beanstalk with Amazon S3 \(p. 831\)](#).

Cleanup

When you finish working with Elastic Beanstalk, you can terminate your environment. Elastic Beanstalk terminates all AWS resources associated with your environment, such as [Amazon EC2 instances \(p. 451\)](#), [database instances \(p. 506\)](#), [load balancers \(p. 470\)](#), security groups, and [alarms \(p. 470\)](#).

To terminate your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

With Elastic Beanstalk, you can easily create a new environment for your application at any time.

Next steps

As you continue to develop your application, you'll probably want to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The [Elastic Beanstalk Command Line Interface \(p. 852\)](#) (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

If you use Visual Studio to develop your application, you can also use the AWS Toolkit for Visual Studio to deploy changed, manage your Elastic Beanstalk environments, and manage other AWS resources. See [The AWS Toolkit for Visual Studio \(p. 166\)](#) for more information.

For developing and testing, you might want to use the Elastic Beanstalk functionality for adding a managed DB instance directly to your environment. For instructions on setting up a database inside your environment, see [Adding a database to your Elastic Beanstalk environment \(p. 506\)](#).

Finally, if you plan to use your application in a production environment, [configure a custom domain name \(p. 534\)](#) for your environment and [enable HTTPS \(p. 652\)](#) for secure connections.

Adding an Amazon RDS DB instance to your .NET application environment

You can use an Amazon Relational Database Service (Amazon RDS) DB instance to store data gathered and modified by your application. The database can be attached to your environment and managed by Elastic Beanstalk, or created and managed externally.

If you are using Amazon RDS for the first time, [add a DB instance \(p. 164\)](#) to a test environment with the Elastic Beanstalk console and verify that your application can connect to it.

To connect to a database, [add the driver \(p. 165\)](#) to your application, load the driver class in your code, and [create a connection string \(p. 165\)](#) with the environment properties provided by Elastic Beanstalk. The configuration and connection code vary depending on the database engine and framework that you use.

Note

For learning purposes or test environments, you can use Elastic Beanstalk to add a DB instance. For production environments, you can create a DB instance outside of your Elastic Beanstalk environment to decouple your environment resources from your database resources. This way, when you terminate your environment, the DB instance isn't deleted. An external DB instance also lets you connect to the same database from multiple environments and perform [blue-green deployments](#). For instructions, see [Using Elastic Beanstalk with Amazon RDS \(p. 820\)](#).

Sections

- [Adding a DB instance to your environment \(p. 164\)](#)
- [Downloading a driver \(p. 165\)](#)
- [Connecting to a database \(p. 165\)](#)

Adding a DB instance to your environment

To add a DB instance to your environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Choose a DB engine, and enter a user name and password.
6. Choose **Apply**.

Adding a DB instance takes about 10 minutes. When the environment update is complete, the DB instance's hostname and other connection information are available to your application through the following environment properties:

Property name	Description	Property value
RDS_HOSTNAME	The hostname of the DB instance.	On the Connectivity & security tab on the Amazon RDS console: Endpoint .
RDS_PORT	The port on which the DB instance accepts connections. The default value varies among DB engines.	On the Connectivity & security tab on the Amazon RDS console: Port .
RDS_DB_NAME	The database name, ebddb .	On the Configuration tab on the Amazon RDS console: DB Name .
RDS_USERNAME	The username that you configured for your database.	On the Configuration tab on the Amazon RDS console: Master username .

Property name	Description	Property value
RDS_PASSWORD	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

For more information about configuring an internal DB instance, see [Adding a database to your Elastic Beanstalk environment](#) (p. 506).

Downloading a driver

Download and install the `EntityFramework` package and a database driver for your development environment with NuGet.

Common entity framework database providers for .NET

- **SQL Server** – `Microsoft.EntityFrameworkCore.SqlServer`
- **MySQL** – `Pomelo.EntityFrameworkCore.MySql`
- **PostgreSQL** – `Npgsql.EntityFrameworkCore.PostgreSQL`

Connecting to a database

Elastic Beanstalk provides connection information for attached DB instances in environment properties. Use `ConfigurationManager.AppSettings` to read the properties and configure a database connection.

Example Helpers.cs - connection string method

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Linq;
using System.Web;

namespace MVC5App.Models
{
    public class Helpers
    {
        public static string GetRDSConnectionString()
        {
            var appConfig = ConfigurationManager.AppSettings;

            string dbname = appConfig["RDS_DB_NAME"];

            if (string.IsNullOrEmpty(dbname)) return null;

            string username = appConfig["RDS_USERNAME"];
            string password = appConfig["RDS_PASSWORD"];
            string hostname = appConfig["RDS_HOSTNAME"];
            string port = appConfig["RDS_PORT"];

            return "Data Source=" + hostname + ";Initial Catalog=" + dbname + ";User ID=" +
                username + ";Password=" + password + ";";
        }
    }
}
```

Use the connection string to initialize your database context.

Example DbContext.cs

```
using System.Data.Entity;
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;

namespace MVC5App.Models
{
    public class RDSContext : DbContext
    {
        public RDSContext()
            : base(GetRDSConnectionString())
        {
        }

        public static RDSContext Create()
        {
            return new RDSContext();
        }
    }
}
```

The AWS Toolkit for Visual Studio

Visual Studio provides templates for different programming languages and application types. You can start with any of these templates. The AWS Toolkit for Visual Studio also provides three project templates that bootstrap development of your application: AWS Console Project, AWS Web Project, and AWS Empty Project. For this example, you'll create a new ASP.NET Web Application.

To create a new ASP.NET web application project

1. In Visual Studio, on the **File** menu, click **New** and then click **Project**.
2. In the **New Project** dialog box, click **Installed Templates**, click **Visual C#**, and then click **Web**. Click **ASP.NET Empty Web Application**, type a project name, and then click **OK**.

To run a project

Do one of the following:

1. Press **F5**.
2. Select **Start Debugging** from the **Debug** menu.

Test locally

Visual Studio makes it easy for you to test your application locally. To test or run ASP.NET web applications, you need a web server. Visual Studio offers several options, such as Internet Information Services (IIS), IIS Express, or the built-in Visual Studio Development Server. To learn about each of these options and to decide which one is best for you, see [Web Servers in Visual Studio for ASP.NET Web Projects](#).

Create an Elastic Beanstalk environment

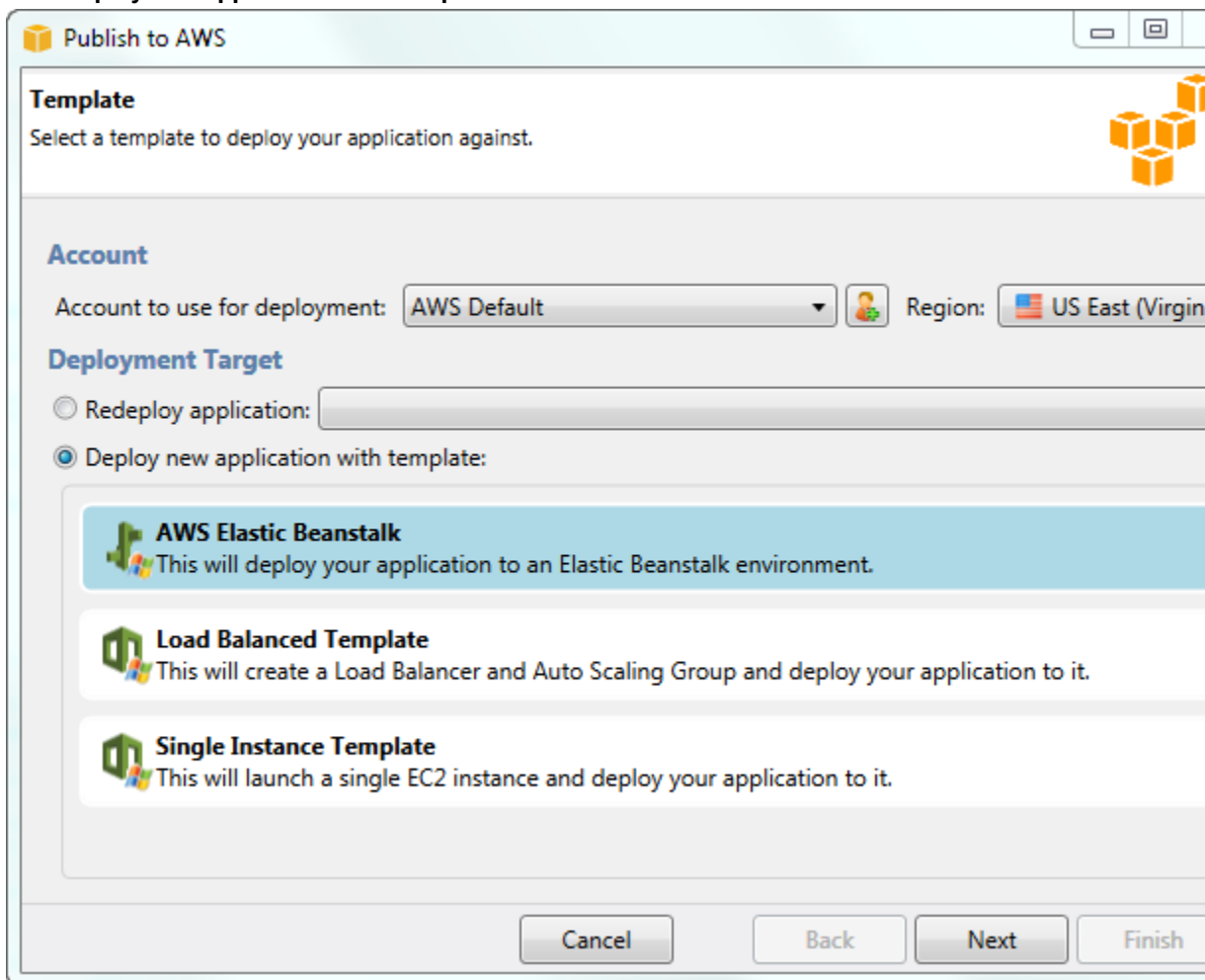
After testing your application, you are ready to deploy it to Elastic Beanstalk.

Note

[Configuration file \(p. 600\)](#) needs to be part of the project to be included in the archive. Alternatively, instead of including the configuration files in the project, you can use Visual Studio to deploy all files in the project folder. In **Solution Explorer**, right-click the project name, and then click **Properties**. Click the **Package/Publish Web** tab. In the **Items to deploy** section, select **All Files in the Project Folder** in the drop-down list.

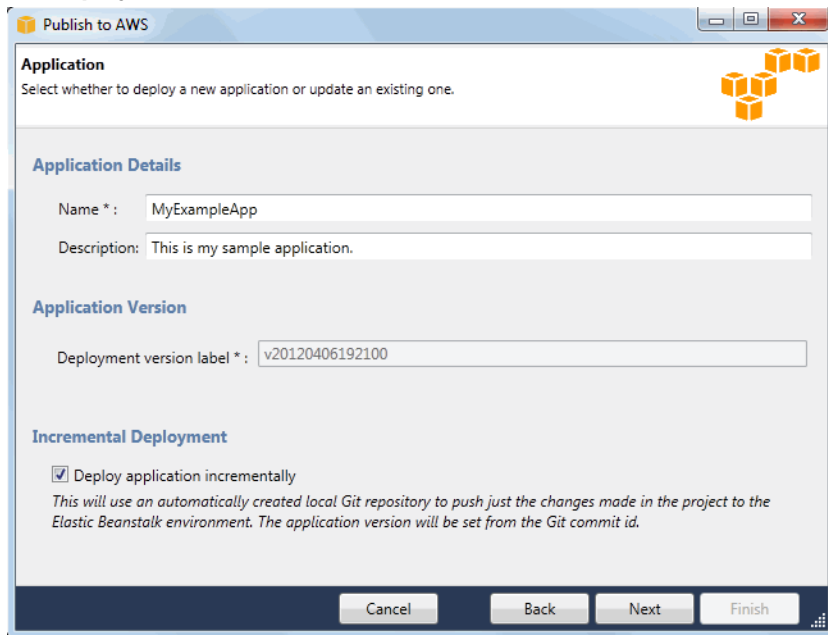
To deploy your application to Elastic Beanstalk using the AWS toolkit for Visual Studio

1. In **Solution Explorer**, right-click your application and then select **Publish to AWS**.
2. In the **Publish to AWS** wizard, enter your account information.
 - a. For **AWS account to use for deployment**, select your account or select **Other** to enter new account information.
 - b. For **Region**, select the region where you want to deploy your application. For information about available AWS Regions, see [AWS Elastic Beanstalk Endpoints and Quotas](#) in the *AWS General Reference*. If you select a region that is not supported by Elastic Beanstalk, then the option to deploy to Elastic Beanstalk will become unavailable.
 - c. Click **Deploy new application with template** and select **Elastic Beanstalk**. Then click **Next**.



3. On the **Application** page, enter your application details.
 - a. For **Name**, type the name of the application.

- b. For **Description**, type a description of the application. This step is optional.
- c. The version label of the application automatically appears in the **Deployment version label**.
- d. Select **Deploy application incrementally** to deploy only the changed files. An incremental deployment is faster because you are updating only the files that changed instead of all the files. If you choose this option, an application version will be set from the Git commit ID. If you choose to not deploy your application incrementally, then you can update the version label in the **Deployment version label** box.



- e. Click **Next**.
4. On the **Environment** page, describe your environment details.
- a. Select **Create a new environment for this application**.
 - b. For **Name**, type a name for your environment.
 - c. For **Description**, characterize your environment. This step is optional.
 - d. Select the **Type** of environment that you want.

You can select either **Load balanced, auto scaled** or a **Single instance** environment. For more information, see [Environment types \(p. 435\)](#).

Note

For single-instance environments, load balancing, autoscaling, and the health check URL settings don't apply.

- e. The environment URL automatically appears in the **Environment URL** once you move your cursor to that box.
- f. Click **Check availability** to make sure the environment URL is available.

Publish to AWS

Environment
Select or define an environment in which the application will run.

Create a new environment for the application:

Name * : MyAppEnvironment

Description: |

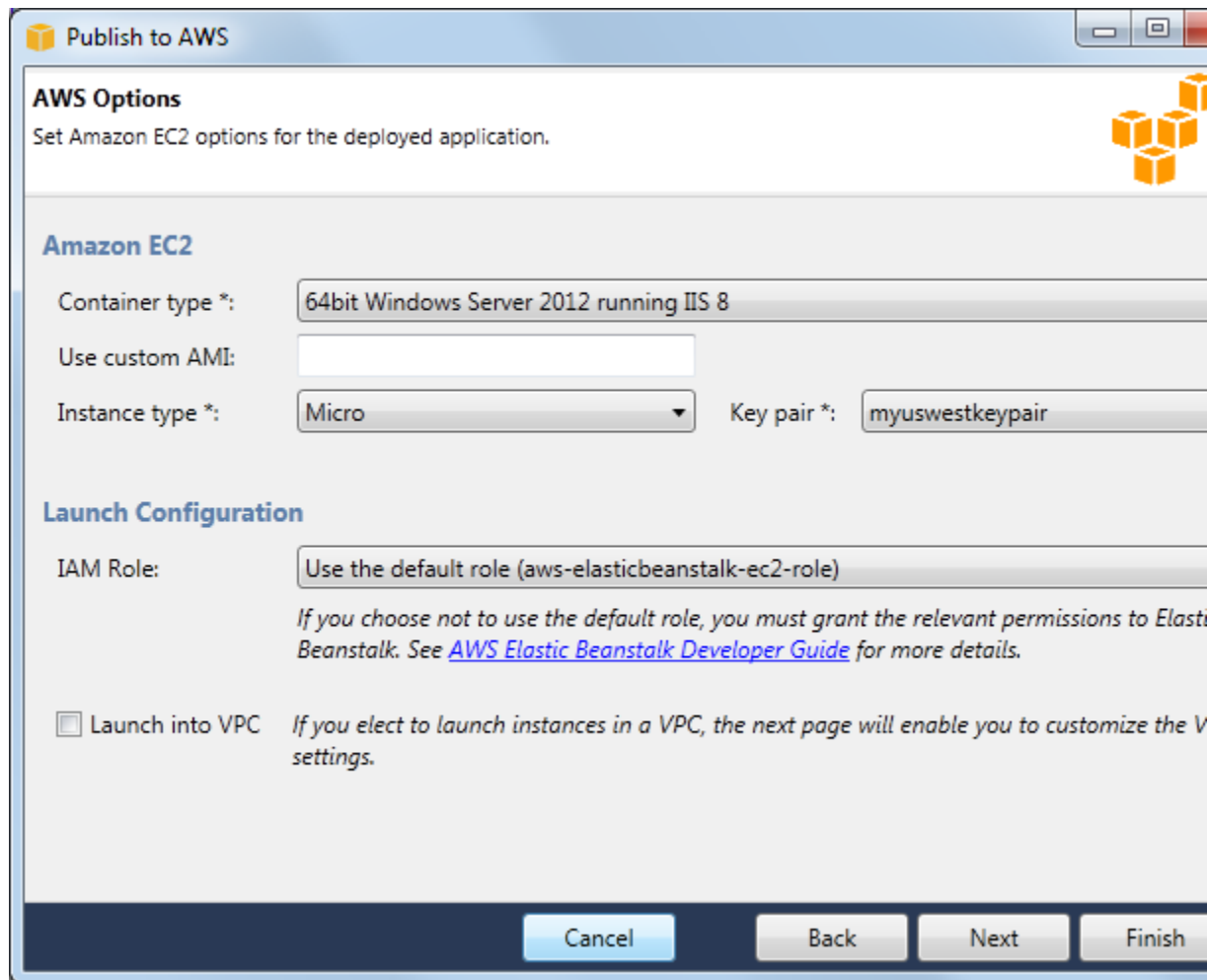
Type: Load balanced, auto scaled ▾

Environment URL:
http:// MyAppEnvironment .elasticbeanstalk.com

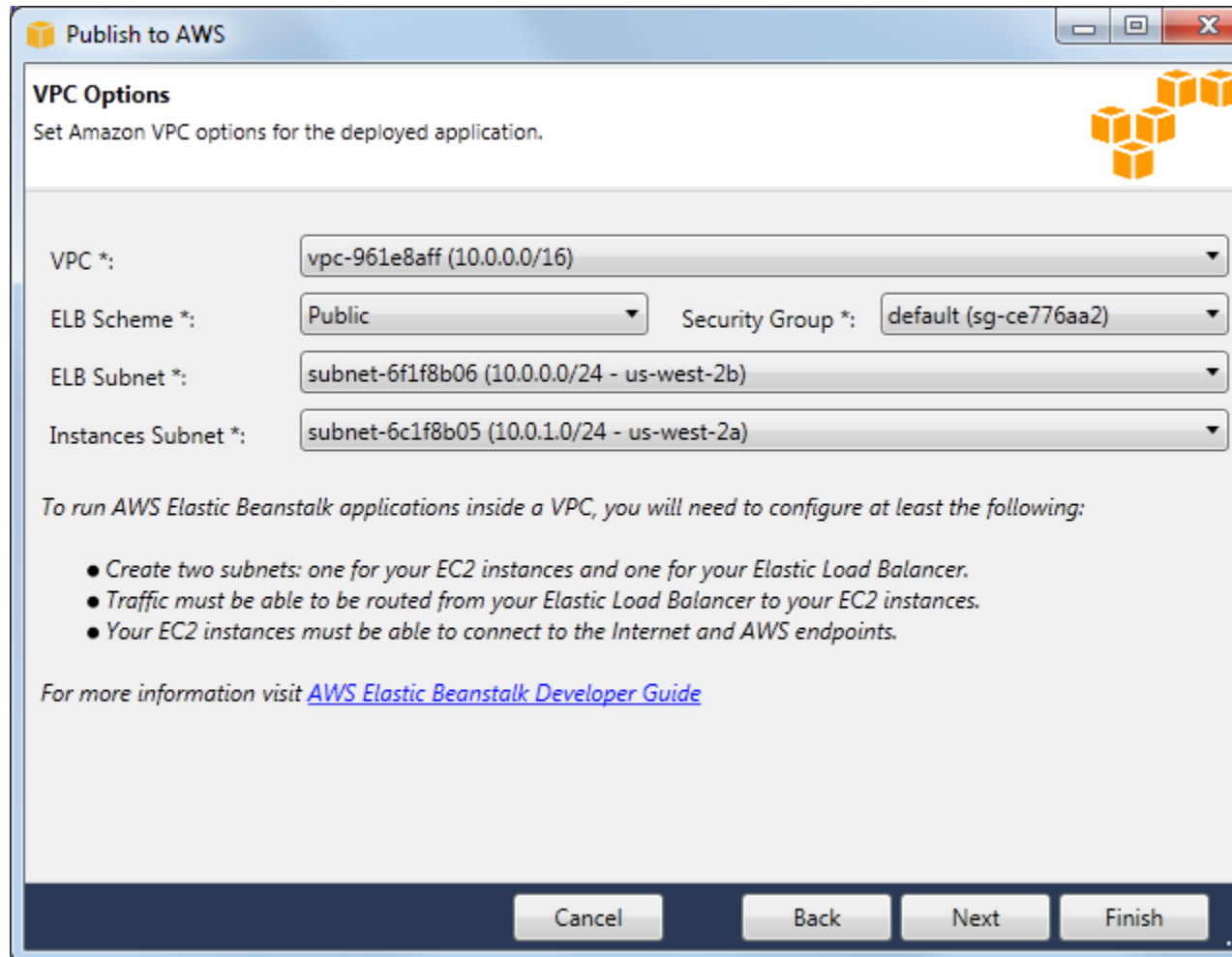
Use an existing environment: ▾

- g. Click **Next**.
5. On the **AWS Options** page, configure additional options and security information for your deployment.
 - a. For **Container Type**, select **64bit Windows Server 2012 running IIS 8** or **64bit Windows Server 2008 running IIS 7.5**.
 - b. For **Instance Type**, select **Micro**.
 - c. For **Key pair**, select **Create new key pair**. Type a name for the new key pair—in this example, we use **myuswestkeypair**—and then click **OK**. A key pair enables remote-desktop access to your Amazon EC2 instances. For more information on Amazon EC2 key pairs, see [Using Credentials](#) in the *Amazon Elastic Compute Cloud User Guide*.
 - d. Select an instance profile.

If you do not have an instance profile, select **Create a default instance profile**. For information about using instance profiles with Elastic Beanstalk, see [Managing Elastic Beanstalk instance profiles](#) (p. 760).
 - e. If you have a custom VPC that you would like to use with your environment, click **Launch into VPC**. You can configure the VPC information on the next page. For more information about Amazon VPC, go to [Amazon Virtual Private Cloud \(Amazon VPC\)](#). For a list of supported nonlegacy container types, see [the section called “Why are some platform versions marked legacy?”](#) (p. 426)



- f. Click **Next**.
6. If you selected to launch your environment inside a VPC, the **VPC Options** page appears; otherwise, the **Additional Options** page appears. Here you'll configure your VPC options.



Publish to AWS

VPC Options
Set Amazon VPC options for the deployed application.

VPC *: vpc-961e8aff (10.0.0.0/16)

ELB Scheme *: Public Security Group *: default (sg-ce776aa2)

ELB Subnet *: subnet-6f1f8b06 (10.0.0.0/24 - us-west-2b)

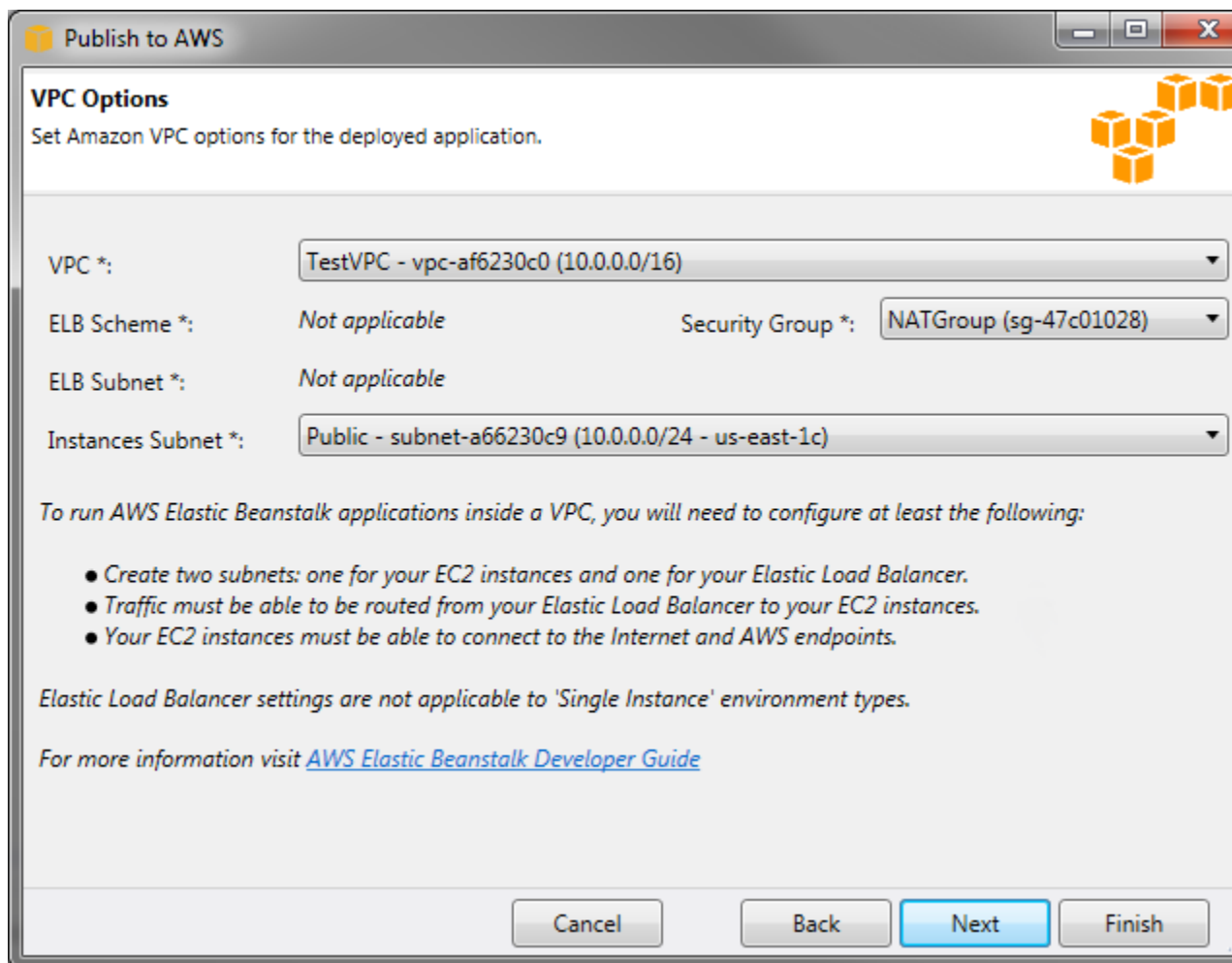
Instances Subnet *: subnet-6c1f8b05 (10.0.1.0/24 - us-west-2a)

To run AWS Elastic Beanstalk applications inside a VPC, you will need to configure at least the following:

- Create two subnets: one for your EC2 instances and one for your Elastic Load Balancer.
- Traffic must be able to be routed from your Elastic Load Balancer to your EC2 instances.
- Your EC2 instances must be able to connect to the Internet and AWS endpoints.

For more information visit [AWS Elastic Beanstalk Developer Guide](#)

Cancel Back Next Finish



- a. Select the VPC ID of the VPC in which you would like to launch your environment.
- b. For a load balanced, autoscaled environment, select **private** for **ELB Scheme** if you do not want your elastic load balancer to be available to the Internet.

For a single-instance environment, this option is not applicable because the environment doesn't have a load balancer. For more information, see [Environment types \(p. 435\)](#).

- c. For a load balanced, autoscaled environment, select the subnets for the elastic load balancer and the EC2 instances. If you created public and private subnets, make sure the elastic load balancer and the EC2 instances are associated with the correct subnet. By default, Amazon VPC creates a default public subnet using 10.0.0.0/24 and a private subnet using 10.0.1.0/24. You can view your existing subnets in the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.

For a single-instance environment, your VPC only needs a public subnet for the instance. Selecting a subnet for the load balancer is not applicable because the environment doesn't have a load balancer. For more information, see [Environment types \(p. 435\)](#).

- d. For a load balanced, autoscaled environment, select the security group you created for your instances, if applicable.

For a single-instance environment, you don't need a NAT device. Select the default security group. Elastic Beanstalk assigns an Elastic IP address to the instance that lets the instance access the Internet.

- e. Click **Next**.
7. On the **Application Options** page, configure your application options.
 - a. For Target framework, select **.NET Framework 4.0**.
 - b. Elastic Load Balancing uses a health check to determine whether the Amazon EC2 instances running your application are healthy. The health check determines an instance's health status by probing a specified URL at a set interval. You can override the default URL to match an existing resource in your application (e.g., `/myapp/index.aspx`) by entering it in the **Application health check URL** box. For more information about application health checks, see [Health check](#) (p. 477).
 - c. Type an email address if you want to receive Amazon Simple Notification Service (Amazon SNS) notifications of important events affecting your application.
 - d. The **Application Environment** section lets you specify environment variables on the Amazon EC2 instances that are running your application. This setting enables greater portability by eliminating the need to recompile your source code as you move between environments.
 - e. Select the application credentials option you want to use to deploy your application.

Publish to AWS

Application Options
Set additional options and credentials for the deployed application.

Application Pool Options

Target framework: .NET Framework 4.0 Enable 32-bit applications

Miscellaneous

Application health check URL *: / Email address for notifications:

Application Environment

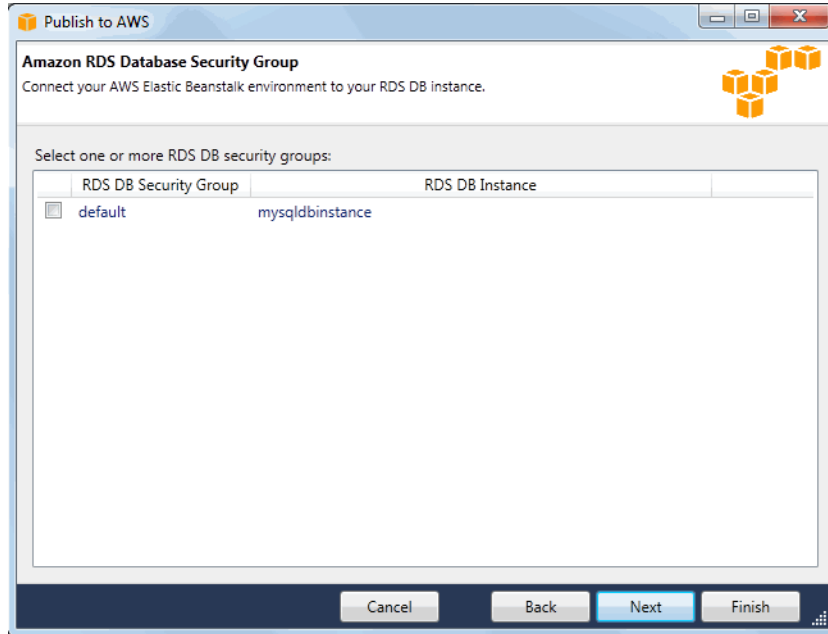
PARAM1
PARAM2
PARAM3
PARAM4
PARAM5

Application Credentials

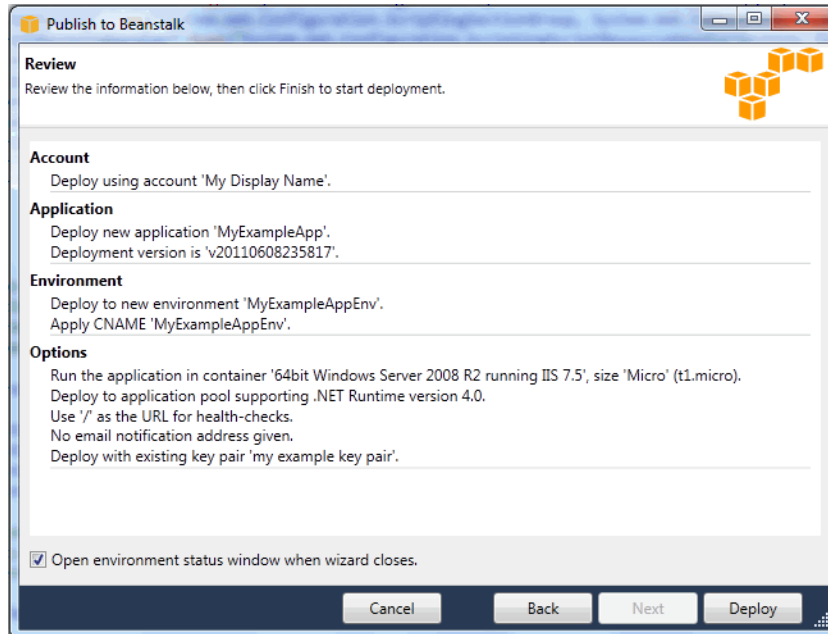
No credentials are required
 Use these credentials:
Access Key:
Secret Key:
 Use credentials for 'My Display Name'
 Use an IAM user:

Cancel Back Next Finish

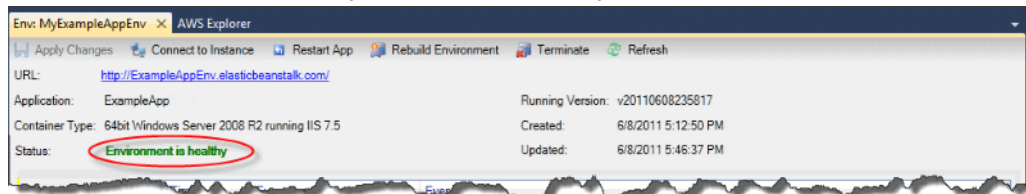
- f. Click **Next**.
8. If you have previously set up an Amazon RDS database, the **Amazon RDS DB Security Group** page appears. If you want to connect your Elastic Beanstalk environment to your Amazon RDS DB Instance, then select one or more security groups. Otherwise, go on to the next step. When you're ready, click **Next**.



9. Review your deployment options. If everything is as you want, click **Deploy**.



Your ASP.NET project will be exported as a web deploy file, uploaded to Amazon S3, and registered as a new application version with Elastic Beanstalk. The Elastic Beanstalk deployment feature will monitor your environment until it becomes available with the newly deployed code. On the env:<environment name> tab, you will see status for your environment.



Terminating an environment

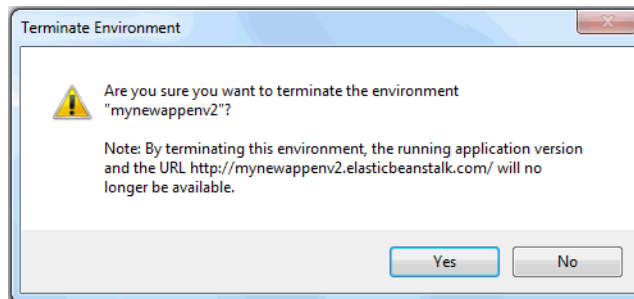
To avoid incurring charges for unused AWS resources, you can terminate a running environment using the AWS Toolkit for Visual Studio.

Note

You can always launch a new environment using the same version later.

To terminate an environment

1. Expand the Elastic Beanstalk node and the application node in **AWS Explorer**. Right-click your application environment and select **Terminate Environment**.
2. When prompted, click **Yes** to confirm that you want to terminate the environment. It will take a few minutes for Elastic Beanstalk to terminate the AWS resources running in the environment.



Note

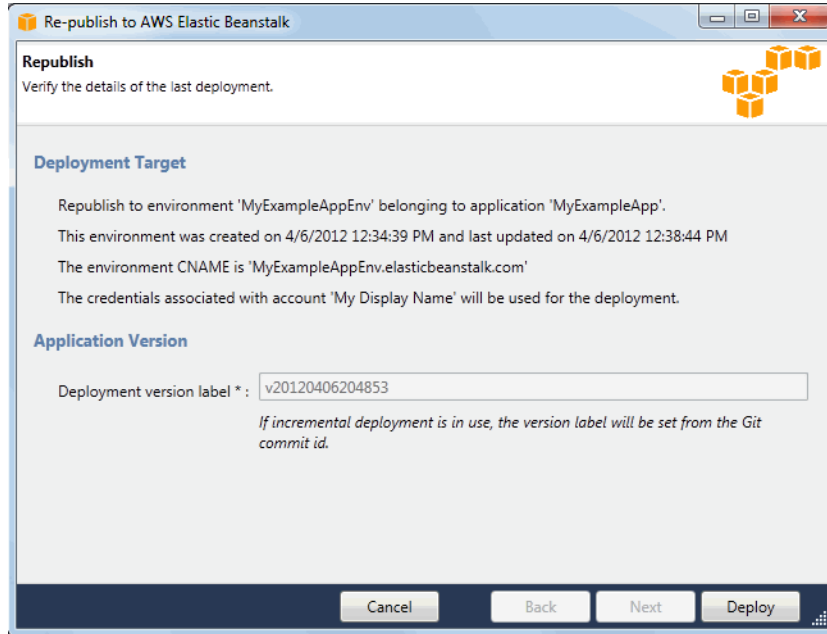
When you terminate your environment, the CNAME associated with the terminated environment becomes available for anyone to use.

Deploying to your environment

Now that you have tested your application, it is easy to edit and redeploy your application and see the results in moments.

To edit and redeploy your ASP.NET web application

1. In **Solution Explorer**, right-click your application, and then click **Republish to Environment <your environment name>**. The **Re-publish to AWS Elastic Beanstalk** wizard opens.



2. Review your deployment details and click **Deploy**.

Note

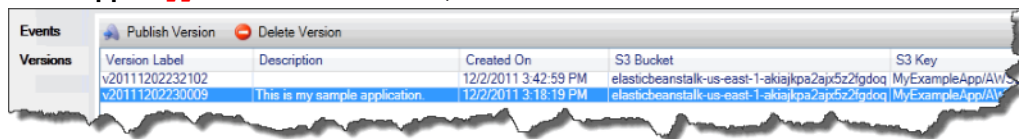
If you want to change any of your settings, you can click **Cancel** and use the **Publish to AWS** wizard instead. For instructions, see [Create an Elastic Beanstalk environment \(p. 166\)](#).

Your updated ASP.NET web project will be exported as a web deploy file with the new version label, uploaded to Amazon S3, and registered as a new application version with Elastic Beanstalk. The Elastic Beanstalk deployment feature monitors your existing environment until it becomes available with the newly deployed code. On the **env:<environment name>** tab, you will see the status of your environment.

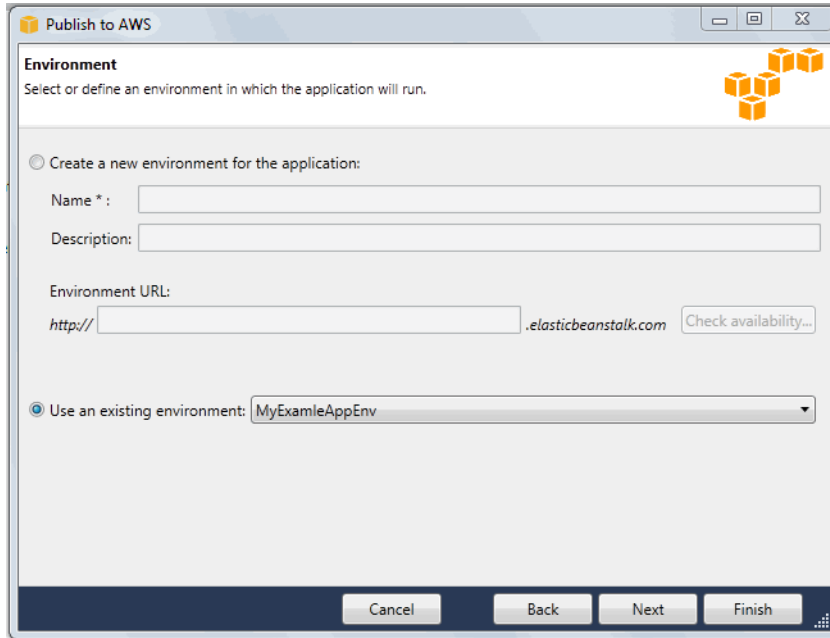
You can also deploy an existing application to an existing environment if, for instance, you need to roll back to a previous application version.

To deploy an application version to an existing environment

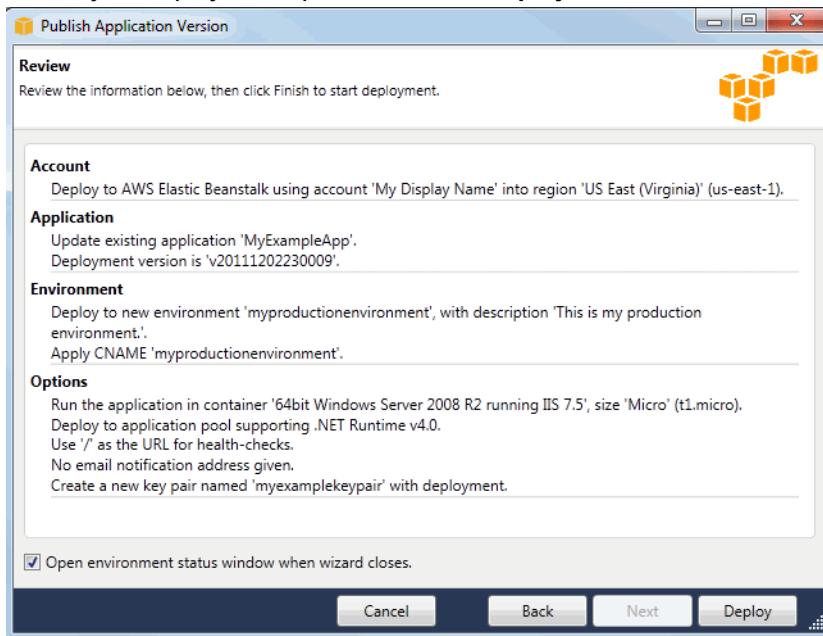
1. Right-click your Elastic Beanstalk application by expanding the Elastic Beanstalk node in **AWS Explorer**. Select **View Status**.
2. In the **App: <application name>** tab, click **Versions**.



3. Click the application version you want to deploy and click **Publish Version**.
4. In the **Publish Application Version** wizard, click **Next**.



5. Review your deployment options, and click **Deploy**.



Your ASP.NET project will be exported as a web deploy file and uploaded to Amazon S3. The Elastic Beanstalk deployment feature will monitor your environment until it becomes available with the newly deployed code. On the `env:<environment name>` tab, you will see status for your environment.

Managing your Elastic Beanstalk application environments

With the AWS Toolkit for Visual Studio and the AWS Management Console, you can change the provisioning and configuration of the AWS resources used by your application environments. For information on how to manage your application environments using the AWS Management Console, see

[Managing environments \(p. 353\)](#). This section discusses the specific service settings you can edit in the AWS Toolkit for Visual Studio as part of your application environment configuration.

Changing environment configurations settings

When you deploy your application, Elastic Beanstalk configures a number of AWS cloud computing services. You can control how these individual services are configured using the AWS Toolkit for Visual Studio.

To edit an application's environment settings

- Expand the Elastic Beanstalk node and your application node. Then right-click your Elastic Beanstalk environment in **AWS Explorer**. Select **View Status**.

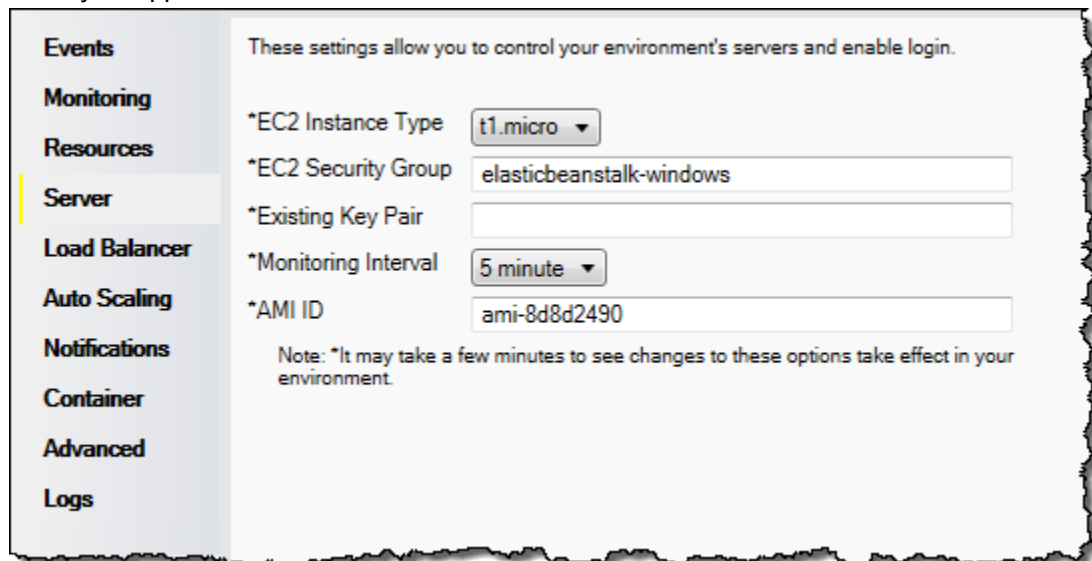
You can now configure settings for the following:

- Server
- Load balancing
- Autoscaling
- Notifications
- Environment properties

Configuring EC2 server instances using the AWS toolkit for Visual Studio

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that you use to launch and manage server instances in Amazon's data centers. You can use Amazon EC2 server instances at any time, for as long as you need, and for any legal purpose. Instances are available in different sizes and configurations. For more information, go to [Amazon EC2](#).

You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration with the **Server** tab inside your application environment tab in the AWS Toolkit for Visual Studio.



Events

Monitoring

Resources

Server

Load Balancer

Auto Scaling

Notifications

Container

Advanced

Logs

These settings allow you to control your environment's servers and enable login.

*EC2 Instance Type

*EC2 Security Group

*Existing Key Pair

*Monitoring Interval

*AMI ID

Note: *It may take a few minutes to see changes to these options take effect in your environment.

Amazon EC2 instance types

Instance type displays the instance types available to your Elastic Beanstalk application. Change the instance type to select a server with the characteristics (including memory size and CPU power) that

are most appropriate to your application. For example, applications with intensive and long-running operations can require more CPU or memory.

For more information about the Amazon EC2 instance types available for your Elastic Beanstalk application, see [Instance Types](#) in the *Amazon Elastic Compute Cloud User Guide*.

Amazon EC2 security groups

You can control access to your Elastic Beanstalk application using an *Amazon EC2 Security Group*. A security group defines firewall rules for your instances. These rules specify which ingress (i.e., incoming) network traffic should be delivered to your instance. All other ingress traffic will be discarded. You can modify rules for a group at any time. The new rules are automatically enforced for all running instances and instances launched in the future.

You can set up your Amazon EC2 security groups using the AWS Management Console or by using the AWS Toolkit for Visual Studio. You can specify which Amazon EC2 Security Groups control access to your Elastic Beanstalk application by entering the names of one or more Amazon EC2 security group names (delimited by commas) into the **EC2 Security Groups** text box.

Note

Make sure port 80 (HTTP) is accessible from 0.0.0.0/0 as the source CIDR range if you want to enable health checks for your application. For more information about health checks, see [Health checks \(p. 182\)](#).

To create a security group using the AWS toolkit for Visual Studio

1. In Visual Studio, in **AWS Explorer**, expand the **Amazon EC2** node, and then double-click **Security Groups**.
2. Click **Create Security Group**, and enter a name and description for your security group.
3. Click **OK**.

For more information on Amazon EC2 Security Groups, see [Using Security Groups](#) in the *Amazon Elastic Compute Cloud User Guide*.

Amazon EC2 key pairs

You can securely log in to the Amazon EC2 instances provisioned for your Elastic Beanstalk application with an Amazon EC2 key pair.

Important

You must create an Amazon EC2 key pair and configure your Elastic Beanstalk–provisioned Amazon EC2 instances to use the Amazon EC2 key pair before you can access your Elastic Beanstalk–provisioned Amazon EC2 instances. You can create your key pair using the **Publish to AWS** wizard inside the AWS Toolkit for Visual Studio when you deploy your application to Elastic Beanstalk. If you want to create additional key pairs using the Toolkit, follow the steps below. Alternatively, you can set up your Amazon EC2 key pairs using the [AWS Management Console](#). For instructions on creating a key pair for Amazon EC2, see the [Amazon Elastic Compute Cloud Getting Started Guide](#).

The **Existing Key Pair** text box lets you specify the name of an Amazon EC2 key pair you can use to securely log in to the Amazon EC2 instances running your Elastic Beanstalk application.

To specify the name of an Amazon EC2 key pair

1. Expand the **Amazon EC2** node and double-click **Key Pairs**.
2. Click **Create Key Pair** and enter the key pair name.
3. Click **OK**.

For more information about Amazon EC2 key pairs, go to [Using Amazon EC2 Credentials](#) in the *Amazon Elastic Compute Cloud User Guide*. For more information about connecting to Amazon EC2 instances, see [Listing and connecting to server instances \(p. 188\)](#).

Monitoring interval

By default, only basic Amazon CloudWatch metrics are enabled. They return data in five-minute periods. You can enable more granular one-minute CloudWatch metrics by selecting **1 minute** for the **Monitoring Interval** in the **Server** section of the **Configuration** tab for your environment in the AWS Toolkit for Eclipse.

Note

Amazon CloudWatch service charges can apply for one-minute interval metrics. See [Amazon CloudWatch](#) for more information.

Custom AMI ID

You can override the default AMI used for your Amazon EC2 instances with your own custom AMI by entering the identifier of your custom AMI into the **Custom AMI ID** box in the **Server** section of the **Configuration** tab for your environment in the AWS Toolkit for Eclipse.

Important

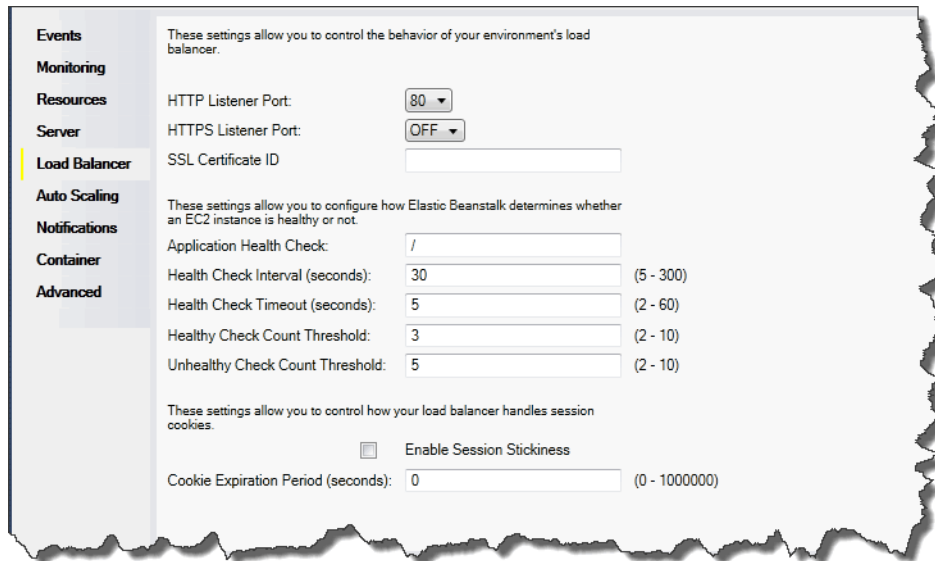
Using your own AMI is an advanced task that you should do with care. If you need a custom AMI, we recommend you start with the default Elastic Beanstalk AMI and then modify it. To be considered healthy, Elastic Beanstalk expects Amazon EC2 instances to meet a set of requirements, including having a running host manager. If these requirements are not met, your environment might not work properly.

Configuring Elastic Load Balancing using the AWS toolkit for Visual Studio

Elastic Load Balancing is an Amazon web service that helps you improve the availability and scalability of your application. This service makes it easy for you to distribute application loads between two or more Amazon EC2 instances. Elastic Load Balancing enables availability through redundancy and supports traffic growth for your application.

Elastic Load Balancing lets you automatically distribute and balance the incoming application traffic among all the instances you are running. The service also makes it easy to add new instances when you need to increase the capacity of your application.

Elastic Beanstalk automatically provisions Elastic Load Balancing when you deploy an application. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration with the **Load Balancer** tab inside your application environment tab in AWS Toolkit for Visual Studio.



The following sections describe the Elastic Load Balancing parameters you can configure for your application.

Ports

The load balancer provisioned to handle requests for your Elastic Beanstalk application sends requests to the Amazon EC2 instances that are running your application. The provisioned load balancer can listen for requests on HTTP and HTTPS ports and route requests to the Amazon EC2 instances in your AWS Elastic Beanstalk application. By default, the load balancer handles requests on the HTTP port. At least one of the ports (either HTTP or HTTPS) must be turned on.



Important

Make sure that the port you specified is not locked down; otherwise, users will not be able to connect to your Elastic Beanstalk application.

Controlling the HTTP port

To turn off the HTTP port, select **OFF** for **HTTP Listener Port**. To turn on the HTTP port, you select an HTTP port (for example, **80**) from the list.

Note

To access your environment using a port other than the default port 80, such as port 8080, add a listener to the existing load balancer and configure the new listener to listen on that port. For example, using the [AWS CLI for Classic load balancers](#), type the following command, replacing *LOAD_BALANCER_NAME* with the name of your load balancer for Elastic Beanstalk.

```
aws elb create-load-balancer-listeners --load-balancer-name LOAD_BALANCER_NAME
--listeners "Protocol=HTTP, LoadBalancerPort=8080, InstanceProtocol=HTTP,
InstancePort=80"
```

For example, using the [AWS CLI for Application Load Balancers](#), type the following command, replacing *LOAD_BALANCER_ARN* with the ARN of your load balancer for Elastic Beanstalk.

```
aws elbv2 create-listener --load-balancer-arn LOAD_BALANCER_ARN --protocol HTTP --port 8080
```

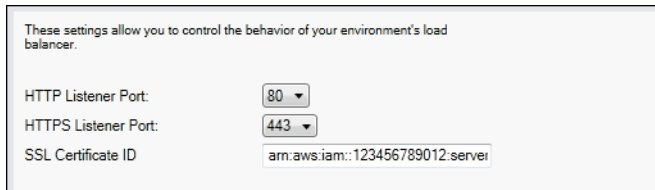
If you want Elastic Beanstalk to monitor your environment, do not remove the listener on port 80.

Controlling the HTTPS port

Elastic Load Balancing supports the HTTPS/TLS protocol to enable traffic encryption for client connections to the load balancer. Connections from the load balancer to the EC2 instances use plaintext encryption. By default, the HTTPS port is turned off.

To turn on the HTTPS port

1. Create a new certificate using AWS Certificate Manager (ACM) or upload a certificate and key to AWS Identity and Access Management (IAM). For more information about requesting an ACM certificate, see [Request a Certificate](#) in the *AWS Certificate Manager User Guide*. For more information about importing third-party certificates into ACM, see [Importing Certificates](#) in the *AWS Certificate Manager User Guide*. If ACM is not [available in your region](#), use AWS Identity and Access Management (IAM) to upload a third-party certificate. The ACM and IAM services store the certificate and provide an Amazon Resource Name (ARN) for the SSL certificate. For more information about creating and uploading certificates to IAM, see [Working with Server Certificates](#) in *IAM User Guide*.
2. Specify the HTTPS port by selecting a port for **HTTPS Listener Port**.



These settings allow you to control the behavior of your environment's load balancer.

HTTP Listener Port:	80
HTTPS Listener Port:	443
SSL Certificate ID:	arn:aws:iam::123456789012:server-

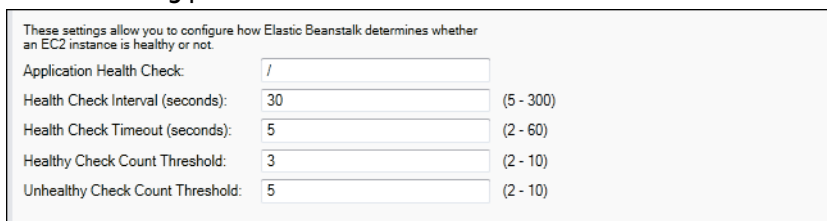
3. For **SSL Certificate ID**, enter the Amazon Resource Name (ARN) of your SSL certificate. For example, `arn:aws:iam::123456789012:server-certificate/abc/certs/build` or `arn:aws:acm:us-east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678`. Use the SSL certificate that you created or uploaded in step 1.

To turn off the HTTPS port, select **OFF** for **HTTPS Listener Port**.

Health checks

The health check definition includes a URL to be queried for instance health. By default, Elastic Beanstalk uses TCP:80 for nonlegacy containers and HTTP:80 for legacy containers. You can override the default URL to match an existing resource in your application (e.g., `/myapp/default.aspx`) by entering it in the **Application Health Check URL** box. If you override the default URL, then Elastic Beanstalk uses HTTP to query the resource. To check if you are using a legacy container type, see [the section called "Why are some platform versions marked legacy?"](#) (p. 426)

You can control the settings for the health check using the **EC2 Instance Health Check** section of the **Load Balancing** panel.



These settings allow you to configure how Elastic Beanstalk determines whether an EC2 instance is healthy or not.

Application Health Check:	/	
Health Check Interval (seconds):	30	(5 - 300)
Health Check Timeout (seconds):	5	(2 - 60)
Healthy Check Count Threshold:	3	(2 - 10)
Unhealthy Check Count Threshold:	5	(2 - 10)

The health check definition includes a URL to be queried for instance health. Override the default URL to match an existing resource in your application (e.g., `/myapp/index.jsp`) by entering it in the **Application Health Check URL** box.

The following list describes the health check parameters you can set for your application.

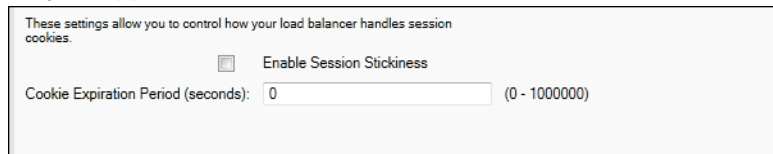
- For **Health Check Interval (seconds)**, enter the number of seconds Elastic Load Balancing waits between health checks for your application's Amazon EC2 instances.
- For **Health Check Timeout (seconds)**, specify the number of seconds Elastic Load Balancing waits for a response before it considers the instance unresponsive.
- For **Healthy Check Count Threshold** and **Unhealthy Check Count Threshold**, specify the number of consecutive successful or unsuccessful URL probes before Elastic Load Balancing changes the instance health status. For example, specifying **5** for **Unhealthy Check Count Threshold** means that the URL would have to return an error message or timeout five consecutive times before Elastic Load Balancing considers the health check failed.

Sessions

By default, a load balancer routes each request independently to the server instance with the smallest load. By comparison, a sticky session binds a user's session to a specific server instance so that all requests coming from the user during the session are sent to the same server instance.

Elastic Beanstalk uses load balancer-generated HTTP cookies when sticky sessions are enabled for an application. The load balancer uses a special load balancer-generated cookie to track the application instance for each request. When the load balancer receives a request, it first checks to see if this cookie is present in the request. If so, the request is sent to the application instance specified in the cookie. If there is no cookie, the load balancer chooses an application instance based on the existing load balancing algorithm. A cookie is inserted into the response for binding subsequent requests from the same user to that application instance. The policy configuration defines a cookie expiry, which establishes the duration of validity for each cookie.

You can use the **Sessions** section on the **Load Balancer** tab to specify whether or not the load balancer for your application allows session stickiness.



These settings allow you to control how your load balancer handles session cookies.

Enable Session Stickiness

Cookie Expiration Period (seconds): (0 - 1000000)

For more information on Elastic Load Balancing, go to the [Elastic Load Balancing Developer Guide](#).

Configuring Auto Scaling using the AWS toolkit for Visual Studio

Amazon EC2 Auto Scaling is an Amazon web service designed to automatically launch or terminate Amazon EC2 instances based on user-defined triggers. Users can set up *Auto Scaling groups* and associate *triggers* with these groups to automatically scale computing resources based on metrics such as bandwidth usage or CPU utilization. Amazon EC2 Auto Scaling works with Amazon CloudWatch to retrieve metrics for the server instances running your application.

Amazon EC2 Auto Scaling lets you take a group of Amazon EC2 instances and set various parameters to have this group automatically increase or decrease in number. Amazon EC2 Auto Scaling can add or remove Amazon EC2 instances from that group to help you seamlessly deal with traffic changes to your application.

Amazon EC2 Auto Scaling also monitors the health of each Amazon EC2 instance that it launches. If any instance terminates unexpectedly, Amazon EC2 Auto Scaling detects the termination and launches a replacement instance. This capability enables you to maintain a fixed, desired number of Amazon EC2 instances automatically.

Elastic Beanstalk provisions Amazon EC2 Auto Scaling for your application. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration with the **Auto Scaling** tab inside your application environment tab in the AWS Toolkit for Visual Studio.

Auto-scaling automatically launches or terminates EC2 instances based on defined metrics and thresholds called triggers. Auto-scaling will also launch a new EC2 instance in the event of a failure. These settings allow you to control auto-scaling behavior.

Minimum Instance Count:	<input type="text" value="1"/>	(1 - 10000)
Maximum Instance Count:	<input type="text" value="4"/>	(1 - 10000)
Availability Zones:	<input type="text" value="Any 1"/>	
Scaling Cooldown Time (seconds):	<input type="text" value="360"/>	(0 - 10000)
Trigger Measurement:	<input type="text" value="NetworkOut"/>	
Trigger Statistic:	<input type="text" value="Average"/>	
Unit of Measurement:	<input type="text" value="Bytes"/>	
Measurement Period (minutes):	<input type="text" value="5"/>	(1 - 600)
Breach Duration (minutes):	<input type="text" value="5"/>	(1 - 600)
Upper Threshold:	<input type="text" value="6000000"/>	(0 - 20000000)
Upper Breach Scalamet Increment:	<input type="text" value="1"/>	
Lower Threshold:	<input type="text" value="2000000"/>	(0 - 20000000)
Lower Breach Scalamet Increment:	<input type="text" value="-1"/>	

The following section discusses how to configure Auto Scaling parameters for your application.

Launch the configuration

You can edit the launch configuration to control how your Elastic Beanstalk application provisions Amazon EC2 Auto Scaling resources.

The **Minimum Instance Count** and **Maximum Instance Count** boxes let you specify the minimum and maximum size of the Auto Scaling group that your Elastic Beanstalk application uses.

Auto-scaling automatically launches or terminates EC2 instances based on defined metrics and thresholds called triggers. Auto-scaling will also launch a new EC2 instance in the event of a failure. These settings allow you to control auto-scaling behavior.

Minimum Instance Count:	<input type="text" value="1"/>	(1 - 10000)
Maximum Instance Count:	<input type="text" value="4"/>	(1 - 10000)
Availability Zones:	<input type="text" value="Any"/>	
Scaling Cooldown Time (seconds):	<input type="text" value="360"/>	(0 - 10000)

Note

To maintain a fixed number of Amazon EC2 instances, set **Minimum Instance Count** and **Maximum Instance Count** to the same value.

The **Availability Zones** box lets you specify the number of Availability Zones you want your Amazon EC2 instances to be in. It is important to set this number if you want to build fault-tolerant applications. If one Availability Zone goes down, your instances will still be running in your other Availability Zones.

Note

Currently, it is not possible to specify which Availability Zone your instance will be in.

Triggers

A *trigger* is an Amazon EC2 Auto Scaling mechanism that you set to tell the system when you want to increase (*scale out*) the number of instances, and when you want to decrease (*scale in*) the number of instances. You can configure triggers to *fire* on any metric published to Amazon CloudWatch, such as CPU utilization, and determine if the conditions you specified have been met. When the upper or lower thresholds of the conditions you have specified for the metric have been breached for the specified period of time, the trigger launches a long-running process called a *Scaling Activity*.

You can define a scaling trigger for your Elastic Beanstalk application using AWS Toolkit for Visual Studio.

Trigger Measurement:	<input type="text" value="NetworkOut"/>	
Trigger Statistic:	<input type="text" value="Average"/>	
Unit of Measurement:	<input type="text" value="Bytes"/>	
Measurement Period (minutes):	<input type="text" value="5"/>	(1 - 600)
Breach Duration (minutes):	<input type="text" value="5"/>	(1 - 600)
Upper Threshold:	<input type="text" value="6000000"/>	(0 - 20000000)
Upper Breach Scalement Increment:	<input type="text" value="1"/>	
Lower Threshold:	<input type="text" value="2000000"/>	(0 - 20000000)
Lower Breach Scalement Increment:	<input type="text" value="-1"/>	

Amazon EC2 Auto Scaling triggers work by watching a specific Amazon CloudWatch metric for an instance. Triggers include CPU utilization, network traffic, and disk activity. Use the **Trigger Measurement** setting to select a metric for your trigger.

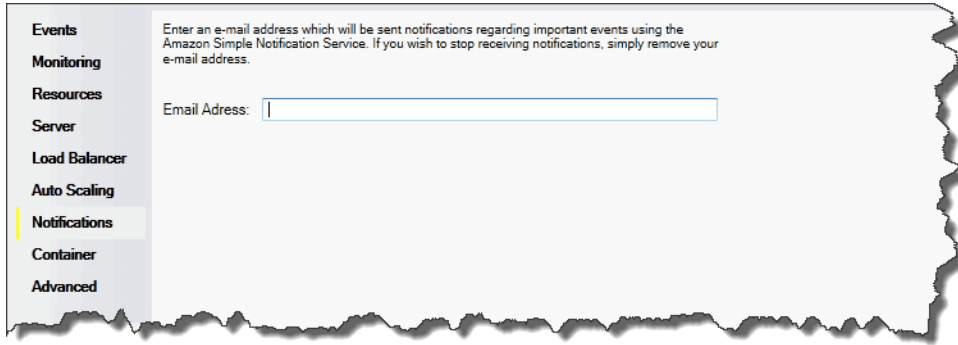
The following list describes the trigger parameters you can configure using the AWS Management Console.

- You can specify which statistic the trigger should use. You can select **Minimum**, **Maximum**, **Sum**, or **Average** for **Trigger Statistic**.
- For **Unit of Measurement**, specify the unit for the trigger measurement.
- The value in the **Measurement Period** box specifies how frequently Amazon CloudWatch measures the metrics for your trigger. The **Breach Duration** is the amount of time a metric can be beyond its defined limit (as specified for the **Upper Threshold** and **Lower Threshold**) before the trigger fires.
- For **Upper Breach Scale Increment** and **Lower Breach Scale Increment**, specify how many Amazon EC2 instances to add or remove when performing a scaling activity.

For more information on Amazon EC2 Auto Scaling, see the *Amazon EC2 Auto Scaling* section on [Amazon Elastic Compute Cloud Documentation](#).

Configuring notifications using AWS toolkit for Visual Studio

Elastic Beanstalk uses the Amazon Simple Notification Service (Amazon SNS) to notify you of important events affecting your application. To enable Amazon SNS notifications, simply enter your email address in the **Email Address** box. To disable these notifications, remove your email address from the box.



Configuring .NET containers using the AWS toolkit for Visual Studio

The **Container/.NET Options** panel lets you fine-tune the behavior of your Amazon EC2 instances and enable or disable Amazon S3 log rotation. You can use the AWS Toolkit for Visual Studio to configure your container information.

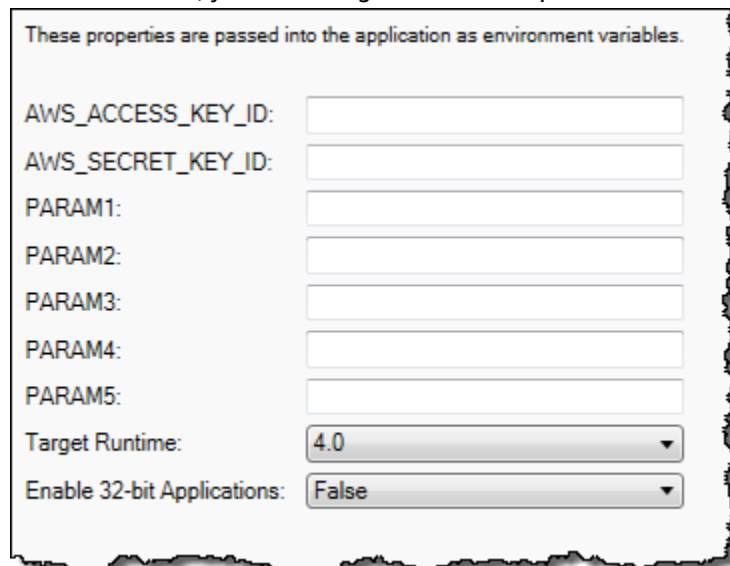
Note

You can modify your configuration settings with zero downtime by swapping the CNAME for your environments. For more information, see [Blue/Green deployments with Elastic Beanstalk](#) (p. 405).

If you want to, you can extend the number of parameters. For information about extending parameters, see [Option settings](#) (p. 601).

To access the Container/.NET options panel for your Elastic Beanstalk application

1. In AWS Toolkit for Visual Studio, expand the Elastic Beanstalk node and your application node.
2. In **AWS Explorer**, double-click your Elastic Beanstalk environment.
3. At the bottom of the **Overview** pane, click the **Configuration** tab.
4. Under **Container**, you can configure container options.

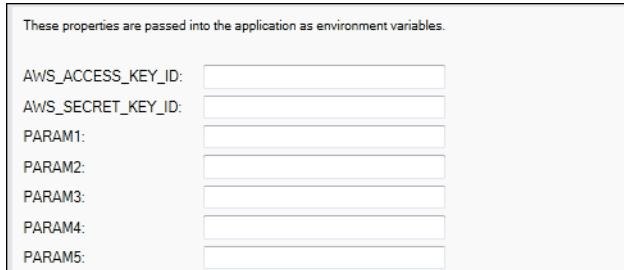


.NET container options

You can choose the version of .NET Framework for your application. Choose either 2.0 or 4.0 for **Target runtime**. Select **Enable 32-bit Applications** if you want to enable 32-bit applications.

Application settings

The **Application Settings** section lets you specify environment variables that you can read from your application code.



These properties are passed into the application as environment variables.

AWS_ACCESS_KEY_ID:	<input type="text"/>
AWS_SECRET_KEY_ID:	<input type="text"/>
PARAM1:	<input type="text"/>
PARAM2:	<input type="text"/>
PARAM3:	<input type="text"/>
PARAM4:	<input type="text"/>
PARAM5:	<input type="text"/>

Managing accounts

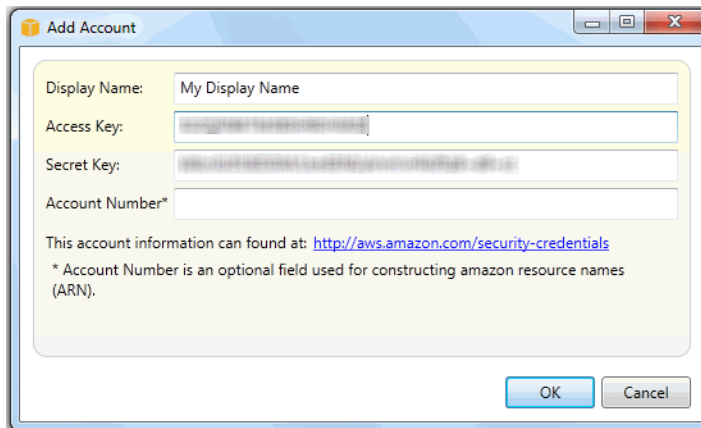
If you want to set up different AWS accounts to perform different tasks, such as testing, staging, and production, you can add, edit, and delete accounts using the AWS Toolkit for Visual Studio.

To manage multiple accounts

1. In Visual Studio, on the **View** menu, click **AWS Explorer**.
2. Beside the **Account** list, click the **Add Account** button.



The **Add Account** dialog box appears.



The 'Add Account' dialog box is shown with the following fields:

- Display Name: My Display Name
- Access Key:
- Secret Key:
- Account Number*:

This account information can found at: <http://aws.amazon.com/security-credentials>
* Account Number is an optional field used for constructing amazon resource names (ARN).

Buttons: OK, Cancel

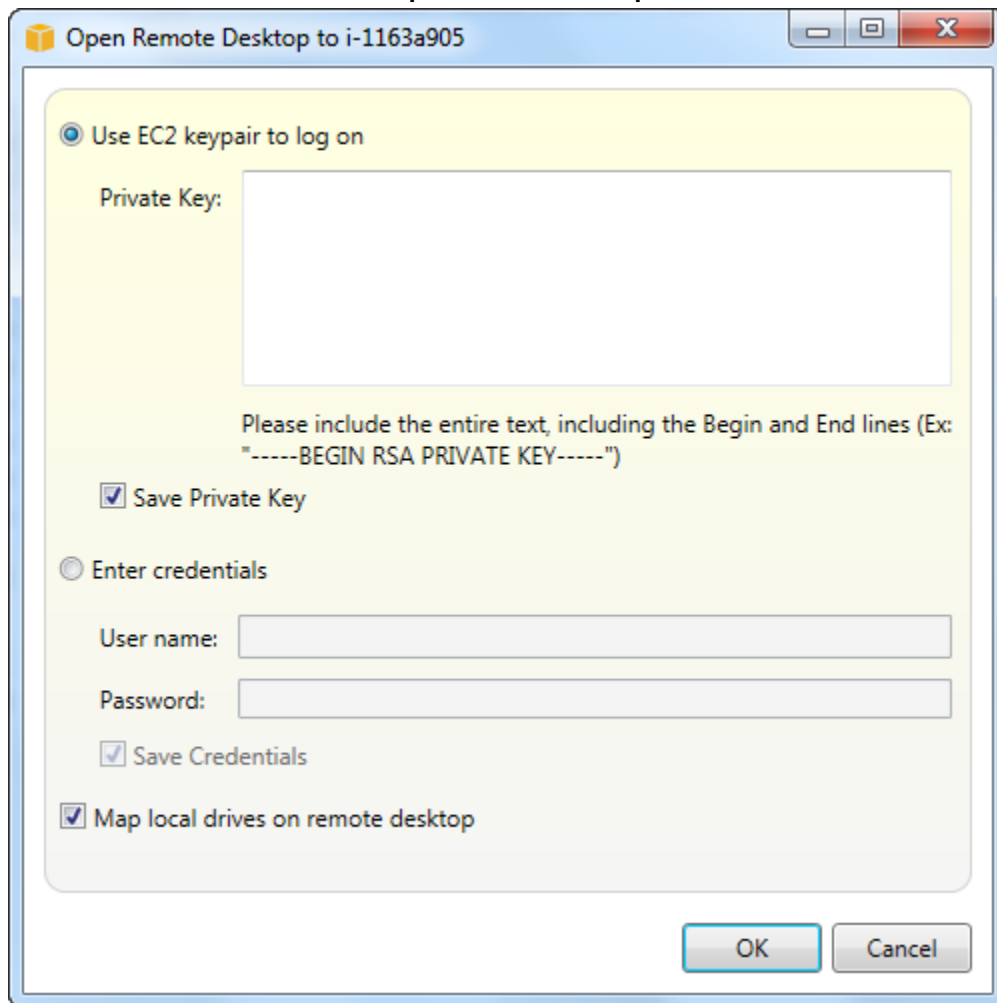
3. Fill in the requested information.
4. Your account information now appears on the **AWS Explorer** tab. When you publish to Elastic Beanstalk, you can select which account you would like to use.

Listing and connecting to server instances

You can view a list of Amazon EC2 instances running your Elastic Beanstalk application environment through the AWS Toolkit for Visual Studio or from the AWS Management Console. You can connect to these instances using Remote Desktop Connection. For information about listing and connecting to your server instances using the AWS Management Console, see [Listing and connecting to server instances \(p. 730\)](#). The following section steps you through viewing and connecting you to your server instances using the AWS Toolkit for Visual Studio.

To view and connect to Amazon EC2 instances for an environment

1. In Visual Studio, in **AWS Explorer**, expand the **Amazon EC2** node and double-click **Instances**.
2. Right-click the instance ID for the Amazon EC2 instance running in your application's load balancer in the **Instance** column and select **Open Remote Desktop** from the context menu.



3. Select **Use EC2 keypair to log on** and paste the contents of your private key file that you used to deploy your application in the **Private key** box. Alternatively, enter your user name and password in the **User name** and **Password** text boxes.

Note

If the key pair is stored inside the Toolkit, the text box does not appear.

4. Click **OK**.

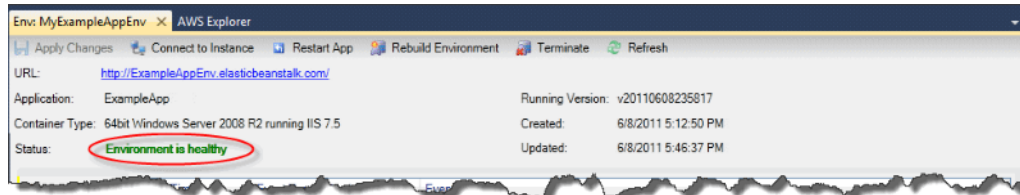
Monitoring application health

When you are running a production website, it is important to know that your application is available and responding to requests. To assist with monitoring your application's responsiveness, Elastic Beanstalk provides features where you can monitor statistics about your application and create alerts that trigger when thresholds are exceeded.

For information about the health monitoring provided by Elastic Beanstalk, see [Basic health reporting \(p. 688\)](#).

You can access operational information about your application by using either the AWS Toolkit for Visual Studio or the AWS Management Console.

The toolkit displays your environment's status and application health in the **Status** field.



To monitor application health

1. In the AWS Toolkit for Visual Studio, in **AWS Explorer**, expand the Elastic Beanstalk node, and then expand your application node.
2. Right-click your Elastic Beanstalk environment, and then click **View Status**.
3. On your application environment tab, click **Monitoring**.

The **Monitoring** panel includes a set of graphs showing resource usage for your particular application environment.



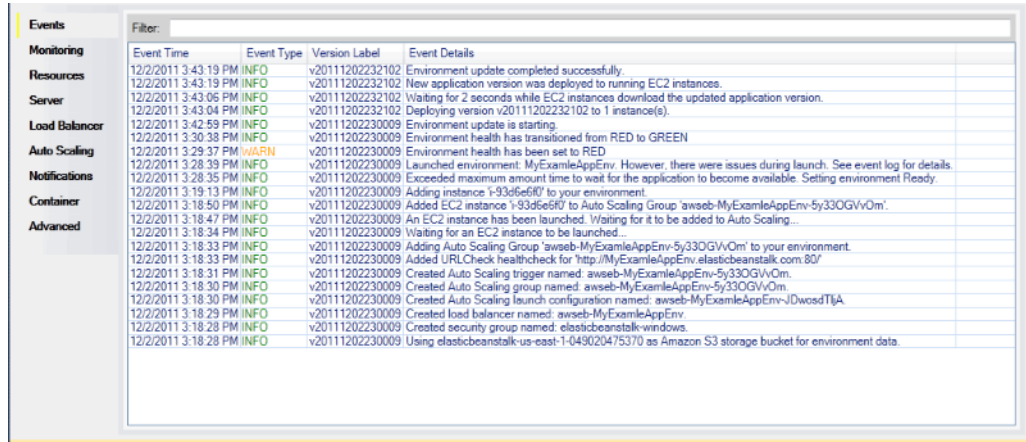
Note

By default, the time range is set to the last hour. To modify this setting, in the **Time Range** list, click a different time range.

You can use the AWS Toolkit for Visual Studio or the AWS Management Console to view events associated with your application.

To view application events

1. In the AWS Toolkit for Visual Studio, in **AWS Explorer**, expand the Elastic Beanstalk node and your application node.
2. Right-click your Elastic Beanstalk environment in **AWS Explorer** and then click **View Status**.
3. In your application environment tab, click **Events**.



Deploying Elastic Beanstalk applications in .NET using the deployment tool

The AWS Toolkit for Visual Studio includes a deployment tool, a command line tool that provides the same functionality as the deployment wizard in the AWS Toolkit. You can use the deployment tool in your build pipeline or in other scripts to automate deployments to Elastic Beanstalk.

The deployment tool supports both initial deployments and redeployments. If you previously deployed your application using the deployment tool, you can redeploy using the deployment wizard within Visual Studio. Similarly, if you have deployed using the wizard, you can redeploy using the deployment tool.

Note

The deployment tool does not apply [recommended values \(p. 537\)](#) for configuration options like the console or EB CLI. Use [configuration files \(p. 600\)](#) to ensure that any settings that you need are configured when you launch your environment.

This chapter walks you through deploying a sample .NET application to Elastic Beanstalk using the deployment tool, and then redeploying the application using an incremental deployment. For a more in-depth discussion about the deployment tool, including the parameter options, see [Deployment Tool](#).

Prerequisites

To use the deployment tool, you need to install the AWS Toolkit for Visual Studio. For information on prerequisites and installation instructions, see [AWS Toolkit for Microsoft Visual Studio](#).

The deployment tool is typically installed in one of the following directories on Windows:

32-bit	64-bit
C:\Program Files\AWS Tools \Deployment Tool\awsdeploy.exe	C:\Program Files (x86)\AWS Tools\Deployment Tool \awsdeploy.exe

Deploy to Elastic Beanstalk

To deploy the sample application to Elastic Beanstalk using the deployment tool, you first need to modify the `ElasticBeanstalkDeploymentSample.txt` configuration file, which is provided in the `Samples` directory. This configuration file contains the information necessary to deploy your application, including the application name, application version, environment name, and your AWS access credentials. After modifying the configuration file, you then use the command line to deploy the sample application. Your web deploy file is uploaded to Amazon S3 and registered as a new application version with Elastic Beanstalk. It will take a few minutes to deploy your application. Once the environment is healthy, the deployment tool outputs a URL for the running application.

To deploy a .NET application to Elastic Beanstalk

1. From the `Samples` subdirectory where the deployment tool is installed, open `ElasticBeanstalkDeploymentSample.txt` and enter your AWS access key and AWS secret key as in the following example.

```
### AWS Access Key and Secret Key used to create and deploy the application instance
AWSAccessKey = AKIAIOSFODNN7EXAMPLE
AWSSecretKey = wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY
```

Note

For API access, you need an access key ID and secret access key. Use IAM user access keys instead of AWS account root user access keys. For more information about creating access keys, see [Managing Access Keys for IAM Users](#) in the *IAM User Guide*.

2. At the command line prompt, type the following:

```
C:\Program Files (x86)\AWS Tools\Deployment Tool>awsdeploy.exe /w Samples
\ElasticBeanstalkDeploymentSample.txt
```

It takes a few minutes to deploy your application. If the deployment succeeds, you will see the message, `Application deployment completed; environment health is Green`.

Note

If you receive the following error, the CNAME already exists.

```
[Error]: Deployment to AWS Elastic Beanstalk failed with exception: DNS name
(MyAppEnv.elasticbeanstalk.com) is not available.
```

Because a CNAME must be unique, you need to change `Environment.CNAME` in `ElasticBeanstalkDeploymentSample.txt`.

3. In your web browser, navigate to the URL of your running application. The URL will be in the form `<CNAME.elasticbeanstalk.com>` (e.g., `MyAppEnv.elasticbeanstalk.com`).

Migrating your on-premises .NET application to Elastic Beanstalk

If you're thinking about migrating your .NET application from on-premises servers to Amazon Web Services (AWS), the .NET Migration Assistant for AWS Elastic Beanstalk might be useful for you. The assistant is an interactive PowerShell utility that migrates a .NET application from Windows Server with IIS running on premises to AWS Elastic Beanstalk. The assistant can migrate an entire website to Elastic Beanstalk with minimal or no changes needed.

For more information about the .NET Migration Assistant for AWS Elastic Beanstalk and to download it, see the <https://github.com/awslabs/windows-web-app-migration-assistant> repository on GitHub.

If your application includes Microsoft SQL Server databases, the assistant's documentation on GitHub includes several options for migrating them.

Resources

There are several places you can go to get additional help when developing your .NET applications:

Resource	Description
.NET Development Forum	Post your questions and get feedback.
.NET Developer Center	One-stop shop for sample code, documentation, tools, and additional resources.
AWS SDK for .NET Documentation	Read about setting up the SDK and running code samples, features of the SDK, and detailed information about the API operations for the SDK.

Deploying Node.js applications to Elastic Beanstalk

Topics

- [Getting started with Node.js on Elastic Beanstalk \(p. 192\)](#)
- [Setting up your Node.js development environment \(p. 193\)](#)
- [Using the Elastic Beanstalk Node.js platform \(p. 195\)](#)
- [Deploying an Express application to Elastic Beanstalk \(p. 203\)](#)
- [Deploying an Express application with clustering to Elastic Beanstalk \(p. 207\)](#)
- [Deploying a Node.js application with DynamoDB to Elastic Beanstalk \(p. 216\)](#)
- [Adding an Amazon RDS DB instance to your Node.js application environment \(p. 225\)](#)
- [Resources \(p. 227\)](#)

AWS Elastic Beanstalk for Node.js makes it easy to deploy, manage, and scale your Node.js web applications using Amazon Web Services. Elastic Beanstalk for Node.js is available to anyone developing or hosting a web application using Node.js. This chapter provides step-by-step instructions for deploying your Node.js web application to Elastic Beanstalk using the Elastic Beanstalk management console, and provides walkthroughs for common tasks such as database integration and working with the Express framework.

After you deploy your Elastic Beanstalk application, you can continue to use EB CLI to manage your application and environment, or you can use the Elastic Beanstalk console, AWS CLI, or the APIs.

The topics in this chapter assume some knowledge of Elastic Beanstalk environments. If you haven't used Elastic Beanstalk before, try the [getting started tutorial \(p. 3\)](#) to learn the basics.

Getting started with Node.js on Elastic Beanstalk

To get started with Node.js applications on AWS Elastic Beanstalk, all you need is an application [source bundle \(p. 342\)](#) to upload as your first application version and to deploy to an environment. When you create an environment, Elastic Beanstalk allocates all of the AWS resources needed to run a highly scalable web application.

Launching an environment with a sample Node.js application

Elastic Beanstalk provides single page sample applications for each platform as well as more complex examples that show the use of additional AWS resources such as Amazon RDS and language or platform-specific features and APIs.

Samples

Environment type	Source bundle	Description	
Web Server	nodejs-v1.zip	Single page application. Use the procedure at Create an Example Application (p. 3) to launch this example.	
Web Server with Amazon RDS	nodejs-express-hiking-v1.zip	Hiking log application that uses the Express framework and an RDS database. Tutorial (p. 203)	
Web Server with Amazon ElastiCache	nodejs-example-express-elasticache.zip	Express web application that uses Amazon ElastiCache for clustering. Clustering enhances your web application's high availability, performance, and security. Tutorial (p. 207)	
Web Server with DynamoDB, Amazon SNS and Amazon SQS	eb-node-express-sample-v1.0.zip Clone the repo at GitHub.com	Express web site that collects user contact information for a new company's marketing campaign. Uses the AWS SDK for JavaScript in Node.js to write entries to a DynamoDB table, and Elastic Beanstalk configuration files to create resources in DynamoDB, Amazon SNS and Amazon SQS. Tutorial (p. 216)	

Next steps

After you have an environment running an application, you can deploy a new version of the application or a completely different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances. For details about application deployment, see [Deploy a New Version of Your Application \(p. 7\)](#).

After you've deployed a sample application or two and are ready to start developing and running Node.js applications locally, see [the next section \(p. 193\)](#) to set up a Node.js development environment with all of the tools that you will need.

Setting up your Node.js development environment

Set up a Node.js development environment to test your application locally prior to deploying it to AWS Elastic Beanstalk. This topic outlines development environment setup steps and links to installation pages for useful tools.

For common setup steps and tools that apply to all languages, see [Configuring your development machine \(p. 849\)](#).

Topics

- [Installing Node.js \(p. 194\)](#)
- [Installing npm \(p. 194\)](#)
- [Installing the AWS SDK for Node.js \(p. 194\)](#)
- [Installing Express \(p. 194\)](#)

Installing Node.js

Install Node.js to run Node.js applications locally. If you don't have a preference, get the latest version supported by Elastic Beanstalk. See [Node.js](#) in the *AWS Elastic Beanstalk Platforms* document for a list of supported versions.

Download Node.js at nodejs.org.

Installing npm

Node.js uses the npm package manager to help you install tools and frameworks for use in your application. Download npm at npmjs.com.

Installing the AWS SDK for Node.js

If you need to manage AWS resources from within your application, install the AWS SDK for JavaScript in Node.js. Install the SDK with npm:

```
$ npm install aws-sdk
```

Visit the [AWS SDK for JavaScript in Node.js](#) homepage for more information.

Installing Express

Express is a web application framework that runs on Node.js. To use it, set up Express and create the project structure. The following walks you through setting up Express on a Linux operating system.

Note

Depending on your permission level to system directories, you might need to prefix some of these commands with `sudo`.

To set up your Express development environment on your local computer

1. Create a directory for your Express application.

```
~$ mkdir node-express  
~$ cd node-express
```

2. Install Express globally so that you have access to the `express` command.

```
~/node-express$ npm install -g express-generator
```

3. Depending on your operating system, you may need to set your path to run the `express` command. If you need to set your path, use the output from the previous step when you installed Express. The following is an example.

```
~/node-express$ export PATH=$PATH:/usr/local/share/npm/bin/express
```

4. Run the `express` command. This generates `package.json`, `app.js`, and a few directories.

```
~/node-express$ express
```

When prompted if you want to continue, type **y**.

5. Set up local dependencies.

```
~/node-express$ npm install
```

6. Verify it works.

```
~/node-express$ npm start
```

You should see output similar to the following:

```
> nodejs@0.0.0 start /home/local/user/node-express  
> node ./bin/www
```

The server runs on port 3000 by default. To test it, run `curl http://localhost:3000` in another terminal, or open a browser on the local computer and go to `http://localhost:3000`.

Press **Ctrl+C** to stop the server.

Using the Elastic Beanstalk Node.js platform

Important

Amazon Linux 2 platform versions are fundamentally different than Amazon Linux AMI platform versions (preceding Amazon Linux 2). These different platform generations are incompatible in several ways. If you are migrating to an Amazon Linux 2 platform version, be sure to read the information in [the section called "Upgrade to Amazon Linux 2" \(p. 426\)](#).

The AWS Elastic Beanstalk Node.js platform is a set of [platform versions](#) for Node.js web applications that run behind an nginx proxy server.

Elastic Beanstalk provides [configuration options \(p. 536\)](#) that you can use to customize the software that runs on the EC2 instances in your Elastic Beanstalk environment. You can configure environment variables needed by your application, enable log rotation to Amazon S3, and map folders in your application source that contain static files to paths served by the proxy server.

Configuration options are available in the Elastic Beanstalk console for [modifying the configuration of a running environment \(p. 547\)](#). To avoid losing your environment's configuration when you terminate it, you can use [saved configurations \(p. 640\)](#) to save your settings and later apply them to another environment.

To save settings in your source code, you can include [configuration files \(p. 600\)](#). Settings in configuration files are applied every time you create an environment or deploy your application. You can also use configuration files to install packages, run scripts, and perform other instance customization operations during deployments.

You can [include a Package.json file \(p. 200\)](#) in your source bundle to install packages during deployment, to provide a start command, and to specify the Node.js version you want your application to use. You can include an [npm-shrinkwrap.json file \(p. 201\)](#) to lock down dependency versions.

The Node.js platform includes a proxy server to serve static assets, forward traffic to your application, and compress responses. You can [extend or override the default proxy configuration \(p. 201\)](#) for advanced scenarios.

You can add a `Procfile` to your source bundle to specify the command that starts your application, as the following example shows. This feature replaces the legacy `NodeCommand` option in the `aws:elasticbeanstalk:container:nodejs` namespace.

Example Procfile

```
web: node index.js
```

For details about `Procfile` usage, expand the *Buildfile and Procfile* section in [the section called “Extending Linux platforms” \(p. 31\)](#).

When you don't provide a `Procfile`, Elastic Beanstalk runs `npm start` if you provide a `package.json` file. If you don't provide that either, Elastic Beanstalk looks for the file `app.js` or `server.js`, in this order, and runs it.

Settings applied in the Elastic Beanstalk console override the same settings in configuration files, if they exist. This lets you have default settings in configuration files, and override them with environment-specific settings in the console. For more information about precedence, and other methods of changing settings, see [Configuration options \(p. 536\)](#).

For details about the various ways you can extend an Elastic Beanstalk Linux-based platform, see [the section called “Extending Linux platforms” \(p. 31\)](#).

Configuring your Node.js environment

The Node.js platform settings let you fine-tune the behavior of your Amazon EC2 instances. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration using the Elastic Beanstalk console.

Use the Elastic Beanstalk console to enable log rotation to Amazon S3 and configure variables that your application can read from the environment.

To configure your Node.js environment in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.

Log options

The **Log Options** section has two settings:

- **Instance profile**– Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3** – Specifies whether log files for your application's Amazon EC2 instances should be copied to the Amazon S3 bucket associated with your application.

Static files

To improve performance, the **Static files** section lets you configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. For each directory,

you set the virtual path to directory mapping. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

For details about configuring static files using the Elastic Beanstalk console, see [the section called “Static files” \(p. 650\)](#).

Environment properties

The **Environment Properties** section lets you specify environment configuration settings on the Amazon EC2 instances that are running your application. These settings are passed in as key-value pairs to the application.

Inside the Node.js environment running in AWS Elastic Beanstalk, you can access the environment variables using `process.env.ENV_VARIABLE` similar to the following example.

```
var endpoint = process.env.API_ENDPOINT
```

The Node.js platform sets the `PORT` environment variable to the port to which the proxy server passes traffic. See [Configuring the proxy server \(p. 201\)](#).

See [Environment properties and other software settings \(p. 516\)](#) for more information.

Configuring an Amazon Linux AMI (preceding Amazon Linux 2) Node.js environment

The following console software configuration categories are supported only on an Elastic Beanstalk Node.js environment that uses an Amazon Linux AMI platform version (preceding Amazon Linux 2).

Container options

On the configuration page, specify the following:

- **Proxy server**– Specifies which web server to use to proxy connections to Node.js. By default, nginx is used. If you select **none**, static file mappings do not take effect, and gzip compression is disabled.
- **Node.js version**– Specifies the version of Node.js. For a list of supported Node.js versions, see [Node.js](#) in the *AWS Elastic Beanstalk Platforms* guide.
- **Gzip compression**– Specifies whether gzip compression is enabled. By default, gzip compression is enabled.
- **Node command**– Lets you enter the command used to start the Node.js application. An empty string (the default) means Elastic Beanstalk will use `app.js`, then `server.js`, and then `npm start` in that order.

Node.js configuration namespace

You can use a [configuration file \(p. 600\)](#) to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be defined by the Elastic Beanstalk service or the platform that you use and are organized into *namespaces*.

The Node.js platform doesn't define any platform-specific namespaces. You can configure the proxy to serve static files by using the `aws:elasticbeanstalk:environment:proxy:staticfiles` namespace. For details and an example, see [the section called “Static files” \(p. 650\)](#).

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See [Configuration options \(p. 536\)](#) for more information.

The Amazon Linux AMI (preceding Amazon Linux 2) Node.js platform

If your Elastic Beanstalk Node.js environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Node.js platform-specific configuration options

Elastic Beanstalk supports some platform-specific configurations options for Amazon Linux AMI Node.js platform versions. You can choose which proxy server to run in front of your application, choose a specific version of Node.js to run, and choose the command used to run your application.

For proxy server, in addition to nginx, you can choose the Apache proxy server. In addition, you can set the none value to the `ProxyServer` option. In this case, Elastic Beanstalk runs your application as standalone, not behind any proxy server. If your environment runs a standalone application, update your code to listen to the port that nginx forwards traffic to.

```
var port = process.env.PORT || 8080;

app.listen(port, function() {
  console.log('Server running at http://127.0.0.1:%s', port);
});
```

Node.js language versions

In terms of supported language version, the Node.js Amazon Linux AMI platform is slightly different than other Elastic Beanstalk managed platforms. Each Node.js platform version supports a few Node.js language versions. For a list of supported Node.js versions, see [Node.js](#) in the *AWS Elastic Beanstalk Platforms* guide.

You can use a platform-specific configuration option to set the language version. For details, see [the section called "Configuring your Node.js environment" \(p. 196\)](#). Alternatively, you can use the Elastic Beanstalk console to update the Node.js version that your environment uses as part of updating your platform version, as shown in the following procedure.

Note

When support for the version of Node.js that you are using is removed from the platform, you must change or remove the version setting prior to doing a [platform update \(p. 415\)](#). This might occur when a security vulnerability is identified for one or more versions of Node.js. When this happens, attempting to update to a new version of the platform that doesn't support the configured [NodeVersion \(p. 595\)](#) fails. To avoid needing to create a new environment, change the `NodeVersion` configuration option to a Node.js version that is supported by both the old platform version and the new one, or [remove the option setting \(p. 547\)](#), and then perform the platform update.

To configure your environment's Node.js version in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, under **Platform**, choose **Change**.
4. On the **Update platform version** dialog, select a Node.js version.

Update platform version

Availability warning
This operation replaces your instances; your application is unavailable during the update. To keep an instance in service during the update, enable rolling updates. Another option is to clone the environment, which creates a newer version of the platform, and then swap the CNAME of the environment when you are ready to deploy the clone. Learn more at [Updating AWS Elastic Beanstalk Environment Platform Version](#), [Rolling Updates](#) and [Deploying Version with Zero Downtime](#).

Platform branch
Node.js running on 64bit Amazon Linux

Current platform version
4.13.0

New platform version
4.13.0 (Recommended)

Current Node.js version
12.14.0

New Node.js version
12.14.1

5. Choose **Save**.

Node.js configuration namespaces

The Node.js Amazon Linux AMI platform defines additional options in the `aws:elasticbeanstalk:container:nodejs:staticfiles` and `aws:elasticbeanstalk:container:nodejs` namespaces.

The following configuration file tells Elastic Beanstalk to use `npm start` to run the application, sets the proxy type to Apache, enables compression, and configures the proxy to serve static files from two source directories: HTML files at the `html` path under the website's root from the `statichtml` source directory, and image files at the `images` path under the website's root from the `staticimages` source directory.

Example `.ebextensions/node-settings.config`

```
option_settings:  
  aws:elasticbeanstalk:container:nodejs:  
    NodeCommand: "npm start"  
    ProxyServer: apache  
    GzipCompression: true  
  aws:elasticbeanstalk:container:nodejs:staticfiles:  
    /html: statichtml
```

```
/images: staticimages
```

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See [Configuration options \(p. 536\)](#) for more information.

Configuring your application's dependencies

Your application might have dependencies on some Node.js modules, like the ones you specify in `require()` statements. You can specify these dependencies using a `package.json` file. Alternatively, you can include your application's dependencies in the source bundle and deploy them with the application. This page discusses both of these ways.

Specifying Node.js dependencies with a `package.json` file

Include a `package.json` file in the root of your project source to specify dependency packages (use the `dependencies` keyword) and to provide a start command (use the `scripts` keyword). Using the `scripts` keyword can replace the legacy `NodeCommand` option in the `aws:elasticbeanstalk:container:nodejs` namespace.

Example `package.json` – Express

```
{
  "name": "my-app",
  "version": "0.0.1",
  "private": true,
  "dependencies": {
    "ejs": "latest",
    "aws-sdk": "latest",
    "express": "latest",
    "body-parser": "latest"
  },
  "scripts": {
    "start": "node app.js"
  }
}
```

When a `package.json` file is present, Elastic Beanstalk runs `npm install` to install dependencies. It also uses the `start` command to start your application.

By default, Elastic Beanstalk installs dependencies in production mode (`npm install --production`). If you want to install development dependencies on your environment instances, set the `NPM_USE_PRODUCTION` [environment property \(p. 518\)](#) to `false`.

You can also use the `engines` keyword in the `package.json` file to specify the Node.js version you want your application to use, as the following example shows. This feature replaces the legacy `NodeVersion` option in the `aws:elasticbeanstalk:container:nodejs` namespace. Be aware that you can only specify a Node.js version that corresponds with your platform branch. For example, if you're using the Node.js 12 platform branch, you can only specify a 12.x.y Node.js version. For valid Node.js versions for each platform branch, see [Node.js](#) in the *AWS Elastic Beanstalk Platforms* guide.

Example `package.json` – Node.js version

```
{
  ...
  "engines": { "node" : "12.16.1" }
}
```

Note

When support for the version of Node.js that you are using is removed from the platform, you must change or remove the Node.js version setting prior to doing a [platform update \(p. 415\)](#). This might occur when a security vulnerability is identified for one or more versions of Node.js. When this happens, attempting to update to a new version of the platform that doesn't support the configured Node.js version fails. To avoid needing to create a new environment, change the Node.js version setting in `package.json` to a Node.js version that is supported by both the old platform version and the new one, or remove the setting, and then deploy the new source bundle. Only then perform the platform update.

Including Node.js dependencies in a `node_modules` directory

To deploy dependency packages to environment instances together with your application code, include them in a directory named `node_modules` in the root of your project source. Node.js looks for dependencies in this directory. For details, see [Loading from `node_modules` Folders](#) in the Node.js documentation.

Note

When you deploy a `node_modules` directory to an Amazon Linux 2 Node.js platform version, Elastic Beanstalk assumes that you're providing your own dependency packages, and avoids installing dependencies specified in a [package.json \(p. 200\)](#) file.

Locking dependencies with `npm shrinkwrap`

The Node.js platform runs `npm install` each time you deploy. When new versions of your dependencies are available, they will be installed when you deploy your application, potentially causing the deployment to take a long time.

You can avoid upgrading dependencies by creating an `npm-shrinkwrap.json` file that locks down your application's dependencies to the current version.

```
$ npm install
$ npm shrinkwrap
wrote npm-shrinkwrap.json
```

Include this file in your source bundle to ensure that dependencies are only installed once.

Configuring the proxy server

Elastic Beanstalk uses `nginx` as the reverse proxy to map your application to your Elastic Load Balancing load balancer on port 80. Elastic Beanstalk provides a default `nginx` configuration that you can either extend or override completely with your own configuration.

By default, Elastic Beanstalk configures the `nginx` proxy to forward requests to your application on port 8080. You can override the default port by setting the `PORT` [environment property \(p. 196\)](#) to the port on which your main application listens.

Notes

- The port that your application listens on doesn't affect the port that the `nginx` server listens to receive requests from the load balancer.
- On Amazon Linux AMI Node.js platform versions (preceding Amazon Linux 2), Elastic Beanstalk configures the `nginx` proxy to forward requests to your application on port 8081. For details about proxy configuration on these Node.js platform versions, see [the section called "Configuring the proxy on Amazon Linux AMI \(preceding Amazon Linux 2\)" \(p. 202\)](#) on this page.

All Amazon Linux 2 platforms support a uniform proxy configuration feature. For details about configuring the proxy server on the new Amazon Corretto platform versions running Amazon Linux 2, expand the *Reverse Proxy Configuration* section in [the section called "Extending Linux platforms" \(p. 31\)](#).

Configuring the proxy on Amazon Linux AMI (preceding Amazon Linux 2)

If your Elastic Beanstalk Node.js environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the information in this section.

Extending and overriding the default proxy configuration

The Node.js platform uses a reverse proxy to relay requests from port 80 on the instance to your application listening on port 8081. Elastic Beanstalk provides a default proxy configuration that you can either extend or override completely with your own configuration.

To extend the default configuration, add `.conf` files to `/etc/nginx/conf.d` with a configuration file. See [Terminating HTTPS on EC2 instances running Node.js \(p. 665\)](#) for an example.

The Node.js platform sets the `PORT` environment variable to the port to which the proxy server passes traffic. Read this variable in your code to configure your application's port.

```
var port = process.env.PORT || 3000;

var server = app.listen(port, function () {
  console.log('Server running at http://127.0.0.1:' + port + '/');
});
```

The default nginx configuration forwards traffic to an upstream server named `nodejs` at `127.0.0.1:8081`. It is possible to remove the default configuration and provide your own in a [configuration file \(p. 600\)](#).

Example `.ebextensions/proxy.config`

The following example removes the default configuration and adds a custom configuration that forwards traffic to port 5000 instead of 8081.

```
files:
  /etc/nginx/conf.d/proxy.conf:
    mode: "000644"
    owner: root
    group: root
    content: |
      upstream nodejs {
        server 127.0.0.1:5000;
        keepalive 256;
      }

      server {
        listen 8080;

        if ($time_iso8601 ~ "^(\\d{4})-(\\d{2})-(\\d{2})T(\\d{2})") {
          set $year $1;
          set $month $2;
          set $day $3;
          set $hour $4;
        }

        access_log /var/log/nginx/healthd/application.log.$year-$month-$day-$hour healthd;
        access_log /var/log/nginx/access.log main;

        location / {
          proxy_pass http://nodejs;
```

```
proxy_set_header    Connection "";
proxy_http_version  1.1;
proxy_set_header    Host          $host;
proxy_set_header    X-Real-IP     $remote_addr;
proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
}

gzip on;
gzip_comp_level 4;
gzip_types text/html text/plain text/css application/json application/x-javascript
text/xml application/xml application/xml+rss text/javascript;

location /static {
    alias /var/app/current/static;
}

}

/opt/elasticbeanstalk/hooks/configdeploy/post/99_kill_default_nginx.sh:
mode: "000755"
owner: root
group: root
content: |
#!/bin/bash -xe
rm -f /etc/nginx/conf.d/00_elastic_beanstalk_proxy.conf
service nginx stop
service nginx start

container_commands:
  removeconfig:
    command: "rm -f /tmp/deployment/config/
#etc#nginx#conf.d#00_elastic_beanstalk_proxy.conf /etc/nginx/
conf.d/00_elastic_beanstalk_proxy.conf"
```

The example configuration, `/etc/nginx/conf.d/proxy.conf`, uses the default configuration at `/etc/nginx/conf.d/00_elastic_beanstalk_proxy.conf` as a base to include the default server block with compression and log settings, and a static file mapping.

The `removeconfig` command removes the container's default configuration to make sure that the proxy server uses the custom configuration. Elastic Beanstalk recreates the default configuration during every configuration deployment. To account for that, the example adds a post-configuration-deployment hook, `/opt/elasticbeanstalk/hooks/configdeploy/post/99_kill_default_nginx.sh`, which removes the default configuration and restarts the proxy server.

Note

The default configuration may change in future versions of the Node.js platform. Use the latest version of the configuration as a base for your customizations to ensure compatibility.

If you override the default configuration, you must define any static file mappings and gzip compression, as the platform will not be able to apply the [standard settings \(p. 197\)](#).

Deploying an Express application to Elastic Beanstalk

This section walks you through deploying a sample application to Elastic Beanstalk using Elastic Beanstalk Command Line Interface (EB CLI) and Git, and then updating the application to use the [Express framework](#).

Prerequisites

This tutorial requires the Node.js language, its package manager called npm, and the Express web application framework. For details on installing these components and setting up your local development environment, see [Setting up your Node.js development environment \(p. 193\)](#).

Note

For this tutorial, you don't need to install the AWS SDK for Node.js, which is also mentioned in [Setting up your Node.js development environment \(p. 193\)](#).

The tutorial also requires the Elastic Beanstalk Command Line Interface (EB CLI). For details on installing and configuring the EB CLI, see [Install the EB CLI \(p. 853\)](#) and [Configure the EB CLI \(p. 860\)](#).

Initialize Git

The prerequisite Node.js and Express setup results in an Express project structure in the `node-express` folder. If you haven't already generated an Express project, run the following command. For more details, see [Installing Express \(p. 194\)](#).

```
~/node-express$ express && npm install
```

Now let's set up a Git repository in this folder.

To set up a Git repository

1. Initialize the Git repository. If you don't have Git installed, download it from the [Git downloads site](#).

```
~/node-express$ git init
```

2. Create a file named `.gitignore` and add the following files and directories to it. These files will be excluded from being added to the repository. This step is not required, but it is recommended.

node-express/.gitignore

```
node_modules/  
.gitignore  
.elasticbeanstalk/
```

Create an Elastic Beanstalk environment

Configure an EB CLI repository for your application and create an Elastic Beanstalk environment running the Node.js platform.

1. Create a repository with the **eb init** command.

```
~/node-express$ eb init --platform node.js --region us-east-2  
Application node-express has been created.
```

This command creates a configuration file in a folder named `.elasticbeanstalk` that specifies settings for creating environments for your application, and creates an Elastic Beanstalk application named after the current folder.

2. Create an environment running a sample application with the **eb create** command.

```
~/node-express$ eb create --sample node-express-env
```

This command creates a load balanced environment with the default settings for the Node.js platform and the following resources:

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or nginx as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
 - **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
 - **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
 - **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
 - **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
 - **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
 - **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
 - **Domain name** – A domain name that routes to your web app in the form `subdomain.region.elasticbeanstalk.com`.
3. When environment creation completes, use the **eb open** command to open the environment's URL in the default browser.

```
~/node-express$ eb open
```

Update the application

After you have created an environment with a sample application, you can update it with your own application. In this step, we update the sample application to use the Express framework.

To update your application to use Express

1. On your local computer, create an `.ebextensions` directory in the top-level directory of your source bundle. In this example, we use `node-express/.ebextensions`.
2. Add a configuration file that sets the Node Command to "npm start":

node-express/.ebextensions/nodecommand.config

```
option_settings:  
  aws:elasticbeanstalk:container:nodejs:  
    NodeCommand: "npm start"
```

For more information, see [Advanced environment customization with configuration files \(.ebextensions\)](#) (p. 600).

3. Stage the files:

```
~/node-express$ git add .
```

```
~/node-express$ git commit -m "First express app"
```

4. Deploy the changes:

```
~/node-express$ eb deploy
```

5. Once the environment is green and ready, refresh the URL to verify it worked. You should see a web page that says **Welcome to Express**.

Next, let's update the Express application to serve static files and add a new page.

To configure static files and add a new page to your Express application

1. Add a second configuration file with the following content:

node-express/.ebextensions/staticfiles.config

```
option_settings:  
  aws:elasticbeanstalk:container:nodejs:staticfiles:  
    /public: /public
```

This setting configures the proxy server to serve files in the `public` folder at the `/public` path of the application. [Serving files statically \(p. 197\)](#) from the proxy reduces the load on your application.

2. Comment out the static mapping in `node-express/app.js`. This step is not required, but it is a good test to confirm that static mappings are configured correctly.

```
// app.use(express.static(path.join(__dirname, 'public')));
```

3. Add your updated files to your local repository and commit your changes.

```
~/node-express$ git add .ebextensions/ app.js  
~/node-express$ git commit -m "Serve stylesheets statically with nginx."
```

4. Add `node-express/routes/hike.js`. Type the following:

```
exports.index = function(req, res) {  
  res.render('hike', {title: 'My Hiking Log'});  
};  
  
exports.add_hike = function(req, res) {  
};
```

5. Update `node-express/app.js` to include three new lines.

First, add the following line to add a `require` for this route:

```
hike = require('./routes/hike');
```

Your file should look similar to the following snippet:

```
var express = require('express');  
var path = require('path');  
var hike = require('./routes/hike');
```

Then, add the following two lines to `node-express/app.js` after `var app = express();`

```
app.get('/hikes', hike.index);  
app.post('/add_hike', hike.add_hike);
```

Your file should look similar to the following snippet:

```
var app = express();  
app.get('/hikes', hike.index);  
app.post('/add_hike', hike.add_hike);
```

6. Copy `node-express/views/index.jade` to `node-express/views/hike.jade`.

```
~/node-express$ cp views/index.jade views/hike.jade
```

7. Add your files to the local repository, commit your changes, and deploy your updated application.

```
~/node-express$ git add .  
~/node-express$ git commit -m "Add hikes route and template."  
~/node-express$ eb deploy
```

8. Your environment will be updated after a few minutes. After your environment is green and ready, verify it worked by refreshing your browser and appending `hikes` at the end of the URL (e.g., `http://node-express-env-syypntcz2q.elasticbeanstalk.com/hikes`).

You should see a web page titled **My Hiking Log**.

Clean up

If you are done working with Elastic Beanstalk, you can terminate your environment.

Use the **eb terminate** command to terminate your environment and all of the resources that it contains.

```
~/node-express$ eb terminate  
The environment "node-express-env" and all associated instances will be terminated.  
To confirm, type the environment name: node-express-env  
INFO: terminateEnvironment is starting.  
...
```

Deploying an Express application with clustering to Elastic Beanstalk

This tutorial walks you through deploying a sample application to Elastic Beanstalk using the Elastic Beanstalk Command Line Interface (EB CLI), and then updating the application to use the [Express](#) framework, [Amazon ElastiCache](#), and clustering. Clustering enhances your web application's high availability, performance, and security. To learn more about Amazon ElastiCache, go to [What Is Amazon ElastiCache for Memcached?](#) in the *Amazon ElastiCache for Memcached User Guide*.

Note

This example creates AWS resources, which you might be charged for. For more information about AWS pricing, see <https://aws.amazon.com/pricing/>. Some services are part of the AWS Free Usage Tier. If you are a new customer, you can test drive these services for free. See <https://aws.amazon.com/free/> for more information.

Prerequisites

This tutorial requires the Node.js language, its package manager called npm, and the Express web application framework. For details on installing these components and setting up your local development environment, see [Setting up your Node.js development environment \(p. 193\)](#).

Note

For this tutorial, you don't need to install the AWS SDK for Node.js, which is also mentioned in [Setting up your Node.js development environment \(p. 193\)](#).

The tutorial also requires the Elastic Beanstalk Command Line Interface (EB CLI). For details on installing and configuring the EB CLI, see [Install the EB CLI \(p. 853\)](#) and [Configure the EB CLI \(p. 860\)](#).

Create an Elastic Beanstalk environment

Configure an EB CLI repository for your application and create an Elastic Beanstalk environment running the Node.js platform.

1. Create a repository with the **eb init** command.

```
~/node-express$ eb init --platform node.js --region us-east-2  
Application node-express has been created.
```

This command creates a configuration file in a folder named `.elasticbeanstalk` that specifies settings for creating environments for your application, and creates an Elastic Beanstalk application named after the current folder.

2. Create an environment running a sample application with the **eb create** command.

```
~/node-express$ eb create --sample node-express-env
```

This command creates a load balanced environment with the default settings for the Node.js platform and the following resources:

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or nginx as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.

- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
 - **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
 - **Domain name** – A domain name that routes to your web app in the form `subdomain.region.elasticbeanstalk.com`.
3. When environment creation completes, use the **eb open** command to open the environment's URL in the default browser.

```
~/node-express$ eb open
```

Update the application

Update the sample application in the Elastic Beanstalk environment to use the Express framework.

You can download the final source code from [nodejs-example-express-elasticache.zip](#).

Note

The prerequisite development environment setup results in an Express project structure in the `node-express` folder. If you haven't already generated an Express project, run the following command. For more details, see [Installing Express \(p. 194\)](#).

```
~/node-express$ express && npm install
```

To update your application to use Express

1. Rename `node-express/app.js` to `node-express/express-app.js`.

```
node-express$ mv app.js express-app.js
```

2. Update the line `var app = express();` in `node-express/express-app.js` to the following:

```
var app = module.exports = express();
```

3. On your local computer, create a file named `node-express/app.js` with the following code.

```
var cluster = require('cluster'),
    app = require('./express-app');

var workers = {},
    count = require('os').cpus().length;

function spawn(){
  var worker = cluster.fork();
  workers[worker.pid] = worker;
  return worker;
}

if (cluster.isMaster) {
  for (var i = 0; i < count; i++) {
    spawn();
  }
  cluster.on('death', function(worker) {
    console.log('worker ' + worker.pid + ' died. spawning a new process...');
    delete workers[worker.pid];
  });
}
```



```
    spawn();  
  });  
} else {  
  app.listen(process.env.PORT || 5000);  
}
```

4. Deploy the updated application.

```
node-express$ eb deploy
```

5. Your environment will be updated after a few minutes. Once the environment is green and ready, refresh the URL to verify it worked. You should see a web page that says "Welcome to Express".

You can access the logs for your EC2 instances running your application. For instructions on accessing your logs, see [Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment \(p. 732\)](#).

Next, let's update the Express application to use Amazon ElastiCache.

To update your Express application to use Amazon ElastiCache

1. On your local computer, create an `.ebextensions` directory in the top-level directory of your source bundle. In this example, we use `node-express/.ebextensions`.
2. Create a configuration file `node-express/.ebextensions/elasticache-iam-with-script.config` with the following snippet. For more information about the configuration file, see [Node.js configuration namespace \(p. 197\)](#). This creates an IAM user with the permissions required to discover the elasticache nodes and writes to a file anytime the cache changes. You can also copy the file from [nodejs-example-express-elasticache.zip](#). For more information on the ElastiCache properties, see [Example: ElastiCache \(p. 628\)](#).

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

```
Resources:  
  MyCacheSecurityGroup:  
    Type: 'AWS::EC2::SecurityGroup'  
    Properties:  
      GroupDescription: "Lock cache down to webserver access only"  
      SecurityGroupIngress:  
        - IpProtocol: tcp  
          FromPort:  
            Fn::GetOptionSetting:  
              OptionName: CachePort  
              DefaultValue: 11211  
          ToPort:  
            Fn::GetOptionSetting:  
              OptionName: CachePort  
              DefaultValue: 11211  
          SourceSecurityGroupName:  
            Ref: AWSEBSecurityGroup  
  MyElastiCache:  
    Type: 'AWS::ElastiCache::CacheCluster'  
    Properties:  
      CacheNodeType:  
        Fn::GetOptionSetting:  
          OptionName: CacheNodeType  
          DefaultValue: cache.t2.micro  
      NumCacheNodes:  
        Fn::GetOptionSetting:  
          OptionName: NumCacheNodes
```

```
        DefaultValue: 1
Engine:
  Fn::GetOptionSetting:
    OptionName: Engine
    DefaultValue: redis
VpcSecurityGroupIds:
  -
    Fn::GetAtt:
      - MyCacheSecurityGroup
      - GroupId
AWSEBAutoScalingGroup :
Metadata :
  ElastiCacheConfig :
    CacheName :
      Ref : MyElastiCache
    CacheSize :
      Fn::GetOptionSetting:
        OptionName : NumCacheNodes
        DefaultValue: 1
WebServerUser :
Type : AWS::IAM::User
Properties :
  Path : "/"
Policies:
  -
    PolicyName: root
    PolicyDocument :
      Statement :
        -
          Effect : Allow
          Action :
            - cloudformation:DescribeStackResource
            - cloudformation:ListStackResources
            - elasticache:DescribeCacheClusters
          Resource : "*"
WebServerKeys :
Type : AWS::IAM::AccessKey
Properties :
  UserName :
    Ref: WebServerUser
Outputs:
WebsiteURL:
  Description: sample output only here to show inline string function parsing
  Value: |
    http://`${Fn::GetAtt` : [ "AWSEBLoadBalancer", "DNSName" ] }`
MyElastiCacheName:
  Description: Name of the elasticache
  Value:
    Ref : MyElastiCache
NumCacheNodes:
  Description: Number of cache nodes in MyElastiCache
  Value:
    Fn::GetOptionSetting:
      OptionName : NumCacheNodes
      DefaultValue: 1
files:
"/etc/cfn/cfn-credentials" :
  content : |
    AWSSecretKey=${Fn::GetAtt` : ["WebServerKeys", "SecretAccessKey"] }`
    mode : "000400"
  owner : root
  group : root
```

```
"/etc/cfn/get-cache-nodes" :
  content : |
    # Define environment variables for command line tools
    export AWS_ELASTICACHE_HOME="/home/ec2-user/elasticache/$(ls /home/ec2-user/
elasticache/)"
    export AWS_CLOUDFORMATION_HOME=/opt/aws/apitools/cfn
    export PATH=$AWS_CLOUDFORMATION_HOME/bin:$AWS_ELASTICACHE_HOME/bin:$PATH
    export AWS_CREDENTIAL_FILE=/etc/cfn/cfn-credentials
    export JAVA_HOME=/usr/lib/jvm/jre

    # Grab the Cache node names and configure the PHP page
    aws cloudformation list-stack-resources --stack `{ "Ref" : "AWS::StackName" }`
--region `{ "Ref" : "AWS::Region" }` --output text | grep MyElasticache | awk '{print
$4}' | xargs -I {} aws elasticache describe-cache-clusters --cache-cluster-id {}
--region `{ "Ref" : "AWS::Region" }` --show-cache-node-info --output text | grep
'^ENDPOINT' | awk '{print $2 ":" $3}' > `{ "Fn::GetOptionSetting" : { "OptionName" :
"NodeListPath", "DefaultValue" : "/var/www/html/nodelist" } }`
    mode : "000500"
    owner : root
    group : root

"/etc/cfn/hooks.d/cfn-cache-change.conf" :
  "content": |
    [cfn-cache-size-change]
    triggers=post.update
    path=Resources.AWSEBAutoScalingGroup.Metadata.ElasticacheConfig
    action=/etc/cfn/get-cache-nodes
    runas=root

sources :
  "/home/ec2-user/elasticache" : "https://elasticache-downloads.s3.amazonaws.com/
AmazonElasticacheCli-latest.zip"

commands:
  make-elasticache-executable:
    command: chmod -R ugo+x /home/ec2-user/elasticache/*/bin/*

packages :
  "yum" :
    "aws-apitools-cfn" : []

container_commands:
  initial_cache_nodes:
    command: /etc/cfn/get-cache-nodes
```

3. On your local computer, create a configuration file `node-express/.ebextensions/elasticache_settings.config` with the following snippet to configure ElastiCache.

```
option_settings:
  "aws:elasticbeanstalk:customoption":
    CacheNodeType: cache.t2.micro
    NumCacheNodes: 1
    Engine: memcached
    NodeListPath: /var/nodelist
```

4. On your local computer, replace `node-express/express-app.js` with the following snippet. This file reads the nodes list from disk (`/var/nodelist`) and configures express to use memcached as a session store if nodes are present. Your file should look like the following.

```
/**
 * Module dependencies.
 */
var express = require('express'),
```

```
    session = require('express-session'),
    bodyParser = require('body-parser'),
    methodOverride = require('method-override'),
    cookieParser = require('cookie-parser'),
    fs = require('fs'),
    filename = '/var/nodelist',
    app = module.exports = express();

var MemcachedStore = require('connect-memcached')(session);

function setup(cacheNodes) {
  app.use(bodyParser.raw());
  app.use(methodOverride());
  if (cacheNodes) {
    app.use(cookieParser());

    console.log('Using memcached store nodes:');
    console.log(cacheNodes);

    app.use(session({
      secret: 'your secret here',
      resave: false,
      saveUninitialized: false,
      store: new MemcachedStore({'hosts': cacheNodes})
    }));
  } else {
    console.log('Not using memcached store. ');
    app.use(cookieParser('your secret here'));
    app.use(session());
  }
}

app.get('/', function(req, resp){
  if (req.session.views) {
    req.session.views++
    resp.setHeader('Content-Type', 'text/html')
    resp.write('Views: ' + req.session.views)
    resp.end()
  } else {
    req.session.views = 1
    resp.end('Refresh the page!')
  }
});

if (!module.parent) {
  console.log('Running express without cluster. ');
  app.listen(process.env.PORT || 5000);
}

// Load elasticache configuration.
fs.readFile(filename, 'UTF8', function(err, data) {
  if (err) throw err;

  var cacheNodes = [];
  if (data) {
    var lines = data.split('\n');
    for (var i = 0 ; i < lines.length ; i++) {
      if (lines[i].length > 0) {
        cacheNodes.push(lines[i]);
      }
    }
  }
  setup(cacheNodes);
});
```

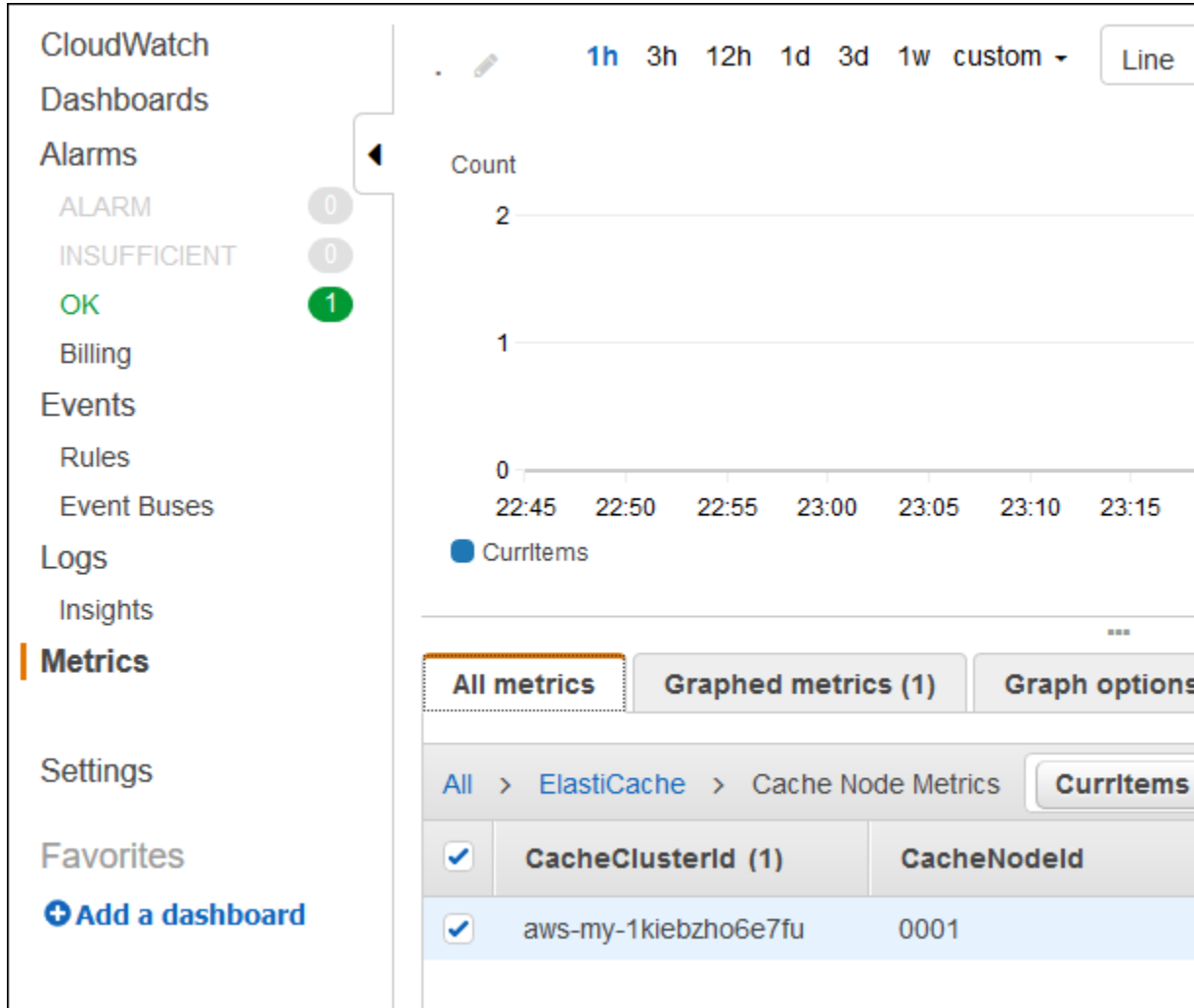
5. On your local computer, update `node-express/package.json` to add four dependencies.

```
{
  "name": "node-express",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "cookie-parser": "*",
    "debug": "~2.6.9",
    "express": "~4.16.0",
    "http-errors": "~1.6.2",
    "jade": "~1.11.0",
    "morgan": "~1.9.0",
    "connect-memcached": "*",
    "express-session": "*",
    "body-parser": "*",
    "method-override": "*"
  }
}
```

6. Deploy the updated application.

```
node-express$ eb deploy
```

7. Your environment will be updated after a few minutes. After your environment is green and ready, verify that the code worked.
 - a. Check the [Amazon CloudWatch console](#) to view your ElastiCache metrics. To view your ElastiCache metrics, select **Metrics** in the left pane, and then search for **CurItems**. Select **ElastiCache > Cache Node Metrics**, and then select your cache node to view the number of items in the cache.



Note

Make sure you are looking at the same region that you deployed your application to.

If you copy and paste your application URL into another web browser and refresh the page, you should see your CurrItem count go up after 5 minutes.

- b. Take a snapshot of your logs. For more information about retrieving logs, see [Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment \(p. 732\)](#).
- c. Check the file `/var/log/nodejs/nodejs.log` in the log bundle. You should see something similar to the following:

```
Using memcached store nodes:  
[ 'aws-my-1oys9co8zt1uo.1iwtrn.0001.use1.cache.amazonaws.com:11211' ]
```

Clean up

If you no longer want to run your application, you can clean up by terminating your environment and deleting your application.

Use the `eb terminate` command to terminate your environment and the `eb delete` command to delete your application.

To terminate your environment

From the directory where you created your local repository, run `eb terminate`.

```
$ eb terminate
```

This process can take a few minutes. Elastic Beanstalk displays a message once the environment is successfully terminated.

Deploying a Node.js application with DynamoDB to Elastic Beanstalk

This tutorial and [sample application](#) walks you through the process of deploying a Node.js application that uses the AWS SDK for JavaScript in Node.js to interact with Amazon DynamoDB. You'll create a DynamoDB table that is external to the AWS Elastic Beanstalk environment, and configure the application to use this external table instead of creating one in the environment. In a production environment, you keep the table independent of the Elastic Beanstalk environment to protect against accidental data loss and enable you to perform [blue/green deployments](#) (p. 405).

The tutorial's sample application uses a DynamoDB table to store user-provided text data. The sample application uses [configuration files](#) (p. 600) to create the table and an Amazon Simple Notification Service topic. It also shows how to use a [package.json file](#) (p. 200) to install packages during deployment.

Sections

- [Prerequisites](#) (p. 216)
- [Launch an Elastic Beanstalk environment](#) (p. 217)
- [Add permissions to your environment's instances](#) (p. 218)
- [Deploy the sample application](#) (p. 219)
- [Create a DynamoDB table](#) (p. 221)
- [Update the application's configuration files](#) (p. 221)
- [Configure your environment for high availability](#) (p. 224)
- [Cleanup](#) (p. 224)
- [Next steps](#) (p. 224)

Prerequisites

- Before you start, download the sample application source bundle from GitHub: [eb-node-express-sample-v1.1.zip](#).
- You will also need a command line terminal or shell to run the commands in the procedures. Example commands are preceded by a prompt symbol (\$) and the name of the current directory, when appropriate:

```
~/eb-project$ this is a command  
this is output
```

Note

You can run all commands in this tutorial on a Linux virtual machine, an OS X machine, or an Amazon EC2 instance running Amazon Linux. If you need a development environment, you can launch a single-instance Elastic Beanstalk environment and connect to it with SSH.

- This tutorial uses a command line ZIP utility to create a source bundle that you can deploy to Elastic Beanstalk. To use the `zip` command in Windows, you can install `UnxUtils`, a lightweight collection of useful command-line utilities like `zip` and `ls`. (Alternatively, you can [use Windows Explorer \(p. 343\)](#) or any other ZIP utility to create source bundle archives.)

To install UnxUtils

1. Download [UnxUtils](#).
2. Extract the archive to a local directory. For example, `C:\Program Files (x86)`.
3. Add the path to the binaries to your Windows PATH user variable. For example, `C:\Program Files (x86)\UnxUtils\usr\local\wbin`.
 - a. Press the Windows key, and then enter **environment variables**.
 - b. Choose **Edit environment variables for your account**.
 - c. Choose **PATH**, and then choose **Edit**.
 - d. Add paths to the **Variable value** field, separated by semicolons. For example: `C:\item1\path;C:\item2\path`
 - e. Choose **OK** twice to apply the new settings.
 - f. Close any running Command Prompt windows, and then reopen a Command Prompt window.
4. Open a new command prompt window and run the `zip` command to verify that it works.

```
> zip -h
Copyright (C) 1990-1999 Info-ZIP
Type 'zip -L' for software license.
...
```

Launch an Elastic Beanstalk environment

You use the Elastic Beanstalk console to launch an Elastic Beanstalk environment. You'll choose the **Node.js** platform and accept the default settings and sample code. After you configure the environment's permissions, you deploy the sample application that you downloaded from GitHub.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link:
console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Elastic Beanstalk takes about five minutes to create the environment with the following resources:

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or nginx as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
- **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

Elastic Beanstalk manages all of these resources. When you terminate your environment, Elastic Beanstalk terminates all of the resources that it contains.

Note

The S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see [Using Elastic Beanstalk with Amazon S3 \(p. 831\)](#).

Add permissions to your environment's instances

Your application runs on one or more EC2 instances behind a load balancer, serving HTTP requests from the Internet. When it receives a request that requires it to use AWS services, the application uses the permissions of the instance it runs on to access those services.

The sample application uses instance permissions to write data to a DynamoDB table, and to send notifications to an Amazon SNS topic with the SDK for JavaScript in Node.js. Add the following managed policies to the default [instance profile \(p. 21\)](#) to grant the EC2 instances in your environment permission to access DynamoDB and Amazon SNS:

- **AmazonDynamoDBFullAccess**
- **AmazonSNSFullAccess**

To add policies to the default instance profile

1. Open the [Roles page](#) in the IAM console.
2. Choose **aws-elasticbeanstalk-ec2-role**.
3. On the **Permissions** tab, choose **Attach policies**.

4. Select the managed policy for the additional services that your application uses. For example, `AmazonSNSFullAccess` or `AmazonDynamoDBFullAccess`.
5. Choose **Attach policy**.

See [Managing Elastic Beanstalk instance profiles \(p. 760\)](#) for more on managing instance profiles.

Deploy the sample application

Now your environment is ready for you to deploy the sample application to it and then run it.

Note

Download the source bundle from GitHub if you haven't already: [eb-node-express-sample-v1.1.zip](#).

To deploy a source bundle

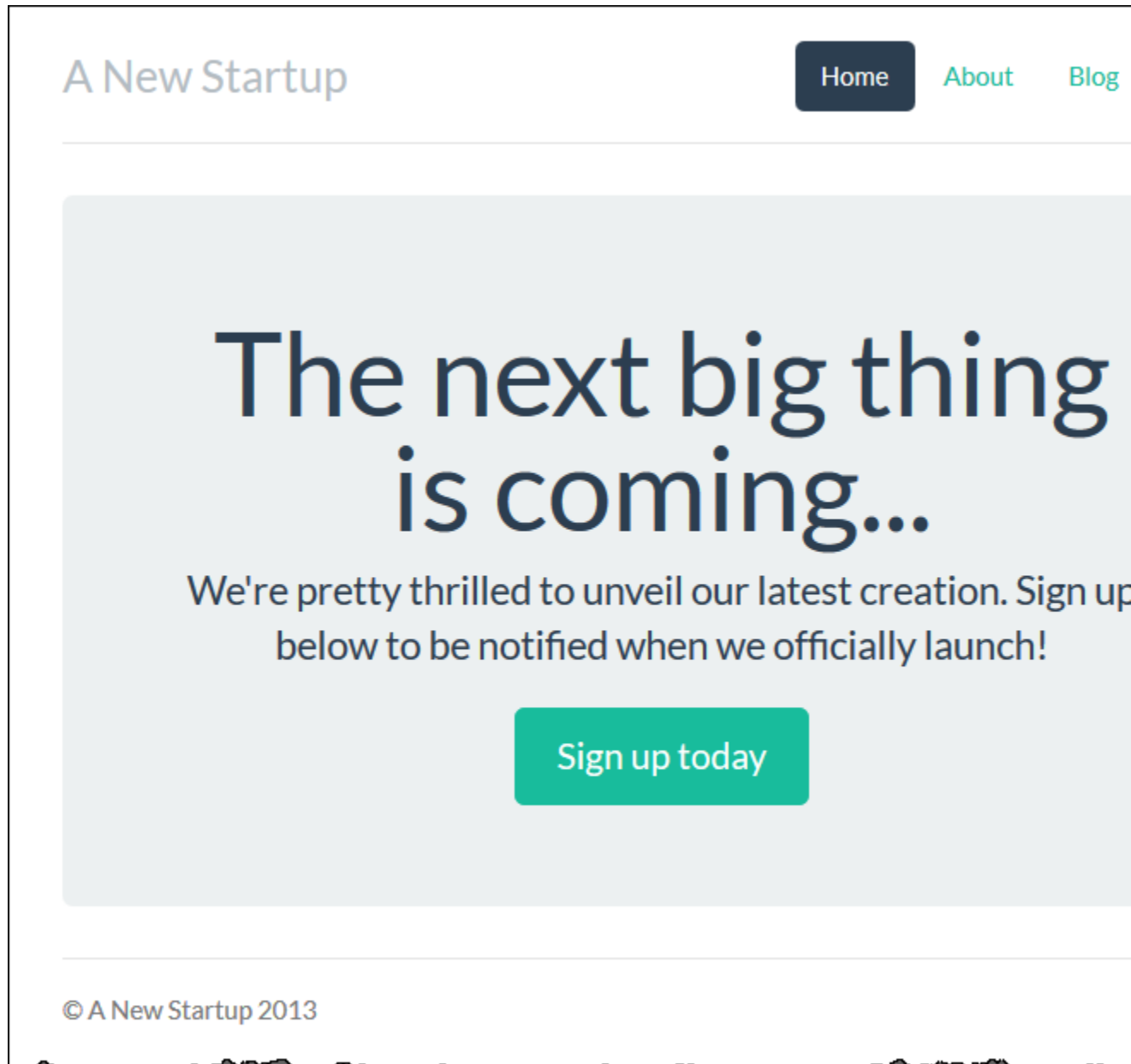
1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

The site collects user contact information and uses a DynamoDB table to store the data. To add an entry, choose **Sign up today**, enter a name and email address, and then choose **Sign Up!**. The web app writes the form contents to the table and triggers an Amazon SNS email notification.



Right now, the Amazon SNS topic is configured with a placeholder email for notifications. You will update the configuration soon, but in the meantime you can verify the DynamoDB table and Amazon SNS topic in the AWS Management Console.

To view the table

1. Open the [Tables page](#) in the DynamoDB console.
2. Find the table that the application created. The name starts with **awseb** and contains **StartupSignupsTable**.
3. Select the table, choose **Items**, and then choose **Start search** to view all items in the table.

The table contains an entry for every email address submitted on the signup site. In addition to writing to the table, the application sends a message to an Amazon SNS topic that has two subscriptions, one for email notifications to you, and another for an Amazon Simple Queue Service queue that a worker application can read from to process requests and send emails to interested customers.

To view the topic

1. Open the [Topics page](#) in the Amazon SNS console.
2. Find the topic that the application created. The name starts with **awseb** and contains **NewSignupTopic**.
3. Choose the topic to view its subscriptions.

The application ([app.js](#)) defines two routes. The root path (/) returns a webpage rendered from an Embedded JavaScript (EJS) template with a form that the user fills out to register their name and email address. Submitting the form sends a POST request with the form data to the `/signup` route, which writes an entry to the DynamoDB table and publishes a message to the Amazon SNS topic to notify the owner of the signup.

The sample application includes [configuration files \(p. 600\)](#) that create the DynamoDB table, Amazon SNS topic, and Amazon SQS queue used by the application. This lets you create a new environment and test the functionality immediately, but has the drawback of tying the DynamoDB table to the environment. For a production environment, you should create the DynamoDB table outside of your environment to avoid losing it when you terminate the environment or update its configuration.

Create a DynamoDB table

To use an external DynamoDB table with an application running in Elastic Beanstalk, first create a table in DynamoDB. When you create a table outside of Elastic Beanstalk, it is completely independent of Elastic Beanstalk and your Elastic Beanstalk environments, and will not be terminated by Elastic Beanstalk.

Create a table with the following settings:

- **Table name** – `nodejs-tutorial`
- **Primary key** – `email`
- Primary key type – **String**

To create a DynamoDB table

1. Open the [Tables page](#) in the DynamoDB management console.
2. Choose **Create table**.
3. Type a **Table name** and **Primary key**.
4. Choose the primary key type.
5. Choose **Create**.

Update the application's configuration files

Update the [configuration files \(p. 600\)](#) in the application source to use the `nodejs-tutorial` table instead of creating a new one.

To update the sample application for production use

1. Extract the project files from the source bundle:

```
~$ mkdir nodejs-tutorial
~$ cd nodejs-tutorial
~/nodejs-tutorial$ unzip ~/Downloads/eb-node-express-sample-v1.0.zip
```

2. Open `.ebextensions/options.config` and change the values of the following settings:

- **NewSignupEmail** – Your email address.
- **STARTUP_SIGNUP_TABLE** – `nodejs-tutorial`

Example `.ebextensions/options.config`

```
option_settings:
  aws:elasticbeanstalk:customoption:
    NewSignupEmail: you@example.com
  aws:elasticbeanstalk:application:environment:
    THEME: "flatly"
    AWS_REGION: `{"Ref" : "AWS::Region"}`
    STARTUP_SIGNUP_TABLE: nodejs-tutorial
    NEW_SIGNUP_TOPIC: `{"Ref" : "NewSignupTopic"}`
  aws:elasticbeanstalk:container:nodejs:
    ProxyServer: nginx
  aws:elasticbeanstalk:container:nodejs:staticfiles:
    /static: /static
  aws:autoscaling:asg:
    Cooldown: "120"
  aws:autoscaling:trigger:
    Unit: "Percent"
    Period: "1"
    BreachDuration: "2"
    UpperThreshold: "75"
    LowerThreshold: "30"
    MeasureName: "CPUUtilization"
```

This configures the application to use the `nodejs-tutorial` table instead of the one created by `.ebextensions/create-dynamodb-table.config`, and sets the email address that the Amazon SNS topic uses for notifications.

3. Remove `.ebextensions/create-dynamodb-table.config`.

```
~/nodejs-tutorial$ rm .ebextensions/create-dynamodb-table.config
```

The next time you deploy the application, the table created by this configuration file will be deleted.

4. Create a source bundle from the modified code.

```
~/nodejs-tutorial$ zip nodejs-tutorial.zip -r * .[^.]*
adding: LICENSE (deflated 65%)
adding: README.md (deflated 56%)
adding: app.js (deflated 63%)
adding: iam_policy.json (deflated 47%)
adding: misc/ (stored 0%)
adding: misc/theme-flow.png (deflated 1%)
adding: npm-shrinkwrap.json (deflated 87%)
adding: package.json (deflated 40%)
adding: static/ (stored 0%)
adding: static/bootstrap/ (stored 0%)
adding: static/bootstrap/css/ (stored 0%)
adding: static/bootstrap/css/jumbotron-narrow.css (deflated 59%)
adding: static/bootstrap/css/theme/ (stored 0%)
adding: static/bootstrap/css/theme/united/ (stored 0%)
```

```
adding: static/bootstrap/css/theme/united/bootstrap.css (deflated 86%)
adding: static/bootstrap/css/theme/amelia/ (stored 0%)
adding: static/bootstrap/css/theme/amelia/bootstrap.css (deflated 86%)
adding: static/bootstrap/css/theme/slate/ (stored 0%)
adding: static/bootstrap/css/theme/slate/bootstrap.css (deflated 87%)
adding: static/bootstrap/css/theme/default/ (stored 0%)
adding: static/bootstrap/css/theme/default/bootstrap.css (deflated 86%)
adding: static/bootstrap/css/theme/flatly/ (stored 0%)
adding: static/bootstrap/css/theme/flatly/bootstrap.css (deflated 86%)
adding: static/bootstrap/LICENSE (deflated 65%)
adding: static/bootstrap/js/ (stored 0%)
adding: static/bootstrap/js/bootstrap.min.js (deflated 74%)
adding: static/bootstrap/fonts/ (stored 0%)
adding: static/bootstrap/fonts/glyphicons-halflings-regular.eot (deflated 1%)
adding: static/bootstrap/fonts/glyphicons-halflings-regular.svg (deflated 73%)
adding: static/bootstrap/fonts/glyphicons-halflings-regular.woff (deflated 1%)
adding: static/bootstrap/fonts/glyphicons-halflings-regular.ttf (deflated 44%)
adding: static/jquery/ (stored 0%)
adding: static/jquery/jquery-1.11.3.min.js (deflated 65%)
adding: static/jquery/MIT-LICENSE.txt (deflated 41%)
adding: views/ (stored 0%)
adding: views/index.ejs (deflated 67%)
adding: .ebextensions/ (stored 0%)
adding: .ebextensions/options.config (deflated 47%)
adding: .ebextensions/create-sns-topic.config (deflated 56%)
```

Deploy the `nodejs-tutorial.zip` source bundle to your environment.

To deploy a source bundle

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

When you deploy, Elastic Beanstalk updates the configuration of the Amazon SNS topic and deletes the DynamoDB table that it created when you deployed the first version of the application.

Now, when you terminate the environment, the `nodejs-tutorial` table will not be deleted. This lets you perform blue/green deployments, modify configuration files, or take down your website without risking data loss.

Open your site in a browser and verify that the form works as you expect. Create a few entries, and then check the DynamoDB console to verify the table.

To view the table

1. Open the [Tables page](#) in the DynamoDB console.
2. Find the `nodejs-tutorial` table.
3. Select the table, choose **Items**, and then choose **Start search** to view all items in the table.

You can also see that Elastic Beanstalk deleted the table that it created previously.

Configure your environment for high availability

Finally, configure your environment's Auto Scaling group with a higher minimum instance count. Run at least two instances at all times to prevent the web servers in your environment from being a single point of failure, and to allow you to deploy changes without taking your site out of service.

To configure your environment's Auto Scaling group for high availability

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Auto Scaling group** section, set **Min instances** to **2**.
6. Choose **Apply**.

Cleanup

When you finish working with Elastic Beanstalk, you can terminate your environment. Elastic Beanstalk terminates all AWS resources associated with your environment, such as [Amazon EC2 instances \(p. 451\)](#), [database instances \(p. 506\)](#), [load balancers \(p. 470\)](#), security groups, and [alarms \(p. \)](#).

To terminate your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

With Elastic Beanstalk, you can easily create a new environment for your application at any time.

You can also delete the external DynamoDB tables that you created.

To delete a DynamoDB table

1. Open the [Tables page](#) in the DynamoDB console.
2. Select a table.
3. Choose **Actions**, and then choose **Delete table**.
4. Choose **Delete**.

Next steps

As you continue to develop your application, you'll probably want to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The [Elastic Beanstalk Command Line Interface \(p. 852\)](#) (EB CLI) provides easy-to-use commands for

creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

The sample application uses configuration files to configure software settings and create AWS resources as part of your environment. See [Advanced environment customization with configuration files \(.ebextensions\) \(p. 600\)](#) for more information about configuration files and their use.

The sample application for this tutorial uses the Express web framework for Node.js. For more information about Express, see the official documentation at expressjs.com.

Finally, if you plan on using your application in a production environment, [configure a custom domain name \(p. 534\)](#) for your environment and [enable HTTPS \(p. 652\)](#) for secure connections.

Adding an Amazon RDS DB instance to your Node.js application environment

You can use an Amazon Relational Database Service (Amazon RDS) DB instance to store data gathered and modified by your application. The database can be attached to your environment and managed by Elastic Beanstalk, or created and managed externally.

If you are using Amazon RDS for the first time, [add a DB instance \(p. 225\)](#) to a test environment with the Elastic Beanstalk Management Console and verify that your application is able to connect to it.

To connect to a database, [add the driver \(p. 226\)](#) to your application, load the driver in your code, and [create a connection object \(p. 227\)](#) with the environment properties provided by Elastic Beanstalk. The configuration and connection code vary depending on the database engine and framework that you use.

Note

For learning purposes or test environments, you can use Elastic Beanstalk to add a DB instance. For production environments, you can create a DB instance outside of your Elastic Beanstalk environment to decouple your environment resources from your database resources. This way, when you terminate your environment, the DB instance isn't deleted. An external DB instance also lets you connect to the same database from multiple environments and perform [blue-green deployments](#). For instructions, see [Using Elastic Beanstalk with Amazon RDS \(p. 820\)](#).

Sections

- [Adding a DB instance to your environment \(p. 225\)](#)
- [Downloading a driver \(p. 226\)](#)
- [Connecting to a database \(p. 227\)](#)

Adding a DB instance to your environment

To add a DB instance to your environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Choose a DB engine, and enter a user name and password.
6. Choose **Apply**.

Adding a DB instance takes about 10 minutes. When the environment update is complete, the DB instance's hostname and other connection information are available to your application through the following environment properties:

Property name	Description	Property value
RDS_HOSTNAME	The hostname of the DB instance.	On the Connectivity & security tab on the Amazon RDS console: Endpoint .
RDS_PORT	The port on which the DB instance accepts connections. The default value varies among DB engines.	On the Connectivity & security tab on the Amazon RDS console: Port .
RDS_DB_NAME	The database name, ebddb .	On the Configuration tab on the Amazon RDS console: DB Name .
RDS_USERNAME	The username that you configured for your database.	On the Configuration tab on the Amazon RDS console: Master username .
RDS_PASSWORD	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

For more information about configuring an internal DB instance, see [Adding a database to your Elastic Beanstalk environment \(p. 506\)](#).

Downloading a driver

Add the database driver to your project's [package.json file \(p. 200\)](#) under dependencies.

Example package.json – Express with MySQL

```
{
  "name": "my-app",
  "version": "0.0.1",
  "private": true,
  "dependencies": {
    "ejs": "latest",
    "aws-sdk": "latest",
    "express": "latest",
    "body-parser": "latest",
    "mysql": "latest"
  },
  "scripts": {
    "start": "node app.js"
  }
}
```

Common driver packages for Node.js

- **MySQL** – `mysql`
- **PostgreSQL** – `pg`
- **SQL Server** – `mssql`
- **Oracle** – `oracle` or `oracledb`

The Oracle package and version depend on the Node.js version you're using:

- **Node.js 6.x, 8.x** – Use the latest version of `oracledb`.
- **Node.js 4.x** – Use the `oracledb` version 2.2.0.
- **Node.js 5.x, 7.x** – Use the latest version of `oracle`. The `oracledb` package doesn't support these Node.js versions.

Connecting to a database

Elastic Beanstalk provides connection information for attached DB instances in environment properties. Use `process.env.VARIABLE` to read the properties and configure a database connection.

Example `app.js` – MySQL database connection

```
var mysql = require('mysql');

var connection = mysql.createConnection({
  host      : process.env.RDS_HOSTNAME,
  user      : process.env.RDS_USERNAME,
  password  : process.env.RDS_PASSWORD,
  port      : process.env.RDS_PORT
});

connection.connect(function(err) {
  if (err) {
    console.error('Database connection failed: ' + err.stack);
    return;
  }

  console.log('Connected to database.');
```

For more information about constructing a connection string using `node-mysql`, see npmjs.org/package/mysql.

Resources

There are several places you can go to get additional help when developing your Node.js applications:

Resource	Description
GitHub	Install the AWS SDK for Node.js using GitHub.
Node.js Development Forum	Post your questions and get feedback.
AWS SDK for Node.js (Developer Preview)	One-stop shop for sample code, documentation, tools, and additional resources.

Creating and deploying PHP applications on Elastic Beanstalk

AWS Elastic Beanstalk for PHP makes it easy to deploy, manage, and scale your PHP web applications using Amazon Web Services. Elastic Beanstalk for PHP is available to anyone developing or hosting a

web application using PHP. This section provides instructions for deploying your PHP web application to Elastic Beanstalk. You can deploy your application in just a few minutes using the Elastic Beanstalk Command Line Interface (EB CLI) or by using the Elastic Beanstalk Management Console. It also provides walkthroughs for common frameworks such as CakePHP and Symfony.

The topics in this chapter assume some knowledge of Elastic Beanstalk environments. If you haven't used Elastic Beanstalk before, try the [getting started tutorial \(p. 3\)](#) to learn the basics.

If you need help with PHP application development, there are several places you can go:

Resource	Description
GitHub	Install the AWS SDK for PHP using GitHub.
PHP Development Forum	Post your questions and get feedback.
PHP Developer Center	One-stop shop for sample code, documentation, tools, and additional resources.
AWS SDK for PHP FAQs	Get answers to commonly asked questions.

Topics

- [Setting up your PHP development environment \(p. 228\)](#)
- [Using the Elastic Beanstalk PHP platform \(p. 230\)](#)
- [Deploying a Laravel application to Elastic Beanstalk \(p. 234\)](#)
- [Deploying a CakePHP application to Elastic Beanstalk \(p. 241\)](#)
- [Deploying a Symfony application to Elastic Beanstalk \(p. 249\)](#)
- [Deploying a high-availability PHP application with an external Amazon RDS database to Elastic Beanstalk \(p. 253\)](#)
- [Deploying a high-availability WordPress website with an external Amazon RDS database to Elastic Beanstalk \(p. 262\)](#)
- [Deploying a high-availability Drupal website with an external Amazon RDS database to Elastic Beanstalk \(p. 273\)](#)
- [Adding an Amazon RDS DB instance to your PHP application environment \(p. 285\)](#)

Setting up your PHP development environment

Set up a PHP development environment to test your application locally prior to deploying it to AWS Elastic Beanstalk. This topic outlines development environment setup steps and links to installation pages for useful tools.

For common setup steps and tools that apply to all languages, see [Configuring your development machine \(p. 849\)](#).

Sections

- [Installing PHP \(p. 229\)](#)
- [Install Composer \(p. 229\)](#)
- [Installing the AWS SDK for PHP \(p. 230\)](#)
- [Installing an IDE or text editor \(p. 230\)](#)

Installing PHP

Install PHP and some common extensions. If you don't have a preference, get the latest version. Depending on your platform and available package manager, the steps will vary.

On Amazon Linux, use yum:

```
$ sudo yum install php
$ sudo yum install php-mbstring
$ sudo yum install php-intl
```

Note

To get specific PHP package versions that match the version on your Elastic Beanstalk [PHP platform version](#), use the command `yum search php` to find available package versions, such as `php72`, `php72-mbstring`, and `php72-intl`. Then use `sudo yum install package` to install them.

On Ubuntu, use apt:

```
$ sudo apt install php-all-dev
$ sudo apt install php-intl
$ sudo apt install php-mbstring
```

On OS-X, use brew:

```
$ brew install php
$ brew install php-intl
```

Note

To get specific PHP package versions that match the version on your Elastic Beanstalk [PHP platform version](#), see [Homebrew Formulae](#) for available PHP versions, such as `php@7.2`. Then use `brew install package` to install them. Depending on the version, `php-intl` might be included in the main PHP package and not exist as a separate package.

On Windows 10, [install the Windows Subsystem for Linux](#) to get Ubuntu and install PHP with apt. For earlier versions, visit the download page at windows.php.net to get PHP, and read [this page](#) for information about extensions.

After installing PHP, reopen your terminal and run `php --version` to ensure that the new version has been installed and is the default.

Install Composer

Composer is a dependency manager for PHP. You can use it to install libraries, track your application's dependencies, and generate projects for popular PHP frameworks.

Install composer with the PHP script from getcomposer.org.

```
$ curl -s https://getcomposer.org/installer | php
```

The installer generates a PHAR file in the current directory. Move this file to a location in your environment PATH so that you can use it as an executable.

```
$ mv composer.phar ~/.local/bin/composer
```

Install libraries with the `require` command.

```
$ composer require twig/twig
```

Composer adds libraries that you install locally to your project's [composer.json file \(p. 233\)](#). When you deploy your project code, Elastic Beanstalk uses Composer to install the libraries listed in this file on your environment's application instances.

If you run into issues installing Composer, see the [composer documentation](#).

Installing the AWS SDK for PHP

If you need to manage AWS resources from within your application, install the AWS SDK for PHP. For example, with the SDK for PHP, you can use Amazon DynamoDB (DynamoDB) to store user and session information without creating a relational database.

Install the SDK for PHP with Composer.

```
$ composer require aws/aws-sdk-php
```

Visit the [AWS SDK for PHP homepage](#) for more information and installation instructions.

Installing an IDE or text editor

Integrated development environments (IDEs) provide a wide range of features that facilitate application development. If you haven't used an IDE for PHP development, try Eclipse and PhpStorm and see which works best for you.

- [Install Eclipse](#)
- [Install PhpStorm](#)

Note

An IDE might add files to your project folder that you might not want to commit to source control. To prevent committing these files to source control, use `.gitignore` or your source control tool's equivalent.

If you just want to begin coding and don't need all of the features of an IDE, consider [installing Sublime Text](#).

Using the Elastic Beanstalk PHP platform

Important

Amazon Linux 2 platform versions are fundamentally different than Amazon Linux AMI platform versions (preceding Amazon Linux 2). These different platform generations are incompatible in several ways. If you are migrating to an Amazon Linux 2 platform version, be sure to read the information in [the section called "Upgrade to Amazon Linux 2" \(p. 426\)](#).

AWS Elastic Beanstalk supports a number of platforms for different versions of the PHP programming language. These platforms support PHP web applications that can run alone or under Composer. Learn more at [PHP](#) in the *AWS Elastic Beanstalk Platforms* document.

Elastic Beanstalk provides [configuration options \(p. 536\)](#) that you can use to customize the software that runs on the EC2 instances in your Elastic Beanstalk environment. You can configure environment variables needed by your application, enable log rotation to Amazon S3, map folders in your application

source that contain static files to paths served by the proxy server, and set common PHP initialization settings.

Configuration options are available in the Elastic Beanstalk console for [modifying the configuration of a running environment \(p. 547\)](#). To avoid losing your environment's configuration when you terminate it, you can use [saved configurations \(p. 640\)](#) to save your settings and later apply them to another environment.

To save settings in your source code, you can include [configuration files \(p. 600\)](#). Settings in configuration files are applied every time you create an environment or deploy your application. You can also use configuration files to install packages, run scripts, and perform other instance customization operations during deployments.

If you use Composer, you can [include a composer.json file \(p. 233\)](#) in your source bundle to install packages during deployment.

For advanced PHP configuration and PHP settings that are not provided as configuration options, you can [use configuration files to provide an INI file \(p. 234\)](#) that can extend and override the default settings applied by Elastic Beanstalk, or install additional extensions.

Settings applied in the Elastic Beanstalk console override the same settings in configuration files, if they exist. This lets you have default settings in configuration files, and override them with environment-specific settings in the console. For more information about precedence, and other methods of changing settings, see [Configuration options \(p. 536\)](#).

For details about the various ways you can extend an Elastic Beanstalk Linux-based platform, see [the section called "Extending Linux platforms" \(p. 31\)](#).

Configuring your PHP environment

You can use the Elastic Beanstalk console to enable log rotation to Amazon S3, configure variables that your application can read from the environment, and change PHP settings.

To configure your PHP environment in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.

PHP settings

- **Document root** – The folder that contains your site's default page. If your welcome page is not at the root of your source bundle, specify the folder that contains it relative to the root path. For example, `/public` if the welcome page is in a folder named `public`.
- **Memory limit** – The maximum amount of memory that a script is allowed to allocate. For example, `512M`.
- **Zlib output compression** – Set to `On` to compress responses.
- **Allow URL fopen** – Set to `Off` to prevent scripts from downloading files from remote locations.
- **Display errors** – Set to `On` to show internal error messages for debugging.
- **Max execution time** – The maximum time in seconds that a script is allowed to run before the environment terminates it.

Log options

The Log Options section has two settings:

- **Instance profile**– Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3** – Specifies whether log files for your application's Amazon EC2 instances should be copied to the Amazon S3 bucket associated with your application.

Static files

To improve performance, the **Static files** section lets you configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. For each directory, you set the virtual path to directory mapping. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

For details about configuring static files using the Elastic Beanstalk console, see [the section called “Static files” \(p. 650\)](#).

Environment properties

The **Environment Properties** section lets you specify environment configuration settings on the Amazon EC2 instances that are running your application. These settings are passed in as key-value pairs to the application.

Your application code can access environment properties by using `$_SERVER` or the `get_cfg_var` function.

```
$endpoint = $_SERVER['API_ENDPOINT'];
```

See [Environment properties and other software settings \(p. 516\)](#) for more information.

The `aws:elasticbeanstalk:container:php:phpini` namespace

You can use a [configuration file \(p. 600\)](#) to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be defined by the Elastic Beanstalk service or the platform that you use and are organized into *namespaces*.

You can use the `aws:elasticbeanstalk:environment:proxy:staticfiles` namespace to configure the environment proxy to serve static files. You define mappings of virtual paths to application directories.

The PHP platform defines options in the `aws:elasticbeanstalk:container:php:phpini` namespace, including one that is not available in the Elastic Beanstalk console. `composer_options` sets custom options to use when installing dependencies using Composer through `composer.phar install`. For more information including available options, go to <http://getcomposer.org/doc/03-cli.md#install>.

The following example [configuration file \(p. 600\)](#) specifies a static files option that maps a directory named `staticimages` to the path `/images`, and shows settings for each of the options available in the `aws:elasticbeanstalk:container:php:phpini` namespace:

Example `.ebextensions/php-settings.config`

```
option_settings:
```

```
aws:elasticbeanstalk:environment:proxy:staticfiles:
  /images: staticimages
aws:elasticbeanstalk:container:php:phpini:
  document_root: /public
  memory_limit: 128M
  zlib.output_compression: "Off"
  allow_url_fopen: "On"
  display_errors: "Off"
  max_execution_time: 60
  composer_options: vendor/package
```

Note

The `aws:elasticbeanstalk:environment:proxy:staticfiles` namespace isn't defined on Amazon Linux AMI PHP platform branches (preceding Amazon Linux 2).

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See [Configuration options \(p. 536\)](#) for more information.

Installing your application's dependencies

Your application might have dependencies on other PHP packages. You can configure your application to install these dependencies on the environment's Amazon Elastic Compute Cloud (Amazon EC2) instances. Alternatively, you can include your application's dependencies in the source bundle and deploy them with the application. The following section discuss both of these ways.

Use a Composer file to install dependencies on instances

Use a `composer.json` file in the root of your project source to use composer to install packages that your application requires on your environment's Amazon EC2 instances.

Example `composer.json`

```
{
  "require": {
    "monolog/monolog": "1.0.*"
  }
}
```

When a `composer.json` file is present, Elastic Beanstalk runs `composer.phar install` to install dependencies. You can add options to append to the command by setting the [composer_options option \(p. 232\)](#) in the `aws:elasticbeanstalk:container:php:phpini` namespace.

Include dependencies in source bundle

If your application has a large number of dependencies, installing them might take a long time. This can increase deployment and scaling operations, because dependencies are installed on every new instance.

To avoid the negative impact on deployment time, use Composer in your development environment to resolve dependencies and install them into the `vendor` folder.

To include dependencies in your application source bundle

1. Run the following command:

```
% composer install
```


2. Include the generated `vendor` folder in the root of your application source bundle.

When Elastic Beanstalk finds a `vendor` folder on the instance, it ignores the `composer.json` file (even if it exists). Your application then uses dependencies from the `vendor` folder.

Updating Composer

[PHP platform versions](#) ship with the latest version of Composer available at the time of release. To keep Composer, PHP, and other libraries up to date, [upgrade your environment \(p. 415\)](#) whenever a platform update is available.

Between platform updates, you can use a [configuration file \(p. 600\)](#) to update Composer on the instances in your environment. You may need to update Composer if you see an error when you try to install packages with a Composer file, or if you are unable to use the latest platform version.

Example `.ebextensions/composer.config`

```
commands:
  01updateComposer:
    command: export COMPOSER_HOME=/root && /usr/bin/composer.phar self-update 1.4.1

option_settings:
  - namespace: aws:elasticbeanstalk:application:environment
    option_name: COMPOSER_HOME
    value: /root
```

This configuration file configures composer to update itself to version 1.4.1. Check the [Composer releases page](#) on GitHub to find the latest version.

Note

If you omit the version number from the `composer.phar self-update` command, Composer will update to the latest version available every time you deploy your source code, and when new instances are provisioned by Auto Scaling. This could cause scaling operations and deployments to fail if a version of Composer is released that is incompatible with your application.

Extending `php.ini`

Use a configuration file with a `files` block to add a `.ini` file to `/etc/php.d/` on the instances in your environment. The main configuration file, `php.ini`, pulls in settings from files in this folder in alphabetical order. Many extensions are enabled by default by files in this folder.

Example `.ebextensions/mongo.config`

```
files:
  "/etc/php.d/99mongo.ini" :
    mode: "000755"
    owner: root
    group: root
    content: |
      extension=mongo.so
```

Deploying a Laravel application to Elastic Beanstalk

Laravel is an open source, model-view-controller (MVC) framework for PHP. This tutorial walks you through the process of generating a Laravel application, deploying it to an AWS Elastic Beanstalk

environment, and configuring it to connect to an Amazon Relational Database Service (Amazon RDS) database instance.

Sections

- [Prerequisites \(p. 235\)](#)
- [Launch an Elastic Beanstalk environment \(p. 235\)](#)
- [Install Laravel and generate a website \(p. 236\)](#)
- [Deploy your application \(p. 237\)](#)
- [Configure Composer settings \(p. 237\)](#)
- [Add a database to your environment \(p. 238\)](#)
- [Cleanup \(p. 240\)](#)
- [Next steps \(p. 241\)](#)

Prerequisites

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in [Getting started using Elastic Beanstalk \(p. 3\)](#) to launch your first Elastic Beanstalk environment.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command  
this is output
```

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Laravel requires PHP 5.5.9 or later and the `mbstring` extension for PHP. In this tutorial we use PHP 7.0 and the corresponding Elastic Beanstalk platform version. Install PHP and Composer by following the instructions at [Setting up your PHP development environment \(p. 228\)](#).

Launch an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. Choose the **PHP** platform and accept the default settings and sample code.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or nginx as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
- **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see [Using Elastic Beanstalk with Amazon S3 \(p. 831\)](#).

Install Laravel and generate a website

Composer can install Laravel and create a working project with one command:

```
~$ composer create-project --prefer-dist laravel/laravel eb-laravel
Installing laravel/laravel (v5.5.28)
- Installing laravel/laravel (v5.5.28): Downloading (100%)
Created project in eb-laravel
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies (including require-dev)

Package operations: 70 installs, 0 updates, 0 removals
- Installing symfony/thanks (v1.0.7): Downloading (100%)
- Installing hamcrest/hamcrest-php (v2.0.0): Downloading (100%)
- Installing mockery/mockery (1.0): Downloading (100%)
- Installing vlucas/phpdotenv (v2.4.0): Downloading (100%)
- Installing symfony/css-selector (v3.4.8): Downloading (100%)
- Installing tijsverkoyen/css-to-inline-styles (2.2.1): Downloading (100%)
```

...

Composer installs Laravel and its dependencies, and generates a default project.

If you run into any issues installing Laravel, go to the installation topic in the official documentation: <https://laravel.com/docs/5.2>

Deploy your application

Create a [source bundle](#) (p. 342) containing the files created by Composer. The following command creates a source bundle named `laravel-default.zip`. It excludes files in the `vendor` folder, which take up a lot of space and are not necessary for deploying your application to Elastic Beanstalk.

```
~/eb-laravel$ zip ../laravel-default.zip -r * .[^.]* -x "vendor/*"
```

Upload the source bundle to Elastic Beanstalk to deploy Laravel to your environment.

To deploy a source bundle

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Note

To optimize the source bundle further, initialize a Git repository and use the [git archive command](#) (p. 343) to create the source bundle. The default Laravel project includes a `.gitignore` file that tells Git to exclude the `vendor` folder and other files that are not required for deployment.

Configure Composer settings

When the deployment completes, click the URL to open your Laravel application in the browser:

Forbidden

You don't have permission to access / on this server.

What's this? By default, Elastic Beanstalk serves the root of your project at the root path of the web site. In this case, though, the default page (`index.php`) is one level down in the `public` folder. You can verify this by adding `/public` to the URL. For example, `http://laravel.us-east-2.elasticbeanstalk.com/public`.

To serve the Laravel application at the root path, use the Elastic Beanstalk console to configure the *document root* for the web site.

To configure your web site's document root

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. For **Document Root**, enter `/public`.
6. Choose **Apply**.
7. When the update is complete, click the URL to reopen your site in the browser.



The image shows the Laravel logo in a large, light gray font. Below the logo is a horizontal navigation bar with five links: DOCUMENTATION, LARACASTS, NEWS, FORGE, and GITHUB, all in uppercase letters.

DOCUMENTATION

LARACASTS

NEWS

FORGE

GITHUB

So far, so good. Next you'll add a database to your environment and configure Laravel to connect to it.

Add a database to your environment

Launch an RDS DB instance in your Elastic Beanstalk environment. You can use MySQL, SQLServer, or PostgreSQL databases with Laravel on Elastic Beanstalk. For this example, we'll use MySQL.

Note

Running an Amazon RDS instance in your Elastic Beanstalk environment is great for development and testing, but it ties the lifecycle of your database to your environment. See [Launching and connecting to an external Amazon RDS instance in a default VPC \(p. 820\)](#) for instructions on connecting to a database running outside of your environment.

To add an RDS DB instance to your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. For **Engine**, choose **mysql**.
6. Type a master **username** and **password**. Elastic Beanstalk will provide these values to your application using environment properties.

7. Choose **Apply**.

Creating a database instance takes about 10 minutes. In the meantime, you can update your source code to read connection information from the environment. Elastic Beanstalk provides connection details using environment variables, such as `RDS_HOSTNAME`, that you can access from your application.

Laravel's database configuration is stored in a file named `database.php` in the `config` folder in your project code. Find the `mysql` entry and modify the `host`, `database`, `username`, and `password` variables to read the corresponding values from Elastic Beanstalk:

Example `~/Eb-laravel/config/database.php`

```
...
'connections' => [

    'sqlite' => [
        'driver' => 'sqlite',
        'database' => env('DB_DATABASE', database_path('database.sqlite')),
        'prefix' => '',
    ],

    'mysql' => [
        'driver' => 'mysql',
        'host' => env('RDS_HOSTNAME', '127.0.0.1'),
        'port' => env('RDS_PORT', '3306'),
        'database' => env('RDS_DB_NAME', 'forge'),
        'username' => env('RDS_USERNAME', 'forge'),
        'password' => env('RDS_PASSWORD', ''),
        'unix_socket' => env('DB_SOCKET', ''),
        'charset' => 'utf8mb4',
        'collation' => 'utf8mb4_unicode_ci',
        'prefix' => '',
        'strict' => true,
        'engine' => null,
    ],

    ...
]
```

To verify that the database connection is configured correctly, add code to `index.php` to connect to the database and add some code to the default response:

Example `~/Eb-laravel/public/index.php`

```
...
if(DB::connection()->getDatabaseName())
{
    echo "Connected to database ".DB::connection()->getDatabaseName();
}
$response->send();
...
```

When the DB instance has finished launching, bundle and deploy the updated application to your environment.

To update your Elastic Beanstalk environment

1. Create a new source bundle:

```
~/eb-laravel$ zip ../laravel-v2-rds.zip -r * .[^.]* -x "vendor/*"
```

2. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.

3. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

4. Choose **Upload and Deploy**.
5. Choose **Browse**, and upload `laravel-v2-rds.zip`.
6. Choose **Deploy**.

Deploying a new version of your application takes less than a minute. When the deployment is complete, refresh the web page again to verify that the database connection succeeded:

Connected to database ebdb

Laravel

DOCUMENTATION

LARACASTS

NEWS

FORGE

GITHUB

Cleanup

When you finish working with Elastic Beanstalk, you can terminate your environment. Elastic Beanstalk terminates all AWS resources associated with your environment, such as [Amazon EC2 instances \(p. 451\)](#), [database instances \(p. 506\)](#), [load balancers \(p. 470\)](#), security groups, and [alarms \(p. \)](#).

To terminate your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

With Elastic Beanstalk, you can easily create a new environment for your application at any time.

In addition, you can terminate database resources that you created outside of your Elastic Beanstalk environment. When you terminate an Amazon RDS DB instance, you can take a snapshot and restore the data to another instance later.

To terminate your RDS DB instance

1. Open the [Amazon RDS console](#).
2. Choose **Databases**.
3. Choose your DB instance.

4. Choose **Actions**, and then choose **Delete**.
5. Choose whether to create a snapshot, and then choose **Delete**.

Next steps

For more information about Laravel, go to the tutorial at laravel.com.

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The [Elastic Beanstalk Command Line Interface \(p. 852\)](#) (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

In this tutorial, you used the Elastic Beanstalk console to configure composer options. To make this configuration part of your application source, you can use a configuration file like the following.

Example `.ebextensions/composer.config`

```
option_settings:  
  aws:elasticbeanstalk:container:php:phpini:  
    document_root: /public
```

For more information, see [Advanced environment customization with configuration files \(.ebextensions\)](#) (p. 600).

Running an Amazon RDS DB instance in your Elastic Beanstalk environment is great for development and testing, but it ties the lifecycle of your database to your environment. See [Adding an Amazon RDS DB instance to your PHP application environment \(p. 285\)](#) for instructions on connecting to a database running outside of your environment.

Finally, if you plan on using your application in a production environment, you will want to [configure a custom domain name \(p. 534\)](#) for your environment and [enable HTTPS \(p. 652\)](#) for secure connections.

Deploying a CakePHP application to Elastic Beanstalk

CakePHP is an open source, MVC framework for PHP. This tutorial walks you through the process of generating a CakePHP project, deploying it to an Elastic Beanstalk environment, and configuring it to connect to an Amazon RDS database instance.

Sections

- [Prerequisites \(p. 241\)](#)
- [Launch an Elastic Beanstalk environment \(p. 242\)](#)
- [Install CakePHP and generate a website \(p. 243\)](#)
- [Deploy your application \(p. 243\)](#)
- [Add a database to your environment \(p. 246\)](#)
- [Cleanup \(p. 248\)](#)
- [Next steps \(p. 248\)](#)

Prerequisites

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in [Getting started using Elastic Beanstalk \(p. 3\)](#) to launch your first Elastic Beanstalk environment.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command  
this is output
```

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

CakePHP requires PHP 5.5.9 or newer, and the `mbstring` and `intl` extensions for PHP. In this tutorial we use PHP 7.0 and the corresponding Elastic Beanstalk platform version. Install PHP and Composer by following the instructions at [Setting up your PHP development environment \(p. 228\)](#).

Launch an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. Choose the **PHP** platform and accept the default settings and sample code.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or nginx as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.

- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
- **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see [Using Elastic Beanstalk with Amazon S3 \(p. 831\)](#).

Install CakePHP and generate a website

Composer can install CakePHP and create a working project with one command:

```
~$ composer create-project --prefer-dist cakephp/app eb-cake
Installing cakephp/app (3.6.0)
- Installing cakephp/app (3.6.0): Loading from cache
Created project in eb-cake2
Loading composer repositories with package information
Updating dependencies (including require-dev)

Package operations: 46 installs, 0 updates, 0 removals
- Installing cakephp/plugin-installer (1.1.0): Loading from cache
- Installing aura/intl (3.0.0): Loading from cache
...
```

Composer installs CakePHP and around 20 dependencies, and generates a default project.

If you run into any issues installing CakePHP, visit the installation topic in the official documentation: <http://book.cakephp.org/3.0/en/installation.html>

Deploy your application

Create a [source bundle \(p. 342\)](#) containing the files created by Composer. The following command creates a source bundle named `cake-default.zip`. It excludes files in the `vendor` folder, which take up a lot of space and are not necessary for deploying your application to Elastic Beanstalk.

```
eb-cake zip ../cake-default.zip -r * .[^.]* -x "vendor/*"
```

Upload the source bundle to Elastic Beanstalk to deploy CakePHP to your environment.

To deploy a source bundle

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Note

To optimize the source bundle further, initialize a Git repository and use the [git archive command \(p. 343\)](#) to create the source bundle. The default Symfony project includes a `.gitignore` file that tells Git to exclude the `vendor` folder and other files that are not required for deployment.

When the process completes, click the URL to open your CakePHP application in the browser:



Welcome to CakePHP 3.6.1 Red Velvet. Build fast. Grow solid.



Please be aware that this page will not be shown if you turn off debug mode unless you replace src/Template/Pages/home.ctp with

Environment

- Your version of PHP is 5.6.0 or higher (detected 7.1.13).
- Your version of PHP has the mbstring extension loaded.
- Your version of PHP has the openssl extension loaded.
- Your version of PHP has the intl extension loaded.

Filesystem

- Your tmp directory is writable.
- Your logs directory is writable.
- The *FileEngine* is being used for core caching.
edit config/app.php

Database

- CakePHP is NOT able to connect to the database.
Connection to database could not be established: SQLSTATE[HY000]
[2002] No such file or directory

DebugKit

- DebugKit is loaded.

Editing this Page

- To change the content of this page, edit:
src/Template/Pages/home.ctp.
- You can also add some CSS styles for your pages at: webroot/css/.

Getting Started

- [CakePHP 3.0 Docs](#)
- [The 15 min Bookmarker Tutorial](#)
- [The 15 min Blog Tutorial](#)
- [The 15 min CMS Tutorial](#)

More about Cake

So far, so good. Next you'll add a database to your environment and configure CakePHP to connect to it.

Add a database to your environment

Launch an Amazon RDS database instance in your Elastic Beanstalk environment. You can use MySQL, SQLServer, or PostgreSQL databases with CakePHP on Elastic Beanstalk. For this example, we'll use PostgreSQL.

To add an Amazon RDS DB instance to your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. Under **Database**, choose **Edit**.
5. For **DB engine**, choose **postgres**.
6. Type a master **username** and **password**. Elastic Beanstalk will provide these values to your application using environment properties.
7. Choose **Apply**.

Creating a database instance takes about 10 minutes. In the meantime, you can update your source code to read connection information from the environment. Elastic Beanstalk provides connection details using environment variables such as `RDS_HOSTNAME` that you can access from your application.

CakePHP's database configuration is in a file named `app.php` in the `config` folder in your project code. Open this file and add some code that reads the environment variables from `$_SERVER` and assigns them to local variables. Insert the highlighted lines in the below example after the first line (`<?php`):

Example `~/Eb-cake/config/app.php`

```
<?php
if (!defined('RDS_HOSTNAME')) {
    define('RDS_HOSTNAME', $_SERVER['RDS_HOSTNAME']);
    define('RDS_USERNAME', $_SERVER['RDS_USERNAME']);
    define('RDS_PASSWORD', $_SERVER['RDS_PASSWORD']);
    define('RDS_DB_NAME', $_SERVER['RDS_DB_NAME']);
}
return [
    ...
```

The database connection is configured further down in `app.php`. Find the following section and modify the default datasources configuration with the name of the driver that matches your database engine (`mysql`, `sqlserver`, or `postgres`), and set the `host`, `username`, `password` and `database` variables to read the corresponding values from Elastic Beanstalk:

Example `~/Eb-cake/config/app.php`

```
...
    /**
     * Connection information used by the ORM to connect
     * to your application's datastores.
```

```
* Drivers include Mysql Postgres Sqlite Sqlserver
* See vendor\cakephp\cakephp\src\Database\Driver for complete list
*/
'Datasources' => [
    'default' => [
        'className' => 'Cake\Database\Connection',
        'driver' => 'Cake\Database\Driver\Postgres',
        'persistent' => false,
        'host' => RDS_HOSTNAME,
        /*
        * CakePHP will use the default DB port based on the driver selected
        * MySQL on MAMP uses port 8889, MAMP users will want to uncomment
        * the following line and set the port accordingly
        */
        //'port' => 'non_standard_port_number',
        'username' => RDS_USERNAME,
        'password' => RDS_PASSWORD,
        'database' => RDS_DB_NAME,
        /*
        * You do not need to set this flag to use full utf-8 encoding (internal
        default since CakePHP 3.6).
        */
        //'encoding' => 'utf8mb4',
        'timezone' => 'UTC',
        'flags' => [],
        'cacheMetadata' => true,
        'log' => false,
    ],
    ...
]
```

When the DB instance has finished launching, bundle up and deploy the updated application to your environment:

To update your Elastic Beanstalk environment

1. Create a new source bundle:

```
~/eb-cake$ zip ../cake-v2-rds.zip -r * .[^.]* -x "vendor/*"
```

2. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
3. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.


Note

If you have many environments, use the search bar to filter the environment list.

4. Choose **Upload and Deploy**.
5. Choose **Browse** and upload `cake-v2-rds.zip`.
6. Choose **Deploy**.

Deploying a new version of your application takes less than a minute. When the deployment is complete, refresh the web page again to verify that the database connection succeeded:

Database

 CakePHP is able to connect to the database.

Cleanup

When you finish working with Elastic Beanstalk, you can terminate your environment. Elastic Beanstalk terminates all AWS resources associated with your environment, such as [Amazon EC2 instances \(p. 451\)](#), [database instances \(p. 506\)](#), [load balancers \(p. 470\)](#), security groups, and [alarms \(p. 470\)](#).

To terminate your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

With Elastic Beanstalk, you can easily create a new environment for your application at any time.

In addition, you can terminate database resources that you created outside of your Elastic Beanstalk environment. When you terminate an Amazon RDS DB instance, you can take a snapshot and restore the data to another instance later.

To terminate your RDS DB instance

1. Open the [Amazon RDS console](#).
2. Choose **Databases**.
3. Choose your DB instance.
4. Choose **Actions**, and then choose **Delete**.
5. Choose whether to create a snapshot, and then choose **Delete**.

Next steps

For more information about CakePHP, read the book at book.cakephp.org.

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The [Elastic Beanstalk Command Line Interface \(p. 852\)](#) (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

Running an Amazon RDS DB instance in your Elastic Beanstalk environment is great for development and testing, but it ties the lifecycle of your database to your environment. See [Adding an Amazon RDS DB instance to your PHP application environment \(p. 285\)](#) for instructions on connecting to a database running outside of your environment.

Finally, if you plan on using your application in a production environment, you will want to [configure a custom domain name \(p. 534\)](#) for your environment and [enable HTTPS \(p. 652\)](#) for secure connections.

Deploying a Symfony application to Elastic Beanstalk

[Symfony](#) is an open source framework for developing dynamic PHP web applications. This tutorial walks you through the process of generating a Symfony application and deploying it to an AWS Elastic Beanstalk environment.

Sections

- [Prerequisites](#) (p. 249)
- [Launch an Elastic Beanstalk environment](#) (p. 249)
- [Install Symfony and generate a website](#) (p. 250)
- [Deploy your application](#) (p. 251)
- [Configure Composer settings](#) (p. 251)
- [Cleanup](#) (p. 252)
- [Next steps](#) (p. 253)

Prerequisites

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in [Getting started using Elastic Beanstalk](#) (p. 3) to launch your first Elastic Beanstalk environment.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command  
this is output
```

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Symfony 4.3 requires PHP 7.1 or later and the `intl` extension for PHP. In this tutorial we use PHP 7.2 and the corresponding Elastic Beanstalk platform version. Install PHP and Composer by following the instructions at [Setting up your PHP development environment](#) (p. 228).

Launch an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. Choose the **PHP** platform and accept the default settings and sample code.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link:
console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or nginx as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
- **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see [Using Elastic Beanstalk with Amazon S3](#) (p. 831).

Install Symfony and generate a website

Composer can install Symfony and create a working project with one command:

```
~$ composer create-project symfony/website-skeleton eb-symfony
Installing symfony/website-skeleton (v4.3.99)
  - Installing symfony/website-skeleton (v4.3.99): Downloading (100%)
Created project in eb-symfony
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Installing symfony/flex (v1.4.5): Downloading (100%)
Symfony operations: 1 recipe (539d006017ad5ef71beab4a2e2870e9a)
  - Configuring symfony/flex (>=1.0): From github.com/symfony/recipes:master
Loading composer repositories with package information
Updating dependencies (including require-dev)
Restricting packages listed in "symfony/symfony" to "4.3.*"
```

```
Package operations: 103 installs, 0 updates, 0 removals
- Installing ocradius/package-versions (1.4.0): Loading from cache
...
```

Composer installs Symfony and its dependencies, and generates a default project.

If you run into any issues installing Symfony, go to the installation topic in the official documentation: symfony.com/doc/current/setup.html

Deploy your application

Go to the project directory.

```
~$ cd eb-symfony
```

Create a [source bundle \(p. 342\)](#) containing the files created by Composer. The following command creates a source bundle named `symfony-default.zip`. It excludes files in the `vendor` folder, which take up a lot of space and are not necessary for deploying your application to Elastic Beanstalk.

```
eb-symfony$ zip ../symfony-default.zip -r * .[^.]* -x "vendor/*"
```

Upload the source bundle to Elastic Beanstalk to deploy Symfony to your environment.

To deploy a source bundle

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Note

To optimize the source bundle further, initialize a Git repository and use the [git archive command \(p. 343\)](#) to create the source bundle. The default Symfony project includes a `.gitignore` file that tells Git to exclude the `vendor` folder and other files that are not required for deployment.

Configure Composer settings

When the deployment completes, click the URL to open your Symfony application in the browser:

Forbidden

You don't have permission to access / on this server.

What's this? By default, Elastic Beanstalk serves the root of your project at the root path of the web site. In this case, though, the default page (`app.php`) is one level down in the web

folder. You can verify this by adding `/public` to the URL. For example, `http://symfony.us-east-2.elasticbeanstalk.com/public`.

To serve the Symfony application at the root path, use the Elastic Beanstalk console to configure the *document root* for the web site.

To configure your web site's document root

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. For **Document root**, enter `/public`.
6. Choose **Apply**.
7. When the update is complete, click the URL to reopen your site in the browser.

Welcome to Symfony 4.3.4



Your application is now ready. You can start working on it at:

`/var/app/current/`

What's next?



Read the documentation to learn

[How to create your first page in Symfony](#)

You're seeing this page because debug mode is enabled and you haven't configured any hom

Cleanup

When you finish working with Elastic Beanstalk, you can terminate your environment. Elastic Beanstalk terminates all AWS resources associated with your environment, such as [Amazon EC2](#)

[instances \(p. 451\)](#), [database instances \(p. 506\)](#), [load balancers \(p. 470\)](#), security groups, and [alarms \(p. 470\)](#).

To terminate your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

With Elastic Beanstalk, you can easily create a new environment for your application at any time.

Next steps

For more information about Symfony, see [What is Symfony?](#) at symfony.com.

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The [Elastic Beanstalk Command Line Interface \(p. 852\)](#) (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

In this tutorial, you used the Elastic Beanstalk console to configure composer options. To make this configuration part of your application source, you can use a configuration file like the following.

Example `.ebextensions/composer.config`

```
option_settings:  
  aws:elasticbeanstalk:container:php:phpini:  
    document_root: /public
```

For more information, see [Advanced environment customization with configuration files \(.ebextensions\) \(p. 600\)](#).

Symfony uses its own configuration files to configure database connections. For instructions on connecting to a database with Symfony, see [Connecting to a database with Symfony \(p. 287\)](#).

Finally, if you plan on using your application in a production environment, you will want to [configure a custom domain name \(p. 534\)](#) for your environment and [enable HTTPS \(p. 652\)](#) for secure connections.

Deploying a high-availability PHP application with an external Amazon RDS database to Elastic Beanstalk

This tutorial walks you through the process of [launching an RDS DB instance \(p. 820\)](#) external to AWS Elastic Beanstalk, and configuring a high-availability environment running a PHP application to connect to it. Running a DB instance external to Elastic Beanstalk decouples the database from the lifecycle of your environment. This lets you connect to the same database from multiple environments, swap out one database for another, or perform a blue/green deployment without affecting your database.

The tutorial uses a [sample PHP application](#) that uses a MySQL database to store user-provided text data. The sample application uses [configuration files \(p. 600\)](#) to configure [PHP settings \(p. 232\)](#)

and to create a table in the database for the application to use. It also shows how to use a [Composer file \(p. 233\)](#) to install packages during deployment.

Sections

- [Prerequisites \(p. 254\)](#)
- [Launch a DB instance in Amazon RDS \(p. 254\)](#)
- [Create an Elastic Beanstalk environment \(p. 256\)](#)
- [Configure security groups, environment properties, and scaling \(p. 257\)](#)
- [Deploy the sample application \(p. 260\)](#)
- [Cleanup \(p. 260\)](#)
- [Next steps \(p. 261\)](#)

Prerequisites

Before you start, download the sample application source bundle from GitHub: [eb-demo-php-simple-app-1.3.zip](#)

The procedures in this tutorial for Amazon Relational Database Service (Amazon RDS) tasks assume that you are launching resources in a default [Amazon Virtual Private Cloud \(Amazon VPC\)](#). All new accounts include a default VPC in each region. If you don't have a default VPC, the procedures will vary. See [Using Elastic Beanstalk with Amazon RDS \(p. 820\)](#) for instructions for EC2-Classic and custom VPC platforms.

Launch a DB instance in Amazon RDS

To use an external database with an application running in Elastic Beanstalk, first launch a DB instance with Amazon RDS. When you launch an instance with Amazon RDS, it is completely independent of Elastic Beanstalk and your Elastic Beanstalk environments, and will not be terminated or monitored by Elastic Beanstalk.

Use the Amazon RDS console to launch a Multi-AZ **MySQL** DB instance. Choosing a Multi-AZ deployment ensures that your database will fail over and continue to be available if the master DB instance goes out of service.

To launch an RDS DB instance in a default VPC

1. Open the [RDS console](#).
2. Choose **Databases** in the navigation pane.
3. Choose **Create database**.
4. Choose **Standard Create**.

Important

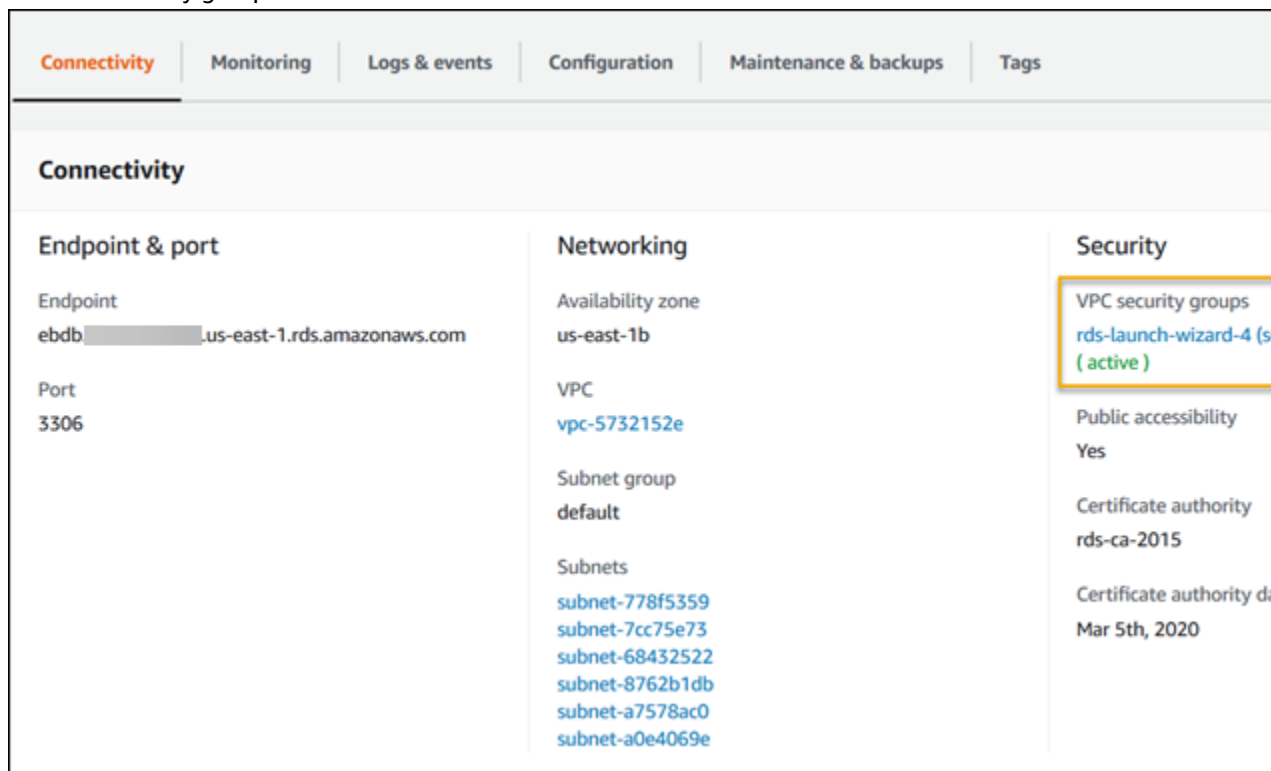
Do not choose **Easy Create**. It does not allow you to configure the necessary settings to launch this RDS DB.

5. Under **Additional configuration**, for **Initial database name**, type **ebdb**.
6. Review the default settings carefully and adjust as necessary. Pay attention to the following options:
 - **DB instance class** – Choose an instance size that has an appropriate amount of memory and CPU power for your workload.
 - **Multi-AZ deployment** – For high availability, set this to **Create an Aurora Replica/Reader node in a different AZ**.
 - **Master username** and **Master password** – The database username and password. Make a note of these settings because you'll use them later.
7. Verify the default settings for the remaining options, and then choose **Create database**.

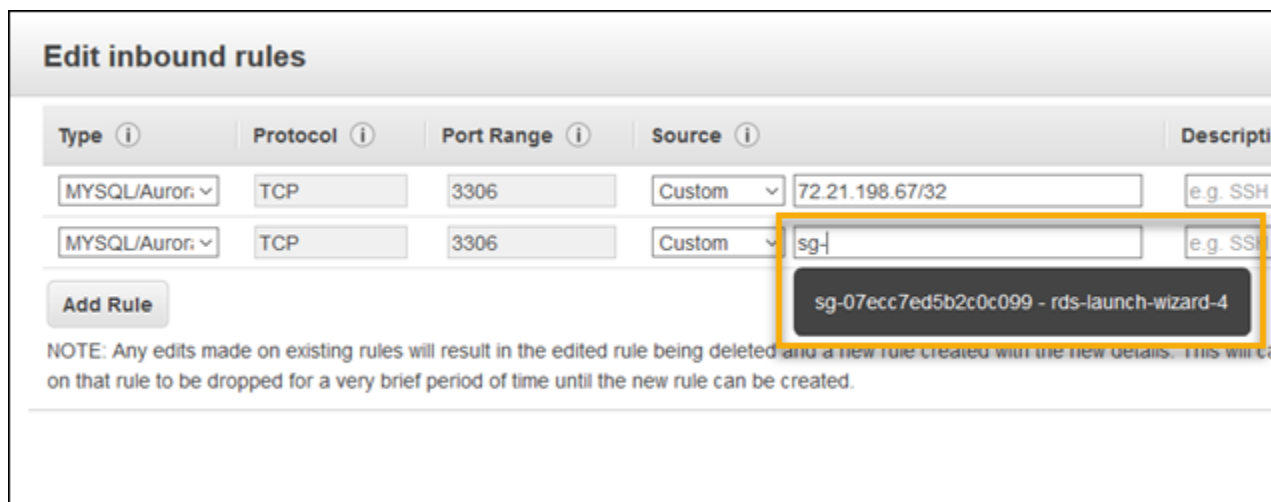
Next, modify the security group attached to your DB instance to allow inbound traffic on the appropriate port. This is the same security group that you will attach to your Elastic Beanstalk environment later, so the rule that you add will grant ingress permission to other resources in the same security group.

To modify the inbound rules on your RDS instance's security group

1. Open the [Amazon RDS console](#).
2. Choose **Databases**.
3. Choose the name of your DB instance to view its details.
4. In the **Connectivity** section, make a note of the **Subnets**, **Security groups**, and **Endpoint** shown on this page so you can use this information later.
5. Under **Security**, you can see the security group associated with the DB instance. Open the link to view the security group in the Amazon EC2 console.



6. In the security group details, choose **Inbound**.
7. Choose **Edit**.
8. Choose **Add Rule**.
9. For **Type**, choose the DB engine that your application uses.
10. For **Source**, type **sg-** to view a list of available security groups. Choose the current security group to allow resources in the security group to receive traffic on the database port from other resources in the same group.



11. Choose **Save**.

Creating a DB instance takes about 10 minutes. In the meantime, create your Elastic Beanstalk environment.

Create an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. Choose the **PHP** platform and accept the default settings and sample code. After you launch the environment, you can configure the environment to connect to the database, then deploy the sample application that you downloaded from GitHub.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link:
console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or nginx as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.

- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
- **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains. The RDS DB instance that you launched is outside of your environment, so you are responsible for managing its lifecycle.

Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see [Using Elastic Beanstalk with Amazon S3 \(p. 831\)](#).

Configure security groups, environment properties, and scaling

Add the security group of your DB instance to your running environment. This procedure causes Elastic Beanstalk to reprovision all instances in your environment with the additional security group attached.

To add a security group to your environment

- Do one of the following:
 - To add a security group using the Elastic Beanstalk console
 - a. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
 - b. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note
If you have many environments, use the search bar to filter the environment list.
 - c. In the navigation pane, choose **Configuration**.
 - d. In the **Instances** configuration category, choose **Edit**.
 - e. Under **EC2 security groups**, choose the security group to attach to the instances, in addition to the instance security group that Elastic Beanstalk creates.
 - f. Choose **Apply**.
 - g. Read the warning, and then choose **Confirm**.
 - To add a security group using a [configuration file \(p. 600\)](#), use the `securitygroup-addexisting.config` example file.

Next, use environment properties to pass the connection information to your environment. The sample application uses a default set of properties that match the ones that Elastic Beanstalk configures when you provision a database within your environment.

To configure environment properties for an Amazon RDS DB instance

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. In the **Environment properties** section, define the variables that your application reads to construct a connection string. For compatibility with environments that have an integrated RDS DB instance, use the following names and values. You can find all values, except for your password, in the [RDS console](#).

Property name	Description	Property value
RDS_HOSTNAME	The hostname of the DB instance.	On the Connectivity & security tab on the Amazon RDS console: Endpoint .
RDS_PORT	The port on which the DB instance accepts connections. The default value varies among DB engines.	On the Connectivity & security tab on the Amazon RDS console: Port .
RDS_DB_NAME	The database name, ebdb .	On the Configuration tab on the Amazon RDS console: DB Name .
RDS_USERNAME	The username that you configured for your database.	On the Configuration tab on the Amazon RDS console: Master username .
RDS_PASSWORD	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

Environment Properties

The following properties are passed into the application as environment variables. [Learn more.](#)

Property Name	Property Value
RDS_DB_NAME	<input type="text" value="ebdb"/>
RDS_HOSTNAME	<input type="text" value="webapp-db.jzcb5mpan"/>
RDS_PORT	<input type="text" value="5432"/>
RDS_USERNAME	<input type="text" value="webapp-admin"/>
<input type="text" value="RDS_PASSWORD"/>	<input type="text" value="kUj5uKxmWDMYc403"/>

[Cancel](#)

6. Choose **Apply**.

Finally, configure your environment's Auto Scaling group with a higher minimum instance count. Run at least two instances at all times to prevent the web servers in your environment from being a single point of failure, and to allow you to deploy changes without taking your site out of service.

To configure your environment's Auto Scaling group for high availability

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Auto Scaling group** section, set **Min instances** to **2**.
6. Choose **Apply**.

Deploy the sample application

Now your environment is ready to run the sample application and connect to Amazon RDS. Deploy the sample application to your environment.

Note

Download the source bundle from GitHub, if you haven't already: [eb-demo-php-simple-app-1.3.zip](#)

To deploy a source bundle

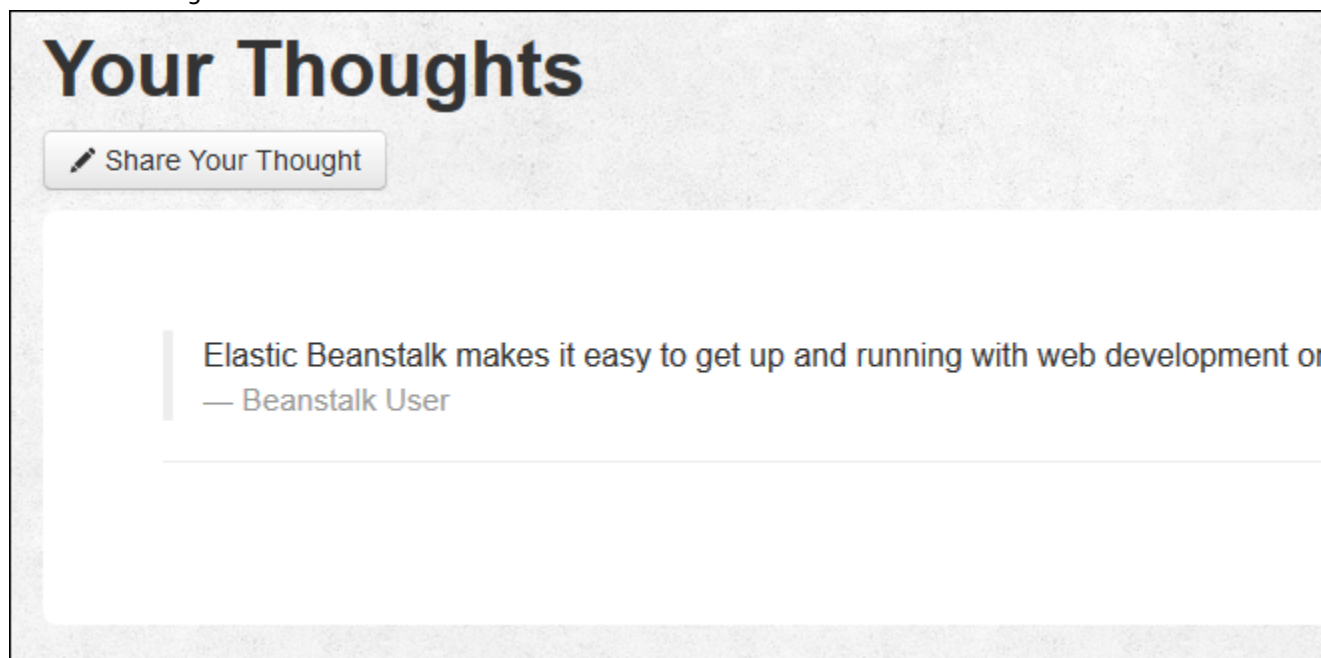
1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

The site collects user comments and uses a MySQL database to store the data. To add a comment, choose **Share Your Thought**, enter a comment, and then choose **Submit Your Thought**. The web app writes the comment to the database so that any instance in the environment can read it, and it won't be lost if instances go out of service.



Cleanup

When you finish working with Elastic Beanstalk, you can terminate your environment. Elastic Beanstalk terminates all AWS resources associated with your environment, such as [Amazon EC2](#)

[instances \(p. 451\)](#), [database instances \(p. 506\)](#), [load balancers \(p. 470\)](#), security groups, and [alarms \(p. 470\)](#).

To terminate your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

With Elastic Beanstalk, you can easily create a new environment for your application at any time.

In addition, you can terminate database resources that you created outside of your Elastic Beanstalk environment. When you terminate an Amazon RDS DB instance, you can take a snapshot and restore the data to another instance later.

To terminate your RDS DB instance

1. Open the [Amazon RDS console](#).
2. Choose **Databases**.
3. Choose your DB instance.
4. Choose **Actions**, and then choose **Delete**.
5. Choose whether to create a snapshot, and then choose **Delete**.

Next steps

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The [Elastic Beanstalk Command Line Interface \(p. 852\)](#) (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

The sample application uses configuration files to configure PHP settings and create a table in the database if it doesn't already exist. You can also use a configuration file to configure the security group settings of your instances during environment creation to avoid time-consuming configuration updates. See [Advanced environment customization with configuration files \(.ebextensions\) \(p. 600\)](#) for more information.

For development and testing, you might want to use the Elastic Beanstalk functionality for adding a managed DB instance directly to your environment. For instructions on setting up a database inside your environment, see [Adding a database to your Elastic Beanstalk environment \(p. 506\)](#).

If you need a high-performance database, consider using [Amazon Aurora](#). Amazon Aurora is a MySQL-compatible database engine that offers commercial database features at low cost. To connect your application to a different database, repeat the [security group configuration \(p. 254\)](#) steps and [update the RDS-related environment properties \(p. 257\)](#).

Finally, if you plan on using your application in a production environment, you will want to [configure a custom domain name \(p. 534\)](#) for your environment and [enable HTTPS \(p. 652\)](#) for secure connections.

Deploying a high-availability WordPress website with an external Amazon RDS database to Elastic Beanstalk

This tutorial describes how to [launch an Amazon RDS DB instance \(p. 820\)](#) that is external to AWS Elastic Beanstalk, then how to configure a high-availability environment running a WordPress website to connect to it. The website uses Amazon Elastic File System (Amazon EFS) as the shared storage for uploaded files.

Running a DB instance external to Elastic Beanstalk decouples the database from the lifecycle of your environment. This lets you connect to the same database from multiple environments, swap out one database for another, or perform a [blue/green deployment \(p. 405\)](#) without affecting your database.

This tutorial was developed with WordPress version 4.9.5 and PHP 7.0.

Topics

- [Prerequisites \(p. 262\)](#)
- [Launch a DB instance in Amazon RDS \(p. 263\)](#)
- [Download WordPress \(p. 265\)](#)
- [Launch an Elastic Beanstalk environment \(p. 265\)](#)
- [Configure security groups and environment properties \(p. 266\)](#)
- [Configure and deploy your application \(p. 268\)](#)
- [Install WordPress \(p. 270\)](#)
- [Update keys and salts \(p. 270\)](#)
- [Remove access restrictions \(p. 271\)](#)
- [Configure your Auto Scaling group \(p. 271\)](#)
- [Upgrade WordPress \(p. 272\)](#)
- [Clean up \(p. 272\)](#)
- [Next steps \(p. 273\)](#)

Prerequisites

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in [Getting started using Elastic Beanstalk \(p. 3\)](#) to launch your first Elastic Beanstalk environment.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command  
this is output
```

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Default VPC

The Amazon Relational Database Service (Amazon RDS) procedures in this tutorial assume that you are launching resources in a default [Amazon Virtual Private Cloud](#) (Amazon VPC). All new accounts include a default VPC in each AWS Region. If you don't have a default VPC, the procedures will vary. See [Using Elastic Beanstalk with Amazon RDS \(p. 820\)](#) for instructions for EC2-Classic and custom VPC platforms.

AWS Regions

The sample application uses Amazon EFS, which only works in AWS Regions that support Amazon EFS. To learn about supported AWS Regions, see [Amazon Elastic File System Endpoints and Quotas](#) in the *AWS General Reference*.

Launch a DB instance in Amazon RDS

When you launch an instance with Amazon RDS, it's completely independent of Elastic Beanstalk and your Elastic Beanstalk environments, and will not be terminated or monitored by Elastic Beanstalk.

In the following steps you'll use the Amazon RDS console to:

- Launch a database with the **MySQL** engine.
- Enable a **Multi-AZ deployment**. This creates a standby in a different Availability Zone (AZ) to provide data redundancy, eliminate I/O freezes, and minimize latency spikes during system backups.

To launch an RDS DB instance in a default VPC

1. Open the [RDS console](#).
2. Choose **Databases** in the navigation pane.
3. Choose **Create database**.
4. Choose **Standard Create**.
 - Important**
Do not choose **Easy Create**. It does not allow you to configure the necessary settings to launch this RDS DB.
5. Under **Additional configuration**, for **Initial database name**, type **ebdb**.
6. Review the default settings carefully and adjust as necessary. Pay attention to the following options:
 - **DB instance class** – Choose an instance size that has an appropriate amount of memory and CPU power for your workload.
 - **Multi-AZ deployment** – For high availability, set this to **Create an Aurora Replica/Reader node in a different AZ**.
 - **Master username** and **Master password** – The database username and password. Make a note of these settings because you'll use them later.
7. Verify the default settings for the remaining options, and then choose **Create database**.

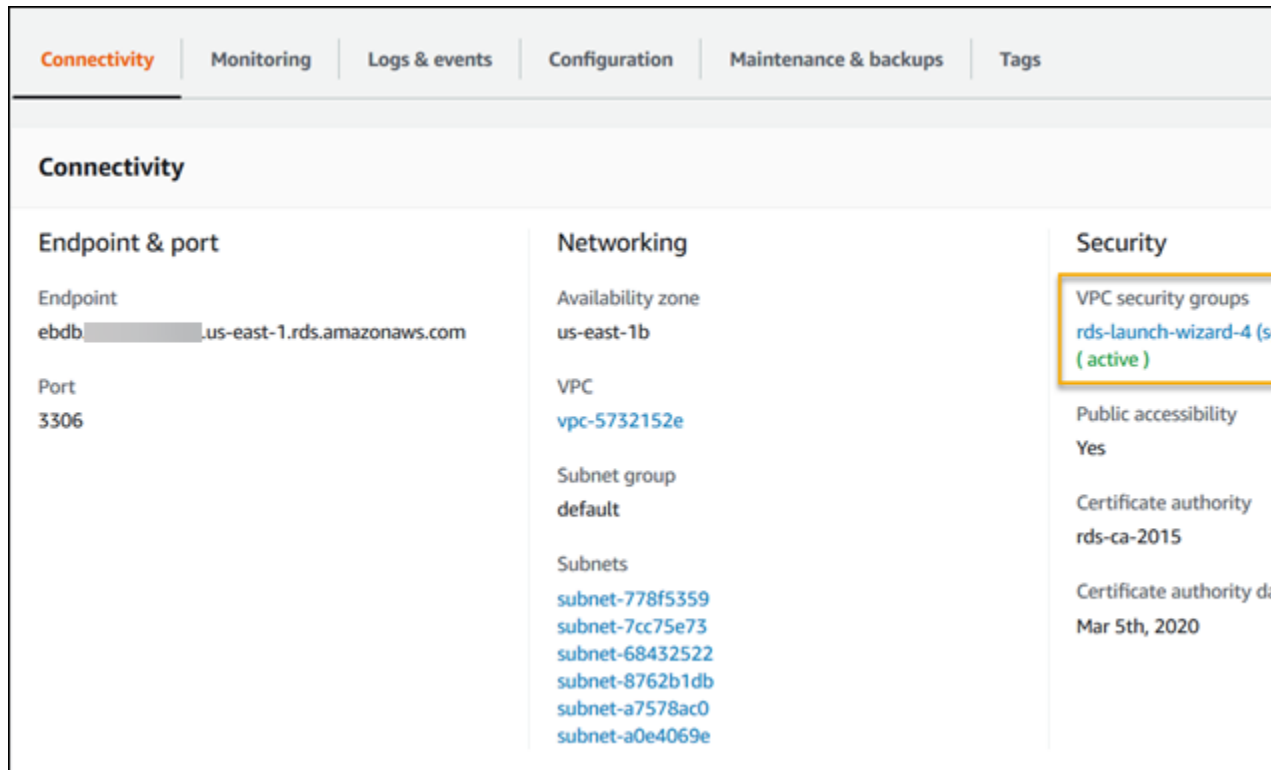
After your DB instance is created, modify the security group attached to it in order to allow inbound traffic on the appropriate port..

Note

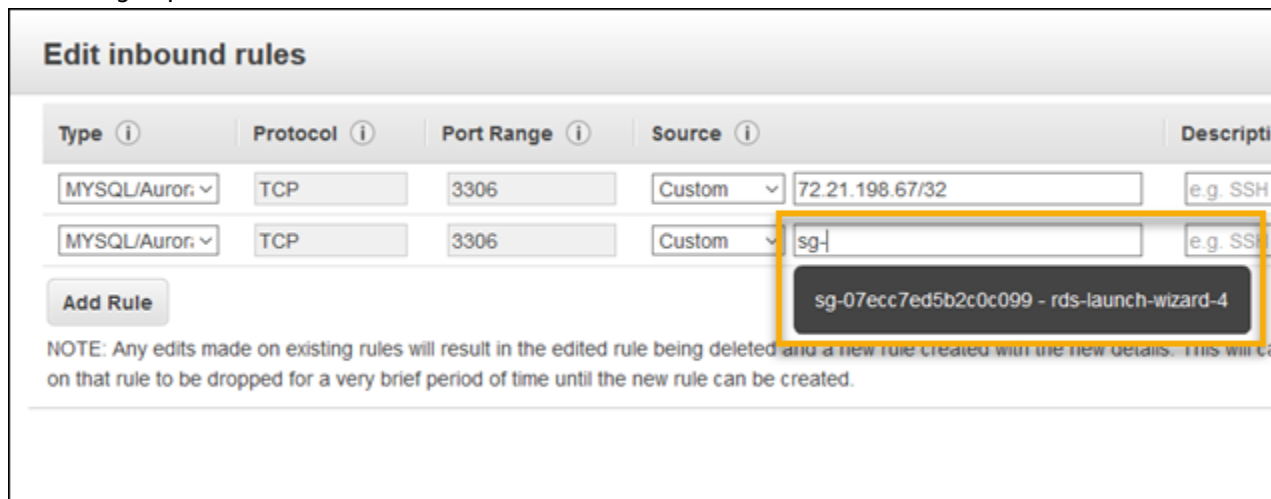
This is the same security group that you'll attach to your Elastic Beanstalk environment later, so the rule that you add now will grant ingress permission to other resources in the same security group.

To modify the inbound rules on your RDS instance's security group

1. Open the [Amazon RDS console](#).
2. Choose **Databases**.
3. Choose the name of your DB instance to view its details.
4. In the **Connectivity** section, make a note of the **Subnets**, **Security groups**, and **Endpoint** shown on this page so you can use this information later.
5. Under **Security**, you can see the security group associated with the DB instance. Open the link to view the security group in the Amazon EC2 console.



6. In the security group details, choose **Inbound**.
7. Choose **Edit**.
8. Choose **Add Rule**.
9. For **Type**, choose the DB engine that your application uses.
10. For **Source**, type **sg-** to view a list of available security groups. Choose the current security group to allow resources in the security group to receive traffic on the database port from other resources in the same group.



11. Choose **Save**.

Creating a DB instance takes about 10 minutes. In the meantime, download WordPress and create your Elastic Beanstalk environment.

Download WordPress

To prepare to deploy WordPress using AWS Elastic Beanstalk, you must copy the WordPress files to your computer and provide the correct configuration information.

To create a WordPress project

1. Download WordPress from wordpress.org.

```
~$ curl https://wordpress.org/wordpress-4.9.5.tar.gz -o wordpress.tar.gz
```

2. Download the configuration files from the sample repository.

```
~$ wget https://github.com/aws-samples/eb-php-wordpress/releases/download/v1.1/eb-php-wordpress-v1.zip
```

3. Extract WordPress and change the name of the folder.

```
~$ tar -xvf wordpress.tar.gz
~$ mv wordpress wordpress-beanstalk
~$ cd wordpress-beanstalk
```

4. Extract the configuration files over the WordPress installation.

```
~/wordpress-beanstalk$ unzip ../eb-php-wordpress-v1.zip
creating: .ebextensions/
inflating: .ebextensions/dev.config
inflating: .ebextensions/efs-create.config
inflating: .ebextensions/efs-mount.config
inflating: .ebextensions/loadbalancer-sg.config
inflating: .ebextensions/wordpress.config
inflating: LICENSE
inflating: README.md
inflating: wp-config.php
```

Launch an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. After you launch the environment, you can configure it to connect to the database, then deploy the WordPress code to the environment.

In the following steps, you'll use the Elastic Beanstalk console to:

- Create an Elastic Beanstalk application using the managed **PHP** platform.
- Accept the default settings and sample code.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link:
console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.

5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about five minutes and creates the following resources.

Elastic Beanstalk created resources

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or nginx as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
- **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Because the Amazon RDS instance that you launched is outside of your environment, you are responsible for managing its lifecycle.

Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see [Using Elastic Beanstalk with Amazon S3 \(p. 831\)](#).

Configure security groups and environment properties

Add the security group of your DB instance to your running environment. This procedure causes Elastic Beanstalk to reprovision all instances in your environment with the additional security group attached.

To add a security group to your environment

- Do one of the following:

- To add a security group using the Elastic Beanstalk console
 - a. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
 - b. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

- c. In the navigation pane, choose **Configuration**.
 - d. In the **Instances** configuration category, choose **Edit**.
 - e. Under **EC2 security groups**, choose the security group to attach to the instances, in addition to the instance security group that Elastic Beanstalk creates.
 - f. Choose **Apply**.
 - g. Read the warning, and then choose **Confirm**.
- To add a security group using a [configuration file \(p. 600\)](#), use the `securitygroup-addexisting.config` example file.

Next, use environment properties to pass the connection information to your environment.

The WordPress application uses a default set of properties that match the ones that Elastic Beanstalk configures when you provision a database within your environment.

To configure environment properties for an Amazon RDS DB instance

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. In the **Environment properties** section, define the variables that your application reads to construct a connection string. For compatibility with environments that have an integrated RDS DB instance, use the following names and values. You can find all values, except for your password, in the [RDS console](#).

Property name	Description	Property value
RDS_HOSTNAME	The hostname of the DB instance.	On the Connectivity & security tab on the Amazon RDS console: Endpoint .
RDS_PORT	The port on which the DB instance accepts connections. The default value varies among DB engines.	On the Connectivity & security tab on the Amazon RDS console: Port .
RDS_DB_NAME	The database name, ebdb .	On the Configuration tab on the Amazon RDS console: DB Name .
RDS_USERNAME	The username that you configured for your database.	On the Configuration tab on the Amazon RDS console: Master username .

Property name	Description	Property value
RDS_PASSWORD	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

Environment Properties

The following properties are passed into the application as environment variables. [Learn more.](#)

Property Name	Property Value
RDS_DB_NAME	ebdb
RDS_HOSTNAME	webapp-db.jxzc5mpan
RDS_PORT	5432
RDS_USERNAME	webapp-admin
RDS_PASSWORD	kUj5uKxmWDMYc403

Cancel

6. Choose **Apply**.

Configure and deploy your application

Verify that the structure of your `wordpress-beanstalk` folder is correct, as shown.

```
wordpress-beanstalk$ tree -aL 1
.
### .ebextensions
### index.php
### LICENSE
### license.txt
### readme.html
### README.md
### wp-activate.php
### wp-admin
### wp-blog-header.php
### wp-comments-post.php
### wp-config.php
### wp-config-sample.php
```

```
### wp-content
### wp-cron.php
### wp-includes
### wp-links-opml.php
### wp-load.php
### wp-login.php
### wp-mail.php
### wp-settings.php
### wp-signup.php
### wp-trackback.php
### xmlrpc.php
```

The customized `wp-config.php` file from the project repo uses the environment variables that you defined in the previous step to configure the database connection. The `.ebextensions` folder contains configuration files that create additional resources within your Elastic Beanstalk environment.

The configuration files require modification to work with your account. Replace the placeholder values in the files with the appropriate IDs and create a source bundle.

To update configuration files and create a source bundle

1. Modify the configuration files as follows.
 - `.ebextensions/dev.config` – Restricts access to your environment to protect it during the WordPress installation process. Replace the placeholder IP address near the top of the file with the public IP address of the computer you'll use to access your environment's website to complete your WordPress installation.

Note

Depending on your network, you might need to use an IP address block.

- `.ebextensions/efs-create.config` – Creates an EFS file system and mount points in each Availability Zone/subnet in your VPC. Identify your default VPC and subnet IDs in the [Amazon VPC console](#).
2. Create a [source bundle](#) (p. 342) containing the files in your project folder. The following command creates a source bundle named `wordpress-beanstalk.zip`.

```
~/eb-wordpress$ zip ../wordpress-beanstalk.zip -r * .[^.]*
```

Upload the source bundle to Elastic Beanstalk to deploy WordPress to your environment.

To deploy a source bundle

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Install WordPress

To complete your WordPress installation

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose the environment URL to open your site in a browser. You are redirected to a WordPress installation wizard because you haven't configured the site yet.
4. Perform a standard installation. The `wp-config.php` file is already present in the source code and configured to read the database connection information from the environment. You shouldn't be prompted to configure the connection.

Installation takes about a minute to complete.

Update keys and salts

The WordPress configuration file `wp-config.php` also reads values for keys and salts from environment properties. Currently, these properties are all set to `test` by the `wordpress.config` file in the `.ebextensions` folder.

The hash salt can be any value that meets the [environment property requirements \(p. 518\)](#), but you should not store it in source control. Use the Elastic Beanstalk console to set these properties directly on the environment.

To update environment properties

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the navigation pane, choose **Configuration**.
4. Under **Software**, choose **Edit**.
5. For `Environment` properties, modify the following properties:
 - `AUTH_KEY` – The value chosen for `AUTH_KEY`.
 - `SECURE_AUTH_KEY` – The value chosen for `SECURE_AUTH_KEY`.
 - `LOGGED_IN_KEY` – The value chosen for `LOGGED_IN_KEY`.
 - `NONCE_KEY` – The value chosen for `NONCE_KEY`.
 - `AUTH_SALT` – The value chosen for `AUTH_SALT`.
 - `SECURE_AUTH_SALT` – The value chosen for `SECURE_AUTH_SALT`.
 - `LOGGED_IN_SALT` – The value chosen for `LOGGED_IN_SALT`.
 - `NONCE_SALT` — The value chosen for `NONCE_SALT`.
6. Choose **Apply**.

Note

Setting the properties on the environment directly overrides the values in `wordpress.config`.

Remove access restrictions

The sample project includes the configuration file `loadbalancer-sg.config`. It creates a security group and assigns it to the environment's load balancer, using the IP address that you configured in `dev.config`. It restricts HTTP access on port 80 to connections from your network. Otherwise, an outside party could potentially connect to your site before you have installed WordPress and configured your admin account.

Now that you've installed WordPress, remove the configuration file to open the site to the world.

To remove the restriction and update your environment

1. Delete the `.ebextensions/loadbalancer-sg.config` file from your project directory.

```
~/wordpress-beanstalk$ rm .ebextensions/loadbalancer-sg.config
```

2. Create a source bundle.

```
~/eb-wordpress$ zip ../wordpress-beanstalk-v2.zip -r * .[^.]*
```

Upload the source bundle to Elastic Beanstalk to deploy WordPress to your environment.

To deploy a source bundle

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Configure your Auto Scaling group

Finally, configure your environment's Auto Scaling group with a higher minimum instance count. Run at least two instances at all times to prevent the web servers in your environment from being a single point of failure. This also allows you to deploy changes without taking your site out of service.

To configure your environment's Auto Scaling group for high availability

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Auto Scaling group** section, set **Min instances** to **2**.
6. Choose **Apply**.

To support content uploads across multiple instances, the sample project uses Amazon EFS to create a shared file system. Create a post on the site and upload content to store it on the shared file system. View the post and refresh the page multiple times to hit both instances and verify that the shared file system is working.

Upgrade WordPress

To upgrade to a new version of WordPress, back up your site and deploy it to a new environment.

Important

Do not use the update functionality within WordPress or update your source files to use a new version. Both of these actions can result in your post URLs returning 404 errors even though they are still in the database and file system.

To upgrade WordPress

1. In the WordPress admin console, use the export tool to export your posts to an XML file.
2. Deploy and install the new version of WordPress to Elastic Beanstalk with the same steps that you used to install the previous version. To avoid downtime, you can create an environment with the new version.
3. On the new version, install the WordPress Importer tool in the admin console and use it to import the XML file containing your posts. If the posts were created by the admin user on the old version, assign them to the admin user on the new site instead of trying to import the admin user.
4. If you deployed the new version to a separate environment, do a [CNAME swap \(p. 405\)](#) to redirect users from the old site to the new site.

Clean up

When you finish working with Elastic Beanstalk, you can terminate your environment. Elastic Beanstalk terminates all AWS resources associated with your environment, such as [Amazon EC2 instances \(p. 451\)](#), [database instances \(p. 506\)](#), [load balancers \(p. 470\)](#), security groups, and [alarms \(p. 470\)](#).

To terminate your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

With Elastic Beanstalk, you can easily create a new environment for your application at any time.

In addition, you can terminate database resources that you created outside of your Elastic Beanstalk environment. When you terminate an Amazon RDS DB instance, you can take a snapshot and restore the data to another instance later.

To terminate your RDS DB instance

1. Open the [Amazon RDS console](#).
2. Choose **Databases**.

3. Choose your DB instance.
4. Choose **Actions**, and then choose **Delete**.
5. Choose whether to create a snapshot, and then choose **Delete**.

Next steps

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The [Elastic Beanstalk Command Line Interface \(p. 852\)](#) (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

The sample application uses configuration files to configure PHP settings and create a table in the database, if it doesn't already exist. You can also use a configuration file to configure the security group settings of your instances during environment creation to avoid time-consuming configuration updates. See [Advanced environment customization with configuration files \(.ebextensions\) \(p. 600\)](#) for more information.

For development and testing, you might want to use the Elastic Beanstalk functionality for adding a managed DB instance directly to your environment. For instructions on setting up a database inside your environment, see [Adding a database to your Elastic Beanstalk environment \(p. 506\)](#).

If you need a high-performance database, consider using [Amazon Aurora](#). Amazon Aurora is a MySQL-compatible database engine that offers commercial database features at low cost. To connect your application to a different database, repeat the [security group configuration \(p. 254\)](#) steps and [update the RDS-related environment properties \(p. 257\)](#).

Finally, if you plan on using your application in a production environment, you will want to [configure a custom domain name \(p. 534\)](#) for your environment and [enable HTTPS \(p. 652\)](#) for secure connections.

Deploying a high-availability Drupal website with an external Amazon RDS database to Elastic Beanstalk

This tutorial walks you through the process of [launching an RDS DB instance \(p. 820\)](#) external to AWS Elastic Beanstalk. Then it describes configuring a high-availability environment running a Drupal website to connect to it. The website uses Amazon Elastic File System (Amazon EFS) as shared storage for uploaded files. Running a DB instance external to Elastic Beanstalk decouples the database from the lifecycle of your environment, and lets you connect to the same database from multiple environments, swap out one database for another, or perform a blue/green deployment without affecting your database.

Sections

- [Prerequisites \(p. 274\)](#)
- [Launch a DB instance in Amazon RDS \(p. 274\)](#)
- [Launch an Elastic Beanstalk environment \(p. 276\)](#)
- [Configure security settings and environment properties \(p. 277\)](#)
- [Configure and deploy your application \(p. 280\)](#)
- [Install Drupal \(p. 281\)](#)
- [Update Drupal configuration and remove access restrictions \(p. 282\)](#)
- [Configure autoscaling \(p. 284\)](#)
- [Cleanup \(p. 284\)](#)

- [Next steps \(p. 285\)](#)

Prerequisites

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in [Getting started using Elastic Beanstalk \(p. 3\)](#) to launch your first Elastic Beanstalk environment.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command  
this is output
```

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

The procedures in this tutorial for Amazon Relational Database Service (Amazon RDS) tasks assume that you are launching resources in a default [Amazon Virtual Private Cloud](#) (Amazon VPC). All new accounts include a default VPC in each region. If you don't have a default VPC, the procedures will vary. See [Using Elastic Beanstalk with Amazon RDS \(p. 820\)](#) for instructions for EC2-Classic and custom VPC platforms.

The sample application uses Amazon EFS. It only works in AWS Regions that support Amazon EFS. To learn about supporting AWS Regions, see [Amazon Elastic File System Endpoints and Quotas](#) in the *AWS General Reference*.

This tutorial was developed with Drupal version 8.5.3 and PHP 7.0.

Launch a DB instance in Amazon RDS

To use an external database with an application running in Elastic Beanstalk, first launch a DB instance with Amazon RDS. When you launch an instance with Amazon RDS, it is completely independent of Elastic Beanstalk and your Elastic Beanstalk environments, and will not be terminated or monitored by Elastic Beanstalk.

Use the Amazon RDS console to launch a Multi-AZ **MySQL** DB instance. Choosing a Multi-AZ deployment ensures that your database will failover and continue to be available if the master DB instance goes out of service.

To launch an RDS DB instance in a default VPC

1. Open the [RDS console](#).
2. Choose **Databases** in the navigation pane.
3. Choose **Create database**.
4. Choose **Standard Create**.

Important

Do not choose **Easy Create**. It does not allow you to configure the necessary settings to launch this RDS DB.

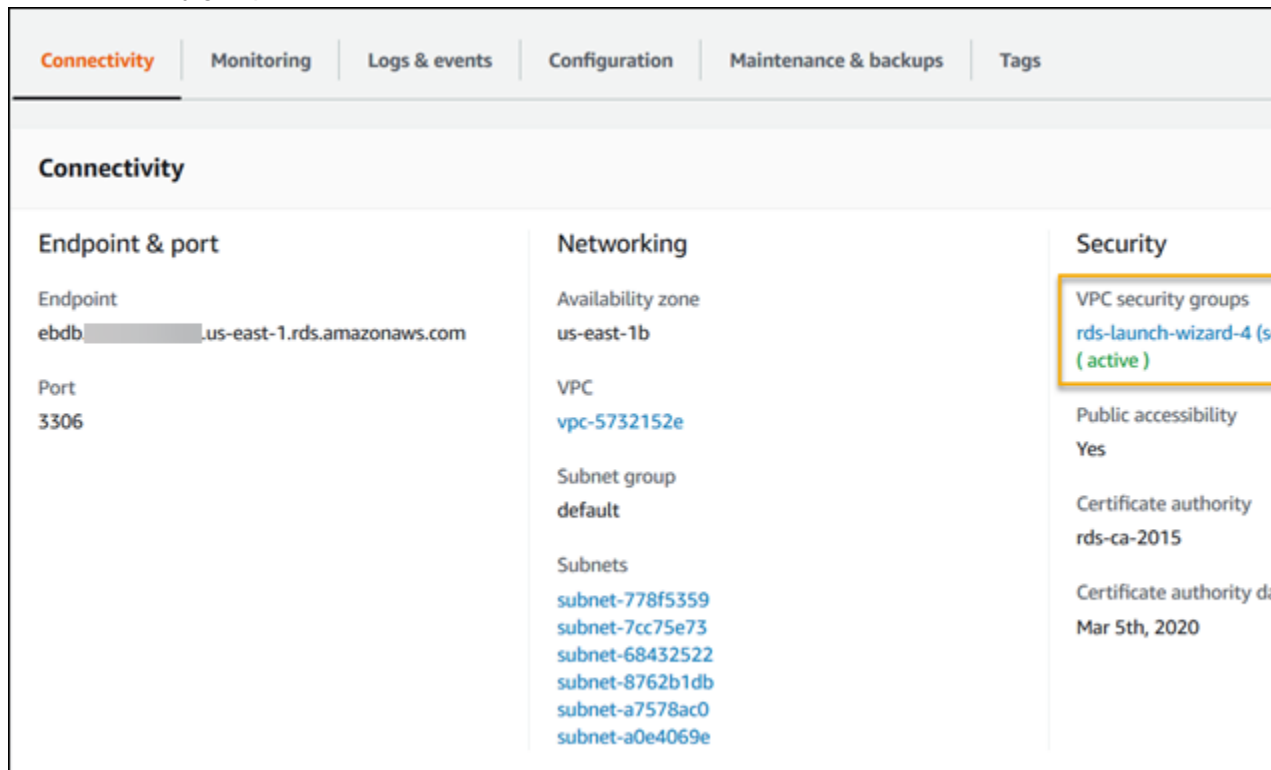
5. Under **Additional configuration**, for **Initial database name**, type **ebdb**.
6. Review the default settings carefully and adjust as necessary. Pay attention to the following options:
 - **DB instance class** – Choose an instance size that has an appropriate amount of memory and CPU power for your workload.

- **Multi-AZ deployment** – For high availability, set this to **Create an Aurora Replica/Reader node in a different AZ**.
 - **Master username** and **Master password** – The database username and password. Make a note of these settings because you'll use them later.
7. Verify the default settings for the remaining options, and then choose **Create database**.

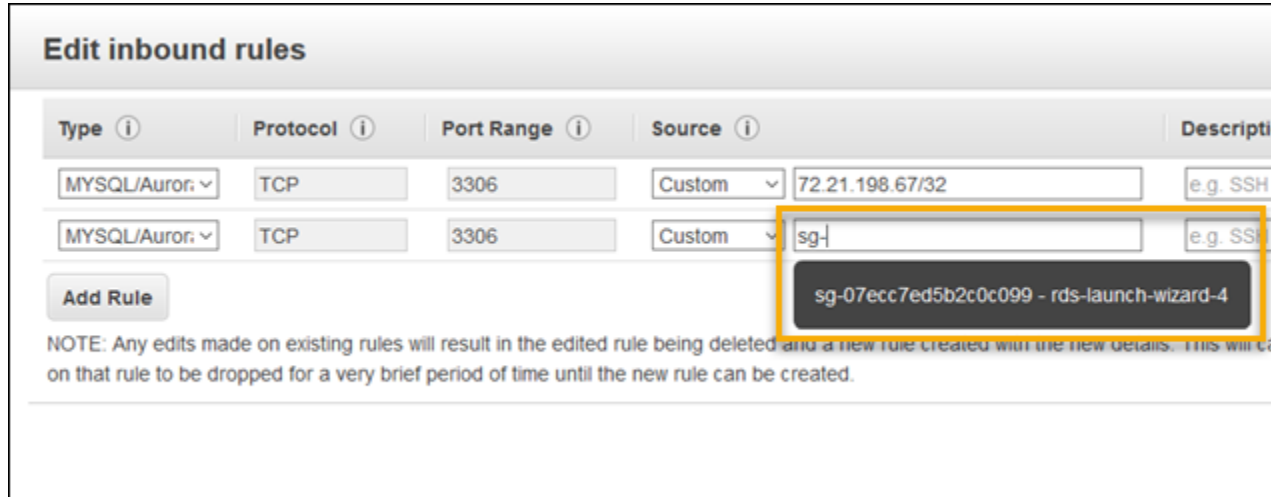
Next, modify the security group attached to your DB instance to allow inbound traffic on the appropriate port. This is the same security group that you will attach to your Elastic Beanstalk environment later, so the rule that you add will grant ingress permission to other resources in the same security group.

To modify the inbound rules on your RDS instance's security group

1. Open the [Amazon RDS console](#).
2. Choose **Databases**.
3. Choose the name of your DB instance to view its details.
4. In the **Connectivity** section, make a note of the **Subnets**, **Security groups**, and **Endpoint** shown on this page so you can use this information later.
5. Under **Security**, you can see the security group associated with the DB instance. Open the link to view the security group in the Amazon EC2 console.



6. In the security group details, choose **Inbound**.
7. Choose **Edit**.
8. Choose **Add Rule**.
9. For **Type**, choose the DB engine that your application uses.
10. For **Source**, type **sg-** to view a list of available security groups. Choose the current security group to allow resources in the security group to receive traffic on the database port from other resources in the same group.



11. Choose **Save**.

Creating a DB instance takes about 10 minutes. In the meantime, launch your Elastic Beanstalk environment.

Launch an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. Choose the **PHP** platform and accept the default settings and sample code. After you launch the environment, you can configure the environment to connect to the database, then deploy the Drupal code to the environment.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link:
console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or nginx as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.

- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
- **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains. The RDS DB instance that you launched is outside of your environment, so you are responsible for managing its lifecycle.

Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see [Using Elastic Beanstalk with Amazon S3 \(p. 831\)](#).

Configure security settings and environment properties

Add the security group of your DB instance to your running environment. This procedure causes Elastic Beanstalk to reprovision all instances in your environment with the additional security group attached.

To add a security group to your environment

- Do one of the following:
 - To add a security group using the Elastic Beanstalk console
 - a. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
 - b. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

- c. In the navigation pane, choose **Configuration**.
 - d. In the **Instances** configuration category, choose **Edit**.
 - e. Under **EC2 security groups**, choose the security group to attach to the instances, in addition to the instance security group that Elastic Beanstalk creates.
 - f. Choose **Apply**.
 - g. Read the warning, and then choose **Confirm**.
- To add a security group using a [configuration file \(p. 600\)](#), use the `securitygroup-addexisting.config` example file.

Next, use environment properties to pass the connection information to your environment. The sample application uses a default set of properties that match the ones that Elastic Beanstalk configures when you provision a database within your environment.

To configure environment properties for an Amazon RDS DB instance

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. In the **Environment properties** section, define the variables that your application reads to construct a connection string. For compatibility with environments that have an integrated RDS DB instance, use the following names and values. You can find all values, except for your password, in the [RDS console](#).

Property name	Description	Property value
RDS_HOSTNAME	The hostname of the DB instance.	On the Connectivity & security tab on the Amazon RDS console: Endpoint .
RDS_PORT	The port on which the DB instance accepts connections. The default value varies among DB engines.	On the Connectivity & security tab on the Amazon RDS console: Port .
RDS_DB_NAME	The database name, ebdb .	On the Configuration tab on the Amazon RDS console: DB Name .
RDS_USERNAME	The username that you configured for your database.	On the Configuration tab on the Amazon RDS console: Master username .
RDS_PASSWORD	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

Environment Properties

The following properties are passed into the application as environment variables. [Learn more.](#)

Property Name	Property Value
RDS_DB_NAME	<input type="text" value="ebdb"/>
RDS_HOSTNAME	<input type="text" value="webapp-db.jxzc5mpan"/>
RDS_PORT	<input type="text" value="5432"/>
RDS_USERNAME	<input type="text" value="webapp-admin"/>
<input type="text" value="RDS_PASSWORD"/>	<input type="text" value="kUj5uKxmWDMYc403"/>

[Cancel](#)

6. Choose **Apply**.

After installing Drupal, you need to connect to the instance with SSH to retrieve some configuration details. Assign an SSH key to your environment's instances.

To configure SSH

1. If you haven't previously created a key pair, open the [key pairs page](#) of the Amazon EC2 console and follow the instructions to create one.
2. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
3. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

4. In the navigation pane, choose **Configuration**.
5. Under **Security**, choose **Edit**.
6. For **EC2 key pair**, choose your key pair.
7. Choose **Apply**.

Configure and deploy your application

To create a Drupal project for Elastic Beanstalk, download the Drupal source code and combine it with the files in the [aws-samples/eb-php-drupal](https://github.com/aws-samples/eb-php-drupal) repository on GitHub.

To create a Drupal project

1. Download Drupal from <https://www.drupal.org/download>.

```
~$ curl https://ftp.drupal.org/files/projects/drupal-8.5.3.tar.gz -o drupal.tar.gz
```

2. Download the configuration files from the sample repository:

```
~$ wget https://github.com/aws-samples/eb-php-drupal/releases/download/v1.1/eb-php-drupal-v1.zip
```

3. Extract Drupal and change the name of the folder.

```
~$ tar -xvf drupal.tar.gz
~$ mv drupal-8.5.3 drupal-beanstalk
~$ cd drupal-beanstalk
```

4. Extract the configuration files over the Drupal installation.

```
~/drupal-beanstalk$ unzip ../eb-php-drupal-v1.zip
creating: .ebextensions/
inflating: .ebextensions/dev.config
inflating: .ebextensions/drupal.config
inflating: .ebextensions/efs-create.config
inflating: .ebextensions/efs-filesystem.template
inflating: .ebextensions/efs-mount.config
inflating: .ebextensions/loadbalancer-sg.config
inflating: LICENSE
inflating: README.md
inflating: beanstalk-settings.php
```

Verify that the structure of your `drupal-beanstalk` folder is correct, as shown.

```
drupal-beanstalk$ tree -aL 1
.
### autoload.php
### beanstalk-settings.php
### composer.json
### composer.lock
### core
### .csslintrc
### .ebextensions
### .ebextensions
### .editorconfig
### .eslintignore
### .eslintrc.json
### example.gitignore
### .gitattributes
### .htaccess
### .ht.router.php
### index.php
### LICENSE
### LICENSE.txt
### modules
```

```
### profiles
### README.md
### README.txt
### robots.txt
### sites
### themes
### update.php
### vendor
### web.config
```

The `beanstalk-settings.php` file from the project repo uses the environment variables that you defined in the previous step to configure the database connection. The `.ebextensions` folder contains configuration files that create additional resources within your Elastic Beanstalk environment.

The configuration files require modification to work with your account. Replace the placeholder values in the files with the appropriate IDs and create a source bundle.

To update configuration files and create a source bundle.

1. Modify the configuration files as follows.
 - `.ebextensions/dev.config` – restricts access to your environment to your IP address to protect it during the Drupal installation process. Replace the placeholder IP address near the top of the file with your public IP address.
 - `.ebextensions/efs-create.config` – creates an EFS file system and mount points in each Availability Zone / subnet in your VPC. Identify your default VPC and subnet IDs in the [Amazon VPC console](#).
2. Create a [source bundle](#) (p. 342) containing the files in your project folder. The following command creates a source bundle named `drupal-beanstalk.zip`. It excludes files in the `vendor` folder, which take up a lot of space and are not necessary for deploying your application to Elastic Beanstalk.

```
~/eb-drupal$ zip ../drupal-beanstalk.zip -r * .[^.]* -x "vendor/*"
```

Upload the source bundle to Elastic Beanstalk to deploy Drupal to your environment.

To deploy a source bundle

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Install Drupal

To complete your Drupal installation

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose the environment URL to open your site in a browser. You are redirected to a Drupal installation wizard because the site has not been configured yet.
4. Perform a standard installation with the following settings for the database:
 - **Database name** – The **DB Name** shown in the Amazon RDS console.
 - **Database username and password** – The **Master Username** and **Master Password** values you entered when creating your database.
 - **Advanced Options > Host** – The **Endpoint** of the DB instance shown in the Amazon RDS console.

Installation takes about a minute to complete.

Update Drupal configuration and remove access restrictions

The Drupal installation process created a file named `settings.php` in the `sites/default` folder on the instance. You need this file in your source code to avoid resetting your site on subsequent deployments, but the file currently contains secrets that you don't want to commit to source. Connect to the application instance to retrieve information from the settings file.

To connect to your application instance with SSH

1. Open the [instances page](#) of the Amazon EC2 console.
2. Choose the application instance. It is the one named after your Elastic Beanstalk environment.
3. Choose **Connect**.
4. Follow the instructions to connect the instance with SSH. The command looks similar to the following.

```
$ ssh -i ~/.ssh/mykey ec2-user@ec2-00-55-33-222.us-west-2.compute.amazonaws.com
```

Get the sync directory id from the last line of the settings file.

```
[ec2-user ~]$ tail -n 1 /var/app/current/sites/default/settings.php
$config_directories['sync'] = 'sites/default/files/
config_4ccfX2sPQm79p1mk5IbUq9S_FokcEN04mxyC-L18-4g_xKj_7j9ydn31kDOYOgnzMu071Tvc4Q/sync';
```

The file also contains the sites current hash key, but you can ignore the current value and use your own.

Assign the sync directory path and hash key to environment properties. The customized settings file from the project repo reads these properties to configure the site during deployment, in addition to the database connection properties that you set earlier.

Drupal configuration properties

- `SYNC_DIR` – The path to the sync directory.
- `HASH_SALT` – Any string value that meets [environment property requirements \(p. 518\)](#).

To configure environment properties in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. Under **Environment properties**, enter key-value pairs.

Name	Value
JDBC_CONNECTION_STRING	<input type="text"/>
XRAY_ENABLED	<input :="" \"n:'"="" fn::getoptionsetting\"="" type="text" value="{ \" {=""/>
<input type="text"/>	<input type="text"/>

6. Choose **Apply**.

Finally, the sample project includes a configuration file (`loadbalancer-sg.config`) that creates a security group and assigns it to the environment's load balancer, using the IP address that you configured in `dev.config` to restrict HTTP access on port 80 to connections from your network. Otherwise, an outside party could potentially connect to your site before you have installed Drupal and configured your admin account.

To update Drupal's configuration and remove access restrictions

1. Delete the `.ebextensions/loadbalancer-sg.config` file from your project directory.

```
~/drupal-beanstalk$ rm .ebextensions/loadbalancer-sg.config
```

2. Copy the customized `settings.php` file into the sites folder.

```
~/drupal-beanstalk$ cp beanstalk-settings.php sites/default/settings.php
```

3. Create a source bundle.

```
~/eb-drupal$ zip ../drupal-beanstalk-v2.zip -r * [^.]* -x "vendor/*"
```

Upload the source bundle to Elastic Beanstalk to deploy Drupal to your environment.

To deploy a source bundle

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Configure autoscaling

Finally, configure your environment's Auto Scaling group with a higher minimum instance count. Run at least two instances at all times to prevent the web servers in your environment from being a single point of failure, and to allow you to deploy changes without taking your site out of service.

To configure your environment's Auto Scaling group for high availability

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Auto Scaling group** section, set **Min instances** to **2**.
6. Choose **Apply**.

To support content uploads across multiple instances, the sample project uses Amazon Elastic File System to create a shared file system. Create a post on the site and upload content to store it on the shared file system. View the post and refresh the page multiple times to hit both instances and verify that the shared file system is working.

Cleanup

When you finish working with Elastic Beanstalk, you can terminate your environment. Elastic Beanstalk terminates all AWS resources associated with your environment, such as [Amazon EC2 instances \(p. 451\)](#), [database instances \(p. 506\)](#), [load balancers \(p. 470\)](#), security groups, and [alarms \(p. 470\)](#).

To terminate your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

With Elastic Beanstalk, you can easily create a new environment for your application at any time.

In addition, you can terminate database resources that you created outside of your Elastic Beanstalk environment. When you terminate an Amazon RDS DB instance, you can take a snapshot and restore the data to another instance later.

To terminate your RDS DB instance

1. Open the [Amazon RDS console](#).
2. Choose **Databases**.
3. Choose your DB instance.
4. Choose **Actions**, and then choose **Delete**.
5. Choose whether to create a snapshot, and then choose **Delete**.

Next steps

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The [Elastic Beanstalk Command Line Interface \(p. 852\)](#) (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

The sample application uses configuration files to configure PHP settings and create a table in the database if it doesn't already exist. You can also use a configuration file to configure your instances' security group settings during environment creation to avoid time-consuming configuration updates. See [Advanced environment customization with configuration files \(.ebextensions\) \(p. 600\)](#) for more information.

For development and testing, you might want to use the Elastic Beanstalk functionality for adding a managed DB instance directly to your environment. For instructions on setting up a database inside your environment, see [Adding a database to your Elastic Beanstalk environment \(p. 506\)](#).

If you need a high-performance database, consider using [Amazon Aurora](#). Amazon Aurora is a MySQL-compatible database engine that offers commercial database features at low cost. To connect your application to a different database, repeat the [security group configuration \(p. 254\)](#) steps and [update the RDS-related environment properties \(p. 257\)](#).

Finally, if you plan on using your application in a production environment, you will want to [configure a custom domain name \(p. 534\)](#) for your environment and [enable HTTPS \(p. 652\)](#) for secure connections.

Adding an Amazon RDS DB instance to your PHP application environment

You can use an Amazon Relational Database Service (Amazon RDS) DB instance to store data gathered and modified by your application. The database can be attached to your environment and managed by Elastic Beanstalk, or created and managed externally.

If you are using Amazon RDS for the first time, [add a DB instance \(p. 286\)](#) to a test environment with the Elastic Beanstalk console and verify that your application can connect to it.

To connect to a database, [add the driver \(p. 287\)](#) to your application, load the driver class in your code, and [create a connection object \(p. 287\)](#) with the environment properties provided by Elastic Beanstalk. The configuration and connection code vary depending on the database engine and framework that you use.

Note

For learning purposes or test environments, you can use Elastic Beanstalk to add a DB instance. For production environments, you can create a DB instance outside of your Elastic Beanstalk environment to decouple your environment resources from your database resources. This way, when you terminate your environment, the DB instance isn't deleted. An external DB instance also lets you connect to the same database from multiple environments and perform [blue-green deployments](#). For instructions, see [Using Elastic Beanstalk with Amazon RDS \(p. 820\)](#).

Sections

- [Adding a DB instance to your environment \(p. 286\)](#)
- [Downloading a driver \(p. 287\)](#)
- [Connecting to a database with a PDO or MySQLi \(p. 287\)](#)
- [Connecting to a database with Symfony \(p. 287\)](#)

Adding a DB instance to your environment

To add a DB instance to your environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Choose a DB engine, and enter a user name and password.
6. Choose **Apply**.

Adding a DB instance takes about 10 minutes. When the environment update is complete, the DB instance's hostname and other connection information are available to your application through the following environment properties:

Property name	Description	Property value
RDS_HOSTNAME	The hostname of the DB instance.	On the Connectivity & security tab on the Amazon RDS console: Endpoint .
RDS_PORT	The port on which the DB instance accepts connections. The default value varies among DB engines.	On the Connectivity & security tab on the Amazon RDS console: Port .
RDS_DB_NAME	The database name, ebdb .	On the Configuration tab on the Amazon RDS console: DB Name .
RDS_USERNAME	The username that you configured for your database.	On the Configuration tab on the Amazon RDS console: Master username .
RDS_PASSWORD	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

For more information about configuring an internal DB instance, see [Adding a database to your Elastic Beanstalk environment](#) (p. 506).

Downloading a driver

To use PHP Data Objects (PDO) to connect to the database, install the driver that matches the database engine that you chose.

- **MySQL** – [PDO_MYSQL](#)
- **PostgreSQL** – [PDO_PGSQL](#)
- **Oracle** – [PDO_OCI](#)
- **SQL Server** – [PDO_SQLSRV](#)

For more information, see <http://php.net/manual/en/pdo.installation.php>.

Connecting to a database with a PDO or MySQLi

You can use `$_SERVER[VARIABLE]` to read connection information from the environment.

For a PDO, create a Data Source Name (DSN) from the host, port, and name. Pass the DSN to the [constructor for the PDO](#) with the database user name and password.

Example Connect to an RDS database with PDO - MySQL

```
<?php
$dbhost = $_SERVER['RDS_HOSTNAME'];
$dbport = $_SERVER['RDS_PORT'];
$dbname = $_SERVER['RDS_DB_NAME'];
$charset = 'utf8' ;

$dsn = "mysql:host={$dbhost};port={$dbport};dbname={$dbname};charset={$charset}";
$username = $_SERVER['RDS_USERNAME'];
$password = $_SERVER['RDS_PASSWORD'];

$pdo = new PDO($dsn, $username, $password);
?>
```

For other drivers, replace `mysql` with the name of your driver – `pgsql`, `oci`, or `sqlsrv`.

For MySQLi, pass the hostname, user name, password, database name, and port to the `mysqli` constructor.

Example Connect to an RDS database with `mysqli_connect()`

```
$link = new mysqli($_SERVER['RDS_HOSTNAME'], $_SERVER['RDS_USERNAME'],
    $_SERVER['RDS_PASSWORD'], $_SERVER['RDS_DB_NAME'], $_SERVER['RDS_PORT']);
```

Connecting to a database with Symfony

For Symfony version 3.2 and newer, you can use `%env(PROPERTY_NAME)%` to set database parameters in a configuration file based on the environment properties set by Elastic Beanstalk.

Example `app/config/parameters.yml`

```
parameters:
    database_driver:    pdo_mysql
    database_host:     '%env(RDS_HOSTNAME)%'
```

```
database_port:      '%env(RDS_PORT)%'  
database_name:     '%env(RDS_DB_NAME)%'  
database_user:     '%env(RDS_USERNAME)%'  
database_password: '%env(RDS_PASSWORD)%'
```

See [External Parameters \(Symfony 3.4\)](#) for more information.

For earlier versions of Symfony, environment variables are only accessible if they start with `SYMFONY__`. This means that the Elastic Beanstalk-defined environment properties are not accessible, and you must define your own environment properties to pass the connection information to Symfony.

To connect to a database with Symfony 2, [create an environment property \(p. 232\)](#) for each parameter. Then, use `property.name` to access the Symfony-transformed variable in a configuration file. For example, an environment property named `SYMFONY__DATABASE__USER` is accessible as `database.user`.

```
database_user:     "%database.user%"
```

See [External Parameters \(Symfony 2.8\)](#) for more information.

Working with Python

This section provides tutorials and information about deploying Python applications using AWS Elastic Beanstalk.

The topics in this chapter assume some knowledge of Elastic Beanstalk environments. If you haven't used Elastic Beanstalk before, try the [getting started tutorial \(p. 3\)](#) to learn the basics.

Topics

- [Setting up your Python development environment \(p. 288\)](#)
- [Using the Elastic Beanstalk Python platform \(p. 290\)](#)
- [Deploying a flask application to Elastic Beanstalk \(p. 295\)](#)
- [Deploying a Django application to Elastic Beanstalk \(p. 301\)](#)
- [Adding an Amazon RDS DB instance to your Python application environment \(p. 311\)](#)
- [Python tools and resources \(p. 313\)](#)

Setting up your Python development environment

Set up a Python development environment to test your application locally prior to deploying it to AWS Elastic Beanstalk. This topic outlines development environment setup steps and links to installation pages for useful tools.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command  
this is output
```

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

For common setup steps and tools that apply to all languages, see [Configuring your development machine \(p. 849\)](#).

Sections

- [Installing Python and pip \(p. 289\)](#)
- [Using a virtual environment \(p. 289\)](#)
- [Configuring a Python project for Elastic Beanstalk \(p. 290\)](#)

Installing Python and pip

For all Python applications that you'll deploy with Elastic Beanstalk, these prerequisites are common:

1. Python 2.7 or 3.4.
2. The `pip` utility, matching your Python version. This is used to install and list dependencies for your project, so that Elastic Beanstalk knows how to set up your application's environment.
3. The `virtualenv` package. This is used to create an environment used to develop and test your application, so that the environment can be replicated by Elastic Beanstalk without installing extra packages that aren't needed by your application.
4. The `awsebcli` package. This is used to initialize your application with the files necessary for deploying with Elastic Beanstalk.
5. A working `ssh` installation. This is used to connect with your running instances when you need to examine or debug a deployment.

For instructions on installing Python, pip, and the EB CLI, see [Install the EB CLI \(p. 853\)](#).

Using a virtual environment

Once you have the prerequisites installed, set up a virtual environment with `virtualenv` to install your application's dependencies. By using a virtual environment, you can discern exactly which packages are needed by your application so that the required packages are installed on the EC2 instances that are running your application.

To set up a virtual environment

1. Open a command-line window and type:

```
virtualenv -p python2.7 /tmp/eb_python_app
```

Replace `eb_python_app` with a name that makes sense for your application (using your application's name or directory name is a good idea). The `virtualenv` command creates a virtual environment for you and prints the results of its actions:

```
Running virtualenv with interpreter /usr/bin/python2.7
New python executable in /tmp/eb_python_app/bin/python2.7
Also creating executable in /tmp/eb_python_app/bin/python
Installing setuptools, pip...done.
```

2. Once your virtual environment is ready, start it by running the `activate` script located in the environment's `bin` directory. For example, to start the `eb_python_app` environment created in the previous step, you would type:

```
./tmp/eb_python_app/bin/activate
```

The virtual environment prints its name (for example: `(eb_python_app)`) at the beginning of each command prompt, reminding you that you're in a virtual Python environment.

Note

Once created, you can restart the virtual environment at any time by running its `activate` script again.

Configuring a Python project for Elastic Beanstalk

You can use the Elastic Beanstalk CLI to prepare your Python applications for deployment with Elastic Beanstalk.

To configure a Python application for deployment with Elastic Beanstalk

1. From within your [virtual environment \(p. 289\)](#), return to the top of your project's directory tree (`python_eb_app`), and type:

```
pip freeze >requirements.txt
```

This command copies the names and versions of the packages that are installed in your virtual environment to `requirements.txt`. For example, if the `PyYAML` package, version `3.11` is installed in your virtual environment, the file will contain the line:

```
PyYAML==3.11
```

This allows Elastic Beanstalk to replicate your application's Python environment using the same packages and same versions that you used to develop and test your application.

2. Configure the EB CLI repository with the `eb init` command. Follow the prompts to choose a region, platform and other options. For detailed instructions, see [Managing Elastic Beanstalk environments with the EB CLI \(p. 864\)](#).

By default, Elastic Beanstalk looks for a file called `application.py` to start your application. If this doesn't exist in the Python project that you've created, some adjustment of your application's environment is necessary. You will also need to set environment variables so that your application's modules can be loaded. See [Using the Elastic Beanstalk Python platform \(p. 290\)](#) for more information.

Using the Elastic Beanstalk Python platform

Important

Amazon Linux 2 platform versions are fundamentally different than Amazon Linux AMI platform versions (preceding Amazon Linux 2). These different platform generations are incompatible in several ways. If you are migrating to an Amazon Linux 2 platform version, be sure to read the information in [the section called "Upgrade to Amazon Linux 2" \(p. 426\)](#).

The AWS Elastic Beanstalk Python platform is a set of [platform versions](#) for Python web applications that can run behind a proxy server with WSGI. Each platform branch corresponds to a version of Python, such as Python 3.4.

Starting with Amazon Linux 2 platform branches, Elastic Beanstalk provides [Gunicorn](#) as the default WSGI server.

You can add a `Procfile` to your source bundle to specify and configure the WSGI server for your application. For details, see [the section called "Procfile" \(p. 293\)](#).

You can use the `Pipfile` and `Pipfile.lock` files created by Pipenv to specify Python package dependencies and other requirements. For details about specifying dependencies, see [the section called "Specifying dependencies" \(p. 294\)](#).

Elastic Beanstalk provides [configuration options \(p. 536\)](#) that you can use to customize the software that runs on the EC2 instances in your Elastic Beanstalk environment. You can configure environment

variables needed by your application, enable log rotation to Amazon S3, and map folders in your application source that contain static files to paths served by the proxy server.

Configuration options are available in the Elastic Beanstalk console for [modifying the configuration of a running environment \(p. 547\)](#). To avoid losing your environment's configuration when you terminate it, you can use [saved configurations \(p. 640\)](#) to save your settings and later apply them to another environment.

To save settings in your source code, you can include [configuration files \(p. 600\)](#). Settings in configuration files are applied every time you create an environment or deploy your application. You can also use configuration files to install packages, run scripts, and perform other instance customization operations during deployments.

Settings applied in the Elastic Beanstalk console override the same settings in configuration files, if they exist. This lets you have default settings in configuration files, and override them with environment-specific settings in the console. For more information about precedence, and other methods of changing settings, see [Configuration options \(p. 536\)](#).

For Python packages available from `pip`, you can include a requirements file in the root of your application source code. Elastic Beanstalk installs any dependency packages specified in a requirements file during deployment. For details, see [the section called "Specifying dependencies" \(p. 294\)](#).

For details about the various ways you can extend an Elastic Beanstalk Linux-based platform, see [the section called "Extending Linux platforms" \(p. 31\)](#).

Configuring your Python environment

The Python platform settings let you fine-tune the behavior of your Amazon EC2 instances. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration using the Elastic Beanstalk console.

Use the Elastic Beanstalk console to configure Python process settings, enable AWS X-Ray, enable log rotation to Amazon S3, and configure variables that your application can read from the environment.

To configure your Python environment in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.

Python settings

- **WSGI Path** – The name of or path to your main application file. For example, `application.py`, or `django/wsgi.py`.
- **NumProcesses** – The number of processes to run on each application instance.
- **NumThreads** – The number of threads to run in each process.

AWS X-Ray settings

- **X-Ray daemon** – Run the AWS X-Ray daemon to process trace data from the [AWS X-Ray SDK for Python](#).

Log options

The Log Options section has two settings:

- **Instance profile**– Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3** – Specifies whether log files for your application's Amazon EC2 instances should be copied to the Amazon S3 bucket associated with your application.

Static files

To improve performance, the **Static files** section lets you configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. For each directory, you set the virtual path to directory mapping. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

For details about configuring static files using the Elastic Beanstalk console, see [the section called “Static files” \(p. 650\)](#).

By default, the proxy server in a Python environment serves any files in a folder named `static` at the `/static` path. For example, if your application source contains a file named `logo.png` in a folder named `static`, the proxy server serves it to users at `subdomain.elasticbeanstalk.com/static/logo.png`. You can configure additional mappings as explained in this section.

Environment properties

You can use environment properties to provide information to your application and configure environment variables. For example, you can create an environment property named `CONNECTION_STRING` that specifies a connection string that your application can use to connect to a database.

Inside the Python environment running in Elastic Beanstalk, these values are accessible using Python's `os.environ` dictionary. For more information, go to <http://docs.python.org/library/os.html>.

You can use code that looks similar to the following to access the keys and parameters:

```
import os
endpoint = os.environ['API_ENDPOINT']
```

Environment properties can also provide information to a framework. For example, you can create a property named `DJANGO_SETTINGS_MODULE` to configure Django to use a specific settings module. Depending on the environment, the value could be `development.settings`, `production.settings`, etc.

See [Environment properties and other software settings \(p. 516\)](#) for more information.

Python configuration namespaces

You can use a [configuration file \(p. 600\)](#) to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be defined by the Elastic Beanstalk service or the platform that you use and are organized into *namespaces*.

The Python platform defines options in the `aws:elasticbeanstalk:environment:proxy:staticfiles` and `aws:elasticbeanstalk:container:python` namespaces.

The following example configuration file specifies configuration option settings to create an environment property named `DJANGO_SETTINGS_MODULE`, two static files options that map a directory

named `statichtml` to the path `/html` and a directory named `staticimages` to the path `/images`, and additional settings in the [aws:elasticbeanstalk:container:python](#) (p. 597) namespace. This namespace contains options that let you specify the location of the WSGI script in your source code, and the number of threads and processes to run in WSGI.

```
option_settings:
  aws:elasticbeanstalk:application:environment:
    DJANGO_SETTINGS_MODULE: production.settings
  aws:elasticbeanstalk:environment:proxy:staticfiles:
    /html: statichtml
    /images: staticimages
  aws:elasticbeanstalk:container:python:
    WSGIPath: ebdjango.wsgi:application
    NumProcesses: 3
    NumThreads: 20
```

Notes

- If you're using an Amazon Linux AMI Python platform version (preceding Amazon Linux 2), replace the value for `WSGIPath` with `ebdjango/wsgi.py`. The value in the example works with the Gunicorn WSGI server, which isn't supported on Amazon Linux AMI platform versions.
- In addition, these older platform versions use a different namespace for configuring static files —`aws:elasticbeanstalk:container:python:staticfiles`. It has the same option names and semantics as the standard static file namespace.

Configuration files also support several keys to further [modify the software on your environment's instances](#) (p. 603). This example uses the [packages](#) (p. 604) key to install Memcached with yum and [container commands](#) (p. 611) to run commands that configure the server during deployment:

```
packages:
  yum:
    libmemcached-devel: '0.31'

container_commands:
  collectstatic:
    command: "django-admin.py collectstatic --noinput"
  01syncdb:
    command: "django-admin.py syncdb --noinput"
    leader_only: true
  02migrate:
    command: "django-admin.py migrate"
    leader_only: true
  03wsgipass:
    command: 'echo "WSGI Pass Authorization On" >> ../wsgi.conf'
  99customize:
    command: "scripts/customize.sh"
```

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See [Configuration options](#) (p. 536) for more information.

Configuring the WSGI server with a Procfile

Important

Amazon Linux 2 platform versions are fundamentally different than Amazon Linux AMI platform versions (preceding Amazon Linux 2). These different platform generations are incompatible in several ways. If you are migrating to an Amazon Linux 2 platform version, be sure to read the information in [the section called "Upgrade to Amazon Linux 2"](#) (p. 426).

You can add a `Procfile` to your source bundle to specify and configure the WSGI server for your application. The following example uses a `Procfile` to specify uWSGI as the server and configure it.

Example Procfile

```
web: uwsgi --http :8000 --wsgi-file application.py --master --processes 4 --threads 2
```

The following example uses a `Procfile` to configure Gunicorn, the default WSGI server.

Example Procfile

```
web: gunicorn --bind :8000 --workers 3 --threads 2 project.wsgi:application
```

Notes

- If you configure any WSGI server other than Gunicorn, be sure to also specify it as a dependency of your application, so that it is installed on your environment instances. For details about dependency specification, see [the section called “Specifying dependencies” \(p. 294\)](#).
- The default port for the WSGI server is 8000. If you specify a different port number in your `Procfile` command, set the `PORT` [environment property \(p. 516\)](#) to this port number too.

When you use a `Procfile`, it overrides `aws:elasticbeanstalk:container:python` namespace options that you set using configuration files.

For details about `Procfile` usage, expand the *Buildfile and Procfile* section in [the section called “Extending Linux platforms” \(p. 31\)](#).

Specifying dependencies using a requirements file

A typical Python application has dependencies on other third-party Python packages. With the Elastic Beanstalk Python platform, you have a few ways to specify Python packages that your application depends on.

Use `pip` and `requirements.txt`

The standard tool for installing Python packages is `pip`. It has a feature that allows you to specify all the packages you need (as well as their versions) in a single requirements file. For more information about the requirements file, go to [Requirements File Format](#).

Create a file named `requirements.txt` and place it in the top-level directory of your source bundle. The following is an example `requirements.txt` file for Django.

```
Django==1.11.3  
MySQL-python==1.2.5
```

In your development environment, you can use the `pip freeze` command to generate your requirements file.

```
~/my-app$ pip freeze > requirements.txt
```

To ensure that your requirements file only contains packages that are actually used by your application, use a [virtual environment \(p. 289\)](#) that only has those packages installed. Outside of a virtual

environment, the output of `pip freeze` will include all `pip` packages installed on your development machine, including those that came with your operating system.

Note

On Amazon Linux AMI Python platform versions, Elastic Beanstalk doesn't natively support Pipenv or Pipfiles. If you use Pipenv to manage your application's dependencies, run the following command to generate a `requirements.txt` file.

```
~/my-app$ pipenv lock -r > requirements.txt
```

To learn more, see [Generating a requirements.txt](#) in the Pipenv documentation.

Use Pipenv and Pipfile

Pipenv is a modern Python packaging tool. It combines package installation with the creation and management of a dependency file and a virtualenv for your application. Pipenv maintains two files: `Pipfile` contains various types of dependencies and requirements, and `Pipfile.lock` is a version snapshot that enables deterministic builds. For more information, see [Pipenv: Python Dev Workflow for Humans](#).

Amazon Linux 2 Python platform versions support Pipenv-based requirements files. Create them on your development environment and include them with the source bundle that you deploy to Elastic Beanstalk.

Note

Amazon Linux AMI Python platform versions (preceding Amazon Linux 2) don't support Pipenv and `Pipfile`.

The following example uses Pipenv to install Django and the Django REST framework.

```
~/my-app$ pipenv install django
~/my-app$ pipenv install.djangorestframework
```

These commands create the files `Pipfile` and `Pipfile.lock`. Place `Pipfile` in the top-level directory of your source bundle to get latest versions of dependency packages installed on your environment instances. Alternatively, include `Pipfile.lock` to get a constant set of package versions reflecting your development environment at the time of the file's creation.

If you include more than one of the requirements files described here, Elastic Beanstalk uses just one of them. The following list shows the precedence, in descending order.

1. `requirements.txt`
2. `Pipfile.lock`
3. `Pipfile`

Deploying a flask application to Elastic Beanstalk

Flask is an open source web application framework for Python. This tutorial walks you through the process of generating a Flask application and deploying it to an AWS Elastic Beanstalk environment.

In this tutorial, you'll do the following:

- [Set up a Python virtual environment with flask \(p. 296\)](#)
- [Create a flask application \(p. 297\)](#)
- [Deploy your site with the EB CLI \(p. 298\)](#)
- [Cleanup \(p. 300\)](#)

Prerequisites

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in [Getting started using Elastic Beanstalk \(p. 3\)](#) to launch your first Elastic Beanstalk environment.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command  
this is output
```

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Flask requires Python 2.7 or 3.4 or newer. In this tutorial we use Python 3.6 and the corresponding Elastic Beanstalk platform version. Install Python by following the instructions at [Setting up your Python development environment \(p. 288\)](#).

The [Flask](#) framework will be installed as part of the tutorial.

This tutorial also uses the Elastic Beanstalk Command Line Interface (EB CLI). For details on installing and configuring the EB CLI, see [Install the EB CLI \(p. 853\)](#) and [Configure the EB CLI \(p. 860\)](#).

Set up a Python virtual environment with flask

Create a project directory and virtual environment for your application, and install Flask.

To set up your project environment

1. Create a project directory.

```
~$ mkdir eb-flask  
~$ cd eb-flask
```

2. Create and activate a virtual environment named `virt`:

```
~/eb-flask$ virtualenv virt  
~$ source virt/bin/activate  
(virt) ~/eb-flask$
```

You will see `(virt)` prepended to your command prompt, indicating that you're in a virtual environment. Use the virtual environment for the rest of this tutorial.

3. Install flask with `pip install`:

```
(virt)~/eb-flask$ pip install flask==1.0.2
```

4. View the installed libraries with `pip freeze`:

```
(virt)~/eb-flask$ pip freeze  
click==6.7  
Flask==1.0.2  
itsdangerous==0.24  
Jinja2==2.10  
MarkupSafe==1.0
```

```
Werkzeug==0.14.1
```

This command lists all of the packages installed in your virtual environment. Because you are in a virtual environment, globally installed packages like the EB CLI are not shown.

5. Save the output from `pip freeze` to a file named `requirements.txt`.

```
(virt)~/eb-flask$ pip freeze > requirements.txt
```

This file tells Elastic Beanstalk to install the libraries during deployment. For more information, see [Specifying dependencies using a requirements file \(p. 294\)](#).

Create a flask application

Next, create an application that you'll deploy using Elastic Beanstalk. We'll create a "Hello World" RESTful web service.

Create a new text file in this directory named `application.py` with the following contents:

Example `~/eb-flask/application.py`

```
from flask import Flask

# print a nice greeting.
def say_hello(username = "World"):
    return '<p>Hello %s!</p>\n' % username

# some bits of text for the page.
header_text = '''
<html>\n<head> <title>EB Flask Test</title> </head>\n<body>'''
instructions = '''
<p><em>Hint</em>: This is a RESTful web service! Append a username
to the URL (for example: <code>/Thelonious</code>) to say hello to
someone specific.</p>\n'''
home_link = '<p><a href="/">Back</a></p>\n'
footer_text = '</body>\n</html>'

# EB looks for an 'application' callable by default.
application = Flask(__name__)

# add a rule for the index page.
application.add_url_rule('/', 'index', (lambda: header_text +
    say_hello() + instructions + footer_text))

# add a rule when the page is accessed with a name appended to the site
# URL.
application.add_url_rule('/<username>', 'hello', (lambda username:
    header_text + say_hello(username) + home_link + footer_text))

# run the app.
if __name__ == "__main__":
    # Setting debug to True enables debug output. This line should be
    # removed before deploying a production app.
    application.debug = True
    application.run()
```

This example prints a customized greeting that varies based on the path used to access the service.

Note

By adding `application.debug = True` before running the application, debug output is enabled in case something goes wrong. It's a good practice for development, but you should

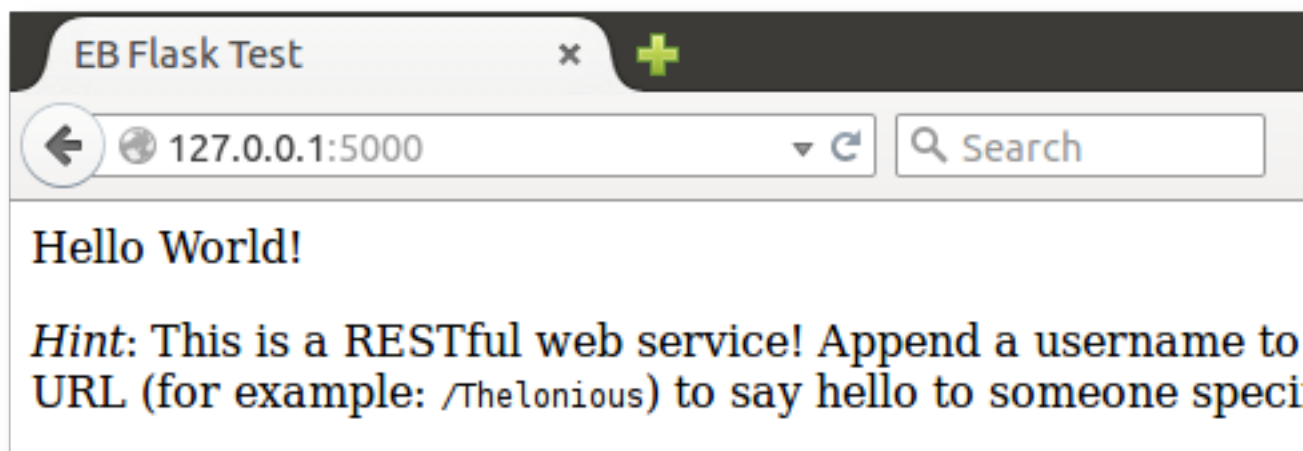
remove debug statements in production code, since debug output can reveal internal aspects of your application.

Using `application.py` as the filename and providing a callable `application` object (the Flask object, in this case) allows Elastic Beanstalk to easily find your application's code.

Run `application.py` with Python:

```
(virt) ~/eb-flask$ python application.py
* Serving Flask app "application" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 313-155-123
```

Open `http://127.0.0.1:5000/` in your web browser. You should see the application running, showing the index page:



Check the server log to see the output from your request. You can stop the web server and return to your virtual environment by typing **Ctrl+C**.

If you got debug output instead, fix the errors and make sure the application is running locally before configuring it for Elastic Beanstalk.

Deploy your site with the EB CLI

You've added everything you need to deploy your application on Elastic Beanstalk. Your project directory should now look like this:

```
~/eb-flask/
|-- virt
|-- application.py
`-- requirements.txt
```

The `virt` folder, however, is not required for the application to run on Elastic Beanstalk. When you deploy, Elastic Beanstalk creates a new virtual environment on the server instances and installs the

libraries listed in `requirements.txt`. To minimize the size of the source bundle that you upload during deployment, add an `.ebignore` (p. 862) file that tells the EB CLI to leave out the `virt` folder.

Example `~/Eb-flask/.ebignore`

```
virt
```

Next, you'll create your application environment and deploy your configured application with Elastic Beanstalk.

To create an environment and deploy your flask application

1. Initialize your EB CLI repository with the `eb init` command:

```
~/eb-flask$ eb init -p python-3.6 flask-tutorial --region us-east-2  
Application flask-tutorial has been created.
```

This command creates a new application named `flask-tutorial` and configures your local repository to create environments with the latest Python 3.6 platform version.

2. (optional) Run `eb init` again to configure a default keypair so that you can connect to the EC2 instance running your application with SSH:

```
~/eb-flask$ eb init  
Do you want to set up SSH for your instances?  
(y/n): y  
Select a keypair.  
1) my-keypair  
2) [ Create new KeyPair ]
```

Select a key pair if you have one already, or follow the prompts to create a new one. If you don't see the prompt or need to change your settings later, run `eb init -i`.

3. Create an environment and deploy your application to it with `eb create`:

```
~/eb-flask$ eb create flask-env
```

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or nginx as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.

- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
- **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

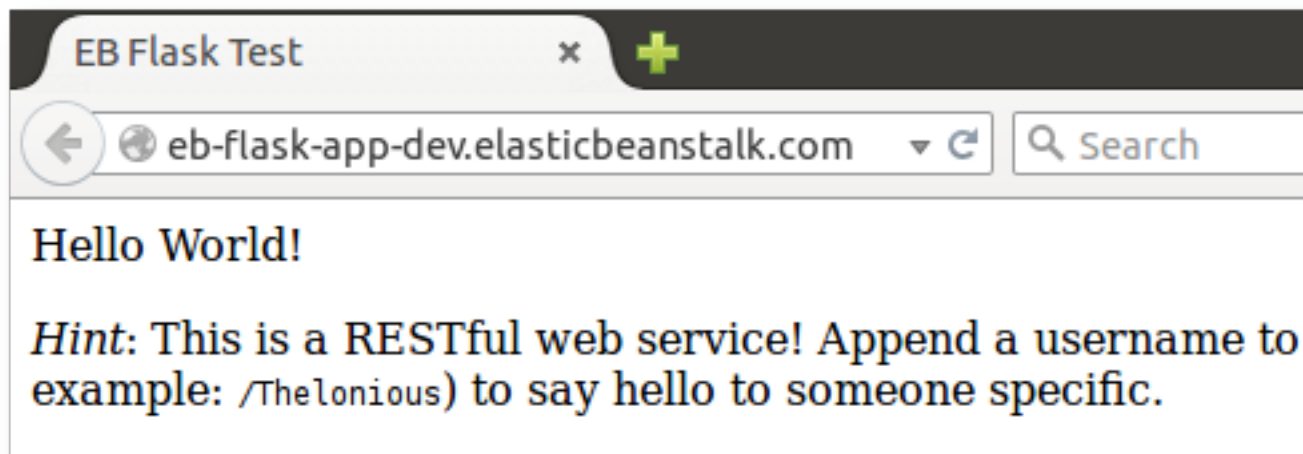
Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see [Using Elastic Beanstalk with Amazon S3 \(p. 831\)](#).

When the environment creation process completes, open your web site with **eb open**:

```
~/eb-flask$ eb open
```

This will open a browser window using the domain name created for your application. You should see the same Flask website that you created and tested locally.



If you don't see your application running, or get an error message, see [Troubleshooting Deployments \(p. 948\)](#) for help with how to determine the cause of the error.

If you *do* see your application running, then congratulations, you've deployed your first Flask application with Elastic Beanstalk!

Cleanup

When you finish working with Elastic Beanstalk, you can terminate your environment. Elastic Beanstalk terminates all AWS resources associated with your environment, such as [Amazon EC2](#)

[instances \(p. 451\)](#), [database instances \(p. 506\)](#), [load balancers \(p. 470\)](#), security groups, and [alarms \(p. 470\)](#).

To terminate your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

With Elastic Beanstalk, you can easily create a new environment for your application at any time.

Or, with the EB CLI:

```
~/eb-flask$ eb terminate flask-env
```

Next steps

For more information about Flask, visit flask.pocoo.org.

If you'd like to try out another Python web framework, check out [Deploying a Django application to Elastic Beanstalk \(p. 301\)](#).

Deploying a Django application to Elastic Beanstalk

This tutorial walks through the deployment of a default, autogenerated [Django](#) website to an AWS Elastic Beanstalk environment running Python. This tutorial shows you how to host a Python web app in the cloud by using an Elastic Beanstalk environment.

In this tutorial, you'll do the following:

- [Set up a Python virtual environment and install Django \(p. 302\)](#)
- [Create a Django project \(p. 303\)](#)
- [Configure your Django application for Elastic Beanstalk \(p. 304\)](#)
- [Deploy your site with the EB CLI \(p. 305\)](#)
- [Update your application \(p. 308\)](#)
- [Clean up \(p. 311\)](#)

Prerequisites

To use any AWS service, including Elastic Beanstalk, you need to have an AWS account and credentials. To learn more and to sign up, visit <https://aws.amazon.com/>.

To follow this tutorial, you should have all of the [Common Prerequisites \(p. 288\)](#) for Python installed, including the following packages:

- Python 3.6
- pip
- virtualenv
- awsebcli

The [Django](#) framework is installed as part of the tutorial.

Note

Creating environments with the EB CLI requires a [service role \(p. 20\)](#). You can create a service role by creating an environment in the Elastic Beanstalk console. If you don't have a service role, the EB CLI attempts to create one when you run `eb create`.

Set up a Python virtual environment and install Django

Create a virtual environment with `virtualenv` and use it to install Django and its dependencies. By using a virtual environment, you can know exactly which packages your application needs, so that the required packages are installed on the Amazon EC2 instances that are running your application.

The following steps demonstrate the commands you must enter for Unix-based systems and Windows, shown on separate tabs.

To set up your virtual environment

1. Create a virtual environment named `eb-virt`.

Unix-based systems

```
~$ virtualenv ~/eb-virt
```

Windows

```
C:\> virtualenv %HOMEPATH%\eb-virt
```

2. Activate the virtual environment.

Unix-based systems

```
~$ source ~/eb-virt/bin/activate  
(eb-virt) ~$
```

Windows

```
C:\>%HOMEPATH%\eb-virt\Scripts\activate  
(eb-virt) C:\>
```

You'll see `(eb-virt)` prepended to your command prompt, indicating that you're in a virtual environment.

Note

The rest of these instructions show the Linux command prompt in your home directory `~$`. On Windows this is `C:\Users\USERNAME>`, where *USERNAME* is your Windows login name.

3. Use `pip` to install Django.

```
(eb-virt)~$ pip install django==2.1.1
```

Note

The Django version you install must be compatible with the Python version on the Elastic Beanstalk Python configuration that you choose for deploying your application. For information about deployment, see [??? \(p. 305\)](#) in this topic.

For more information about current Python platform versions, see [Python](#) in the *AWS Elastic Beanstalk Platforms* document.

For Django version compatibility with Python, see [What Python version can I use with Django?](#) Django 2.2 is incompatible with the Elastic Beanstalk Python 3.6 platform. The latest compatible version is Django 2.1.

4. To verify that Django is installed, enter the following.

```
(eb-virt)~$ pip freeze
Django==2.1.1
...
```

This command lists all of the packages installed in your virtual environment. Later, you use the output of this command to configure your project for use with Elastic Beanstalk.

Create a Django project

Now you are ready to create a Django project and run it on your machine, using the virtual environment.

Note

This tutorial uses SQLite, which is a database engine included in Python. The database is deployed with your project files. For production environments, we recommend that you use Amazon Relational Database Service (Amazon RDS), and that you separate it from your environment. For more information, see [Adding an Amazon RDS DB instance to your Python application environment](#) (p. 311).

To generate a Django application

1. Activate your virtual environment.

Unix-based systems

```
~$ source ~/eb-virt/bin/activate
(eb-virt) ~$
```

Windows

```
C:\>%HOMEPATH%\eb-virt\Scripts\activate
(eb-virt) C:\>
```

You'll see the `(eb-virt)` prefix prepended to your command prompt, indicating that you're in a virtual environment.

Note

The rest of these instructions show the Linux command prompt `~$` in your home directory and the Linux home directory `~/`. On Windows these are `C:\Users\USERNAME>`, where *USERNAME* is your Windows login name.

2. Use the `django-admin startproject ebdjango` command to create a Django project named `ebdjango`.

```
(eb-virt)~$ django-admin startproject ebdjango
```

This command creates a standard Django site named **ebdjango** with the following directory structure.

```
~/ebdjango
|-- ebdjango
|   |-- __init__.py
|   |-- settings.py
```

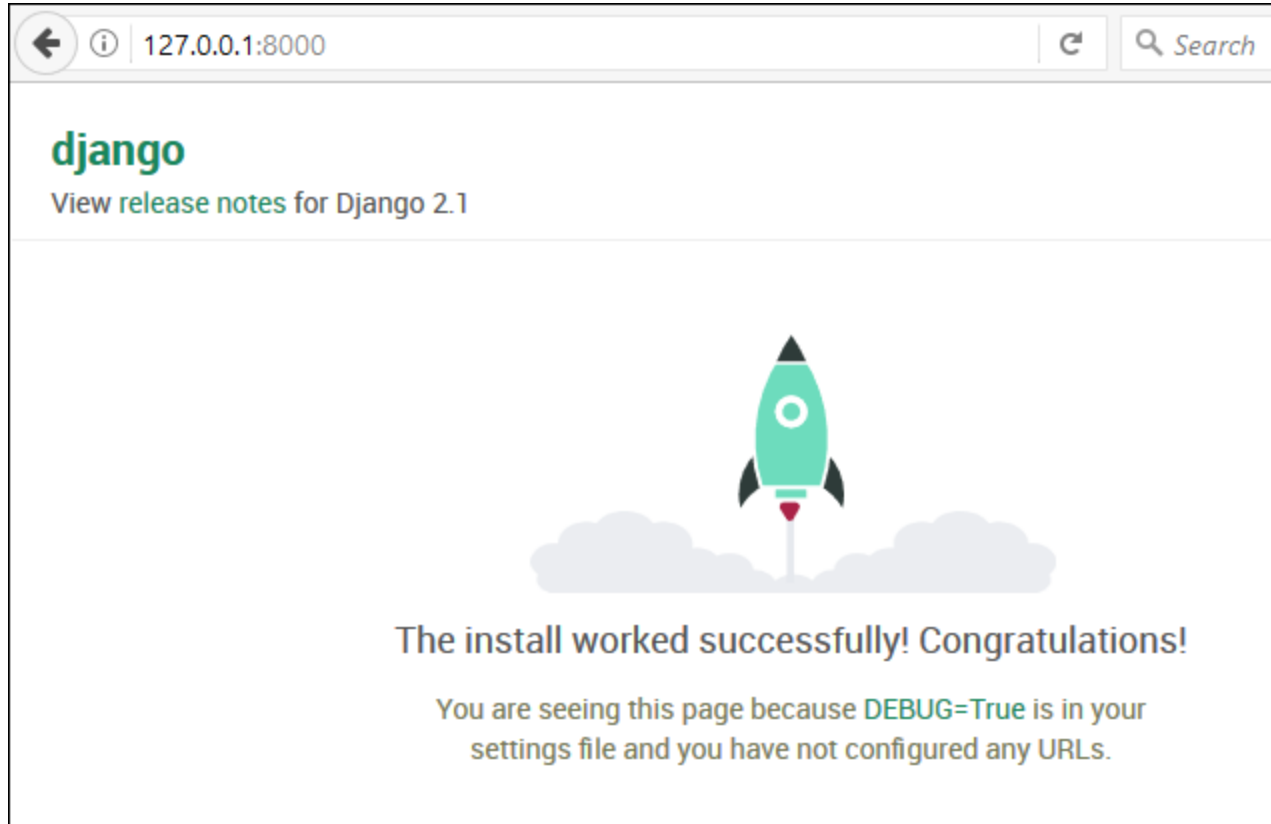
```
| |-- urls.py  
| |-- wsgi.py  
|-- manage.py
```

3. Run your Django site locally with `manage.py runserver`.

```
(eb-virt) ~$ cd ebdjango
```

```
(eb-virt) ~/ebdjango$ python manage.py runserver
```

4. In a web browser, open `http://127.0.0.1:8000/` to view the site.



5. Check the server log to see the output from your request. To stop the web server and return to your virtual environment, press **Ctrl+C**.

```
Django version 2.1.1, using settings 'ebdjango.settings'  
Starting development server at http://127.0.0.1:8000/  
Quit the server with CONTROL-C.  
[07/Sep/2018 20:14:09] "GET / HTTP/1.1" 200 16348  
ctrl+c
```

Configure your Django application for Elastic Beanstalk

Now that you have a Django-powered site on your local machine, you can configure it for deployment with Elastic Beanstalk.

By default, Elastic Beanstalk looks for a file named `application.py` to start your application. Because this doesn't exist in the Django project that you've created, you need to make some adjustments to your

application's environment. You also must set environment variables so that your application's modules can be loaded.

To configure your site for Elastic Beanstalk

1. Activate your virtual environment.

Unix-based systems

```
~/ebdjango$ source ~/eb-virt/bin/activate
```

Windows

```
C:\Users\USERNAME\ebdjango>%HOMEPATH%\eb-virt\Scripts\activate
```

2. Run `pip freeze`, and then save the output to a file named `requirements.txt`.

```
(eb-virt) ~/ebdjango$ pip freeze > requirements.txt
```

Elastic Beanstalk uses `requirements.txt` to determine which package to install on the EC2 instances that run your application.

3. Create a directory named `.ebextensions`.

```
(eb-virt) ~/ebdjango$ mkdir .ebextensions
```

4. In the `.ebextensions` directory, add a [configuration file \(p. 600\)](#) named `django.config` with the following text.

Example `~/ebdjango/.ebextensions/django.config`

```
option_settings:  
  aws:elasticbeanstalk:container:python:  
    WSGIPath: ebdjango.wsgi:application
```

This setting, `WSGIPath`, specifies the location of the WSGI script that Elastic Beanstalk uses to start your application.

5. Deactivate your virtual environment with the `deactivate` command.

```
(eb-virt) ~/ebdjango$ deactivate
```

Reactivate your virtual environment whenever you need to add packages to your application or run your application locally.

Deploy your site with the EB CLI

You've added everything you need to deploy your application on Elastic Beanstalk. Your project directory should now look like this.

```
~/ebdjango/  
|-- .ebextensions  
|   |-- django.config  
|-- ebdjango  
|   |-- __init__.py  
|   |-- settings.py
```



```
| |-- urls.py  
| `-- wsgi.py  
|-- db.sqlite3  
|-- manage.py  
`-- requirements.txt
```

Next, you'll create your application environment and deploy your configured application with Elastic Beanstalk.

Immediately after deployment, you'll edit Django's configuration to add the domain name that Elastic Beanstalk assigned to your application to Django's `ALLOWED_HOSTS`. Then you'll redeploy your application. This is a Django security requirement, designed to prevent HTTP Host header attacks. For more information, see [Host header validation](#).

To create an environment and deploy your Django application

Note

This tutorial uses the EB CLI as a deployment mechanism, but you can also use the Elastic Beanstalk console to deploy a .zip file containing your project's contents.

1. Initialize your EB CLI repository with the **eb init** command.

```
~/ebdjango$ eb init -p python-3.6 django-tutorial  
Application django-tutorial has been created.
```

This command creates an application named `django-tutorial`. It also configures your local repository to create environments with the latest Python 3.6 platform version.

2. (Optional) Run **eb init** again to configure a default key pair so that you can use SSH to connect to the EC2 instance running your application.

```
~/ebdjango$ eb init  
Do you want to set up SSH for your instances?  
(y/n): y  
Select a keypair.  
1) my-keypair  
2) [ Create new KeyPair ]
```

Select a key pair if you have one already, or follow the prompts to create one. If you don't see the prompt or need to change your settings later, run **eb init -i**.

3. Create an environment and deploy your application to it with **eb create**.

```
~/ebdjango$ eb create django-env
```

Note

If you see a "service role required" error message, run **eb create** interactively (without specifying an environment name) and the EB CLI creates the role for you.

This command creates a load balanced Elastic Beanstalk environment named `django-env`. Creating an environment takes about 5 minutes. As Elastic Beanstalk creates the resources needed to run your application, it outputs informational messages that the EB CLI relays to your terminal.

4. When the environment creation process completes, find the domain name of your new environment by running **eb status**.

```
~/ebdjango$ eb status  
Environment details for: django-env  
Application name: django-tutorial
```

```
...  
CNAME: eb-django-app-dev.elasticbeanstalk.com  
...
```

Your environment's domain name is the value of the `CNAME` property.

5. Open the `settings.py` file in the `ebdjango` directory. Locate the `ALLOWED_HOSTS` setting, and then add your application's domain name that you found in the previous step to the setting's value. If you can't find this setting in the file, add it to a new line.

```
...  
ALLOWED_HOSTS = [ 'eb-django-app-dev.elasticbeanstalk.com' ]
```

6. Save the file, and then deploy your application by running **eb deploy**. When you run **eb deploy**, the EB CLI bundles up the contents of your project directory and deploys it to your environment.

```
~/ebdjango$ eb deploy
```

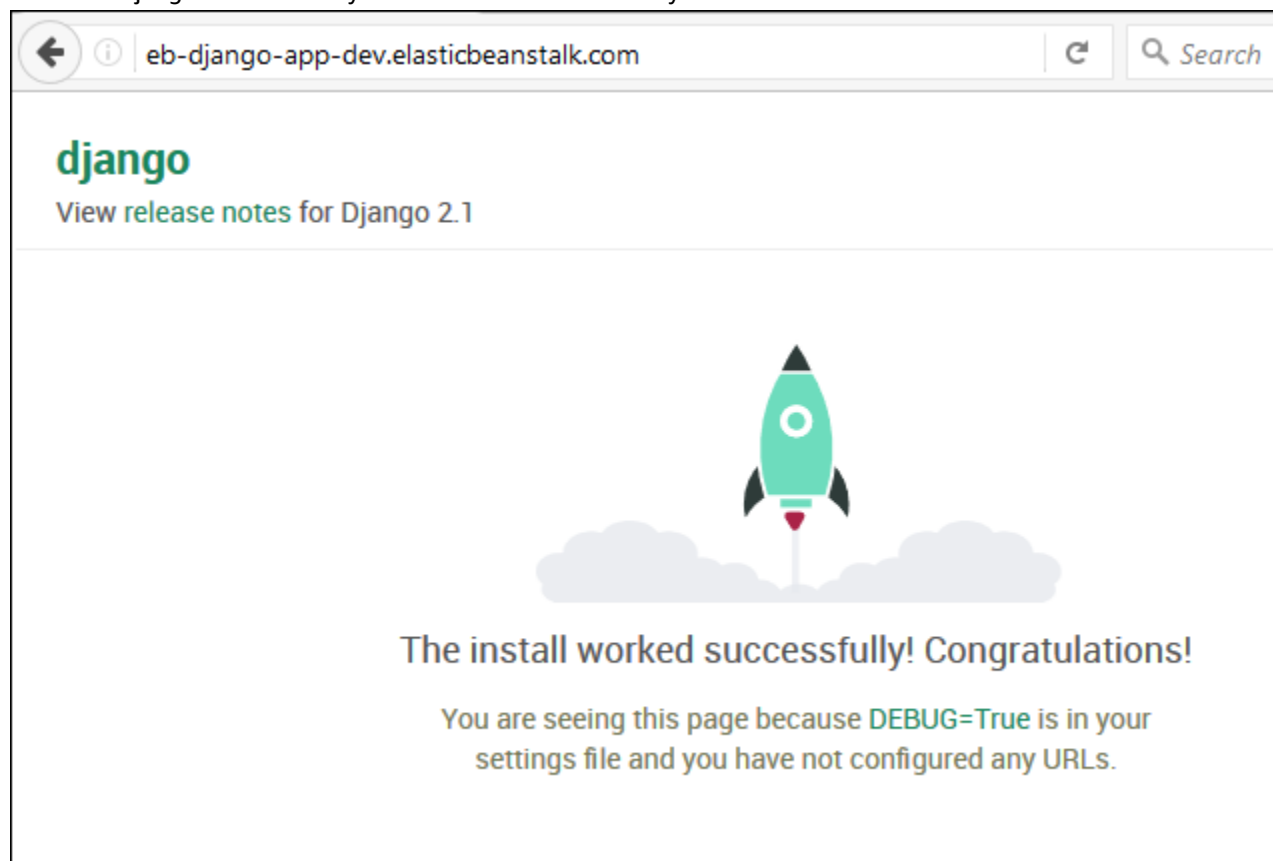
Note

If you are using Git with your project, see [Using the EB CLI with Git \(p. 870\)](#).

7. When the environment update process completes, open your website with **eb open**.

```
~/ebdjango$ eb open
```

This opens a browser window using the domain name created for your application. You should see the same Django website that you created and tested locally.



If you don't see your application running, or get an error message, see [Troubleshooting deployments \(p. 948\)](#) for help with how to determine the cause of the error.

If you *do* see your application running, then congratulations, you've deployed your first Django application with Elastic Beanstalk!

Update your application

Now that you have a running application on Elastic Beanstalk, you can update and redeploy your application or its configuration, and Elastic Beanstalk does the work of updating your instances and starting your new application version.

For this example, we'll enable Django's admin console and configure a few other settings.

Modify your site settings

By default, your Django website uses the UTC time zone to display time. You can change this by specifying a time zone in `settings.py`.

To change your site's time zone

1. Modify the `TIME_ZONE` setting in `settings.py`.

Example `~/ebdjango/ebdjango/settings.py`

```
...
# Internationalization
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'US/Pacific'
USE_I18N = True
USE_L10N = True
USE_TZ = True
```

For a list of time zones, visit [this page](#).

2. Deploy the application to your Elastic Beanstalk environment.

```
~/ebdjango/$ eb deploy
```

Create a site administrator

You can create a site administrator for your Django application to access the admin console directly from the website. Administrator login details are stored securely in the local database image included in the default project that Django generates.

To create a site administrator

1. Initialize your Django application's local database.

```
(eb-virt) ~/ebdjango$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
```

```
Applying admin.0003_logentry_add_action_flag_choices... OK
Applying contenttypes.0002_remove_content_type_name... OK
Applying auth.0002_alter_permission_name_max_length... OK
Applying auth.0003_alter_user_email_max_length... OK
Applying auth.0004_alter_user_username_opts... OK
Applying auth.0005_alter_user_last_login_null... OK
Applying auth.0006_require_contenttypes_0002... OK
Applying auth.0007_alter_validators_add_error_messages... OK
Applying auth.0008_alter_user_username_max_length... OK
Applying auth.0009_alter_user_last_name_max_length... OK
Applying sessions.0001_initial... OK
```

2. Run `manage.py createsuperuser` to create an administrator.

```
(eb-virt) ~/ebdjango$ python manage.py createsuperuser
Username: admin
Email address: me@mydomain.com
Password: *****
Password (again): *****
Superuser created successfully.
```

3. To tell Django where to store static files, define `STATIC_ROOT` in `settings.py`.

Example `~/ebdjango/ebdjango/settings.py`

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.1/howto/static-files/
STATIC_URL = '/static/'
STATIC_ROOT = 'static'
```

4. Run `manage.py collectstatic` to populate the static directory with static assets (JavaScript, CSS, and images) for the admin site.

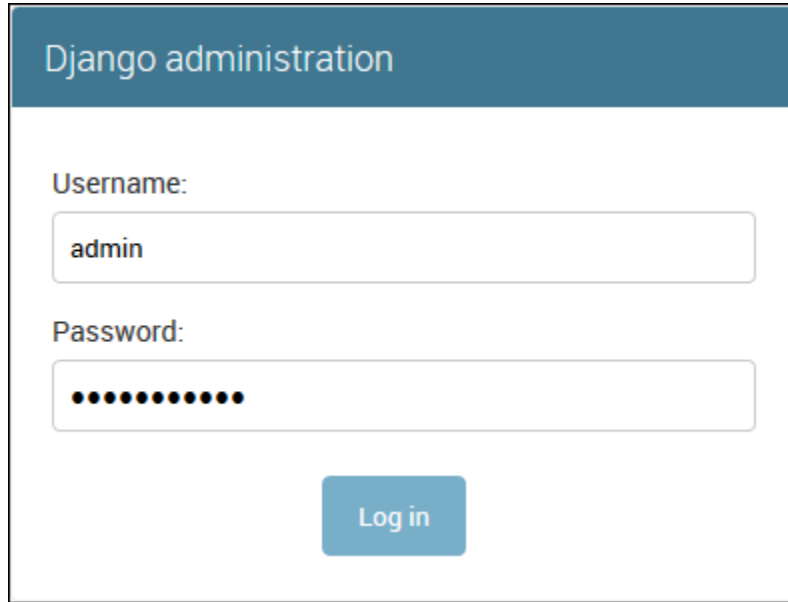
```
(eb-virt) ~/ebdjango$ python manage.py collectstatic
119 static files copied to ~/ebdjango/static
```

5. Deploy your application.

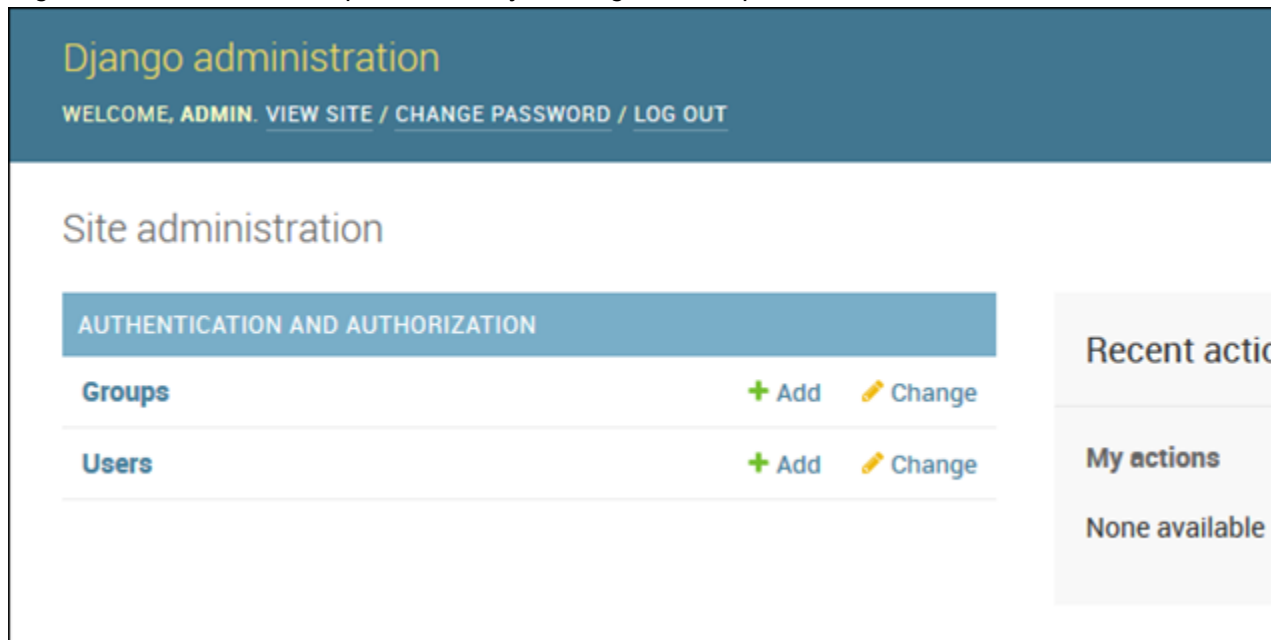
```
~/ebdjango$ eb deploy
```

6. View the admin console by opening the site in your browser, appending `/admin/` to the site URL, such as the following.

```
http://django-env.p33kq46sfh.us-west-2.elasticbeanstalk.com/admin/
```



7. Log in with the username and password that you configured in step 2.



You can use a similar procedure of local updating/testing followed by **eb deploy**. Elastic Beanstalk does the work of updating your live servers, so you can focus on application development instead of server administration!

Add a database migration configuration file

You can add commands to your `.ebextensions` script that are run when your site is updated. This enables you to automatically generate database migrations.

To add a migrate step when your application is deployed

1. Create a [configuration file \(p. 600\)](#) named `db-migrate.config` with the following content.

Example `~/ebdjango/.ebextensions/db-migrate.config`

```
container_commands:
  01_migrate:
    command: "django-admin.py migrate"
    leader_only: true
option_settings:
  aws:elasticbeanstalk:application:environment:
    DJANGO_SETTINGS_MODULE: ebdjango.settings
```

This configuration file runs the `django-admin.py migrate` command during the deployment process, before starting your application. Because it runs before the application starts, you must also configure the `DJANGO_SETTINGS_MODULE` environment variable explicitly (usually `wsgi.py` takes care of this for you during startup). Specifying `leader_only: true` in the command ensures that it is run only once when you're deploying to multiple instances.

2. Deploy your application.

```
~/ebdjango$ eb deploy
```

Clean up

To save instance hours and other AWS resources between development sessions, terminate your Elastic Beanstalk environment with **eb terminate**.

```
~/ebdjango$ eb terminate django-env
```

This command terminates the environment and all of the AWS resources that run within it. It doesn't delete the application, however, so you can always create more environments with the same configuration by running **eb create** again. For more information on EB CLI commands, see [Managing Elastic Beanstalk environments with the EB CLI \(p. 864\)](#).

If you're done with the sample application, you can also remove the project folder and virtual environment.

```
~$ rm -rf ~/eb-virt
~$ rm -rf ~/ebdjango
```

Next steps

For more information about Django, including an in-depth tutorial, see [the official documentation](#).

If you want to try out another Python web framework, check out [Deploying a flask application to Elastic Beanstalk \(p. 295\)](#).

Adding an Amazon RDS DB instance to your Python application environment

You can use an Amazon Relational Database Service (Amazon RDS) DB instance to store data gathered and modified by your application. The database can be attached to your environment and managed by Elastic Beanstalk, or created and managed externally.

If you are using Amazon RDS for the first time, [add a DB instance \(p. 312\)](#) to a test environment with the Elastic Beanstalk Management Console and verify that your application can connect to it.

To connect to a database, [add the driver \(p. 313\)](#) to your application, load the driver in your code, and [create a connection object \(p. 313\)](#) with the environment properties provided by Elastic Beanstalk. The configuration and connection code vary depending on the database engine and framework that you use.

Note

For learning purposes or test environments, you can use Elastic Beanstalk to add a DB instance. For production environments, you can create a DB instance outside of your Elastic Beanstalk environment to decouple your environment resources from your database resources. This way, when you terminate your environment, the DB instance isn't deleted. An external DB instance also lets you connect to the same database from multiple environments and perform [blue-green deployments](#). For instructions, see [Using Elastic Beanstalk with Amazon RDS \(p. 820\)](#).

Sections

- [Adding a DB instance to your environment \(p. 312\)](#)
- [Downloading a driver \(p. 313\)](#)
- [Connecting to a database \(p. 313\)](#)

Adding a DB instance to your environment

To add a DB instance to your environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Choose a DB engine, and enter a user name and password.
6. Choose **Apply**.

Adding a DB instance takes about 10 minutes. When the environment update is complete, the DB instance's hostname and other connection information are available to your application through the following environment properties:

Property name	Description	Property value
RDS_HOSTNAME	The hostname of the DB instance.	On the Connectivity & security tab on the Amazon RDS console: Endpoint .
RDS_PORT	The port on which the DB instance accepts connections. The default value varies among DB engines.	On the Connectivity & security tab on the Amazon RDS console: Port .
RDS_DB_NAME	The database name, ebdb .	On the Configuration tab on the Amazon RDS console: DB Name .

Property name	Description	Property value
RDS_USERNAME	The username that you configured for your database.	On the Configuration tab on the Amazon RDS console: Master username .
RDS_PASSWORD	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

For more information about configuring an internal DB instance, see [Adding a database to your Elastic Beanstalk environment](#) (p. 506).

Downloading a driver

Add the database driver to your project's [requirements file](#) (p. 294).

Example requirements.txt – Django with MySQL

```
Django==2.0.3
mysqlclient==1.3.12
```

Common driver packages for Python

- **MySQL** – MySQL-python (Python 2), mysqlclient (Python 3)
- **PostgreSQL** – psycopg2
- **Oracle** – cx_Oracle
- **SQL Server** – adodbapi

Connecting to a database

Elastic Beanstalk provides connection information for attached DB instances in environment properties. Use `os.environ['VARIABLE']` to read the properties and configure a database connection.

Example Django settings file – DATABASES dictionary

```
import os

if 'RDS_HOSTNAME' in os.environ:
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.mysql',
            'NAME': os.environ[ 'RDS_DB_NAME' ],
            'USER': os.environ[ 'RDS_USERNAME' ],
            'PASSWORD': os.environ[ 'RDS_PASSWORD' ],
            'HOST': os.environ[ 'RDS_HOSTNAME' ],
            'PORT': os.environ[ 'RDS_PORT' ],
        }
    }
```

Python tools and resources

There are several places you can go to get additional help when developing your Python applications:

Resource	Description
Boto (the AWS SDK for Python)	Install Boto using GitHub.
Python Development Forum	Post your questions and get feedback.
Python Developer Center	One-stop shop for sample code, documentation, tools, and additional resources.

Creating and deploying Ruby applications on Elastic Beanstalk

Topics

- [Setting up your Ruby development environment \(p. 314\)](#)
- [Using the Elastic Beanstalk Ruby platform \(p. 316\)](#)
- [Deploying a rails application to Elastic Beanstalk \(p. 320\)](#)
- [Deploying a sinatra application to Elastic Beanstalk \(p. 326\)](#)
- [Adding an Amazon RDS DB instance to your Ruby application environment \(p. 329\)](#)

AWS Elastic Beanstalk for Ruby makes it easy to deploy, manage, and scale your Ruby web applications using Amazon Web Services. Elastic Beanstalk is available to anyone developing or hosting a web application using Ruby. This section provides step-by-step instructions for deploying a sample application to Elastic Beanstalk using the Elastic Beanstalk command line interface (EB CLI), and then updating the application to use the [Rails](#) and [Sinatra](#) web application frameworks.

The topics in this chapter assume some knowledge of Elastic Beanstalk environments. If you haven't used Elastic Beanstalk before, try the [getting started tutorial \(p. 3\)](#) to learn the basics.

Setting up your Ruby development environment

Set up a Ruby development environment to test your application locally prior to deploying it to AWS Elastic Beanstalk. This topic outlines development environment setup steps and links to installation pages for useful tools.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command  
this is output
```

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

For common setup steps and tools that apply to all languages, see [Configuring your development machine for use with Elastic Beanstalk \(p. 849\)](#)

Sections

- [Installing Ruby \(p. 315\)](#)
- [Installing the AWS SDK for Ruby \(p. 316\)](#)

- [Installing an IDE or text editor \(p. 316\)](#)

Installing Ruby

Install GCC if you don't have a C compiler. On Ubuntu, use `apt`.

```
~$ sudo apt install gcc
```

On Amazon Linux, use `yum`.

```
~$ sudo yum install gcc
```

Install RVM to manage Ruby language installations on your machine. Use the commands at rvm.io to get the project keys and run the installation script.

```
~$ gpg --keyserver hkp://keys.gnupg.net --recv-keys key1 key2  
~$ curl -sSL https://get.rvm.io | bash -s stable
```

This script installs RVM in a folder named `.rvm` in your user directory, and modifies your shell profile to load a setup script whenever you open a new terminal. Load the script manually to get started.

```
~$ source ~/.rvm/scripts/rvm
```

Use `rvm get head` to get the latest version.

```
~$ rvm get head
```

View the available versions of Ruby.

```
~$ rvm list known  
# MRI Rubies  
[ruby-]2.1[.10]  
[ruby-]2.2[.10]  
[ruby-]2.3[.7]  
[ruby-]2.4[.4]  
[ruby-]2.5[.1]  
...
```

Check [Ruby](#) in the *AWS Elastic Beanstalk Platforms* document to find the latest version of Ruby available on an Elastic Beanstalk platform. Install that version.

```
~$ rvm install 2.5.1  
Searching for binary rubies, this might take some time.  
Found remote file https://rvm_io.global.ssl.fastly.net/binaries/ubuntu/16.04/x86_64/  
ruby-2.5.1.tar.bz2  
Checking requirements for ubuntu.  
Requirements installation successful.  
ruby-2.5.1 - #configure  
ruby-2.5.1 - #download  
...
```

Test your Ruby installation.

```
~$ ruby --version
```

```
ruby 2.5.1p57 (2018-03-29 revision 63029) [x86_64-linux]
```

Installing the AWS SDK for Ruby

If you need to manage AWS resources from within your application, install the AWS SDK for Ruby. For example, with the SDK for Ruby, you can use Amazon DynamoDB (DynamoDB) to store user and session information without creating a relational database.

Install the SDK for Ruby and its dependencies with the `gem` command.

```
$ gem install aws-sdk
```

Visit the [AWS SDK for Ruby homepage](#) for more information and installation instructions.

Installing an IDE or text editor

Integrated development environments (IDEs) provide a wide range of features that facilitate application development. If you haven't used an IDE for Ruby development, try Aptana and RubyMine and see which works best for you.

- [Install Aptana](#)
- [RubyMine](#)

Note

An IDE might add files to your project folder that you might not want to commit to source control. To prevent committing these files to source control, use `.gitignore` or your source control tool's equivalent.

If you just want to begin coding and don't need all of the features of an IDE, consider [installing Sublime Text](#).

Using the Elastic Beanstalk Ruby platform

Important

Amazon Linux 2 platform versions are fundamentally different than Amazon Linux AMI platform versions (preceding Amazon Linux 2). These different platform generations are incompatible in several ways. If you are migrating to an Amazon Linux 2 platform version, be sure to read the information in [the section called "Upgrade to Amazon Linux 2" \(p. 426\)](#).

The AWS Elastic Beanstalk Ruby platform is a set of [environment configurations](#) for Ruby web applications that can run behind an nginx proxy server under a Puma application server. Each platform branch corresponds to a version of Ruby.

If you use RubyGems, you can [include a Gemfile file \(p. 319\)](#) in your source bundle to install packages during deployment.

Your application might run under a different application server, for example Passenger. You can use a `Procfile` to start a different application server, and a `Gemfile` to install it. For details, see [the section called "Procfile" \(p. 319\)](#).

Note

If you are using an Amazon Linux AMI Ruby platform branch (preceding Amazon Linux 2), be aware that Elastic Beanstalk provides two flavors of these branches - with Puma and with Passenger. If your application needs the Passenger application server, you can use the appropriate Passenger platform branch, and you don't need to do any additional configuration.

Elastic Beanstalk provides [configuration options \(p. 536\)](#) that you can use to customize the software that runs on the Amazon Elastic Compute Cloud (Amazon EC2) instances in your Elastic Beanstalk environment. You can configure environment variables needed by your application, enable log rotation to Amazon S3, and map folders in your application source that contain static files to paths served by the proxy server. The platform also predefines some common environment variables related to Rails and Rack for ease of discovery and use.

Configuration options are available in the Elastic Beanstalk console for [modifying the configuration of a running environment \(p. 547\)](#). To avoid losing your environment's configuration when you terminate it, you can use [saved configurations \(p. 640\)](#) to save your settings and later apply them to another environment.

To save settings in your source code, you can include [configuration files \(p. 600\)](#). Settings in configuration files are applied every time you create an environment or deploy your application. You can also use configuration files to install packages, run scripts, and perform other instance customization operations during deployments.

Settings applied in the Elastic Beanstalk console override the same settings in configuration files, if they exist. This lets you have default settings in configuration files, and override them with environment-specific settings in the console. For more information about precedence, and other methods of changing settings, see [Configuration options \(p. 536\)](#).

For details about the various ways you can extend an Elastic Beanstalk Linux-based platform, see [the section called "Extending Linux platforms" \(p. 31\)](#).

Configuring your Ruby environment

You can use the Elastic Beanstalk console to enable log rotation to Amazon S3 and configure variables that your application can read from the environment.

To access the software configuration settings for your environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.

Log options

The **Log options** section has two settings:

- **Instance profile**– Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3** – Specifies whether log files for your application's Amazon EC2 instances should be copied to the Amazon S3 bucket associated with your application.

Static files

To improve performance, the **Static files** section lets you configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. For each directory, you set the virtual path to directory mapping. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

For details about configuring static files using the Elastic Beanstalk console, see [the section called “Static files” \(p. 650\)](#).

By default, the proxy server in a Ruby environment serves any files in the `public` folder at the `/public` path, and any files in the `public/assets` subfolder at the `/assets` path. For example, if your application source contains a file named `logo.png` in a folder named `public`, the proxy server serves it to users at `subdomain.elasticbeanstalk.com/public/logo.png`. If `logo.png` is in a folder named `assets` inside the `public` folder, the proxy server serves it at `subdomain.elasticbeanstalk.com/assets/logo.png`. You can configure additional mappings as explained in this section.

Environment properties

The **Environment Properties** section lets you specify environment configuration settings on the Amazon EC2 instances that are running your application. Environment properties are passed in as key-value pairs to the application.

The Ruby platform defines the following properties for environment configuration:

- **BUNDLE_WITHOUT** – A colon-separated list of groups to ignore when [installing dependencies](#) from a [Gemfile](#).
- **BUNDLER_DEPLOYMENT_MODE** – Set to `true` (the default) to install dependencies in [deployment mode](#) using Bundler. Set to `false` to run `bundle install` in development mode.

Note

This environment property isn't defined on Amazon Linux AMI Ruby platform branches (preceding Amazon Linux 2).

- **RAILS_SKIP_ASSET_COMPILATION** – Set to `true` to skip running `rake assets:precompile` during deployment.
- **RAILS_SKIP_MIGRATIONS** – Set to `true` to skip running `rake db:migrate` during deployment.
- **RACK_ENV** – Specify the environment stage for Rack. For example, `development`, `production`, or `test`.

Inside the Ruby environment running in Elastic Beanstalk, environment variables are accessible using the `ENV` object. For example, you could read a property named `API_ENDPOINT` to a variable with the following code:

```
endpoint = ENV['API_ENDPOINT']
```

See [Environment properties and other software settings \(p. 516\)](#) for more information.

Ruby configuration namespaces

You can use a [configuration file \(p. 600\)](#) to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be defined by the Elastic Beanstalk service or the platform that you use and are organized into *namespaces*.

You can use the `aws:elasticbeanstalk:environment:proxy:staticfiles` namespace to configure the environment proxy to serve static files. You define mappings of virtual paths to application directories.

The Ruby platform doesn't define any platform-specific namespaces. Instead, it defines environment properties for common Rails and Rack options.

The following configuration file specifies a static files option that maps a directory named `staticimages` to the path `/images`, sets each of the platform defined environment properties, and sets an additional environment property named `LOGGING`.

Example `.ebextensions/ruby-settings.config`

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:staticfiles:
    /images: staticimages
  aws:elasticbeanstalk:application:environment:
    BUNDLE_WITHOUT: test
    BUNDLER_DEPLOYMENT_MODE: true
    RACK_ENV: development
    RAILS_SKIP_ASSET_COMPILATION: true
    RAILS_SKIP_MIGRATIONS: true
    LOGGING: debug
```

Note

The `BUNDLER_DEPLOYMENT_MODE` environment property and the `aws:elasticbeanstalk:environment:proxy:staticfiles` namespace aren't defined on Amazon Linux AMI Ruby platform branches (preceding Amazon Linux 2).

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See [Configuration options \(p. 536\)](#) for more information.

Installing packages with a Gemfile

Use a `Gemfile` file in the root of your project source to use RubyGems to install packages that your application requires.

Example Gemfile

```
source "https://rubygems.org"
gem 'sinatra'
gem 'json'
gem 'rack-parser'
```

When a `Gemfile` file is present, Elastic Beanstalk runs `bundle install` to install dependencies.

Configuring the application process with a Procfile

To specify the command that starts your Ruby application, include a file called `Procfile` at the root of your source bundle.

Note

Elastic Beanstalk doesn't support this feature on Amazon Linux AMI Ruby platform branches (preceding Amazon Linux 2).

For details about writing and using a `Procfile`, expand the *Buildfile and Procfile* section in [the section called "Extending Linux platforms" \(p. 31\)](#).

When you don't provide a `Procfile`, Elastic Beanstalk generates the following default file, which assumes you're using the pre-installed Puma application server.

```
web: puma -C /opt/elasticbeanstalk/config/private/pumaconf.rb
```

If you want to use your own provided Puma server, you can install it using a [Gemfile \(p. 319\)](#). The following example `Procfile` shows how to start it.

Example Procfile

```
web: bundle exec puma -C /opt/elasticbeanstalk/config/private/pumaconf.rb
```

If you want to use the Passenger application server, use the following example files to configure your Ruby environment to install and use Passenger.

1. Use this example file to install Passenger.

Example Gemfile

```
source 'https://rubygems.org'  
gem 'passenger'
```

2. Use this example file to instruct Elastic Beanstalk to start Passenger.

Example Procfile

```
web: bundle exec passenger start /var/app/current --socket /var/run/puma/my_app.sock
```

Note

You don't have to change anything in the configuration of the nginx proxy server to use Passenger. To use other application servers, you might need to customize the nginx configuration to properly forward requests to your application.

Deploying a rails application to Elastic Beanstalk

Rails is an open source, model-view-controller (MVC) framework for Ruby. This tutorial walks you through the process of generating a Rails application and deploying it to an AWS Elastic Beanstalk environment.

Sections

- [Prerequisites \(p. 320\)](#)
- [Launch an Elastic Beanstalk environment \(p. 321\)](#)
- [Install rails and generate a website \(p. 322\)](#)
- [Configure rails settings \(p. 324\)](#)
- [Deploy your application \(p. 325\)](#)
- [Cleanup \(p. 325\)](#)
- [Next steps \(p. 326\)](#)

Prerequisites

Basic Elastic Beanstalk knowledge

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in [Getting started using Elastic Beanstalk \(p. 3\)](#) to launch your first Elastic Beanstalk environment.

Command line

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command  
this is output
```

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Rails dependencies

The Rails framework has the following dependencies. Be sure you have all of them installed.

- **Ruby 2.2.2 or newer** – For installation instructions, see [Setting up your Ruby development environment \(p. 314\)](#).

In this tutorial we use Ruby 2.5.1 and the corresponding Elastic Beanstalk platform version.

- **Node.js** – For installation instructions, see [Installing Node.js via package manager](#).
- **Yarn** – For installation instructions, see [Installation](#) on the *Yarn* website.

Launch an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. Choose the **Ruby** platform and accept the default settings and sample code.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link:
console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or nginx as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.

- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
- **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see [Using Elastic Beanstalk with Amazon S3](#) (p. 831).

Install rails and generate a website

Install Rails and its dependencies with the `gem` command.

```
~$ gem install rails
Fetching: concurrent-ruby-1.0.5.gem (100%)
Successfully installed concurrent-ruby-1.0.5
Fetching: i18n-1.0.1.gem (100%)
Successfully installed i18n-1.0.1
...
```

Test your Rails installation.

```
~$ rails --version
Rails 5.2.0
```

Use `rails new` with the name of the application to create a new Rails project.

```
~$ rails new ~/eb-rails
```

Rails creates a directory with the name specified, generates all of the files needed to run a sample project locally, and then runs `bundler` to install all of the dependencies (Gems) defined in the project's `Gemfile`.

Test your Rails installation by running the default project locally.

```
~$ cd eb-rails
eb-rails $ rails server
=> Booting Puma
=> Rails 5.2.0 application starting in development
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.11.4 (ruby 2.5.1-p57), codename: Love Song
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://0.0.0.0:3000
Use Ctrl+C to stop
...
```

Open `http://localhost:3000` in a web browser to see the default project in action.



Yay! You're on Rails!



This page is only visible in development mode. Add some content to the front page of the application to support production deployment to Elastic Beanstalk. Use `rails generate` to create a controller, route, and view for your welcome page.

```
~/eb-rails$ rails generate controller WelcomePage welcome
create  app/controllers/welcome_page_controller.rb
route  get 'welcome_page/welcome'
invoke erb
create  app/views/welcome_page
create  app/views/welcome_page/welcome.html.erb
invoke test_unit
create  test/controllers/welcome_page_controller_test.rb
invoke helper
create  app/helpers/welcome_page_helper.rb
invoke test_unit
invoke assets
invoke coffee
create  app/assets/javascripts/welcome_page.coffee
invoke scss
create  app/assets/stylesheets/welcome_page.scss.
```

This gives you all you need to access the page at `/welcome_page/welcome`. Before you publish the changes, however, change the content in the view and add a route to make this page appear at the top level of the site.

Use a text editor to edit the content in `app/views/welcome_page/welcome.html.erb`. For this example, you'll use `cat` to simply overwrite the content of the existing file.

Example `app/views/welcome_page/welcome.html.erb`

```
<h1>Welcome!</h1>
<p>This is the front page of my first Rails application on Elastic Beanstalk.</p>
```

Finally, add the following route to `config/routes.rb`:

Example `config/routes.rb`

```
Rails.application.routes.draw do
  get 'welcome_page/welcome'
  root 'welcome_page#welcome'
```

This tells Rails to route requests to the root of the website to the welcome page controller's `welcome` method, which renders the content in the welcome view (`welcome.html.erb`).

Configure rails settings

Use the Elastic Beanstalk console to configure Rails with environment properties. Set the `SECRET_KEY_BASE` environment property to a string of up to 256 alphanumeric characters.

Rails uses this property to create keys. Therefore you should keep it a secret and not store it in source control in plain text. Instead, you provide it to Rails code on your environment through an environment property.

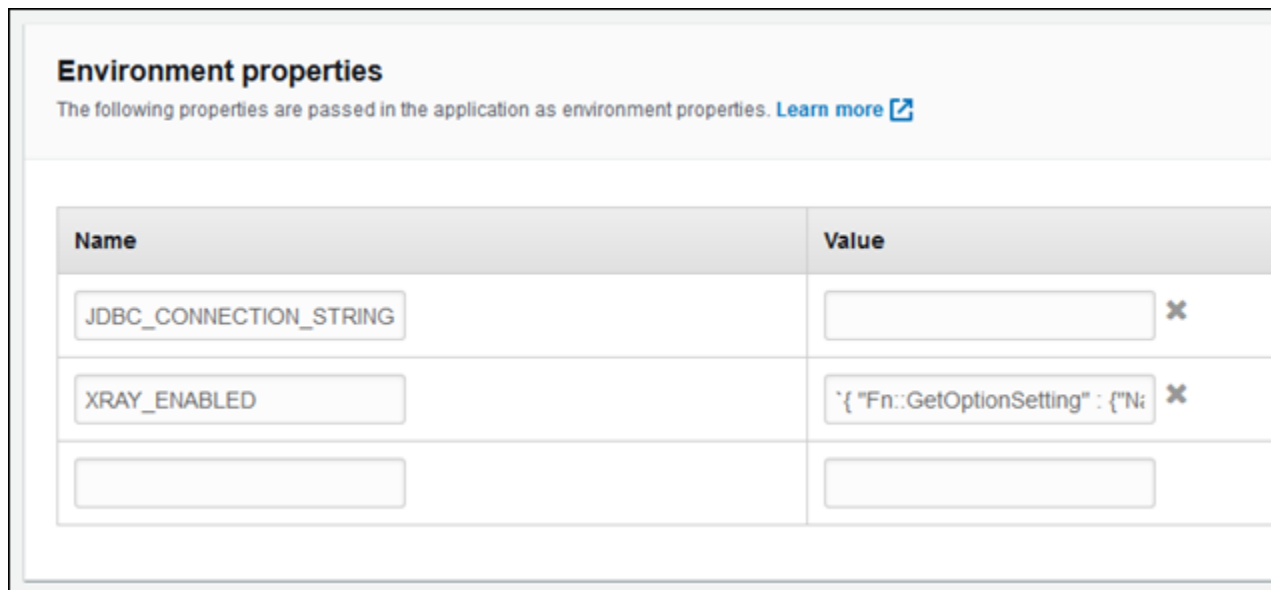
To configure environment properties in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. Under **Environment properties**, enter key-value pairs.



6. Choose **Apply**.

Now you're ready to deploy the site to your environment.

Deploy your application

Create a [source bundle](#) (p. 342) containing the files created by Rails. The following command creates a source bundle named `rails-default.zip`.

```
~/eb-rails$ zip ../rails-default.zip -r * .[^.]*
```

Upload the source bundle to Elastic Beanstalk to deploy Rails to your environment.

To deploy a source bundle

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Cleanup

When you finish working with Elastic Beanstalk, you can terminate your environment. Elastic Beanstalk terminates all AWS resources associated with your environment, such as [Amazon EC2 instances](#) (p. 451), [database instances](#) (p. 506), [load balancers](#) (p. 470), security groups, and [alarms](#) (p. 470).

To terminate your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

With Elastic Beanstalk, you can easily create a new environment for your application at any time.

Next steps

For more information about Rails, visit rubyonrails.org.

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The [Elastic Beanstalk Command Line Interface \(p. 852\)](#) (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

Finally, if you plan on using your application in a production environment, you will want to [configure a custom domain name \(p. 534\)](#) for your environment and [enable HTTPS \(p. 652\)](#) for secure connections.

Deploying a sinatra application to Elastic Beanstalk

This walkthrough shows how to deploy a simple [Sinatra](#) web application to AWS Elastic Beanstalk.

Prerequisites

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in [Getting started using Elastic Beanstalk \(p. 3\)](#) to launch your first Elastic Beanstalk environment.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (\$) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command  
this is output
```

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can [install the Windows Subsystem for Linux](#) to get a Windows-integrated version of Ubuntu and Bash.

Sinatra requires Ruby 1.9 or newer. In this tutorial we use Ruby 2.5.1 and the corresponding Elastic Beanstalk platform version. Install Ruby by following the instructions at [Setting up your Ruby development environment \(p. 314\)](#).

Launch an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. Choose the **Ruby** platform and accept the default settings and sample code.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link:
console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or nginx as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
- **Domain name** – A domain name that routes to your web app in the form `subdomain.region.elasticbeanstalk.com`.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see [Using Elastic Beanstalk with Amazon S3 \(p. 831\)](#).

Write a basic sinatra website

To create and deploy a sinatra application

1. Create a configuration file named **config.ru** with the following contents.

Example config.ru

```
require './helloworld'  
run Sinatra::Application
```

2. Create a Ruby code file named **helloworld.rb** with the following contents.

Example helloworld.rb

```
require 'sinatra'  
get '/' do  
  "Hello World!"  
end
```

3. Create a **Gemfile** with the following contents.

Example Gemfile

```
source 'https://rubygems.org'  
gem 'sinatra'
```

Deploy your application

Create a [source bundle](#) (p. 342) containing the your source files. The following command creates a source bundle named `sinatra-default.zip`.

```
~/eb-sinatra$ zip ../sinatra-default.zip -r * .[^.]*
```

Upload the source bundle to Elastic Beanstalk to deploy Sinatra to your environment.

To deploy a source bundle

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Cleanup

When you finish working with Elastic Beanstalk, you can terminate your environment. Elastic Beanstalk terminates all AWS resources associated with your environment, such as [Amazon EC2](#)

[instances \(p. 451\)](#), [database instances \(p. 506\)](#), [load balancers \(p. 470\)](#), security groups, and [alarms \(p. 470\)](#).

To terminate your Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

With Elastic Beanstalk, you can easily create a new environment for your application at any time.

Next steps

For more information about Sinatra, visit sinatrarb.com.

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The [Elastic Beanstalk Command Line Interface \(p. 852\)](#) (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

Finally, if you plan on using your application in a production environment, you will want to [configure a custom domain name \(p. 534\)](#) for your environment and [enable HTTPS \(p. 652\)](#) for secure connections.

Adding an Amazon RDS DB instance to your Ruby application environment

You can use an Amazon Relational Database Service (Amazon RDS) DB instance to store data gathered and modified by your application. The database can be attached to your environment and managed by Elastic Beanstalk, or created and managed externally.

If you are using Amazon RDS for the first time, [add a DB instance \(p. 330\)](#) to a test environment with the Elastic Beanstalk Management Console and verify that your application can connect to it.

To connect to a database, [add the adapter \(p. 330\)](#) to your application and [configure a connection \(p. 331\)](#) with the environment properties provided by Elastic Beanstalk. The configuration and connection code vary depending on the database engine and framework that you use.

Note

For learning purposes or test environments, you can use Elastic Beanstalk to add a DB instance. For production environments, you can create a DB instance outside of your Elastic Beanstalk environment to decouple your environment resources from your database resources. This way, when you terminate your environment, the DB instance isn't deleted. An external DB instance also lets you connect to the same database from multiple environments and perform [blue-green deployments](#). For instructions, see [Using Elastic Beanstalk with Amazon RDS \(p. 820\)](#).

Sections

- [Adding a DB instance to your environment \(p. 330\)](#)
- [Downloading an adapter \(p. 330\)](#)

- [Connecting to a database \(p. 331\)](#)

Adding a DB instance to your environment

To add a DB instance to your environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Choose a DB engine, and enter a user name and password.
6. Choose **Apply**.

Adding a DB instance takes about 10 minutes. When the environment update is complete, the DB instance's hostname and other connection information are available to your application through the following environment properties:

Property name	Description	Property value
RDS_HOSTNAME	The hostname of the DB instance.	On the Connectivity & security tab on the Amazon RDS console: Endpoint .
RDS_PORT	The port on which the DB instance accepts connections. The default value varies among DB engines.	On the Connectivity & security tab on the Amazon RDS console: Port .
RDS_DB_NAME	The database name, ebdb .	On the Configuration tab on the Amazon RDS console: DB Name .
RDS_USERNAME	The username that you configured for your database.	On the Configuration tab on the Amazon RDS console: Master username .
RDS_PASSWORD	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

For more information about configuring an internal DB instance, see [Adding a database to your Elastic Beanstalk environment \(p. 506\)](#).

Downloading an adapter

Add the database adapter to your project's [gem file \(p. 319\)](#).

Example Gemfile – Rails with MySQL

```
source 'https://rubygems.org'
gem 'puma'
gem 'rails', '4.1.8'
```

```
gem 'mysql2'
```

Common adapter gems for Ruby

- **MySQL** – `mysql2`
- **PostgreSQL** – `pg`
- **Oracle** – `activerecord-oracle_enhanced-adapter`
- **SQL Server** – `sql_server`

Connecting to a database

Elastic Beanstalk provides connection information for attached DB instances in environment properties. Use `ENV['VARIABLE']` to read the properties and configure a database connection.

Example config/database.yml – Ruby on rails database configuration (MySQL)

```
production:
  adapter: mysql2
  encoding: utf8
  database: <%= ENV['RDS_DB_NAME'] %>
  username: <%= ENV['RDS_USERNAME'] %>
  password: <%= ENV['RDS_PASSWORD'] %>
  host: <%= ENV['RDS_HOSTNAME'] %>
  port: <%= ENV['RDS_PORT'] %>
```

Tutorials and samples

Language and framework specific tutorials are spread throughout the AWS Elastic Beanstalk Developer Guide. New and updated tutorials are added to this list as they are published. The most recent updates are shown first.

These tutorials are targeted at intermediate users and may not contain instructions for basic steps such as signing up for AWS. If this is your first time using AWS or Elastic Beanstalk, check out the [Getting Started walkthrough \(p. 3\)](#) to get your first Elastic Beanstalk environment up and running.

- **Ruby on Rails** - [Deploying a rails application to Elastic Beanstalk \(p. 320\)](#)
- **Ruby and Sinatra** - [Deploying a sinatra application to Elastic Beanstalk \(p. 326\)](#)
- **PHP and MySQL HA Configuration** - [Deploying a high-availability PHP application with an external Amazon RDS database to Elastic Beanstalk \(p. 253\)](#)
- **PHP and Laravel** - [Deploying a Laravel application to Elastic Beanstalk \(p. 234\)](#)
- **PHP and CakePHP** - [Deploying a CakePHP application to Elastic Beanstalk \(p. 241\)](#)
- **PHP and Drupal HA Configuration** - [Deploying a high-availability Drupal website with an external Amazon RDS database to Elastic Beanstalk \(p. 273\)](#)
- **PHP and WordPress HA Configuration** - [Deploying a high-availability WordPress website with an external Amazon RDS database to Elastic Beanstalk \(p. 262\)](#)
- **Node.js with DynamoDB HA Configuration** - [Deploying a Node.js application with DynamoDB to Elastic Beanstalk \(p. 216\)](#)
- **ASP.NET Core** - [Deploying an ASP.NET core application with Elastic Beanstalk \(p. 155\)](#)
- **Python and Flask** - [Deploying a flask application to Elastic Beanstalk \(p. 295\)](#)
- **Python and Django** - [Deploying a Django application to Elastic Beanstalk \(p. 301\)](#)
- **Node.js and Express** - [Deploying an Express application to Elastic Beanstalk \(p. 203\)](#)
- **Docker, PHP and nginx** - [Multicontainer Docker environments with the Elastic Beanstalk console \(p. 66\)](#)
- **.NET Framework (IIS and ASP.NET)** - [Tutorial: How to deploy a .NET sample application using Elastic Beanstalk \(p. 149\)](#)

You can download the sample applications used by Elastic Beanstalk when you create an environment without providing a source bundle with the following links:

- **Single Container Docker** – [docker-singlecontainer-v1.zip](#)
- **Multicontainer Docker** – [docker-multicontainer-v2.zip](#)
- **Preconfigured Docker (Glassfish)** – [docker-glassfish-v1.zip](#)
- **Go** – [go-v1.zip](#)
- **Java SE** – [java-se-jetty-gradle-v3.zip](#)
- **Tomcat (default)** – [java-tomcat-v3.zip](#)
- **Tomcat 7** – [java7-tomcat7.zip](#)
- **.NET** – [dotnet-asp-v1.zip](#)
- **Node.js** – [nodejs-v1.zip](#)
- **PHP** – [php-v1.zip](#)
- **Python** – [python-v1.zip](#)
- **Ruby (Passenger Standalone)** – [ruby-passenger-v3.zip](#)
- **Ruby (Puma)** – [ruby-puma-v3.zip](#)

More involved sample applications that show the use of additional web frameworks, libraries and tools are available as open source projects on GitHub:

- **Load Balanced WordPress** ([tutorial \(p. 262\)](#)) – Configuration files for installing WordPress securely and running it in a load balanced AWS Elastic Beanstalk environment.
- **Load Balanced Drupal** ([tutorial \(p. 273\)](#)) – Configuration files and instructions for installing Drupal securely and running it in a load balanced AWS Elastic Beanstalk environment.
- **Scorekeep** - RESTful web API that uses the Spring framework and the AWS SDK for Java to provide an interface for creating and managing users, sessions, and games. The API is bundled with an Angular 1.5 web app that consumes the API over HTTP. Includes branches that show integration with Amazon Cognito, AWS X-Ray, and Amazon Relational Database Service.

The application uses features of the Java SE platform to download dependencies and build on-instance, minimizing the size of the source bundle. The application also includes nginx configuration files that override the default configuration to serve the frontend web app statically on port 80 through the proxy, and route requests to paths under `/api` to the API running on `localhost:5000`.

- **Does it Have Snakes?** - Tomcat application that shows the use of RDS in a Java EE web application in AWS Elastic Beanstalk. The project shows the use of Servlets, JSPs, Simple Tag Support, Tag Files, JDBC, SQL, Log4J, Bootstrap, Jackson, and Elastic Beanstalk configuration files.
- **Locust Load Generator** - This project shows the use of Java SE platform features to install and run **Locust**, a load generating tool written in Python. The project includes configuration files that install and configure Locust, a build script that configures a DynamoDB table, and a Procfile that runs Locust.
- **Share Your Thoughts** ([tutorial \(p. 253\)](#)) - PHP application that shows the use of MySQL on Amazon RDS, Composer, and configuration files.
- **A New Startup** ([tutorial \(p. 216\)](#)) - Node.js sample application that shows the use of DynamoDB, the AWS SDK for JavaScript in Node.js, npm package management, and configuration files.

Managing and configuring Elastic Beanstalk applications

The first step in using AWS Elastic Beanstalk is to create an application, which represents your web application in AWS. In Elastic Beanstalk an application serves as a container for the environments that run your web app and for versions of your web app's source code, saved configurations, logs, and other artifacts that you create while using Elastic Beanstalk.

To create an application

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose **Create a new application**.
3. Use the on-screen form to provide an application name.
4. Optionally, provide a description, and add tag keys and values.
5. Choose **Create**.

Elastic Beanstalk

Create new application

Application information

Application Name

Maximum length of 100 characters, not including forward slash (/).

Description

Tags

Apply up to 50 tags. You can use tags to group and filter your resources. A tag is a key-value pair. The key must be unique for each resource and is case-sensitive. [Learn more](#)

Key	Value	
<input type="text"/>	<input type="text"/>	<input type="button" value="Remove"/>

50 remaining

Ca

After creating the application, the console prompts you to create an environment for it. For detailed information about all of the options available, see [Creating an Elastic Beanstalk environment \(p. 363\)](#).

If you no longer need an application, you can delete it.

Warning

Deleting an application terminates all associated environments and deletes all application versions and saved configurations that belong to the application.

To delete an application

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Applications**, and then select your application on the list.
3. Choose **Actions**, and then choose **Delete application**.

Topics

- [Elastic Beanstalk application management console \(p. 336\)](#)
- [Managing application versions \(p. 337\)](#)
- [Create an application source bundle \(p. 342\)](#)
- [Tagging Elastic Beanstalk application resources \(p. 349\)](#)

Elastic Beanstalk application management console

You can use the AWS Elastic Beanstalk console to manage applications, application versions, and saved configurations.

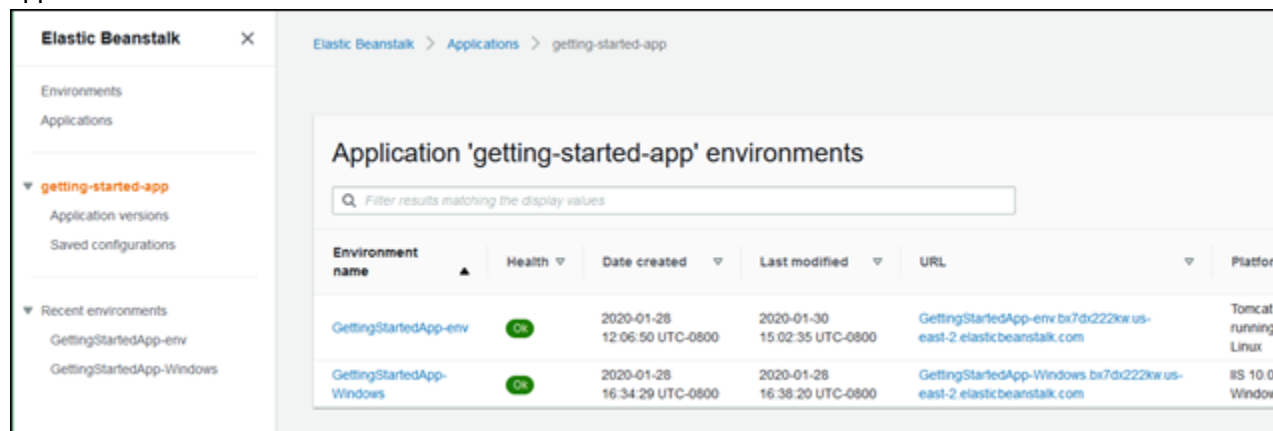
To access the application management console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.

Note

If you have many applications, use the search bar to filter the application list.

The application overview page shows a list with an overview of all environments associated with the application.



3. You have a few ways to continue:
 - a. Choose the **Actions** drop-down menu, and then choose one of the application management actions. To launch an environment in this application, you can directly choose **Create a new environment**. For details, see [the section called "Creating environments" \(p. 363\)](#).
 - b. Choose an environment name to go to the [environment management console \(p. 353\)](#) for that environment, where you can configure, monitor, or manage the environment.
 - c. Choose **Application versions** following the application name in the navigation pane to view and manage the application versions for your application.

An application version is an uploaded version of your application code. You can upload new versions, deploy an existing version to any of the application's environments, or delete old versions. For more information, see [Managing application versions \(p. 337\)](#).

- d. Choose **Saved configurations** following the application name in the navigation pane to view and manage configurations saved from running environments.

A saved configuration is a collection of settings that you can use to restore an environment's settings to a previous state, or to create an environment with the same settings. For more information see [Using Elastic Beanstalk saved configurations \(p. 640\)](#).

Managing application versions

Elastic Beanstalk creates an application version whenever you upload source code. This usually occurs when you create an environment or upload and deploy code using the [environment management console \(p. 353\)](#) or [EB CLI \(p. 852\)](#). Elastic Beanstalk deletes these application versions according to the application's lifecycle policy and when you delete the application. For details about application lifecycle policy, see [Configuring application version lifecycle settings \(p. 339\)](#).

You can also upload a source bundle without deploying it from the [application management console \(p. 336\)](#). Elastic Beanstalk stores source bundles in Amazon Simple Storage Service (Amazon S3) and doesn't automatically delete them.

You can apply tags to an application version when you create it, and edit tags of existing application versions. For details, see [Tagging application versions \(p. 340\)](#).

To create a new application version

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.

Note

If you have many applications, use the search bar to filter the application list.

3. In the navigation pane, find your application's name and choose **Application versions**.
4. Choose **Upload**. Use the on-screen form to upload your application's [source bundle \(p. 342\)](#).

Note

The source bundle's file size limit is 512 MB.

5. Optionally, provide a brief description, and add tag keys and values.
6. Choose **Upload**.

The file you specified is associated with your application. You can deploy the application version to a new or existing environment.

Over time, your application can accumulate many application versions. To save storage space and avoid hitting the [application version quota](#), it's a good idea to delete application versions that you no longer need.

Note

Deleting an application version doesn't affect environments currently running that version.

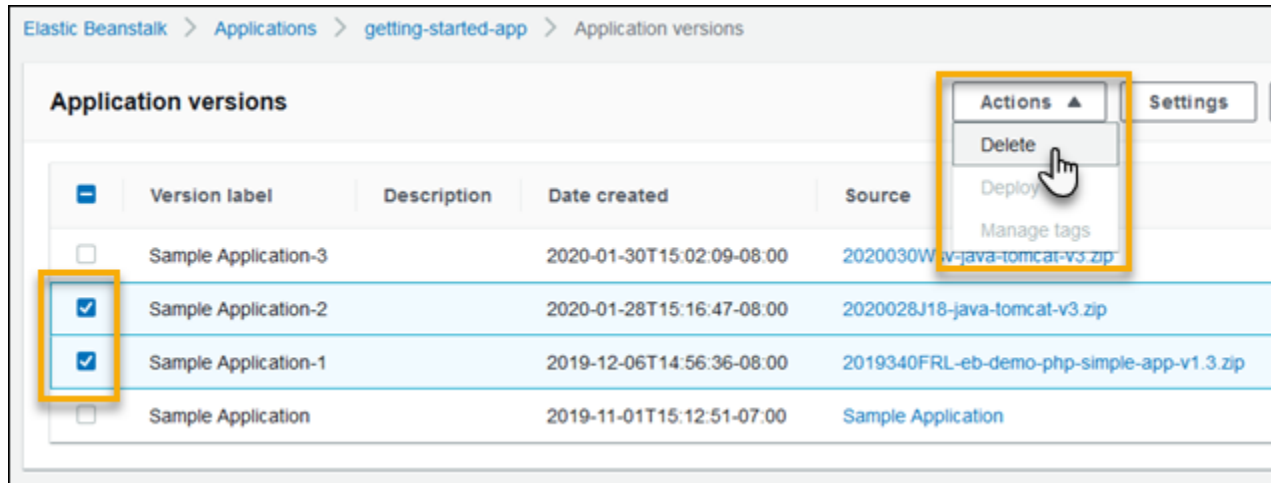
To delete an application version

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.

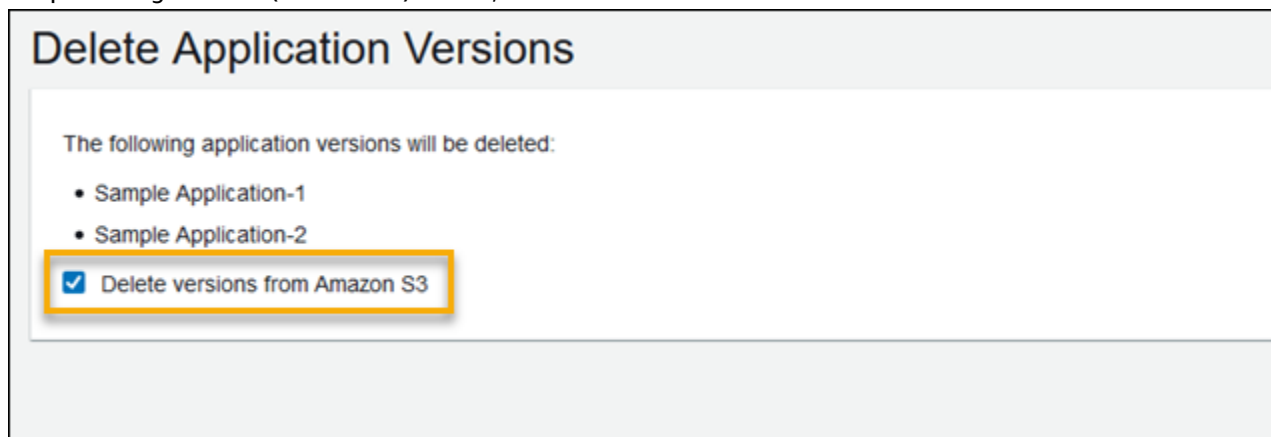
Note

If you have many applications, use the search bar to filter the application list.

3. In the navigation pane, find your application's name and choose **Application versions**.
4. Select one or more application versions that you want to delete.



5. Choose **Actions**, then choose **Delete**.
6. (Optional) To leave the application source bundle for these application versions in your Amazon Simple Storage Service (Amazon S3) bucket, clear the box for **Delete versions from Amazon S3**.



7. Choose **Delete**.

You can also configure Elastic Beanstalk to delete old versions automatically by configuring application version lifecycle settings. If you configure these lifecycle settings, they're applied when you create new application versions. For example, if you configure a maximum of 25 application versions, Elastic Beanstalk deletes the oldest version when you upload a 26th version. If you set a maximum age of 90 days, any versions older than 90 days are deleted when you upload a new version. For details, see [the section called "Version lifecycle" \(p. 339\)](#).

If you don't choose to delete the source bundle from Amazon S3, Elastic Beanstalk still deletes the version from its records. However, the source bundle is left in your [Elastic Beanstalk storage bucket \(p. 831\)](#). The application version quota applies only to versions Elastic Beanstalk tracks. Therefore, you can delete versions to stay within the quota, but retain all source bundles in Amazon S3.

Note

The application version quota doesn't apply to source bundles, but you might still incur Amazon S3 charges, and retain personal information beyond the time you need it. Elastic Beanstalk never deletes source bundles automatically. You should delete source bundles when you no longer need them.

Configuring application version lifecycle settings

Each time you upload a new version of your application with the Elastic Beanstalk console or the EB CLI, Elastic Beanstalk creates an [application version](#) (p. 337). If you don't delete versions that you no longer use, you will eventually reach the [application version quota](#) and be unable to create new versions of that application.

You can avoid hitting the quota by applying an *application version lifecycle policy* to your applications. A lifecycle policy tells Elastic Beanstalk to delete application versions that are old, or to delete application versions when the total number of versions for an application exceeds a specified number.

Elastic Beanstalk applies an application's lifecycle policy each time you create a new application version, and deletes up to 100 versions each time the lifecycle policy is applied. Elastic Beanstalk deletes old versions after creating the new version, and does not count the new version towards the maximum number of versions defined in the policy.

Elastic Beanstalk does not delete application versions that are currently being used by an environment, or application versions deployed to environments that were terminated less than ten weeks before the policy was triggered.

The application version quota applies across all applications in a region. If you have several applications, configure each application with a lifecycle policy appropriate to avoid reaching the quota. For example, if you have 10 applications in a region and the quota is 1,000 application versions, consider setting a lifecycle policy with a quota of 99 application versions for all applications, or set other values in each application as long as the total is less than 1,000 application versions. Elastic Beanstalk only applies the policy if the application version creation succeeds, so if you have already reached the quota, you must delete some versions manually prior to creating a new version.

By default, Elastic Beanstalk leaves the application version's [source bundle](#) (p. 342) in Amazon S3 to prevent loss of data. You can delete the source bundle to save space.

You can set the lifecycle settings through the Elastic Beanstalk CLI and APIs. See [eb appversion](#) (p. 886), [CreateApplication](#) (using the `ResourceLifecycleConfig` parameter), and [UpdateApplicationResourceLifecycle](#) for details.

Setting the application lifecycle settings in the console

You can specify the lifecycle settings in the Elastic Beanstalk console.

To specify your application lifecycle settings

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.

Note

If you have many applications, use the search bar to filter the application list.

3. In the navigation pane, find your application's name and choose **Application versions**.
4. Choose **Settings**.
5. Use the on-screen form to configure application lifecycle settings.
6. Choose **Save**.

Application version lifecycle settings ✕

Configure a lifecycle policy to limit the number of application versions to retain for future deployments. This policy will not delete application versions that are currently deployed or are in the process of being created. [Learn more](#)

Lifecycle policy

Enable

Lifecycle rule

Set the application versions limit by total count

200 Application Versions

Set the application versions limit by age

180 days

Retention

Delete source bundle from S3

Service role

On the settings page, you can do the following.

- Configure lifecycle settings based on the total count of application versions or the age of application versions.
- Specify whether to delete the source bundle from S3 when the application version is deleted.
- Specify the service role under which the application version is deleted. To include all permissions required for version deletion, choose the default Elastic Beanstalk service role, named `aws-elasticbeanstalk-service-role`, or another service role using the Elastic Beanstalk managed service policies. For more information, see [Managing Elastic Beanstalk service roles \(p. 764\)](#).

Tagging application versions

You can apply tags to your AWS Elastic Beanstalk application versions. Tags are key-value pairs associated with AWS resources. For information about Elastic Beanstalk resource tagging, use cases, tag key and value constraints, and supported resource types, see [Tagging Elastic Beanstalk application resources \(p. 349\)](#).

You can specify tags when you create an application version. In an existing application version, you can add or remove tags, and update the values of existing tags. You can add up to 50 tags to each application version.

Adding tags during application version creation

When you use the Elastic Beanstalk console to [create an environment](#) (p. 365), and you choose to upload a version of your application code, you can specify tag keys and values to associate with the new application version.

You can also use the Elastic Beanstalk console to [upload an application version](#) (p. 337) without immediately using it in an environment. You can specify tag keys and values when you upload an application version.

With the AWS CLI or other API-based clients, add tags by using the `--tags` parameter on the [create-application-version](#) command.

```
$ aws elasticbeanstalk create-application-version \  
  --tags Key=mytag1,Value=value1 Key=mytag2,Value=value2 \  
  --application-name my-app --version-label v1
```

When you use the EB CLI to create or update an environment, an application version is created from the code that you deploy. There isn't a direct way to tag an application version during its creation through the EB CLI. See the following section to learn about adding tags to an existing application version.

Managing tags of an existing application version

You can add, update, and delete tags in an existing Elastic Beanstalk application version.

To manage an application version's tags using the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.

Note

If you have many applications, use the search bar to filter the application list.

3. In the navigation pane, find your application's name and choose **Application versions**.
4. Select the application version you want to manage.
5. Choose **Actions**, and then choose **Manage tags**.
6. Use the on-screen form to add, update, or delete tags.
7. Choose **Apply**.

If you use the EB CLI to update your application version, use [eb tags](#) (p. 930) to add, update, delete, or list tags.

For example, the following command lists the tags in an application version.

```
~/workspace/my-app$ eb tags --list --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:applicationversion/my-app/my-version"
```

The following command updates the tag `mytag1` and deletes the tag `mytag2`.

```
~/workspace/my-app$ eb tags --update mytag1=newvalue --delete mytag2 \  
  --application-version my-app/my-version
```

```
--resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:applicationversion/my-app/my-version"
```

For a complete list of options and more examples, see [eb tags](#) (p. 930).

With the AWS CLI or other API-based clients, use the [list-tags-for-resource](#) command to list the tags of an application version.

```
$ aws elasticbeanstalk list-tags-for-resource --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-id:applicationversion/my-app/my-version"
```

Use the [update-tags-for-resource](#) command to add, update, or delete tags in an application version.

```
$ aws elasticbeanstalk update-tags-for-resource \
  --tags-to-add Key=mytag1,Value=newvalue --tags-to-remove mytag2 \
  --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-id:applicationversion/my-app/my-version"
```

Specify both tags to add and tags to update in the `--tags-to-add` parameter of [update-tags-for-resource](#). A nonexistent tag is added, and an existing tag's value is updated.

Note

To use some of the EB CLI and AWS CLI commands with an Elastic Beanstalk application version, you need the application version's ARN. You can retrieve the ARN by using the following command.

```
$ aws elasticbeanstalk describe-application-versions --application-name my-app --version-label my-version
```

Create an application source bundle

When you use the AWS Elastic Beanstalk console to deploy a new application or an application version, you'll need to upload a source bundle. Your source bundle must meet the following requirements:

- Consist of a single ZIP file or WAR file (you can include multiple WAR files inside your ZIP file)
- Not exceed 512 MB
- Not include a parent folder or top-level directory (subdirectories are fine)

If you want to deploy a worker application that processes periodic background tasks, your application source bundle must also include a `cron.yaml` file. For more information, see [Periodic tasks](#) (p. 440).

If you are deploying your application with the Elastic Beanstalk Command Line Interface (EB CLI), the AWS Toolkit for Eclipse, or the AWS Toolkit for Visual Studio, the ZIP or WAR file will automatically be structured correctly. For more information, see [Using the Elastic Beanstalk command line interface \(EB CLI\)](#) (p. 852), [Creating and deploying Java applications on Elastic Beanstalk](#) (p. 94), and [The AWS Toolkit for Visual Studio](#) (p. 166).

Sections

- [Creating a source bundle from the command line](#) (p. 343)
- [Creating a source bundle with Git](#) (p. 343)
- [Zipping files in Mac OS X Finder or Windows explorer](#) (p. 343)

- [Creating a source bundle for a .NET application \(p. 347\)](#)
- [Testing your source bundle \(p. 348\)](#)

Creating a source bundle from the command line

Create a source bundle using the `zip` command. To include hidden files and folders, use a pattern like the following.

```
~/myapp$ zip ../myapp.zip -r * .[^.]*
  adding: app.js (deflated 63%)
  adding: index.js (deflated 44%)
  adding: manual.js (deflated 64%)
  adding: package.json (deflated 40%)
  adding: restify.js (deflated 85%)
  adding: .ebextensions/ (stored 0%)
  adding: .ebextensions/xray.config (stored 0%)
```

This ensures that Elastic Beanstalk [configuration files \(p. 600\)](#) and other files and folders that start with a period are included in the archive.

For Tomcat web applications, use `jar` to create a web archive.

```
~/myapp$ jar -cvf myapp.war .
```

The above commands include hidden files that may increase your source bundle size unnecessarily. For more control, use a more detailed file pattern, or [create your source bundle with Git \(p. 343\)](#).

Creating a source bundle with Git

If you're using Git to manage your application source code, use the `git archive` command to create your source bundle.

```
$ git archive -v -o myapp.zip --format=zip HEAD
```

`git archive` only includes files that are stored in git, and excludes ignored files and git files. This helps keep your source bundle as small as possible. For more information, go to the [git-archive manual page](#).

Zippping files in Mac OS X Finder or Windows explorer

When you create a ZIP file in Mac OS X Finder or Windows Explorer, make sure you zip the files and subfolders themselves, rather than zipping the parent folder.

Note

The graphical user interface (GUI) on Mac OS X and Linux-based operating systems does not display files and folders with names that begin with a period (.). Use the command line instead of the GUI to compress your application if the ZIP file must include a hidden folder, such as `.ebextensions`. For command line procedures to create a ZIP file on Mac OS X or a Linux-based operating system, see [Creating a source bundle from the command line \(p. 343\)](#).

Example

Suppose you have a Python project folder labeled `myapp`, which includes the following files and subfolders:

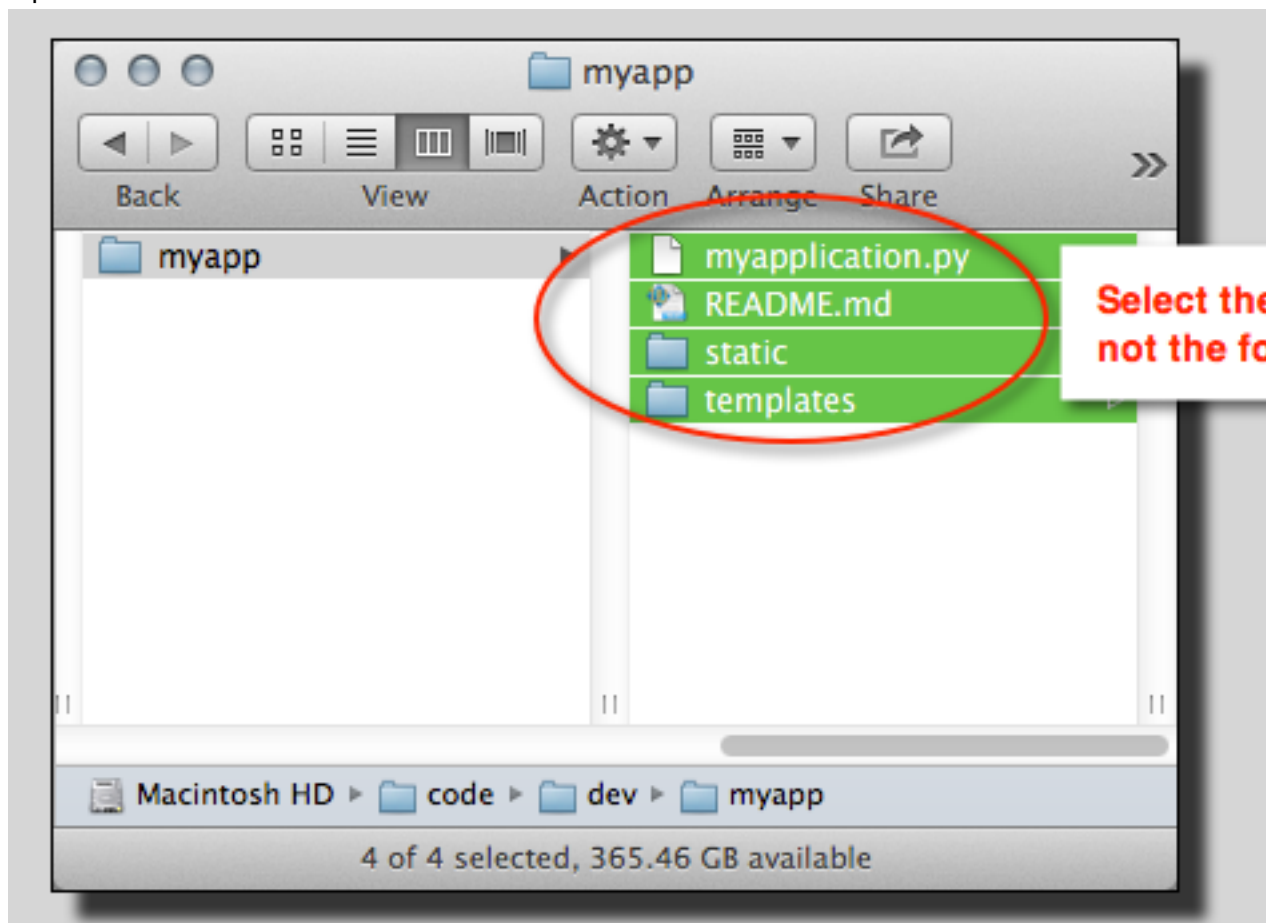
```
myapplication.py
README.md
static/
static/css
static/css/styles.css
static/img
static/img/favicon.ico
static/img/logo.png
templates/
templates/base.html
templates/index.html
```

As noted in the list of requirements above, your source bundle must be compressed without a parent folder, so that its decompressed structure does not include an extra top-level directory. In this example, no `myapp` folder should be created when the files are decompressed (or, at the command line, no `myapp` segment should be added to the file paths).

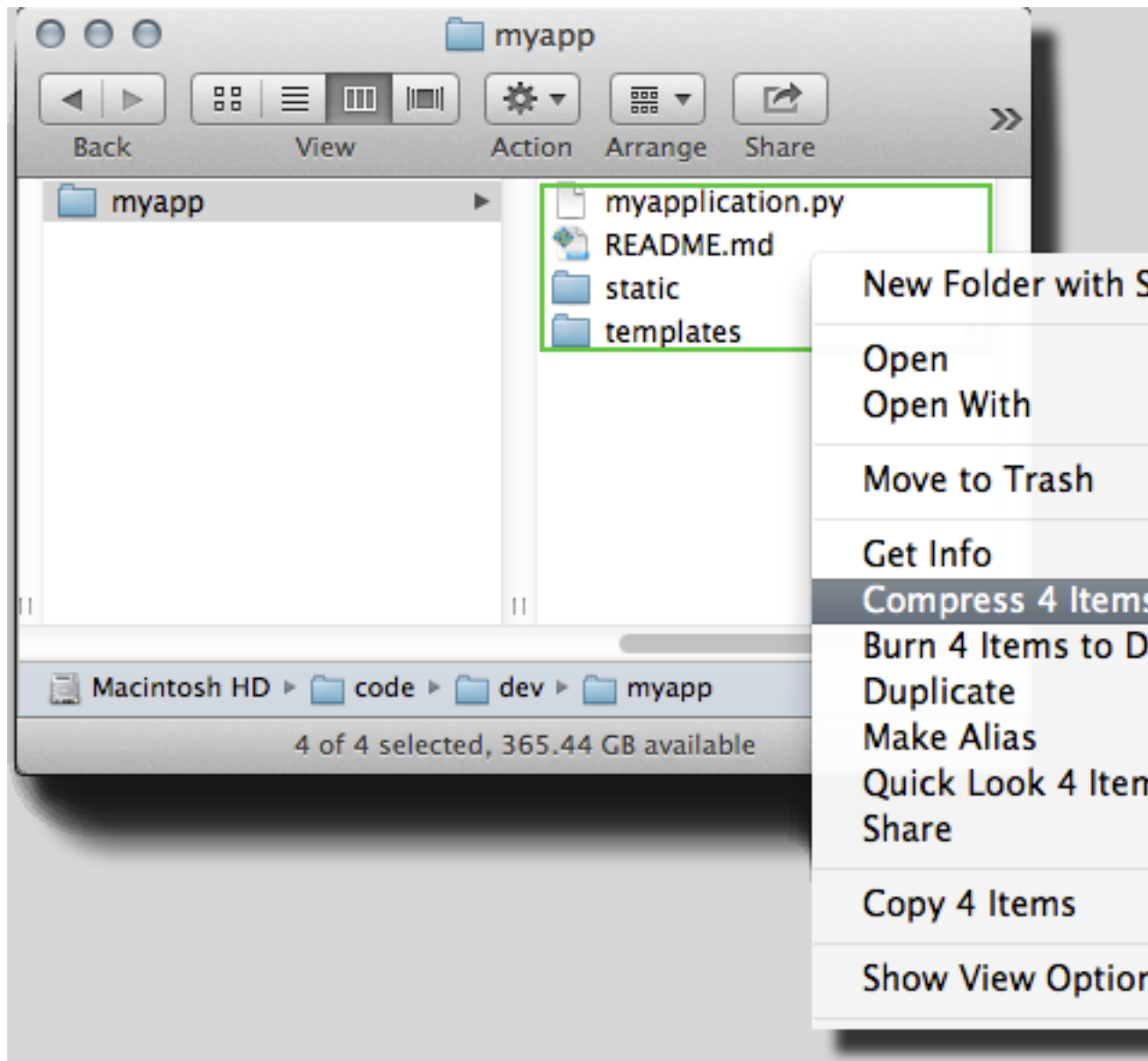
This sample file structure is used throughout this topic to illustrate how to zip files.

To zip files in Mac OS X Finder

1. Open your top-level project folder and select all the files and subfolders within it. Do not select the top-level folder itself.

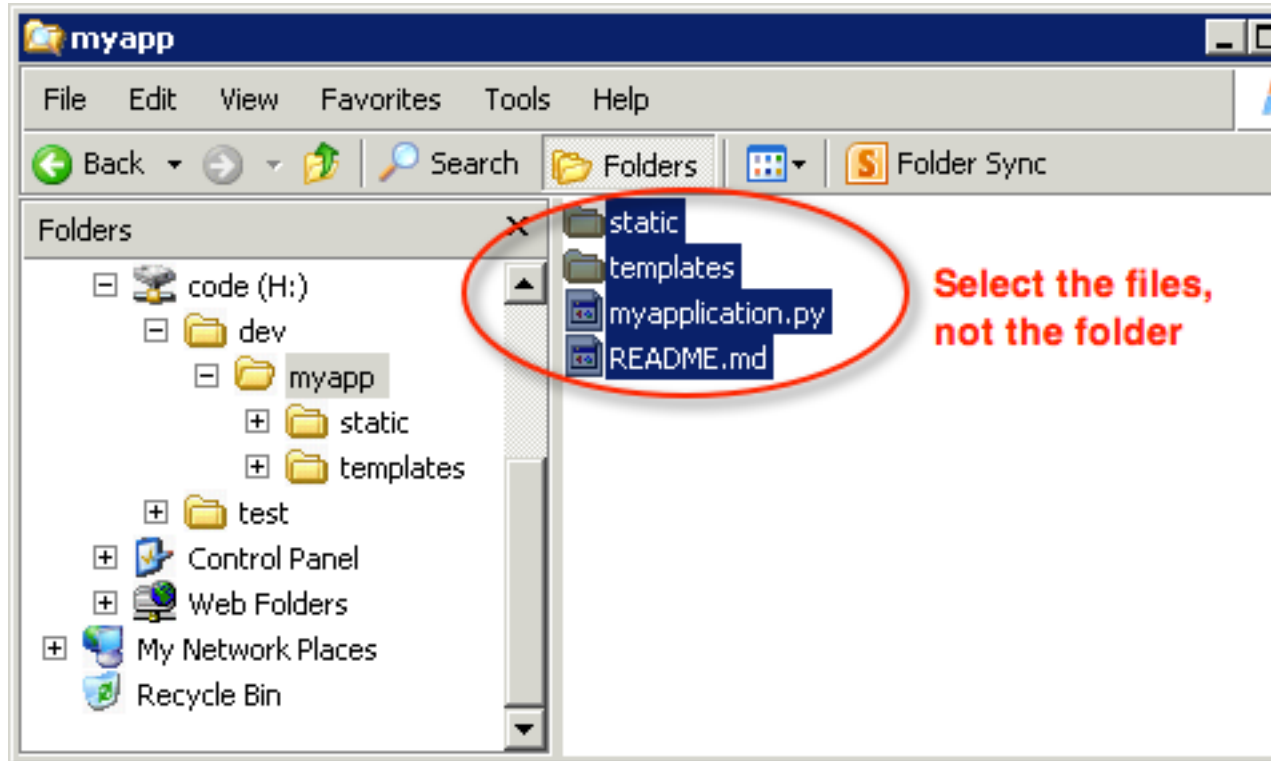


2. Right-click the selected files, and then choose **Compress X items**, where X is the number of files and subfolders you've selected.

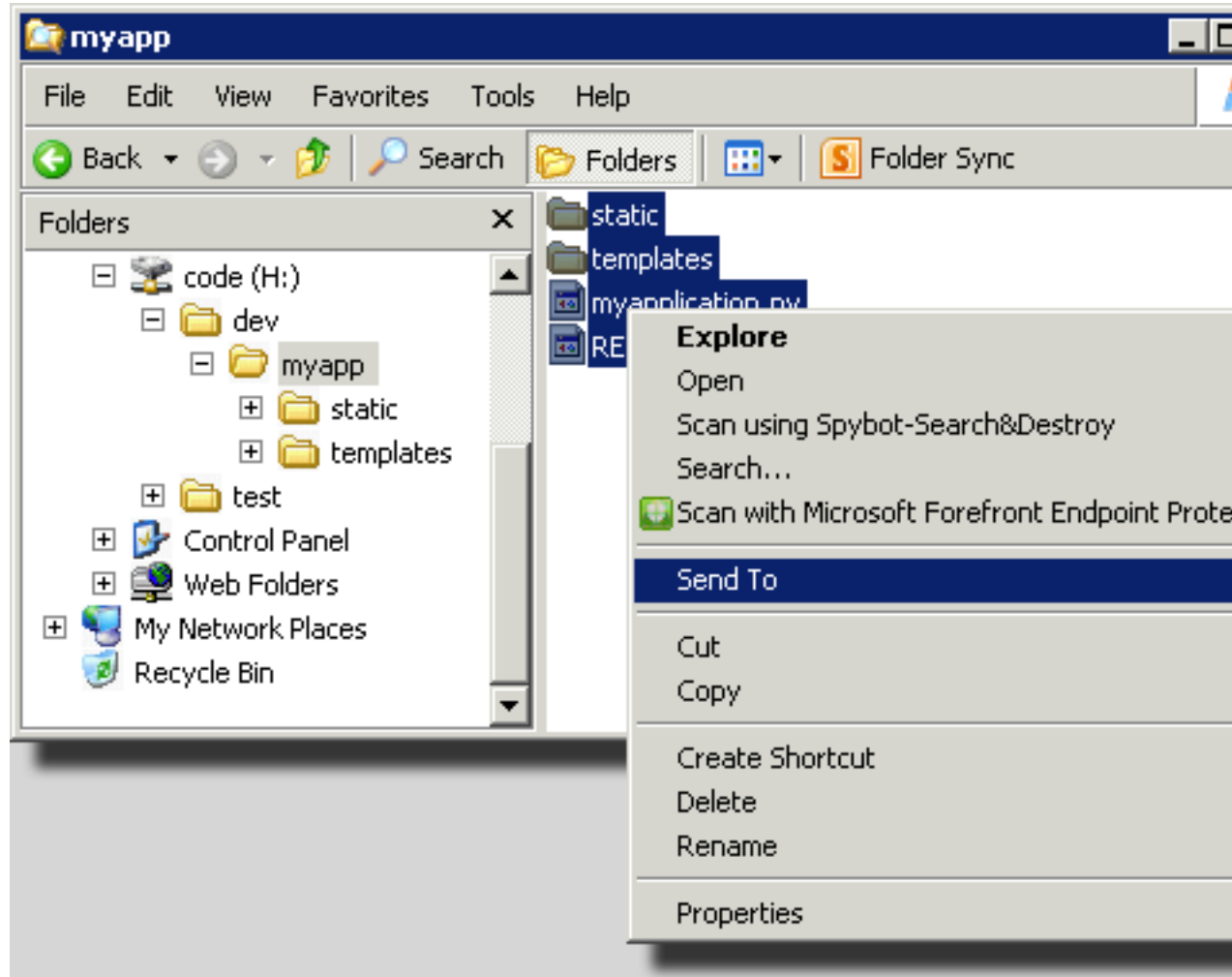


To zip files in Windows explorer

1. Open your top-level project folder and select all the files and subfolders within it. Do not select the top-level folder itself.



2. Right-click the selected files, choose **Send to**, and then choose **Compressed (zipped) folder**.



Creating a source bundle for a .NET application

If you use Visual Studio, you can use the deployment tool included in the AWS Toolkit for Visual Studio to deploy your .NET application to Elastic Beanstalk. For more information, see [Deploying Elastic Beanstalk applications in .NET using the deployment tool \(p. 190\)](#).

If you need to manually create a source bundle for your .NET application, you cannot simply create a ZIP file that contains the project directory. You must create a web deployment package for your project that is suitable for deployment to Elastic Beanstalk. There are several methods you can use to create a deployment package:

- Create the deployment package using the **Publish Web** wizard in Visual Studio. For more information, go to [How to: Create a Web Deployment Package in Visual Studio](#).

Important

When creating the web deployment package, you must start the **Site name** with `Default Web Site`.

- If you have a .NET project, you can create the deployment package using the **msbuild** command as shown in the following example.

Important

The `DeployIisAppPath` parameter must begin with `Default Web Site`.

```
C:/> msbuild <web_app>.csproj /t:Package /p:DeployIisAppPath="Default Web Site"
```

- If you have a website project, you can use the IIS Web Deploy tool to create the deployment package. For more information, go to [Packaging and Restoring a Web site](#).

Important

The `apphostconfig` parameter must begin with `Default Web Site`.

If you are deploying multiple applications or an ASP.NET Core application, put your `.ebextensions` folder in the root of the source bundle, side by side with the application bundles and manifest file:

```
~/workspace/source-bundle/  
|-- .ebextensions  
|   |-- environmentvariables.config  
|   |-- healthcheckurl.config  
|-- AspNetCore101HelloWorld.zip  
|-- AspNetCoreHelloWorld.zip  
|-- aws-windows-deployment-manifest.json  
|-- VS2015AspNetWebApiApp.zip
```

Testing your source bundle

You may want to test your source bundle locally before you upload it to Elastic Beanstalk. Because Elastic Beanstalk essentially uses the command line to extract the files, it's best to do your tests from the command line rather than with a GUI tool.

To test the file extraction in Mac OS X or Linux

1. Open a terminal window (Mac OS X) or connect to the Linux server. Navigate to the directory that contains your source bundle.
2. Using the `unzip` or `tar xzf` command, decompress the archive.
3. Ensure that the decompressed files appear in the same folder as the archive itself, rather than in a new top-level folder or directory.

Note

If you use Mac OS X Finder to decompress the archive, a new top-level folder will be created, no matter how you structured the archive itself. For best results, use the command line.

To test the file extraction in Windows

1. Download or install a program that allows you to extract compressed files via the command line. For example, you can download the free `unzip.exe` program from <http://stahlforce.com/dev/index.php?tool=zipunzip>.
2. If necessary, copy the executable file to the directory that contains your source bundle. If you've installed a system-wide tool, you can skip this step.
3. Using the appropriate command, decompress the archive. If you downloaded `unzip.exe` using the link in step 1, the command is `unzip <archive-name>`.
4. Ensure that the decompressed files appear in the same folder as the archive itself, rather than in a new top-level folder or directory.

Tagging Elastic Beanstalk application resources

You can apply tags to resources of your AWS Elastic Beanstalk applications. Tags are key-value pairs associated with AWS resources. Tags can help you categorize resources. They're particularly useful if you manage many resources as part of multiple AWS applications.

Here are some ways to use tagging with Elastic Beanstalk resources:

- *Deployment stages* – Identify resources associated with different stages of your application, such as development, beta, and production.
- *Cost allocation* – Use cost allocation reports to track your usage of AWS resources associated with various expense accounts. The reports include both tagged and untagged resources, and they aggregate costs according to tags. For information about how cost allocation reports use tags, see [Use Cost Allocation Tags for Custom Billing Reports](#) in the *AWS Billing and Cost Management User Guide*.
- *Access control* – Use tags to manage permissions to requests and resources. For example, a user who can only create and manage beta environments should only have access to beta stage resources. For details, see [Using tags to control access to Elastic Beanstalk resources \(p. 807\)](#).

You can add up to 50 tags to each resource. Environments are slightly different: Elastic Beanstalk adds three default system tags to environments, and you can't edit or delete these tags. In addition to the default tags, you can add up to 47 additional tags to each environment.

The following constraints apply to tag keys and values:

- Keys and values can contain letters, numbers, white space, and the following symbols: `_ . : / = + - @`
- Keys can contain up to 127 characters. Values can contain up to 255 characters.

Note

These length limits are for Unicode characters in UTF-8. For other multibyte encodings, the limits might be lower.

- Keys are case sensitive.
- Keys cannot begin with `aws:` or `elasticbeanstalk:`.

Resources you can tag

The following are the types of Elastic Beanstalk resources that you can tag, and links to specific topics about managing tags for each of them:

- [Applications \(p. 349\)](#)
- [Environments \(p. 513\)](#)
- [Application versions \(p. 340\)](#)
- [Saved configurations \(p. 644\)](#)
- [Custom platform versions \(p. 48\)](#)

Tagging applications

You can apply tags to your AWS Elastic Beanstalk applications. Tags are key-value pairs associated with AWS resources. For information about Elastic Beanstalk resource tagging, use cases, tag key and value constraints, and supported resource types, see [Tagging Elastic Beanstalk application resources \(p. 349\)](#).

You can specify tags when you create an application. In an existing application, you can add or remove tags, and update the values of existing tags. You can add up to 50 tags to each application.

Adding tags during application creation

When you use the Elastic Beanstalk console to [create an application \(p. 334\)](#), you can specify tag keys and values in the **Create New Application** dialog box.

Elastic Beanstalk

Create new application

Application information

Application Name

Maximum length of 100 characters, not including forward slash (/).

Description

Tags

Apply up to 50 tags. You can use tags to group and filter your resources. A tag is a key-value pair. The key must be resource and is case-sensitive. [Learn more](#)

Key	Value	Remove
<input type="text"/>	<input type="text"/>	<input type="button" value="Remove"/>

50 remaining

If you use the EB CLI to create an application, use the `--tags` option with [eb init \(p. 907\)](#) to add tags.

```
~/workspace/my-app$ eb init --tags mytag1=value1,mytag2=value2
```

With the AWS CLI or other API-based clients, add tags by using the `--tags` parameter on the [create-application](#) command.

```
$ aws elasticbeanstalk create-application \  
  --tags Key=mytag1,Value=value1 Key=mytag2,Value=value2 \  
  --application-name my-app --version-label v1
```

Managing tags of an existing application

You can add, update, and delete tags in an existing Elastic Beanstalk application.

To manage an application's tags in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.

Note

If you have many applications, use the search bar to filter the application list.

3. Choose **Actions**, and then choose **Manage tags**.
4. Use the on-screen form to add, update, or delete tags.
5. Choose **Apply**.

If you use the EB CLI to update your application, use [eb tags \(p. 930\)](#) to add, update, delete, or list tags.

For example, the following command lists the tags in an application.

```
~/workspace/my-app$ eb tags --list --resource "arn:aws:elasticbeanstalk:us-east-2:my-  
account-id:application/my-app"
```

The following command updates the tag `mytag1` and deletes the tag `mytag2`.

```
~/workspace/my-app$ eb tags --update mytag1=newvalue --delete mytag2 \  
  --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:application/my-app"
```

For a complete list of options and more examples, see [eb tags \(p. 930\)](#).

With the AWS CLI or other API-based clients, use the [list-tags-for-resource](#) command to list the tags of an application.

```
$ aws elasticbeanstalk list-tags-for-resource --resource-arn "arn:aws:elasticbeanstalk:us-  
east-2:my-account-id:application/my-app"
```

Use the [update-tags-for-resource](#) command to add, update, or delete tags in an application.

```
$ aws elasticbeanstalk update-tags-for-resource \  
  --tags-to-add Key=mytag1,Value=newvalue --tags-to-remove mytag2 \  
  --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-id:application/my-app"
```

Specify both tags to add and tags to update in the `--tags-to-add` parameter of [update-tags-for-resource](#). A nonexistent tag is added, and an existing tag's value is updated.

Note

To use some of the EB CLI and AWS CLI commands with an Elastic Beanstalk application, you need the application's ARN. You can retrieve the ARN by using the following command.

```
$ aws elasticbeanstalk describe-applications --application-names my-app
```

Managing environments

AWS Elastic Beanstalk makes it easy to create new environments for your application. You can create and manage separate environments for development, testing, and production use, and you can [deploy any version \(p. 397\)](#) of your application to any environment. Environments can be long-running or temporary. When you terminate an environment, you can save its configuration to recreate it later.

As you develop your application, you will deploy it often, possibly to several different environments for different purposes. Elastic Beanstalk lets you [configure how deployments are performed \(p. 400\)](#). You can deploy to all of the instances in your environment simultaneously, or split a deployment into batches with rolling deployments.

[Configuration changes \(p. 408\)](#) are processed separately from deployments, and have their own scope. For example, if you change the type of the EC2 instances running your application, all of the instances must be replaced. On the other hand, if you modify the configuration of the environment's load balancer, that change can be made in-place without interrupting service or lowering capacity. You can also apply configuration changes that modify the instances in your environment in batches with [rolling configuration updates \(p. 409\)](#).

Note

Modify the resources in your environment only by using Elastic Beanstalk. If you modify resources using another service's console, CLI commands, or SDKs, Elastic Beanstalk won't be able to accurately monitor the state of those resources, and you won't be able to save the configuration or reliably recreate the environment. Out-of-band-changes can also cause issues when updating or terminating an environment.

When you launch an environment, you choose a platform version. We update platforms periodically with new platform versions to provide performance improvements and new features. You can [update your environment to the latest platform version \(p. 415\)](#) at any time.

As your application grows in complexity, you can split it into multiple components, each running in a separate environment. For long-running workloads, you can launch [worker environments \(p. 437\)](#) that process jobs from an Amazon Simple Queue Service (Amazon SQS) queue.

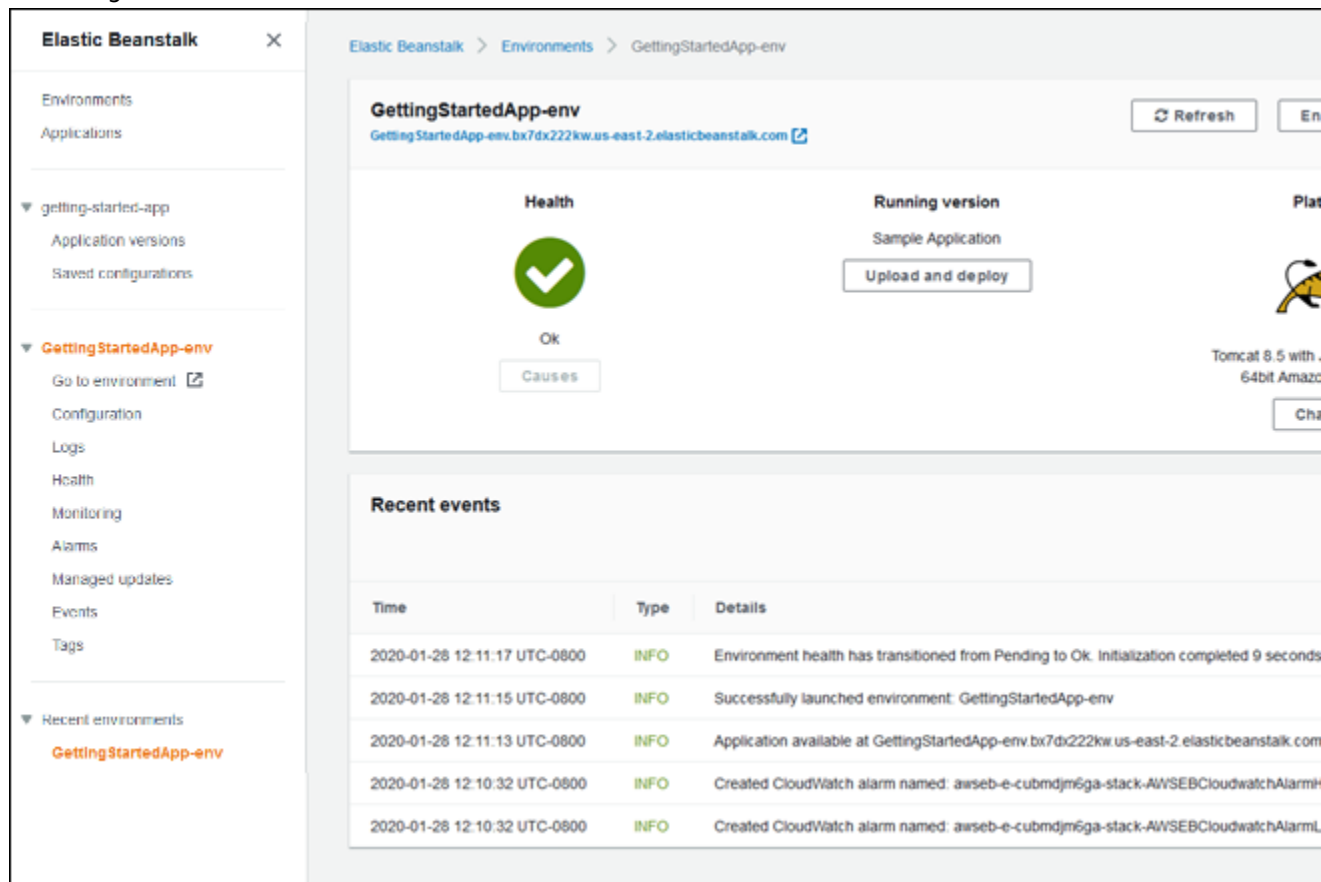
Topics

- [Using the Elastic Beanstalk environment management console \(p. 353\)](#)
- [Creating an Elastic Beanstalk environment \(p. 363\)](#)
- [Deploying applications to Elastic Beanstalk environments \(p. 397\)](#)
- [Configuration changes \(p. 408\)](#)
- [Updating your Elastic Beanstalk environment's platform version \(p. 415\)](#)
- [Canceling environment configuration updates and application deployments \(p. 432\)](#)
- [Rebuilding Elastic Beanstalk environments \(p. 433\)](#)
- [Environment types \(p. 435\)](#)
- [Elastic Beanstalk worker environments \(p. 437\)](#)
- [Creating links between Elastic Beanstalk environments \(p. 444\)](#)

Using the Elastic Beanstalk environment management console

The Elastic Beanstalk console provides a management page for each of your AWS Elastic Beanstalk environments. From this page, you can manage your environment's configuration and perform common

actions including restarting the web servers running in your environment, cloning the environment, or rebuilding it from scratch.



To access the environment management console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

You see the environment overview page. The console's navigation pane shows the name of the application to which the environment belongs, with related application management pages, and the environment name, with environment management pages.

Topics

- [Environment overview](#) (p. 355)
- [Environment actions](#) (p. 356)
- [Configuration](#) (p. 358)
- [Logs](#) (p. 358)
- [Health](#) (p. 359)
- [Monitoring](#) (p. 359)
- [Alarms](#) (p. 360)
- [Managed updates](#) (p. 361)

- [Events \(p. 361\)](#)
- [Tags \(p. 362\)](#)

Environment overview

To view the environment overview page, choose the environment name on the navigation pane, if it's the current environment. Alternatively, navigate to the environment from the application page or from the main environment list on the **Environments** page.

The top pane on the environment overview page shows top level information about your environment. This includes its name, its URL, its current health status, the name of the currently deployed application version, and the platform version that the application is running on. Below the overview pane you can see the five most recent environment events.

Choose **Refresh** to update the information shown. The overview page contains the following information and options.

URL

The environment's **URL** is located at the top of the overview, below the environment name. This is the URL of the web application that the environment is running.

Health

The overall health of the environment. With [Enhanced health reporting and monitoring \(p. 691\)](#) enabled, the environment status is shown with a **Causes** button you can choose to view more information about the current status.

For [Basic health reporting \(p. 688\)](#) environments, a link to the [Monitoring Console \(p. 685\)](#) is shown.

Running version


The name of the application version that is deployed and running on your environment. Choose **Upload and deploy** to upload a [source bundle \(p. 342\)](#) and deploy it to your environment. This option creates a new application version.

Platform

Shows the name of the platform version running on your environment—typically a combination of the architecture, operating system (OS), language, and application server (collectively, the *platform branch*), with a specific platform version number. Choose **Change** to select a different platform version. This option is available only if another version of the platform branch is available.

Updating the platform version using this option replaces instances running in your environment with new instances.

Update platform version

 **Availability warning**

This operation replaces your instances; your application is unavailable during the update. To keep an instance in service during the update, enable rolling updates. Another option is to clone the current environment, which creates a newer version of the platform, and then swap the CNAME of the environment when you are ready to deploy the clone. Learn more at [Updating AWS Elastic Beanstalk Environments with Rolling Updates](#) and [Deploying Version with Zero Downtime](#).

Platform branch
Tomcat 8.5 with Java 8 running on 64bit Amazon Linux

Current platform version
3.3.1

New platform version
3.3.2 (Recommended) ▼

Cancel

Note

When you first use Elastic Beanstalk, only the latest (recommended) version of each platform branch is available for use. **Change** first becomes available when a new platform version is released for the branch. After upgrading, you have the option to change back to the previous version.

Recent events

The **Recent Events** section of the environment overview page shows the most recent events emitted by your environment. This list is updated in real time when your environment is being updated.

Choose **Show all** to open the **Events** page.

Environment actions

The environment overview page contains an **Environment actions** menu that you can use to perform common operations on your environment. This menu is shown on the right side of the environment header under the **Create New Environment** option.

Note

Some actions are only available under certain conditions, and will be disabled unless these conditions are met.

Load configuration

Load a previously saved configuration. Configurations are saved to your application and can be loaded by any associated environment. If you've made changes to your environment's configuration, you can load a saved configuration to undo those changes. You can also load a configuration that you saved from a different environment running the same application to propagate configuration changes between them.

Save configuration

Save the current configuration of your environment to your application. Before you make changes to your environment's configuration, save the current configuration so that you can roll back later, if needed. You can also apply a saved configuration when you launch a new environment.

Swap environment URLs

Swap the CNAME of the current environment with a new environment. After a CNAME swap, all traffic to the application using the environment URL goes to the new environment. When you are ready to deploy a new version of your application, you can launch a separate environment under the new version. When the new environment is ready to start taking requests, perform a CNAME swap to start routing traffic to the new environment with no interruption of service. For more information, see [Blue/Green deployments with Elastic Beanstalk](#) (p. 405).

Clone environment

Launch a new environment with the same configuration as your currently running environment.

Clone with latest platform

Clone your current environment with the latest version of the in-use Elastic Beanstalk platform. This option is available only when a newer version of the current environment's platform is available for use.

Abort current operation

Stop an in-progress environment update. Aborting an operation can cause some of the instances in your environment to be in a different state than others, depending on how far the operation progressed. This option is available only when your environment is being updated.

Restart app servers

Restart the web server running on your environment's instances. This option does not terminate or restart any AWS resources. If your environment is acting strangely in response to some bad requests, restarting the application server can restore functionality temporarily while you troubleshoot the root cause.

Rebuild environment

Terminate all resources in the running environment and build a new environment with the same settings. This operation takes several minutes, equivalent to deploying a new environment from scratch. Any Amazon RDS instances running in your environment's data tier are deleted during a rebuild. If you need the data, create a snapshot. You can create a snapshot manually [in the RDS console](#) or configure your data tier's Deletion Policy to create a snapshot automatically before deleting the instance (this is the default setting when you create a data tier).

Terminate environment

Terminate all resources in the running environment, and remove the environment from the application. If you have an RDS instance running in a data tier and need to retain the data, be sure to take a snapshot

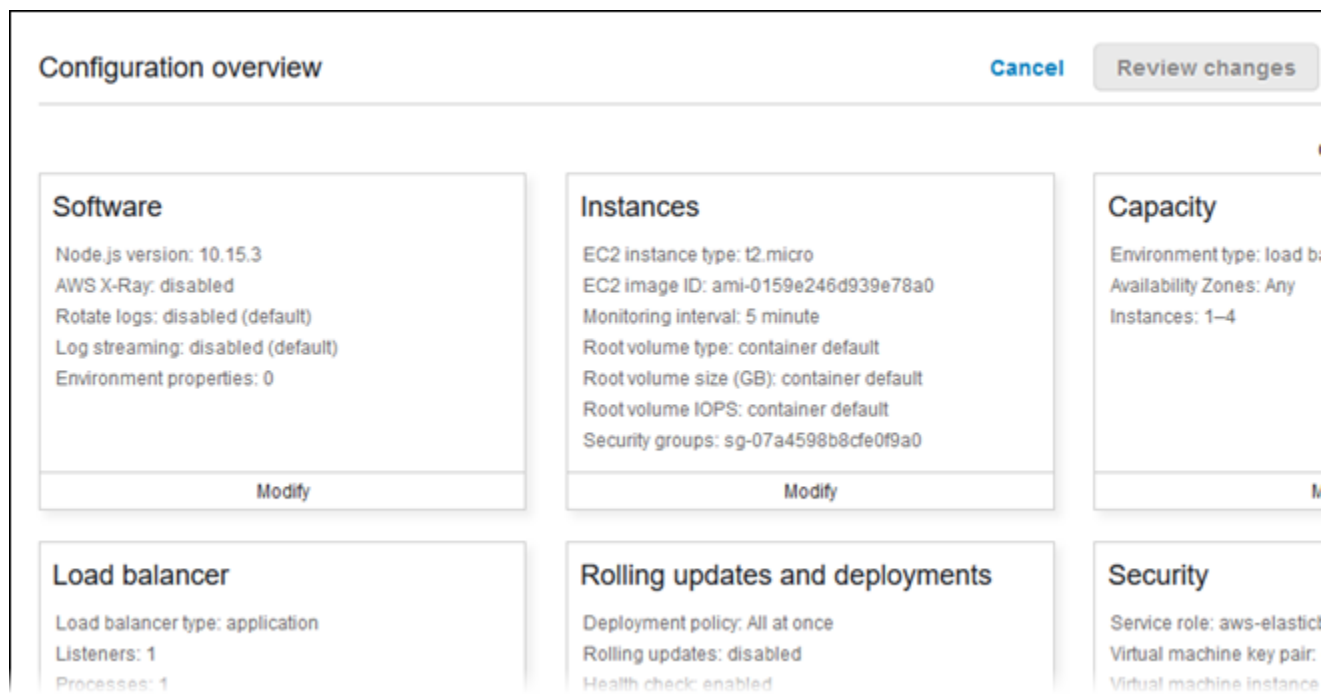
before terminating your environment. You can create a snapshot manually [in the RDS console](#) or configure your data tier's Deletion Policy to create a snapshot automatically before deleting the instance (this is the default setting when you create a data tier).

Restore environment

If the environment has been terminated in the last hour, you can restore it from this page. After an hour, you can [restore it from the application overview page](#) (p. 434).

Configuration

The **Configuration overview** page shows the current configuration of your environment and its resources, including Amazon EC2 instances, load balancer, notifications, and health monitoring settings. Use the settings on this page to customize the behavior of your environment during deployments, enable additional features, and modify the instance type and other settings that you chose during environment creation.

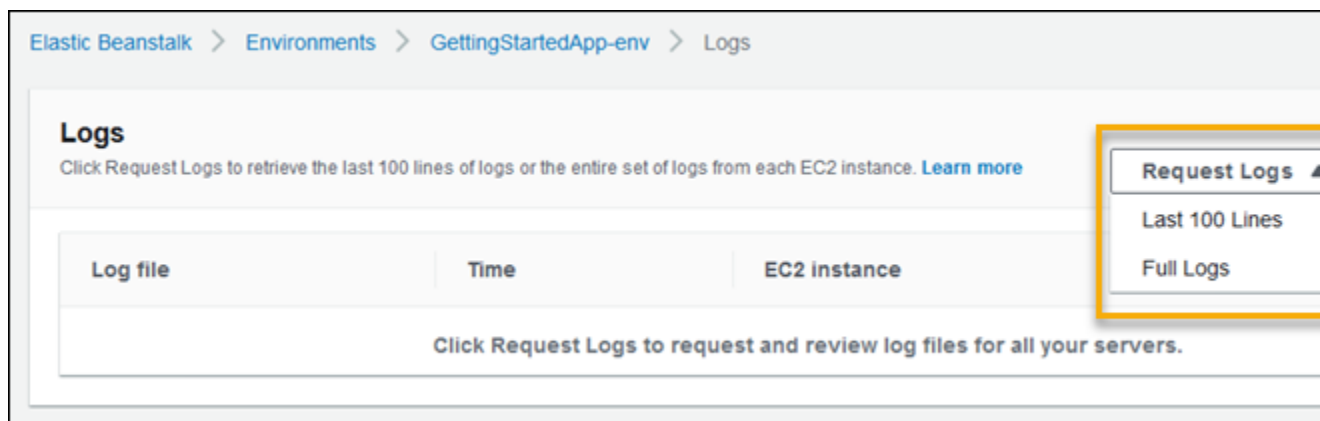


For more information, see [Configuring Elastic Beanstalk environments](#) (p. 446).

Logs

The **Logs** page lets you retrieve logs from the EC2 instances in your environment. When you request logs, Elastic Beanstalk sends a command to the instances, which then upload logs to your Elastic Beanstalk storage bucket in Amazon S3. When you request logs on this page, Elastic Beanstalk automatically deletes them from Amazon S3 after 15 minutes.

You can also configure your environment's instances to upload logs to Amazon S3 for permanent storage after they have been rotated locally.



For more information, see [Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment \(p. 732\)](#).

Health

If enhanced health monitoring is enabled, the **Enhanced health overview** page shows live health information about every instance in your environment. Enhanced health monitoring enables Elastic Beanstalk to closely monitor the resources in your environment so that it can assess the health of your application more accurately.

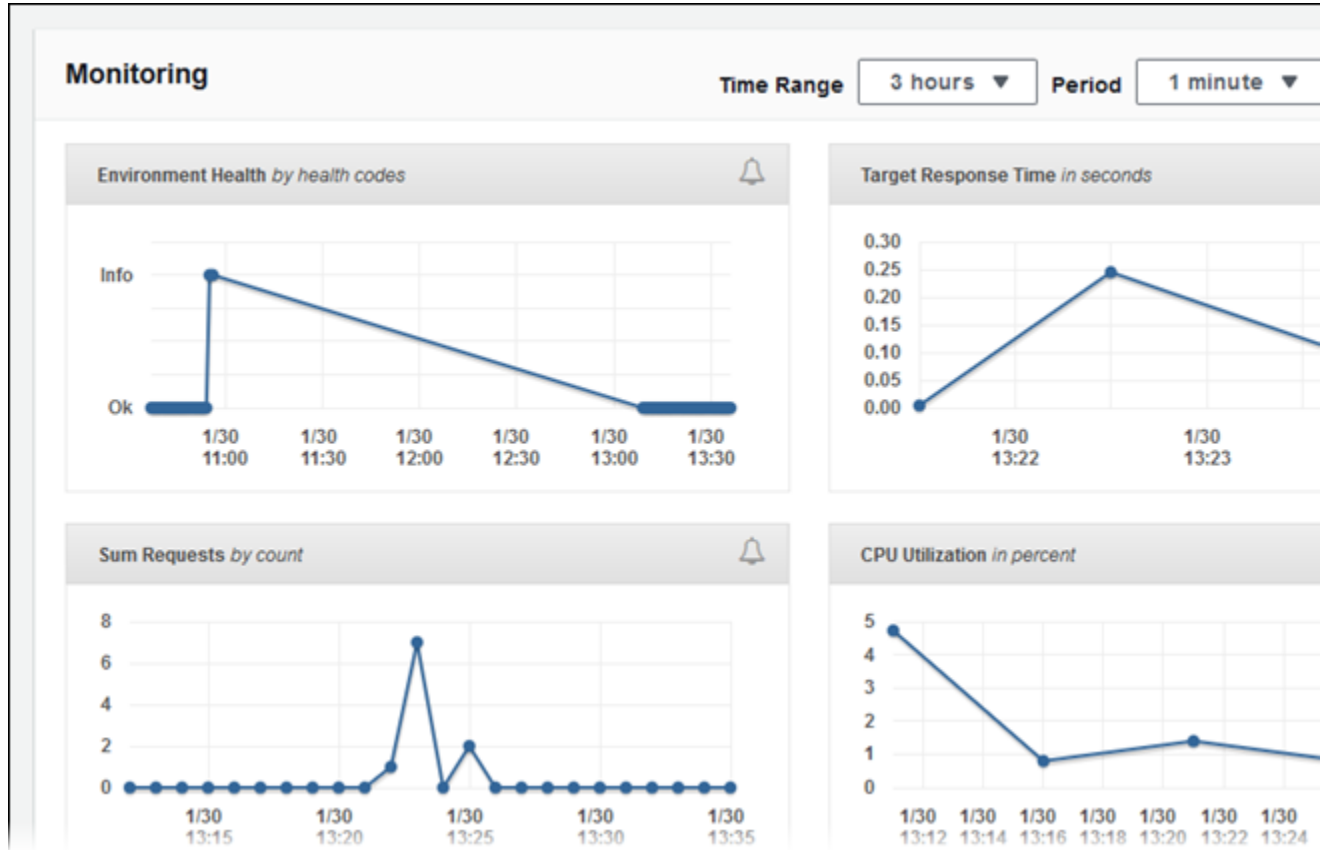
When enhanced health monitoring is enabled, this page shows information about the requests served by the instances in your environment and metrics from the operating system, including latency, load, and CPU utilization.

Instance ID	Status	Running	Deployment ID	Requests/sec	2xx Responses	3xx Responses	4xx Responses	5xx Responses	P99 Latency	P90 Latency
Overall	Ok	N/A	N/A	0.4	100%	0.0%	0.0%	0.0%	0.002	0.00
i-00227807c4c4a1334	Ok	2 hours	3	0.2	2	0	0	0	0.002	0.00
i-03280193ba1ba4171	Ok	19 days	3	0.2	2	0	0	0	0.001	0.00

For more information, see [Enhanced health reporting and monitoring \(p. 691\)](#).

Monitoring

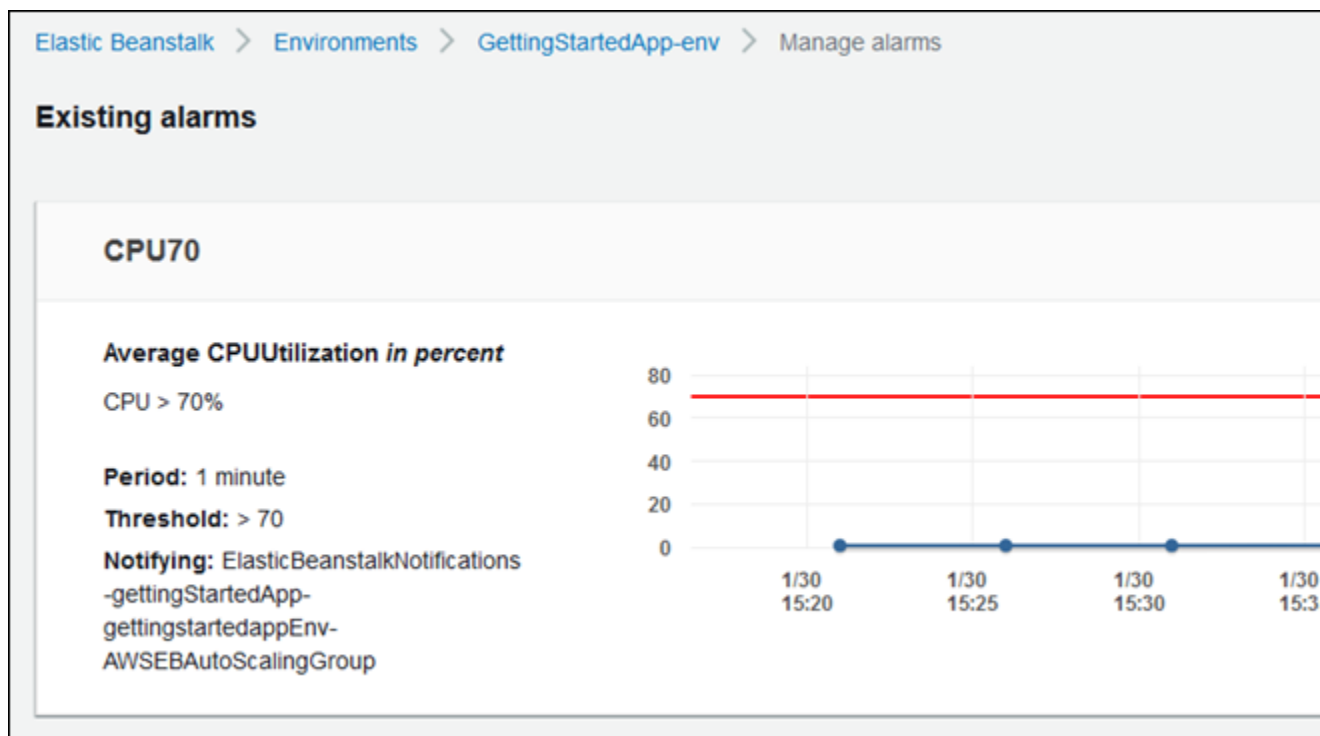
The **Monitoring** page shows an overview of health information for your environment. This includes the default set of metrics provided by Elastic Load Balancing and Amazon EC2, and graphs that show how the environment's health has changed over time. You can use the options on this page to configure additional graphs for resource-specific metrics, and add alarms for any metric supported by the in-use health reporting system.



For more information, see [Monitoring environment health in the AWS management console \(p. 685\)](#).

Alarms

The **Existing alarms** page shows information about any alarms that you have configured for your environment. You can use the options on this page to modify or delete alarms.



For more information, see [Manage alarms](#) (p. 725).

Managed updates

The **Managed updates overview** page shows information about upcoming and completed managed platform updates and instance replacement. These features let you configure your environment to update to the latest platform version automatically during a weekly maintenance window that you choose.


In between platform releases, you can choose to have your environment replace all of its Amazon EC2 instances during the maintenance window. This can help alleviate issues that occur when your application runs for extended periods of time.

For more information, see [Managed platform updates](#) (p. 420).

Events

The **Events** page shows the event stream for your environment. Elastic Beanstalk outputs event messages whenever you interact with the environment, and when any of your environment's resources are created or modified as a result.

Elastic Beanstalk > Environments > GettingStartedApp-env > Events

 **Click the link to be routed to the previous Beanstalk Console**
Switch to the previous console

Events

Severity < 1 2 3 4 5 6 7 .

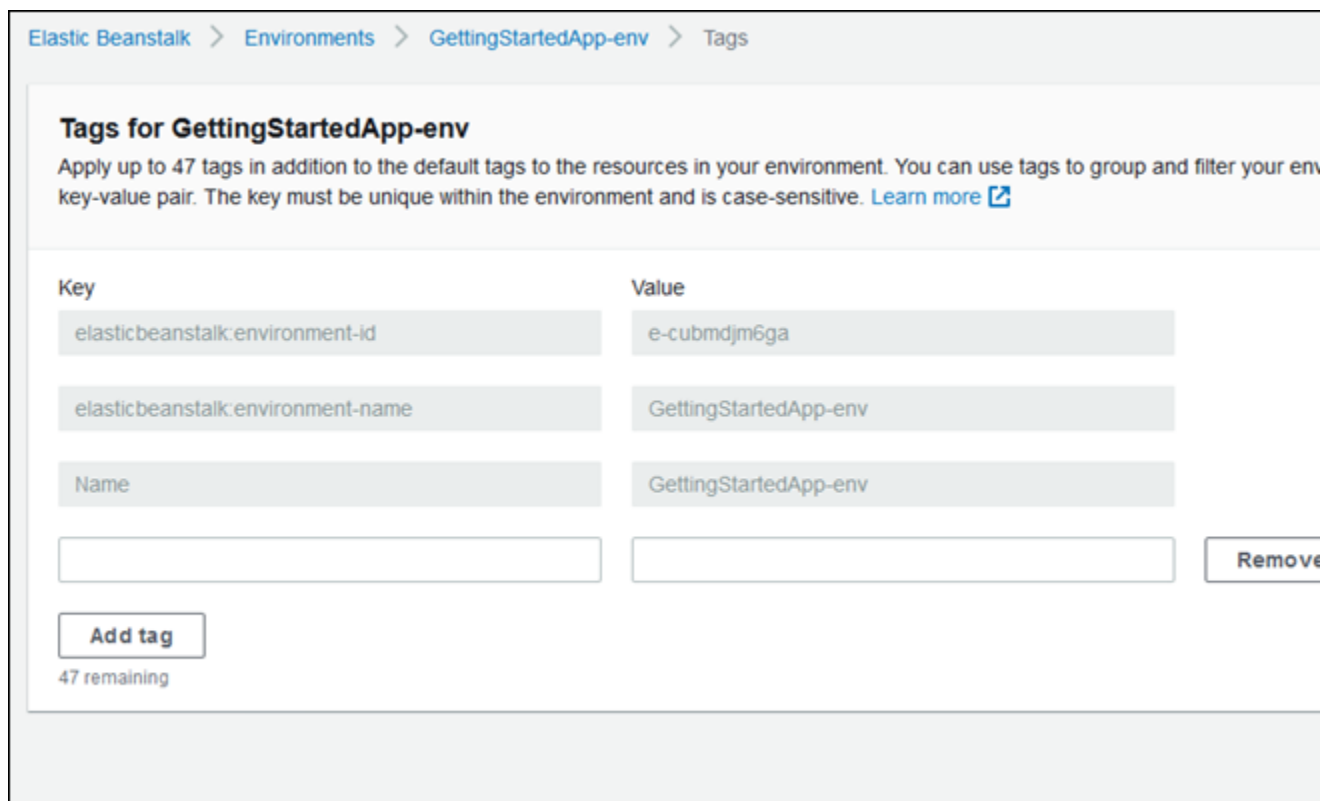
Time	Type	Details
2020-03-09 17:14:06 UTC-0700	INFO	createConfigurationTemplate completed success
2020-03-09 17:14:06 UTC-0700	INFO	createConfigurationTemplate is starting.
2020-03-03 04:16:55 UTC-0800	INFO	Environment health has transitioned from Info to update completed 85 seconds ago and took 15
2020-03-03 04:16:07 UTC-0800	INFO	Environment update completed successfully.
2020-03-03 04:16:07 UTC-0800	INFO	Successfully deployed new configuration to envi

For more information, see [Viewing an Elastic Beanstalk environment's event stream \(p. 728\)](#).

Tags

The **Tags** page shows the tags that Elastic Beanstalk applied to the environment when you created it, and any tags that you added. You can add, edit, and delete custom tags. You can't edit or delete the tags that Elastic Beanstalk applied.

Environment tags are applied to every resource that Elastic Beanstalk creates to support your application.



For more information, see [Tagging resources in your Elastic Beanstalk environments \(p. 513\)](#).

Creating an Elastic Beanstalk environment

An AWS Elastic Beanstalk *environment* is a collection of AWS resources running an application version. You can deploy multiple environments when you need to run multiple versions of an application. For example, you might have development, integration, and production environments.

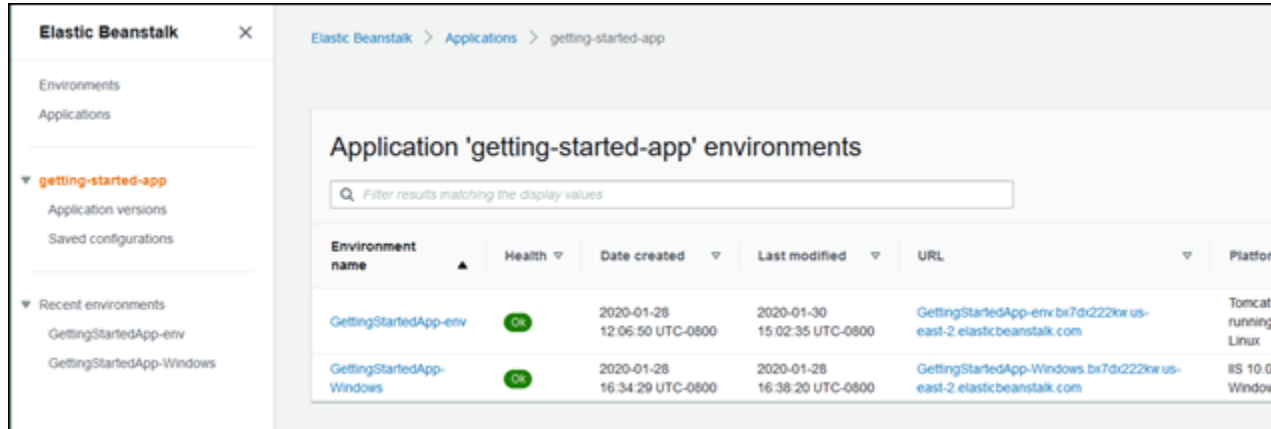
The following procedure launches a new environment running the default application. These steps are simplified to get your environment up and running quickly, using default option values. For detailed instructions with descriptions of the many options you can use to configure the resources that Elastic Beanstalk deploys on your behalf, see [The create new environment wizard \(p. 365\)](#).

Notes

- For instructions on creating and managing environments with the EB CLI, see [Managing Elastic Beanstalk environments with the EB CLI \(p. 864\)](#).
- Creating an environment requires the permissions in the Elastic Beanstalk full access managed policy. See [Elastic Beanstalk user policy \(p. 23\)](#) for details.

To launch an environment with a sample application (console)

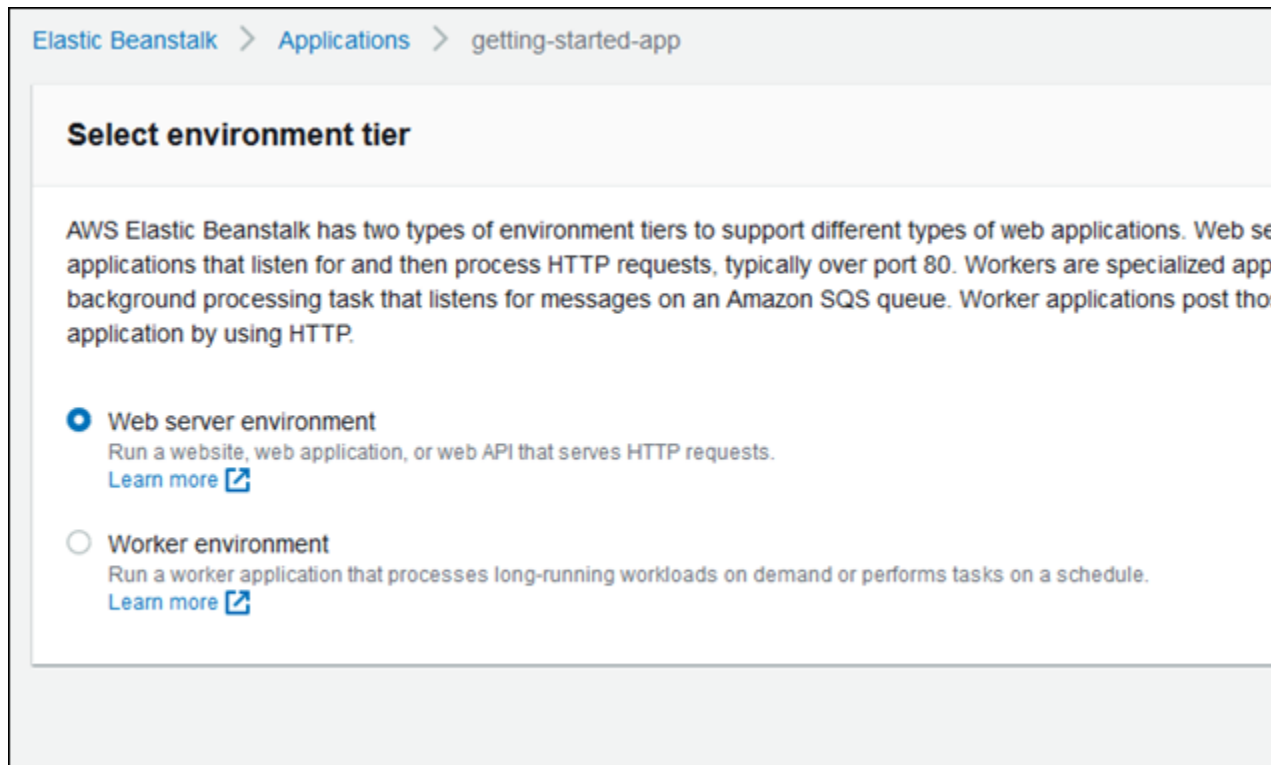
1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose an existing application's name on the list or [create one \(p. 334\)](#).
3. On the application overview page, choose **Create a new environment**.



4. Choose the **Web server environment** or **Worker environment** environment tier (p. 13). You can't change an environment's tier after creation.

Note

The [.NET on Windows Server platform](#) (p. 137) doesn't support the worker environment tier.



5. For **Platform**, select the platform and platform branch that match the language used by your application.

Note

Elastic Beanstalk supports multiple [versions](#) (p. 29) for most of the platforms that are listed. By default, the console selects the recommended version for the platform and platform branch you choose. If your application requires different version, you can select it here, or by choosing **Configure more options**, as described in step 7. For information about supported platform versions, see [the section called "Supported platforms"](#) (p. 29).

6. For **Application code**, choose **Sample application**.

7. To further customize your environment, choose **Configure more options**. You can set the following options only during environment creation:

- Environment name
- Domain name
- Platform version
- VPC
- Tier

You can change the following settings after environment creation, but they require new instances or other resources to be provisioned and can take a long time to apply:

- Instance type, root volume, key pair, and AWS Identity and Access Management (IAM) role
- Internal Amazon RDS database
- Load balancer

For details on all available settings, see [The create new environment wizard \(p. 365\)](#).

8. Choose **Create environment**.

While Elastic Beanstalk creates your environment, you are redirected to the [Elastic Beanstalk console \(p. 353\)](#). When the environment health turns green, choose the URL next to the environment name to view the running application. This URL is generally accessible from the internet unless you configure your environment to use a [custom VPC with an internal load balancer \(p. 380\)](#).

Topics

- [The create new environment wizard \(p. 365\)](#)
- [Clone an Elastic Beanstalk environment \(p. 384\)](#)
- [Terminate an Elastic Beanstalk environment \(p. 386\)](#)
- [Creating Elastic Beanstalk environments with the AWS CLI \(p. 387\)](#)
- [Creating Elastic Beanstalk environments with the API \(p. 388\)](#)
- [Constructing a Launch Now URL \(p. 391\)](#)
- [Creating and updating groups of Elastic Beanstalk environments \(p. 395\)](#)

The create new environment wizard

In [Creating an Elastic Beanstalk environment \(p. 363\)](#) we show how to open the **Create new environment** wizard and quickly create an environment. Choose **Create environment** to launch an environment with a default environment name, automatically generated domain, sample application code, and recommended settings.

This topic describes the **Create new environment** wizard and all the ways you can use it to configure the environment you want to create.

Wizard main page

The **Create New Environment** wizard main page starts with naming information for the new environment. Set the environment's name and subdomain, and create a description for your environment. Be aware that these environment settings cannot change after the environment is created.

Environment information

Choose the name, subdomain, and description for your environment. These cannot be changed later.

Application name
getting-started-app

Environment name
GettingStartedApp-env-1

Domain
Leave blank for autogenerated value .us-east-2.elasticbeanstalk

Check availability

Description

- **Name** – Enter a name for the environment. The form provides a generated name.
- **Domain** – (web server environments) Enter a unique domain name for your environment. The default name is the environment's name. You can enter a different domain name. Elastic Beanstalk uses this name to create a unique CNAME for the environment. To check whether the domain name you want is available, choose **Check Availability**.
- **Description** – Enter a description for this environment.

Select a platform for the new environment

You can create a new environment from two types of platforms:

- Managed platform
- Custom platform

Managed platform

In most cases you use an Elastic Beanstalk managed platform for your new environment. When the new environment wizard starts, it selects the **Managed platform** option by default, as shown in the following screenshot.

Platform

Managed platform
Platforms published and maintained by AWS Elastic Beanstalk. [Learn more](#)

Custom platform
Platforms created and owned by you.

Platform

Platform branch

Platform version

Select a platform, a platform branch within that platform, and a specific platform version in the branch. When you select a platform branch, the recommended version within the branch is selected by default. In addition, you can select any platform version you've used before.

Note

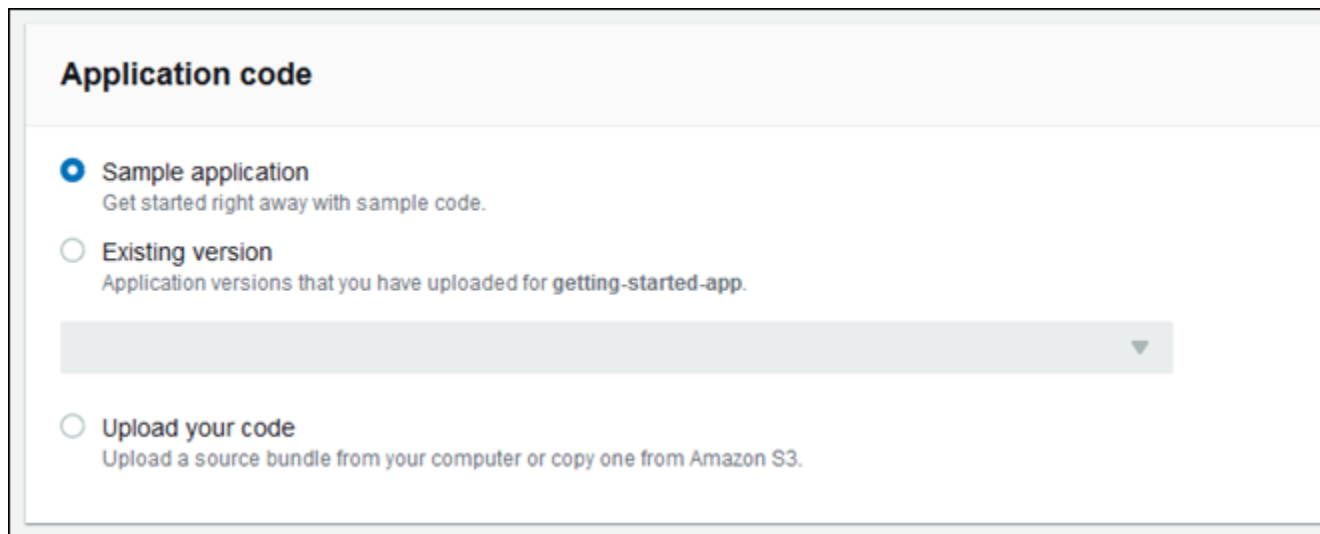
For a production environment, we recommend that you choose a platform version in a supported platform branch. For details about platform branch states, see the *Platform Branch* definition in the [the section called "Platforms glossary" \(p. 25\)](#).

Custom platform

If an off-the-shelf platform doesn't meet your needs, you can create a new environment from a custom platform. To specify a custom platform, choose the **Custom platform** option, and then select one of the available custom platforms. If there are no custom platforms available, this option is dimmed.

Provide application code

Now that you have selected the platform to use, the next step is to provide your application code.



Application code

Sample application
Get started right away with sample code.

Existing version
Application versions that you have uploaded for **getting-started-app**.

Upload your code
Upload a source bundle from your computer or copy one from Amazon S3.

You have several options:

- You can use the sample application that Elastic Beanstalk provides for each platform.
- You can use code that you already deployed to Elastic Beanstalk. Choose **Existing version** and your application in the **Application code** section.
- You can upload new code. Choose **Upload your code**, and then choose **Upload**. You can upload new application code from a local file, or you can specify the URL for the Amazon S3 bucket that contains your application code.

Note

Depending on the platform version you selected, you can upload your application in a ZIP [source bundle](#) (p. 342), a [WAR file](#) (p. 101), or a [plaintext Docker configuration](#) (p. 51). The file size limit is 512 MB.

When you choose to upload new code, you can also provide tags to associate with your new code. For more information about tagging an application version, see [the section called “Tagging application versions”](#) (p. 340).

Application code

- Sample application**
Get started right away with sample code.
- Existing version**
Application versions that you have uploaded for `getting-started-app`.

- Upload your code**
Upload a source bundle from your computer or copy one from Amazon S3.

▼ Source code origin

(Maximum size 512 MB)

- Local file**
- Public S3 URL**

File name : **java-tomcat-v3.zip**

File successfully uploaded

Version label

Unique name for this version of your application code.

▼ Application code tags

Apply up to 50 tags. You can use tags to group and filter your resources. A tag is a key-value pair. The key is within the resource and is case-sensitive. [Learn more](#)

Key

Value

50 remaining

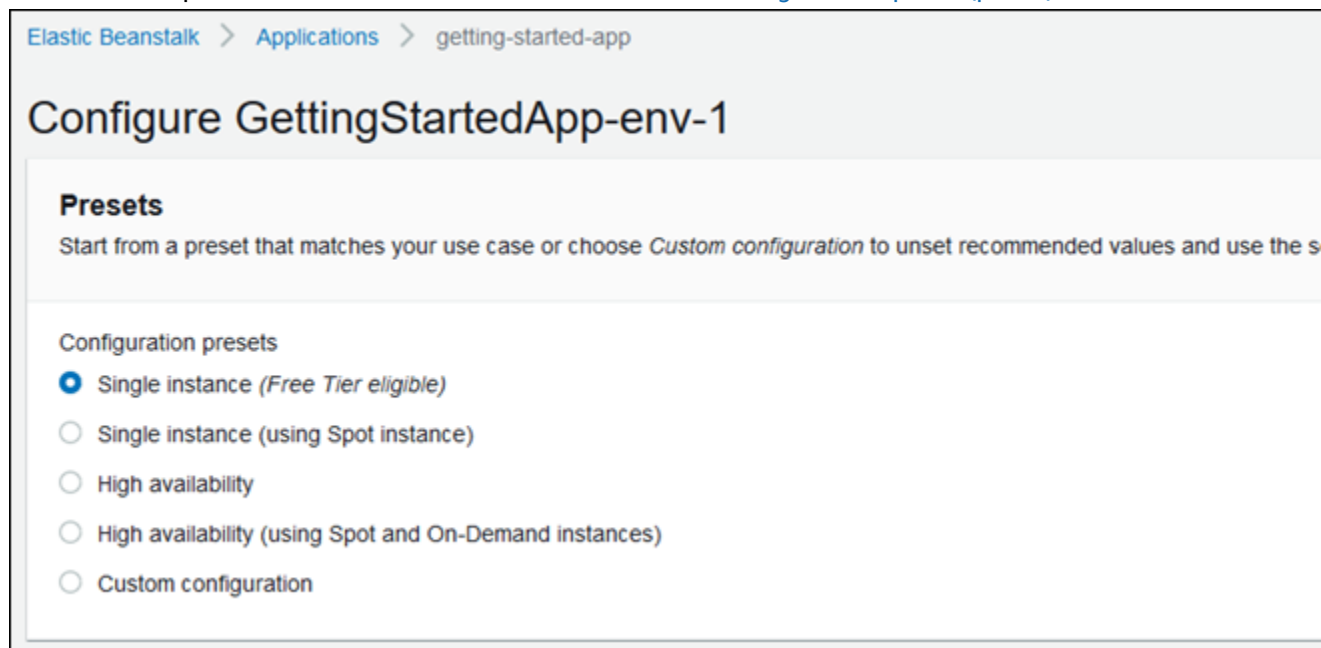
For quick environment creation using default configuration options, you can now choose **Create environment**. Choose **Configure more options** to make additional configuration changes, as described in the following sections.

Wizard configuration page

When you choose **Configure more options**, the wizard shows the **Configure** page. On this page you can select a configuration preset, change the platform version you want your environment to use, or make specific configuration choices for the new environment.

Choose a preset configuration

On the **Presets** section of the page, Elastic Beanstalk provides several configuration presets for different use cases. Each preset includes recommended values for several [configuration options](#) (p. 536).

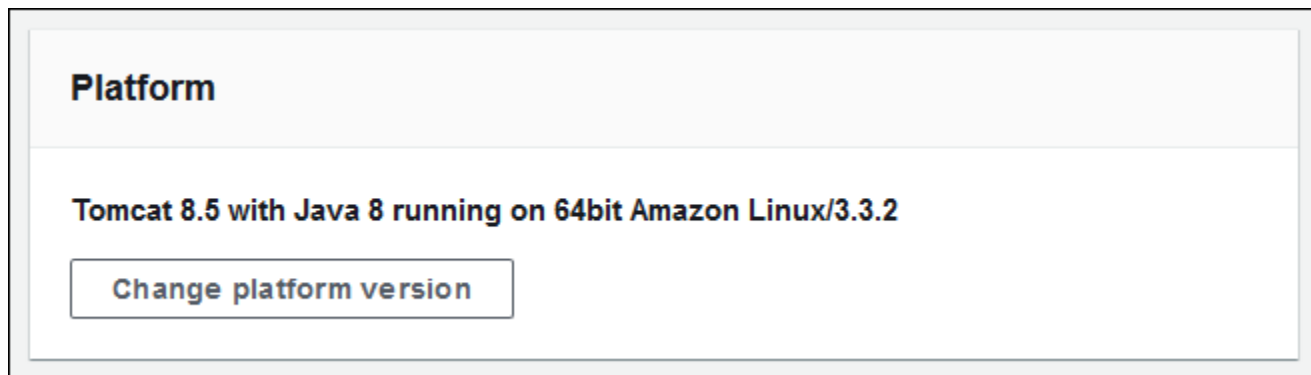


The **High availability** presets include a load balancer, and are recommended for production environments. Choose them if you want an environment that can run multiple instances for high availability and scale in response to load. The **Single instance** presets are primarily recommended for development. Two of the presets enable Spot Instance requests. For details about Elastic Beanstalk capacity configuration, see [Auto Scaling group](#) (p. 457).

The last preset, **Custom configuration**, removes all recommended values except role settings and uses the API defaults. Choose this option if you are deploying a source bundle with [configuration files](#) (p. 600) that set configuration options. **Custom configuration** is also selected automatically if you modify either the **Low cost** or **High availability** configuration presets.

Change the platform version

On the **Platform** section of the page, you can change the platform version that your new environment will use. You can choose the recommended version in any platform branch, or any platform version that you've used in the past.



Customize your configuration

In addition to (or instead of) choosing a configuration preset, you can fine-tune [configuration options](#) (p. 536) in your environment. The **Configure** wizard shows several configuration categories. Each configuration category displays a summary of values for a group of configuration settings. Choose **Edit** to edit this group of settings.

Configuration Categories

- [Software settings](#) (p. 371)
- [Instances](#) (p. 372)
- [Capacity](#) (p. 373)
- [Load balancer](#) (p. 373)
- [Rolling updates and deployments](#) (p. 374)
- [Security](#) (p. 376)
- [Monitoring](#) (p. 377)
- [Managed updates](#) (p. 378)
- [Notifications](#) (p. 379)
- [Network](#) (p. 380)
- [Database](#) (p. 381)
- [Tags](#) (p. 382)
- [Worker environment](#) (p. 383)

Software settings

Use the **Modify software** configuration page to configure the software on the Amazon Elastic Compute Cloud (Amazon EC2) instances that run your application. You can configure environment properties, AWS X-Ray debugging, instance log storing and streaming, and platform-specific settings. For details, see [the section called "Software settings"](#) (p. 516).

Elastic Beanstalk > Applications > getting-started-app

Modify software

The following settings control platform behavior and let you pass key-value pairs in as OS environment variables. [Learn more](#)

Platform options

Target .NET runtime
4.0

Enable 32-bit applications
False

AWS X-Ray

X-Ray daemon

Instances

Use the **Modify instances** configuration page to configure the Amazon EC2 instances that run your application. For details, see [the section called "Amazon EC2 instances" \(p. 451\)](#).

Elastic Beanstalk > Applications > getting-started-app

Modify instances

Amazon CloudWatch monitoring

The time interval between when metrics are reported from the EC2 instances.

Monitoring interval
5 minute

Root volume (boot device)

Root volume type
(Container default)

Capacity

Use the **Modify capacity** configuration page to configure the compute capacity of your environment and **Auto Scaling group** settings to optimize the number and type of instances you're using. You can also change your environment capacity based on triggers or on a schedule.

A load balanced environment can run multiple instances for high availability and prevent downtime during configuration updates and deployments. In a load balanced environment, the domain name maps to the load balancer. In a single-instance environment, it maps to an elastic IP address on the instance.

Warning

A single-instance environment isn't production ready. If the instance becomes unstable during deployment, or Elastic Beanstalk terminates and restarts the instance during a configuration update, your application can be unavailable for a period of time. Use single-instance environments for development, testing, or staging. Use load-balanced environments for production.

For more information about environment capacity settings, see [the section called "Auto Scaling group" \(p. 457\)](#).

Elastic Beanstalk > Applications > getting-started-app

Modify capacity

Configure the compute capacity of your environment and Auto Scaling settings to optimize the number of instances used.

Auto Scaling Group

Environment type
Load balanced

Instances
Min 1
Max 2

Fleet composition
Choose a mix of On-Demand and Spot instances with multiple instance types. Spot instances are automatically launched at the lowest available price.

On-Demand instances
 Combine purchase options and instances

Maximum spot price
The maximum price per instance-hour, in USD, that you're willing to pay for a Spot instance. Setting a custom price limits your chances to fulfill your Spot instances.

Load balancer

Use the **Modify load balancer** configuration page to select a load balancer type and to configure settings for it. In a load balanced environment, your environment's load balancer is the entry point for all traffic headed for your application. Elastic Beanstalk supports several types of load balancer. By default, the Elastic Beanstalk console creates an Application Load Balancer and configures it to serve HTTP traffic on port 80.

Note

You can only select your environment's load balancer type during environment creation.

For more information about load balancer types and settings, see [the section called "Load balancer"](#) (p. 470) and [the section called "HTTPS"](#) (p. 652).

Elastic Beanstalk > Applications > getting-started-app

Modify load balancer

Application Load Balancer

Application layer load balancer—routing HTTP and HTTPS traffic based on protocol, port, and route to environment processes.

Classic Load Balancer

Previous generation — HTTP, HTTPS, and

Network Load Balancer

Ultra-high performance and static IP addresses for your application.

Application Load Balancer

You can specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port using a your environment processes. By default, we've configured your load balancer with a standard web server on port 80.

Action

<input type="checkbox"/>	Port	Protocol	SSL certificate
<input type="checkbox"/>	80	HTTP	--

Processes

For each environment process, you can specify the protocol and port that the load balancer uses to route requests to

Rolling updates and deployments

Use the **Modify rolling updates and deployments** configuration page to configure how Elastic Beanstalk processes application deployments and configuration updates for your environment.

Application deployments happen when you upload an updated application source bundle and deploy it to your environment. For more information about configuring deployments, see [the section called “Deployment options”](#) (p. 400).

The screenshot shows the AWS Elastic Beanstalk console configuration page for 'Modify rolling updates and deployments'. The breadcrumb navigation at the top reads: 'Elastic Beanstalk > Environments > GettingStartedApp-env > Configuration'. The main heading is 'Modify rolling updates and deployments'. Below this is a section titled 'Application deployments' with a sub-heading 'Choose how AWS Elastic Beanstalk propagates source code changes and software configuration updates. [Learn more](#)'. The configuration options are: 'Deployment policy' is a dropdown menu set to 'All at once'; 'Batch size' has two radio buttons, 'Percentage' (selected) and 'Fixed'; below 'Batch size' is a spinner set to '100' followed by '% of instances at a time'; 'Traffic split' has a spinner set to '10' followed by '% to new application version'; and 'Traffic splitting evaluation time' has a spinner set to '5' followed by 'minutes'.

Configuration changes that modify the [launch configuration](#) (p. 556) or [VPC settings](#) (p. 568) require terminating all instances in your environment and replacing them. For more information about setting the update type and other options, see [the section called “Configuration changes”](#) (p. 408).

Configuration updates

Changes to virtual machine settings and VPC configuration trigger rolling updates to replace the instances in your environment. [Learn more](#)

Rolling update type
Rolling based on Health

Batch size
1
The maximum number of instances to replace in each phase of the update.

Minimum capacity
1
The minimum number of instances to keep in service at all times.

Pause time
hh:mm:ss
Pause the update for up to an hour between each batch.

Security

Use the **Modify security** configuration page to configure service and instance security settings.

For a description of Elastic Beanstalk security concepts, see [Permissions](#) (p. 20). For more information about configuring environment security settings, see [the section called "Security"](#) (p. 510).

Elastic Beanstalk > Applications > getting-started-app

Modify security

Service role

Service role

aws-elasticbeanstalk-service-role ▼ ↻

Virtual machine permissions

EC2 key pair

-- Choose a key pair -- ▼ ↻

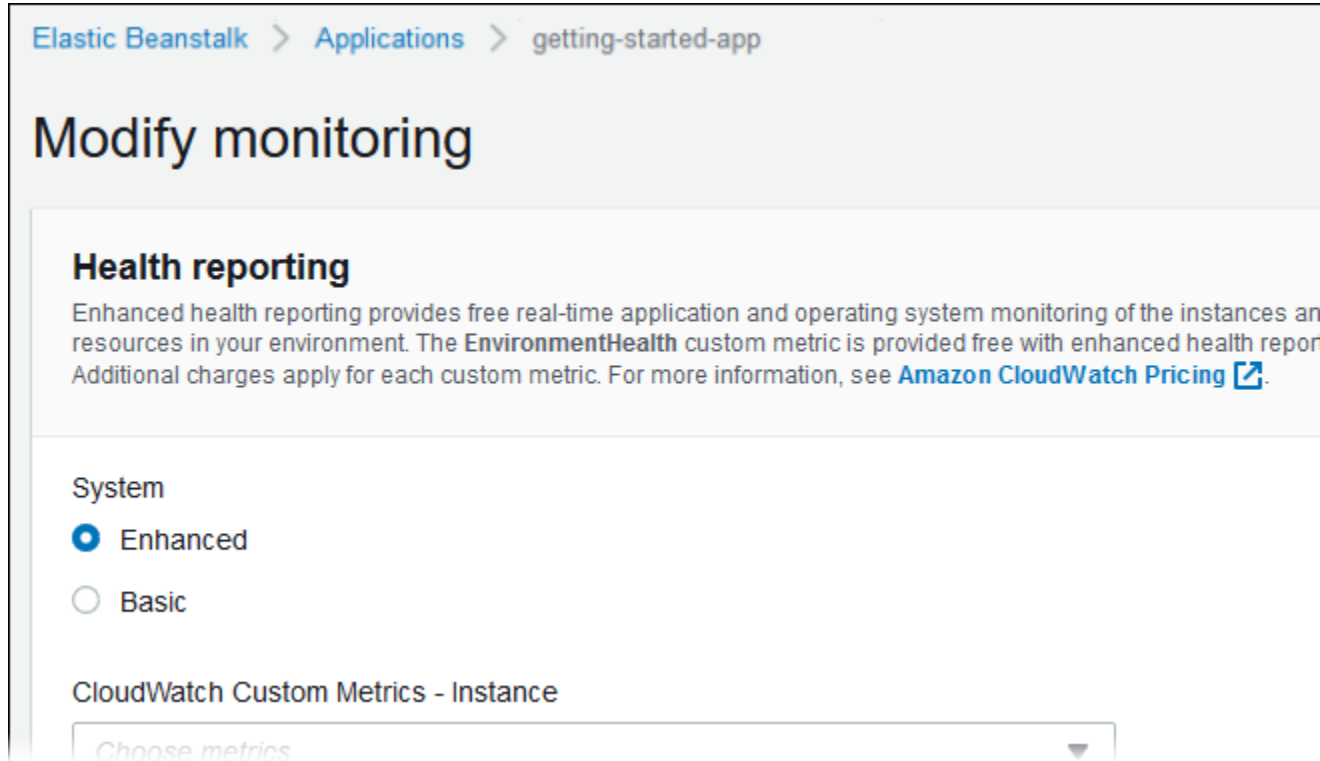
IAM instance profile

aws-elasticbeanstalk-ec2-role ▼ ↻

Cancel

Monitoring

Use the **Modify monitoring** configuration page to configure health reporting, monitoring rules, and health event streaming. For details, see [the section called "Enable enhanced health" \(p. 698\)](#), [the section called "Enhanced health rules" \(p. 710\)](#), and [the section called "Streaming environment health" \(p. 750\)](#).



Managed updates

Use the **Modify managed updates** configuration page to configure managed platform updates. You can decide if you want them enabled, set the schedule, and configure other properties. For details, see [the section called "Managed updates" \(p. 420\)](#).

Elastic Beanstalk > Applications > getting-started-app

Modify managed updates

Managed platform updates

Enable managed platform updates to apply platform updates automatically during a weekly maintenance window that your application stays available during the update process.

Managed updates
 Enabled

Weekly update window
Tuesday at 12 : 00 UTC
Any available managed updates will run between Tuesday, 4:00 AM and Tuesday, 6:00 AM (-0800 GMT).

Update level
Minor and patch

Instance replacement
If enabled, an instance replacement will be scheduled if no other updates are available.
 Enabled

Notifications

Use the **Modify notifications** configuration page to specify an email address to receive [email notifications](#) (p. 526) for important events from your environment.

Elastic Beanstalk > Applications > getting-started-app

Modify notifications

Email notifications

Enter an email address to receive email notifications for important events from your environment. [Learn more](#)

Email

Network


If you have created a [custom VPC \(p. 530\)](#), the **Modify network** configuration page to configure your environment to use it. If you don't choose a VPC, Elastic Beanstalk uses the default VPC and subnets.

Elastic Beanstalk > Applications > getting-started-app

Modify network

Virtual private cloud (VPC)

VPC
Launch your environment in a custom VPC instead of the default VPC. You can create a VPC and subnets in the VPC management console.


vpc-0f9c96ae77f3c49c1 (172.31.0.0/16) | private-public 

[Create custom VPC](#)

Load balancer settings

Assign your load balancer to a subnet in each Availability Zone (AZ) in which your application runs. For a publicly accessible application, choose **Public** and choose public subnets.

Visibility
Make your load balancer internal if your application serves requests only from connected VPCs. Public load balancers serve requests from the Internet.

Public 

Load balancer subnets

Database

Use the **Modify database** configuration page to add an Amazon Relational Database Service (Amazon RDS) database to your environment for development and testing. Elastic Beanstalk provides connection information to your instances by setting environment properties for the database hostname, user name, password, table name, and port.

For details, see [the section called “Database” \(p. 506\)](#).

Elastic Beanstalk > Applications > getting-started-app



Modify database

Add an Amazon RDS SQL database to your environment for development and testing. AWS Elastic Beanstalk provides information to your instances by setting environment properties for the database hostname, username, password, and so on. When you add a database to your environment, its lifecycle is tied to your environment's. For production environments, you can configure your instances to connect to a database. [Learn more](#)

Restore a snapshot

Restore an existing snapshot in your account, or create a new database.


Snapshot

None  

Database settings

Choose an engine and instance type for your environment's database.

Engine

mysql 

Engine version

Tags

Use the **Modify tags** configuration page to add [tags](#) to the resources in your environment. For more information about environment tagging, see [Tagging resources in your Elastic Beanstalk environments](#) (p. 513).

Elastic Beanstalk > Applications > getting-started-app

Modify tags

Apply up to 50 tags to the resources in your environment in addition to the default tags.

Key	Value	
<input type="text" value="mytag1"/>	<input type="text" value="value1"/>	<input type="button" value="Remove"/>

49 remaining

Worker environment

If you're creating a *worker tier environment*, use the **Modify worker** configuration page to configure the worker environment. The worker daemon on the instances in your environment pulls items from an Amazon Simple Queue Service (Amazon SQS) queue and relays them as post messages to your worker application. You can choose the Amazon SQS queue that the worker daemon reads from (auto-generated or existing). You can also configure the messages that the worker daemon sends to your application.

For more information, see [the section called "Worker environments" \(p. 437\)](#).

Elastic Beanstalk > Applications > getting-started-app

Modify worker

You can create a new Amazon SQS queue for your worker application or pull work items from an existing queue. The instances in your environment pulls an item from the queue and relays it in the body of a POST request to a local HTTP endpoint on localhost.

Queue

Worker queue

SQS queue from which to read work items.

Messages

HTTP path

Clone an Elastic Beanstalk environment

You can use an existing Elastic Beanstalk environment as the basis for a new environment by cloning the existing environment. For example, you might want to create a clone so that you can use a newer version of the solution stack used by the original environment's platform. Elastic Beanstalk configures the clone with the same environment settings used by the original environment. By cloning an existing environment instead of creating a new environment, you don't have to manually configure option settings, environment variables, and other settings. Elastic Beanstalk also creates a copy of any AWS resource associated with the original environment. However, during the cloning process, Elastic Beanstalk doesn't copy data from Amazon RDS to the clone. After you create the clone environment, you can modify environment configuration settings as needed.

Note

Elastic Beanstalk doesn't include any unmanaged changes to resources in the clone. Changes to AWS resources that you make using tools other than the Elastic Beanstalk console, command-line tools, or API are considered unmanaged changes.

AWS management console

To clone an environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Environment actions**, and then do one of the following:
 - Choose **Clone environment** to clone the environment without any changes to the solution stack version.
 - Choose **Clone with latest platform** to clone the environment, but with a newer version of the original environment's solution stack.
4. On the **Clone environment** page, review the information in the **Original Environment** section to verify that you chose the environment from which you want to create a clone.
5. In the **New Environment** section, you can optionally change the **Environment name**, **Environment URL**, **Description**, **Platform version**, and **Service role** values that Elastic Beanstalk automatically set based on the original environment.

Note

If the platform version used in the original environment isn't the one recommended for use in the platform branch, you are warned that a different platform version is recommended. Choose **Platform version**, and you can see the recommended platform version on the list—for example, **3.3.2 (Recommended)**.

Elastic Beanstalk > Environments > GettingStartedApp-env > Clone environment

Clone environment

You can launch a new environment based on an existing environment's configuration settings while optionally choosing a version for the new environment. [Learn more](#)

Original environment

Environment name

GettingStartedApp-env

Environment URL

GettingStartedApp-env.gap8pzvmti.us-east-2.elasticbeanstalk.com

Platform

Tomcat 8.5 with Java 8 running on 64bit Amazon Linux/3.3.1

New environment

Environment name

GettingStartedApp-env-1

Environment URL

gettingstartedapp-env-1

.us-east-2.elasticbeanstal

Check availability

Description

Clone of GettingStartedApp-env

Platform branch

Tomcat 8.5 with Java 8 running on 64bit Amazon Linux

Platform version

3.3.1



Warning

A different platform version is recommended.

6. When you are ready, choose **Clone**.

Elastic Beanstalk command line interface (EB CLI)

Use the **eb clone** command to clone a running environment, as follows.

```
~/workspace/my-app$ eb clone my-env1
Enter name for Environment Clone
(default is my-env1-clone): my-env2
Enter DNS CNAME prefix
(default is my-env1-clone): my-env2
```

You can specify the name of the source environment in the clone command, or leave it out to clone the default environment for the current project folder. The EB CLI prompts you to enter a name and DNS prefix for the new environment.

By default, **eb clone** creates the new environment with the latest available version of the source environment's platform. To force the EB CLI to use the same version, even if there is a newer version available, use the **--exact** option.

```
~/workspace/my-app$ eb clone --exact
```

For more information about this command, see [eb clone \(p. 888\)](#).

Terminate an Elastic Beanstalk environment

You can terminate a running AWS Elastic Beanstalk environment using the Elastic Beanstalk console to avoid incurring charges for unused AWS resources. For more information about terminating an environment using the AWS Toolkit for Eclipse, see [Terminating an environment \(p. 137\)](#).

Note

You can always launch a new environment using the same version later. If you have data from an environment that you want to preserve, create a snapshot of your current database instance before you terminate the environment. You can use it later as the basis for new DB instance when you create a new environment. For more information, see [Creating a DB Snapshot](#) in the [Amazon Relational Database Service User Guide](#).

Elastic Beanstalk might fail to terminate your environment. One common reason for this failure is that another environment's security group has a dependency on the security group of the environment you're trying to terminate. A way to avoid this condition is described in [Security groups \(p. 455\)](#) on the *EC2 Instances* page of this guide.

Elastic Beanstalk console

To terminate an environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

Note

When you terminate your environment, the CNAME associated with the terminated environment becomes available for anyone to use.

It takes a few minutes for Elastic Beanstalk to terminate the AWS resources running in the environment.

AWS CLI

To terminate an environment

- Run the following command.

```
$ aws elasticbeanstalk terminate-environment --environment-name my-env
```

API

To terminate an environment

- Call `TerminateEnvironment` with the following parameter:

```
EnvironmentName = SampleAppEnv
```

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv  
&Operation=TerminateEnvironment  
&AuthParams
```

Creating Elastic Beanstalk environments with the AWS CLI

1. Check if the CNAME for the environment is available.

```
$ aws elasticbeanstalk check-dns-availability --cname-prefix my-cname  
{  
  "Available": true,  
  "FullyQualifiedCNAME": "my-cname.elasticbeanstalk.com"  
}
```

2. Make sure your application version exists.

```
$ aws elasticbeanstalk describe-application-versions --application-name my-app --  
version-label v1
```

If you don't have an application version for your source yet, create it. For example, the following command creates an application version from a source bundle in Amazon Simple Storage Service (Amazon S3).

```
$ aws elasticbeanstalk create-application-version --application-name my-app --version-  
label v1 --source-bundle S3Bucket=my-bucket,S3Key=my-source-bundle.zip
```

3. Create a configuration template for the application.

```
$ aws elasticbeanstalk create-configuration-template --application-name my-app --  
template-name v1 --solution-stack-name "64bit Amazon Linux 2015.03 v2.0.0 running Ruby  
2.2 (Passenger Standalone)"
```

4. Create environment.

```
$ aws elasticbeanstalk create-environment --cname-prefix my-cname --application-  
name my-app --template-name v1 --version-label v1 --environment-name v1clone --option-  
settings file://options.txt
```

Option Settings are defined in the **options.txt** file:

```
[  
  {  
    "Namespace": "aws:autoscaling:launchconfiguration",  
    "OptionName": "IamInstanceProfile",  
    "Value": "aws-elasticbeanstalk-ec2-role"  
  }  
]
```

The above option setting defines the IAM instance profile. You can specify the ARN or the profile name.

5. Determine if the new environment is Green and Ready.

```
$ aws elasticbeanstalk describe-environments --environment-names my-env
```

If the new environment does not come up Green and Ready, you should decide if you want to retry the operation or leave the environment in its current state for investigation. Make sure to terminate the environment after you are finished, and clean up any unused resources.

Note

You can adjust the timeout period if the environment doesn't launch in a reasonable time.

Creating Elastic Beanstalk environments with the API

1. Call CheckDNSAvailability with the following parameter:

- CNAMEPrefix = SampleApp

Example

```
https://elasticbeanstalk.us-east-2.amazonaws.com/?CNAMEPrefix=sampleapplication  
&Operation=CheckDNSAvailability  
&AuthParams
```

2. Call DescribeApplicationVersions with the following parameters:

- ApplicationName = SampleApp
- VersionLabel = Version2

Example

```
https://elasticbeanstalk.us-east-2.amazonaws.com/?ApplicationName=SampleApp
```

```
&VersionLabel=Version2
&Operation=DescribeApplicationVersions
&AuthParams
```

3. Call `CreateConfigurationTemplate` with the following parameters:

- `ApplicationName` = SampleApp
- `TemplateName` = MyConfigTemplate
- `SolutionStackName` = 64bit%20Amazon%20Linux%202015.03%20v2.0.0%20running%20Ruby%202.2%20(Passenger%20Standalone)

Example

```
https://elasticbeanstalk.us-east-2.amazonaws.com/?ApplicationName=SampleApp
&TemplateName=MyConfigTemplate
&Operation=CreateConfigurationTemplate
&SolutionStackName=64bit%20Amazon%20Linux%202015.03%20v2.0.0%20running%20Ruby
%202.2%20(Passenger%20Standalone)
&AuthParams
```

4. Call `CreateEnvironment` with one of the following sets of parameters.

a. Use the following for a web server environment tier:

- `EnvironmentName` = SampleAppEnv2
- `VersionLabel` = Version2
- `Description` = description
- `TemplateName` = MyConfigTemplate
- `ApplicationName` = SampleApp
- `CNAMEPrefix` = sampleapplication
- `OptionSettings.member.1.Namespace` = aws:autoscaling:launchconfiguration
- `OptionSettings.member.1.OptionName` = IamInstanceProfile
- `OptionSettings.member.1.Value` = aws-elasticbeanstalk-ec2-role

Example

```
https://elasticbeanstalk.us-east-2.amazonaws.com/?ApplicationName=SampleApp
&VersionLabel=Version2
&EnvironmentName=SampleAppEnv2
&TemplateName=MyConfigTemplate
&CNAMEPrefix=sampleapplication
&Description=description
&Operation=CreateEnvironment
&OptionSettings.member.1.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.1.OptionName=IamInstanceProfile
&OptionSettings.member.1.Value=aws-elasticbeanstalk-ec2-role
&AuthParams
```

b. Use the following for a worker environment tier:

- `EnvironmentName` = SampleAppEnv2
- `VersionLabel` = Version2
- `Description` = description
- `TemplateName` = MyConfigTemplate
- `ApplicationName` = SampleApp

- Tier = Worker
- OptionSettings.member.1.Namespace = aws:autoscaling:launchconfiguration
- OptionSettings.member.1.OptionName = IamInstanceProfile
- OptionSettings.member.1.Value = aws-elasticbeanstalk-ec2-role
- OptionSettings.member.2.Namespace = aws:elasticbeanstalk:sqs
- OptionSettings.member.2.OptionName = WorkerQueueURL
- OptionSettings.member.2.Value = sqsd.elasticbeanstalk.us-east-2.amazonaws.com
- OptionSettings.member.3.Namespace = aws:elasticbeanstalk:sqs
- OptionSettings.member.3.OptionName = HttpPath
- OptionSettings.member.3.Value = /
- OptionSettings.member.4.Namespace = aws:elasticbeanstalk:sqs
- OptionSettings.member.4.OptionName = MimeType
- OptionSettings.member.4.Value = application/json
- OptionSettings.member.5.Namespace = aws:elasticbeanstalk:sqs
- OptionSettings.member.5.OptionName = HttpConnections
- OptionSettings.member.5.Value = 75
- OptionSettings.member.6.Namespace = aws:elasticbeanstalk:sqs
- OptionSettings.member.6.OptionName = ConnectTimeout
- OptionSettings.member.6.Value = 10
- OptionSettings.member.7.Namespace = aws:elasticbeanstalk:sqs
- OptionSettings.member.7.OptionName = InactivityTimeout
- OptionSettings.member.7.Value = 10
- OptionSettings.member.8.Namespace = aws:elasticbeanstalk:sqs
- OptionSettings.member.8.OptionName = VisibilityTimeout
- OptionSettings.member.8.Value = 60
- OptionSettings.member.9.Namespace = aws:elasticbeanstalk:sqs
- OptionSettings.member.9.OptionName = RetentionPeriod
- OptionSettings.member.9.Value = 345600

Example

```
https://elasticbeanstalk.us-east-2.amazonaws.com/?ApplicationName=SampleApp
&VersionLabel=Version2
&EnvironmentName=SampleAppEnv2
&TemplateName=MyConfigTemplate
&Description=description
&Tier=Worker
&Operation=CreateEnvironment
&OptionSettings.member.1.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.1.OptionName=IamInstanceProfile
&OptionSettings.member.1.Value=aws-elasticbeanstalk-ec2-role
&OptionSettings.member.2.Namespace=aws%3Aelasticbeanstalk%3Asqs
&OptionSettings.member.2.OptionName=WorkerQueueURL
&OptionSettings.member.2.Value=sqsd.elasticbeanstalk.us-east-2.amazonaws.com
&OptionSettings.member.3.Namespace=aws%3Aelasticbeanstalk%3Asqs
&OptionSettings.member.3.OptionName=HttpPath
&OptionSettings.member.3.Value=%2F
&OptionSettings.member.4.Namespace=aws%3Aelasticbeanstalk%3Asqs
&OptionSettings.member.4.OptionName=MimeType
&OptionSettings.member.4.Value=application%2Fjson
```

```
&OptionSettings.member.5.Namespace=aws%3Aelasticbeanstalk%3Asqs  
&OptionSettings.member.5.OptionName=HttpConnections  
&OptionSettings.member.5.Value=75  
&OptionSettings.member.6.Namespace=aws%3Aelasticbeanstalk%3Asqs  
&OptionSettings.member.6.OptionName=ConnectTimeout  
&OptionSettings.member.6.Value=10  
&OptionSettings.member.7.Namespace=aws%3Aelasticbeanstalk%3Asqs  
&OptionSettings.member.7.OptionName=InactivityTimeout  
&OptionSettings.member.7.Value=10  
&OptionSettings.member.8.Namespace=aws%3Aelasticbeanstalk%3Asqs  
&OptionSettings.member.8.OptionName=VisibilityTimeout  
&OptionSettings.member.8.Value=60  
&OptionSettings.member.9.Namespace=aws%3Aelasticbeanstalk%3Asqs  
&OptionSettings.member.9.OptionName=RetentionPeriod  
&OptionSettings.member.9.Value=345600  
&AuthParams
```

Constructing a Launch Now URL

You can construct a custom uniform resource locator (URL) so that anyone can quickly deploy and run a predetermined web application in AWS Elastic Beanstalk. This URL is called a Launch Now URL. You might need a Launch Now URL, for example, to demonstrate a web application that is built to run on Elastic Beanstalk. With a Launch Now URL, you can use parameters to add the required information to the Create Application wizard in advance. When you do, anyone can use the URL link to launch an Elastic Beanstalk environment with your web application source in just a few steps. This means users don't need to manually upload or specify the location of the application source bundle or provide any additional input to the wizard.

A Launch Now URL gives Elastic Beanstalk the minimum information required to create an application: the application name, solution stack, instance type, and environment type. Elastic Beanstalk uses default values for other configuration details that are not explicitly specified in your custom Launch Now URL.

A Launch Now URL uses standard URL syntax. For more information, see [RFC 3986 - Uniform Resource Identifier \(URI\): Generic Syntax](#).

URL parameters

The URL must contain the following parameters, which are case-sensitive:

- **region** – Specify an AWS Region. For a list of regions supported by Elastic Beanstalk, see [AWS Elastic Beanstalk Endpoints and Quotas](#) in the *AWS General Reference*.
- **applicationName** – Specify the name of your application. Elastic Beanstalk displays the application name in the Elastic Beanstalk console to distinguish it from other applications. By default, the application name also forms the basis of the environment name and environment URL.
- **platform** – Specify the platform version to use for the environment. Use one of the following methods, then URL-encode your choice:
 - Specify a platform ARN without a version. Elastic Beanstalk selects the latest platform version of the corresponding platform major version. For example, to select the latest Python 3.6 platform version, specify:

Python 3.6 running on 64bit Amazon Linux
 - Specify the platform name. Elastic Beanstalk selects the latest version of the platform's latest language runtime. For example:

Python

For a description of all available platforms and their versions, see [Elastic Beanstalk supported platforms \(p. 29\)](#).

You can use the [AWS Command Line Interface \(AWS CLI\)](#) to get a list of available platform versions with their respective ARNs. The `list-platform-versions` command lists detailed information about all available platform versions. The `--filters` argument allows you to scope down the list. For example, you can list all platform versions of a specific language.

The following example queries for all Python platform versions, and pipes the output through a series of commands. The result is a list of platform version ARNs (without the `/version` tail), in human-readable format, with no URL encoding.

```
$ aws elasticbeanstalk list-platform-versions --filters
  'Type="PlatformName",Operator="contains",Values="Python"' | grep PlatformArn | awk -F
  '""' '{print $4}' | awk -F '/' '{print $2}'
Preconfigured Docker - Python 3.4 running on 64bit Debian
Preconfigured Docker - Python 3.4 running on 64bit Debian
Python 2.6 running on 32bit Amazon Linux
Python 2.6 running on 32bit Amazon Linux 2014.03
...
Python 3.6 running on 64bit Amazon Linux
```

The following example adds a Perl command to the last example, to URL-encode the output.

```
$ aws elasticbeanstalk list-platform-versions --filters
  'Type="PlatformName",Operator="contains",Values="Python"' | grep PlatformArn | awk
  -F '""' '{print $4}' | awk -F '/' '{print $2}' | perl -MURI::Escape -ne 'chomp;print
  uri_escape($_),"\n"'
Preconfigured%20Docker%20-%20Python%203.4%20running%20on%2064bit%20Debian
Preconfigured%20Docker%20-%20Python%203.4%20running%20on%2064bit%20Debian
Python%202.6%20running%20on%2032bit%20Amazon%20Linux
Python%202.6%20running%20on%2032bit%20Amazon%20Linux%202014.03
...
Python%203.6%20running%20on%2064bit%20Amazon%20Linux
```

A Launch Now URL can optionally contain the following parameters. If you don't include the optional parameters in your Launch Now URL, Elastic Beanstalk uses default values to create and run your application. When you don't include the `sourceBundleUrl` parameter, Elastic Beanstalk uses the default sample application for the specified **platform**.

- **sourceBundleUrl** – Specify the location of your web application source bundle in URL format. For example, if you uploaded your source bundle to an Amazon S3 bucket, you might specify the value of the `sourceBundleUrl` parameter as `https://mybucket.s3.amazonaws.com/myobject`.

Note

You can specify the value of the `sourceBundleUrl` parameter as an HTTP URL, but the user's web browser will convert characters as needed by applying HTML URL encoding.

- **environmentType** – Specify whether the environment is load balancing and automatically scaling or just a single instance. For more information, see [Environment types \(p. 435\)](#). You can specify either `LoadBalancing` or `SingleInstance` as the parameter value.
- **tierName** – Specify whether the environment supports a web application that processes web requests or a web application that runs background jobs. For more information, see [Elastic Beanstalk worker environments \(p. 437\)](#). You can specify either `WebServer` or `Worker`,
- **instanceType** – Specify a server with the characteristics (including memory size and CPU power) that are most appropriate to your application. To see the instance types that are available in your Elastic

Beanstalk region, see [InstanceType \(p. 558\)](#) in [Configuration options \(p. 536\)](#). To see the detailed specifications for each Amazon EC2 instance type, see [Instance Types](#).

- **withVpc** – Specify whether to create the environment in an Amazon VPC. You can specify either `true` or `false`. For more information about using Elastic Beanstalk with Amazon VPC, see [Using Elastic Beanstalk with Amazon VPC \(p. 834\)](#).
- **withRds** – Specify whether to create an Amazon RDS database instance with this environment. For more information, see [Using Elastic Beanstalk with Amazon RDS \(p. 820\)](#). You can specify either `true` or `false`.
- **rdsDBEngine** – Specify the database engine that you want to use for your Amazon EC2 instances in this environment. You can specify `mysql`, `oracle-se1`, `sqlserver-ex`, `sqlserver-web`, or `sqlserver-se`. The default value is `mysql`.
- **rdsDBAllocatedStorage** – Specify the allocated database storage size in gigabytes. You can specify the following values:
 - **MySQL** – 5 to 1024. The default is 5.
 - **Oracle** – 10 to 1024. The default is 10.
 - **Microsoft SQL Server Express Edition** – 30.
 - **Microsoft SQL Server Web Edition** – 30.
 - **Microsoft SQL Server Standard Edition** – 200.
- **rdsDBInstanceClass** – Specify the database instance type. The default value is `db.t2.micro` (`db.m1.large` for an environment not running in an Amazon VPC). For a list of database instance classes supported by Amazon RDS, see [DB Instance Class](#) in the *Amazon Relational Database Service User Guide*.
- **rdsMultiAZDatabase** – Specify whether Elastic Beanstalk needs to create the database instance across multiple Availability Zones. You can specify either `true` or `false`. For more information about multiple Availability Zone deployments with Amazon RDS, go to [Regions and Availability Zones](#) in the *Amazon Relational Database Service User Guide*.
- **rdsDBDeletionPolicy** – Specify whether to delete or snapshot the database instance on environment termination. You can specify either `Delete` or `Snapshot`.

Example

The following is an example Launch Now URL. After you construct your own, you can give it to your users. For example, you might want to embed the URL on a webpage or in training materials. When users create an application using the Launch Now URL, the Elastic Beanstalk Create an Application wizard requires no additional input.

```
https://console.aws.amazon.com/elasticbeanstalk/home?region=us-west-2#/newApplication?applicationName=YourCompanySampleApp&platform=PHP%207.3%20running%20on%2064bit%20Amazon%20Linux&sourceBundleUrl=http://s3.amazonaws.com/mybucket/myobject&environmentType=SingleInstance&tierName=WebServer&instanceType=m1.small&withVpc=true&withRds=true
```

When users choose a Launch Now URL, Elastic Beanstalk displays a page similar to the following.



Create a web app

Create a new application and environment with a sample application or your own code. By creating an environment, you create an AWS Elastic Beanstalk to manage AWS resources and permissions on your behalf. [Learn more](#)

Application information

Application name

Up to 100 Unicode characters, not including forward slash (/).

Environment information

Choose the name, subdomain, and description for your environment. These cannot be changed later.

Environment name

Domain

Description

Base configuration

Tier Web Server ([Choose tier](#))

Platform Preconfigured platform

Platforms published and maintained by AWS Elastic Beanstalk.

Custom platform **NEW**

Platforms created and owned by you. [Learn more](#)

Application code Sample application

Get started right away with sample code.

Upload your code

Upload a source bundle from your computer or copy one from Amazon S3.

394

 ZIP or WAR

To use the Launch Now URL

1. Choose the Launch Now URL.
2. When the Elastic Beanstalk console opens, on the **Create a web app** page, choose **Review and launch** to view the settings Elastic Beanstalk will use to create the application and launch the environment in which the application runs.
3. On the **Configure** page, choose **Create app** to create the application.

Creating and updating groups of Elastic Beanstalk environments

With the AWS Elastic Beanstalk `Compose Environments` API, you can create and update groups of Elastic Beanstalk environments within a single application. Each environment in the group can run a separate component of a service-oriented architecture application. The `Compose Environments` API takes a list of application versions and an optional group name. Elastic Beanstalk creates an environment for each application version, or, if the environments already exist, deploys the application versions to them.

Create links between Elastic Beanstalk environments to designate one environment as a dependency of another. When you create a group of environments with the `Compose Environments` API, Elastic Beanstalk creates dependent environments only after their dependencies are up and running. For more information on environment links, see [Creating links between Elastic Beanstalk environments \(p. 444\)](#).

The `Compose Environments` API uses an [environment manifest \(p. 645\)](#) to store configuration details that are shared by groups of environments. Each component application must have an `env.yaml` configuration file in its application source bundle that specifies the parameters used to create its environment.

`Compose Environments` requires the `EnvironmentName` and `SolutionStack` to be specified in the environment manifest for each component application.

You can use the `Compose Environments` API with the Elastic Beanstalk command line interface (EB CLI), the AWS CLI, or an SDK. See [Managing multiple Elastic Beanstalk environments as a group with the EB CLI \(p. 881\)](#) for EB CLI instructions.

Using the Compose Environments API

For example, you could make an application named `Media Library` that lets users upload and manage images and videos stored in Amazon Simple Storage Service (Amazon S3). The application has a front-end environment, `front`, that runs a web application that lets users upload and download individual files, view their library, and initiate batch processing jobs.

Instead of processing the jobs directly, the front-end application adds jobs to an Amazon SQS queue. The second environment, `worker`, pulls jobs from the queue and processes them. `worker` uses a G2 instance type that has a high-performance GPU, while `front` can run on a more cost-effective generic instance type.

You would organize the project folder, `Media Library`, into separate directories for each component, with each directory containing an environment definition file (`env.yaml`) with the source code for each:

```
~/workspace/media-library
|-- front
|   |-- env.yaml
|-- worker
|   |-- env.yaml
```

The following listings show the `env.yaml` file for each component application.

~/workspace/media-library/front/env.yaml

```
EnvironmentName: front+
EnvironmentLinks:
  "WORKERQUEUE" : "worker+"
AWSConfigurationTemplateVersion: 1.1.0.0
EnvironmentTier:
  Name: WebServer
  Type: Standard
SolutionStack: 64bit Amazon Linux 2015.09 v2.0.4 running Java 8
OptionSettings:
  aws:autoscaling:launchconfiguration:
    InstanceType: m4.large
```

~/workspace/media-library/worker/env.yaml

```
EnvironmentName: worker+
AWSConfigurationTemplateVersion: 1.1.0.0
EnvironmentTier:
  Name: Worker
  Type: SQS/HTTP
SolutionStack: 64bit Amazon Linux 2015.09 v2.0.4 running Java 8
OptionSettings:
  aws:autoscaling:launchconfiguration:
    InstanceType: g2.2xlarge
```

After [creating an application version \(p. 337\)](#) for the front-end (`front-v1`) and worker (`worker-v1`) application components, you call the Compose Environments API with the version names. In this example, we use the AWS CLI to call the API.

```
# Create application versions for each component:
~$ aws elasticbeanstalk create-application-version --application-name media-library --
version-label front-v1 --process --source-bundle S3Bucket="my-bucket",S3Key="front-v1.zip"
{
  "ApplicationVersion": {
    "ApplicationName": "media-library",
    "VersionLabel": "front-v1",
    "Description": "",
    "DateCreated": "2015-11-03T23:01:25.412Z",
    "DateUpdated": "2015-11-03T23:01:25.412Z",
    "SourceBundle": {
      "S3Bucket": "my-bucket",
      "S3Key": "front-v1.zip"
    }
  }
}
~$ aws elasticbeanstalk create-application-version --application-name media-library --
version-label worker-v1 --process --source-bundle S3Bucket="my-bucket",S3Key="worker-
v1.zip"
{
  "ApplicationVersion": {
    "ApplicationName": "media-library",
    "VersionLabel": "worker-v1",
    "Description": "",
    "DateCreated": "2015-11-03T23:01:48.151Z",
    "DateUpdated": "2015-11-03T23:01:48.151Z",
    "SourceBundle": {
      "S3Bucket": "my-bucket",
      "S3Key": "worker-v1.zip"
    }
  }
}
```

```

}
# Create environments:
~$ aws elasticbeanstalk compose-environments --application-name media-library --group-name
dev --version-labels front-v1 worker-v1

```

The third call creates two environments, `front-dev` and `worker-dev`. The API creates the names of the environments by concatenating the `EnvironmentName` specified in the `env.yaml` file with the `group` name option specified in the `Compose Environments` call, separated by a hyphen. The total length of these two options and the hyphen must not exceed the maximum allowed environment name length of 23 characters.

The application running in the `front-dev` environment can access the name of the Amazon SQS queue attached to the `worker-dev` environment by reading the `WORKERQUEUE` variable. For more information on environment links, see [Creating links between Elastic Beanstalk environments \(p. 444\)](#).

Deploying applications to Elastic Beanstalk environments

You can use the AWS Elastic Beanstalk console to upload an updated [source bundle \(p. 342\)](#) and deploy it to your Elastic Beanstalk environment, or redeploy a previously uploaded version.

Each deployment is identified by a deployment ID. Deployment IDs start at 1 and increment by one with each deployment and instance configuration change. If you enable [enhanced health reporting \(p. 691\)](#), Elastic Beanstalk displays the deployment ID in both the [health console \(p. 701\)](#) and the [EB CLI \(p. 875\)](#) when it reports instance health status. The deployment ID helps you determine the state of your environment when a rolling update fails.

Elastic Beanstalk provides several deployment policies and settings. For details about configuring a policy and additional settings, see [the section called "Deployment options" \(p. 400\)](#). The following table lists the policies and the kinds of environments that support them.

Supported deployment policies

Deployment policy	Load-balanced environments	Single-instance environments	Legacy Windows Server environments†
All at once	✓	✓	✓
Rolling	✓	×	✓
Rolling with an additional batch	✓	×	×
Immutable	✓	✓	×
Traffic splitting	✓ (Application Load Balancer)	×	×

† In this table, a *Legacy Windows Server environment* is an environment based on a [Windows Server platform configuration](#) that uses an IIS version earlier than IIS 8.5.

Warning

Some policies replace all instances during the deployment or update. This causes all accumulated [Amazon EC2 burst balances](#) to be lost. It happens in the following cases:

- Managed platform updates with instance replacement enabled
- Immutable updates

- Deployments with immutable updates or traffic splitting enabled

Choosing a deployment policy

Choosing the right deployment policy for your application is a tradeoff of a few considerations, and depends on your particular needs. The [the section called “Deployment options” \(p. 400\)](#) page has more information about each policy, and a detailed description of the workings of some of them.



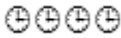
The following list provides summary information about the different deployment policies and adds related considerations.

- **All at once** – The quickest deployment method. Suitable if you can accept a short loss of service, and if quick deployments are important to you. With this method, Elastic Beanstalk deploys the new application version to each instance. Then, the web proxy or application server might need to restart. As a result, your application might be unavailable to users (or have low availability) for a short time.
- **Rolling** – Avoids downtime and minimizes reduced availability, at a cost of a longer deployment time. Suitable if you can't accept any period of completely lost service. With this method, your application is deployed to your environment one batch of instances at a time. Most bandwidth is retained throughout the deployment.
- **Rolling with additional batch** – Avoids any reduced availability, at a cost of an even longer deployment time compared to the *Rolling* method. Suitable if you must maintain the same bandwidth throughout the deployment. With this method, Elastic Beanstalk launches an extra batch of instances, then performs a rolling deployment. Launching the extra batch takes time, and ensures that the same bandwidth is retained throughout the deployment.
- **Immutable** – A slower deployment method, that ensures your new application version is always deployed to new instances, instead of updating existing instances. It also has the additional advantage of a quick and safe rollback in case the deployment fails. With this method, Elastic Beanstalk performs an [immutable update \(p. 412\)](#) to deploy your application. In an immutable update, a second Auto Scaling group is launched in your environment and the new version serves traffic alongside the old version until the new instances pass health checks.
- **Traffic splitting** – A canary testing deployment method. Suitable if you want to test the health of your new application version using a portion of incoming traffic, while keeping the rest of the traffic served by the old application version.

The following table compares deployment method properties.

Deployment methods

Method	Impact of failed deployment	Deploy time	Zero down	No DNS change	Rollback procedure	Code deployed to
All at once	Downtime	⊕	×	✓	Manual redeploy	Existing instances
Rolling	Single batch out of service; any successful batches before failure running new application version	⊕⊕†	✓	✓	Manual redeploy	Existing instances
Rolling with an additional batch	Minimal if first batch fails; otherwise, similar to Rolling	⊕⊕⊕†	✓	✓	Manual redeploy	New and existing instances

Method	Impact of failed deployment	Deploy time	Zero down	No DNS change	Rollback process	Code deployed to
Immutable	Minimal		✓	✓	Terminate new instances	New instances
Traffic splitting	Percentage of client traffic routed to new version temporarily impacted		✓	✓	Reroute new traffic and terminate new instances	New instances
Blue/green	Minimal		✓	✗	Swap URL	New instances

† *Varies depending on batch size.*

†† *Varies depending on **evaluation time** option setting.*

Deploying a new application version

You can perform deployments from your environment's dashboard.

To deploy a new application version to an Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Upload and deploy**.
4. Use the on-screen form to upload the application source bundle.
5. Choose **Deploy**.

Redeploying a previous version

You can also deploy a previously uploaded version of your application to any of its environments from the application versions page.

To deploy an existing application version to an existing environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.

Note

If you have many applications, use the search bar to filter the application list.

3. In the navigation pane, find your application's name and choose **Application versions**.
4. Select the application version to deploy.
5. Choose **Actions**, and then choose **Deploy**.
6. Select an environment, and then choose **Deploy**.

Other ways to deploy your application

If you deploy often, consider using the [Elastic Beanstalk Command Line Interface \(p. 852\)](#) (EB CLI) to manage your environments. The EB CLI creates a repository alongside your source code. It can also create a source bundle, upload it to Elastic Beanstalk, and deploy it with a single command.

For deployments that depend on resource configuration changes or a new version that can't run alongside the old version, you can launch a new environment with the new version and perform a CNAME swap for a [blue/green deployment \(p. 405\)](#).

Deployment policies and settings

AWS Elastic Beanstalk provides several options for how [deployments \(p. 397\)](#) are processed, including deployment policies (*All at once*, *Rolling*, *Rolling with additional batch*, *Immutable*, and *Traffic splitting*) and options that let you configure batch size and health check behavior during deployments. By default, your environment uses all-at-once deployments. If you created the environment with the EB CLI and it's an automatically scaling environment (you didn't specify the `--single` option), it uses rolling deployments.

With *rolling deployments*, Elastic Beanstalk splits the environment's Amazon EC2 instances into batches and deploys the new version of the application to one batch at a time. It leaves the rest of the instances in the environment running the old version of the application. During a rolling deployment, some instances serve requests with the old version of the application, while instances in completed batches serve other requests with the new version. For details, see [the section called "How rolling deployments work" \(p. 403\)](#).

To maintain full capacity during deployments, you can configure your environment to launch a new batch of instances before taking any instances out of service. This option is known as a *rolling deployment with an additional batch*. When the deployment completes, Elastic Beanstalk terminates the additional batch of instances.

Immutable deployments perform an [immutable update \(p. 412\)](#) to launch a full set of new instances running the new version of the application in a separate Auto Scaling group, alongside the instances running the old version. Immutable deployments can prevent issues caused by partially completed rolling deployments. If the new instances don't pass health checks, Elastic Beanstalk terminates them, leaving the original instances untouched.

Traffic-splitting deployments let you perform canary testing as part of your application deployment. In a traffic-splitting deployment, Elastic Beanstalk launches a full set of new instances just like during an immutable deployment. It then forwards a specified percentage of incoming client traffic to the new application version for a specified evaluation period. If the new instances stay healthy, Elastic Beanstalk forwards all traffic to them and terminates the old ones. If the new instances don't pass health checks, or if you choose to abort the deployment, Elastic Beanstalk moves traffic back to the old instances and terminates the new ones. There's never any service interruption. For details, see [the section called "How traffic-splitting deployments work" \(p. 404\)](#).

Warning

Some policies replace all instances during the deployment or update. This causes all accumulated [Amazon EC2 burst balances](#) to be lost. It happens in the following cases:

- Managed platform updates with instance replacement enabled
- Immutable updates
- Deployments with immutable updates or traffic splitting enabled

If your application doesn't pass all health checks, but still operates correctly at a lower health status, you can allow instances to pass health checks with a lower status, such as `Warning`, by modifying the

Healthy threshold option. If your deployments fail because they don't pass health checks and you need to force an update regardless of health status, specify the **Ignore health check** option.

When you specify a batch size for rolling updates, Elastic Beanstalk also uses that value for rolling application restarts. Use rolling restarts when you need to restart the proxy and application servers running on your environment's instances without downtime.

Configuring application deployments

In the [environment management console \(p. 353\)](#), enable and configure batched application version deployments by editing **Updates and Deployments** on the environment's **Configuration** page.

To configure deployments (console)

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Rolling updates and deployments** configuration category, choose **Edit**.
5. In the **Application Deployments** section, choose a **Deployment policy**, batch settings, and health check options.
6. Choose **Apply**.

The **Application deployments** section of the **Rolling updates and deployments** page has the following options for application deployments:

- **Deployment policy** – Choose from the following deployment options:
 - **All at once** – Deploy the new version to all instances simultaneously. All instances in your environment are out of service for a short time while the deployment occurs.
 - **Rolling** – Deploy the new version in batches. Each batch is taken out of service during the deployment phase, reducing your environment's capacity by the number of instances in a batch.
 - **Rolling with additional batch** – Deploy the new version in batches, but first launch a new batch of instances to ensure full capacity during the deployment process.
 - **Immutable** – Deploy the new version to a fresh group of instances by performing an [immutable update \(p. 412\)](#).
 - **Traffic splitting** – Deploy the new version to a fresh group of instances and temporarily split incoming client traffic between the existing application version and the new one.

For the **Rolling** and **Rolling with additional batch** deployment policies you can configure:

- **Batch size** – The size of the set of instances to deploy in each batch.

Choose **Percentage** to configure a percentage of the total number of EC2 instances in the Auto Scaling group (up to 100 percent), or choose **Fixed** to configure a fixed number of instances (up to the maximum instance count in your environment's Auto Scaling configuration).

For the **Traffic splitting** deployment policy you can configure the following:

- **Traffic split** – The initial percentage of incoming client traffic that Elastic Beanstalk shifts to environment instances running the new application version you're deploying.

- **Traffic splitting evaluation time** – The time period, in minutes, that Elastic Beanstalk waits after an initial healthy deployment before proceeding to shift all incoming client traffic to the new application version that you're deploying.

Elastic Beanstalk > Environments > GettingStartedApp-env > Configuration

Modify rolling updates and deployments

Application deployments

Choose how AWS Elastic Beanstalk propagates source code changes and software configuration updates. [Learn more](#)

Deployment policy

All at once

Batch size:

Percentage

Fixed

100 % of instances at a time

Traffic split

10 % to new application version

Traffic splitting evaluation time

5 minutes

The **Deployment preferences** section contains options related to health checks.

- **Ignore health check** – Prevents a deployment from rolling back when a batch fails to become healthy within the **Command timeout**.
- **Healthy threshold** – Lowers the threshold at which an instance is considered healthy during rolling deployments, rolling updates, and immutable updates.
- **Command timeout** – The number of seconds to wait for an instance to become healthy before canceling the deployment or, if **Ignore health check** is set, to continue to the next batch.

The screenshot displays the 'Deployment preferences' section in the AWS Elastic Beanstalk console. It includes three settings: 'Ignore health check' set to 'False', 'Healthy threshold' set to 'Ok', and 'Command timeout' set to '600'. Each setting has a brief description below it.

Deployment preferences
Customize health check requirements and deployment timeouts.

Ignore health check
False
Don't fail deployments due to health check failures.

Healthy threshold
Ok
Lower the threshold for an instance in a batch to pass health checks during an update or deployment.

Command timeout
600
Change the amount of time in seconds that AWS Elastic Beanstalk allows an instance to complete deployment commands.

How rolling deployments work

When processing a batch, Elastic Beanstalk detaches all instances in the batch from the load balancer, deploys the new application version, and then reattaches the instances. If you enable [connection draining](#) (p. 476), Elastic Beanstalk drains existing connections from the Amazon EC2 instances in each batch before beginning the deployment.

After reattaching the instances in a batch to the load balancer, Elastic Load Balancing waits until they pass a minimum number of Elastic Load Balancing health checks (the **Healthy check count threshold** value), and then starts routing traffic to them. If no [health check URL](#) (p. 477) is configured, this can happen very quickly, because an instance will pass the health check as soon as it can accept a TCP connection. If a health check URL is configured, the load balancer doesn't route traffic to the updated instances until they return a 200 OK status code in response to an HTTP GET request to the health check URL.

Elastic Beanstalk waits until all instances in a batch are healthy before moving on to the next batch. With [basic health reporting](#) (p. 688), instance health depends on the Elastic Load Balancing health check status. When all instances in the batch pass enough health checks to be considered healthy by Elastic Load Balancing, the batch is complete. If [enhanced health reporting](#) (p. 691) is enabled, Elastic Beanstalk considers several other factors, including the result of incoming requests. With enhanced health reporting, all instances must pass 12 consecutive health checks with an [OK status](#) (p. 707) within two minutes for web server environments, and 18 health checks within three minutes for worker environments.

If a batch of instances does not become healthy within the [command timeout](#) (p. 401), the deployment fails. After a failed deployment, [check the health of the instances in your environment](#) (p. 701) for information about the cause of the failure. Then perform another deployment with a fixed or known good version of your application to roll back.

If a deployment fails after one or more batches completed successfully, the completed batches run the new version of your application while any pending batches continue to run the old version. You can identify the version running on the instances in your environment on the [health page](#) (p. 702)

in the console. This page displays the deployment ID of the most recent deployment that executed on each instance in your environment. If you terminate instances from the failed deployment, Elastic Beanstalk replaces them with instances running the application version from the most recent successful deployment.

How traffic-splitting deployments work

Traffic-splitting deployments allow you to perform canary testing. You direct some incoming client traffic to your new application version to verify the application's health before committing to the new version and directing all traffic to it.

During a traffic-splitting deployment, Elastic Beanstalk creates a new set of instances in a separate temporary Auto Scaling group. Elastic Beanstalk then instructs the load balancer to direct a certain percentage of your environment's incoming traffic to the new instances. Then, for a configured amount of time, Elastic Beanstalk tracks the health of the new set of instances. If all is well, Elastic Beanstalk shifts remaining traffic to the new instances and attaches them to the environment's original Auto Scaling group, replacing the old instances. Then Elastic Beanstalk cleans up—terminates the old instances and removes the temporary Auto Scaling group.

Note

The environment's capacity doesn't change during a traffic-splitting deployment. Elastic Beanstalk launches the same number of instances in the temporary Auto Scaling group as there are in the original Auto Scaling group at the time the deployment starts. It then maintains a constant number of instances in both Auto Scaling groups for the deployment duration. Take this fact into account when configuring the environment's traffic splitting evaluation time.

Rolling back the deployment to the previous application version is quick and doesn't impact service to client traffic. If the new instances don't pass health checks, or if you choose to abort the deployment, Elastic Beanstalk moves traffic back to the old instances and terminates the new ones. You can abort any deployment by using the environment overview page in the Elastic Beanstalk console, and choosing **Abort current operation** in **Environment actions**. You can also call the [AbortEnvironmentUpdate](#) API or the equivalent AWS CLI command.

Traffic-splitting deployments require an Application Load Balancer. Elastic Beanstalk uses this load balancer type by default when you create your environment using the Elastic Beanstalk console or the EB CLI.

Deployment option namespaces

You can use the [configuration options \(p. 536\)](#) in the `aws:elasticbeanstalk:command` (p. 571) namespace to configure your deployments. If you choose the traffic-splitting policy, additional options for this policy are available in the `aws:elasticbeanstalk:trafficsplitting` (p. 581) namespace.

Use the `DeploymentPolicy` option to set the deployment type. The following values are supported:

- `AllAtOnce` – Disables rolling deployments and always deploys to all instances simultaneously.
- `Rolling` – Enables standard rolling deployments.
- `RollingWithAdditionalBatch` – Launches an extra batch of instances, before starting the deployment, to maintain full capacity.
- `Immutable` – Performs an [immutable update \(p. 412\)](#) for every deployment.
- `TrafficSplitting` – Performs traffic-splitting deployments to canary-test your application deployments.

When you enable rolling deployments, set the `BatchSize` and `BatchSizeType` options to configure the size of each batch. For example, to deploy 25 percent of all instances in each batch, specify the following options and values.

Example `.ebextensions/rolling-updates.config`

```
option_settings:
  aws:elasticbeanstalk:command:
    DeploymentPolicy: Rolling
    BatchSizeType: Percentage
    BatchSize: 25
```

To deploy to five instances in each batch, regardless of the number of instances running, and to bring up an extra batch of five instances running the new version before pulling any instances out of service, specify the following options and values.

Example `.ebextensions/rolling-additionalbatch.config`

```
option_settings:
  aws:elasticbeanstalk:command:
    DeploymentPolicy: RollingWithAdditionalBatch
    BatchSizeType: Fixed
    BatchSize: 5
```

To perform an immutable update for each deployment with a health check threshold of **Warning**, and proceed with the deployment even if instances in a batch don't pass health checks within a timeout of 15 minutes, specify the following options and values.

Example `.ebextensions/immutable-ignorehealth.config`

```
option_settings:
  aws:elasticbeanstalk:command:
    DeploymentPolicy: Immutable
    HealthCheckSuccessThreshold: Warning
    IgnoreHealthCheck: true
    Timeout: "900"
```

To perform traffic-splitting deployments, forwarding 15 percent of client traffic to the new application version and evaluating health for 10 minutes, specify the following options and values.

Example `.ebextensions/traffic-splitting.config`

```
option_settings:
  aws:elasticbeanstalk:command:
    DeploymentPolicy: TrafficSplitting
  aws:elasticbeanstalk:trafficsplitting:
    NewVersionPercent: "15"
    EvaluationTime: "10"
```

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See [Recommended values \(p. 537\)](#) for details.

Blue/Green deployments with Elastic Beanstalk

Because AWS Elastic Beanstalk performs an in-place update when you update your application versions, your application can become unavailable to users for a short period of time. You can avoid this downtime by performing a blue/green deployment, where you deploy the new version to a separate environment, and then swap CNAMEs of the two environments to redirect traffic to the new version instantly.

A blue/green deployment is also required when you want to update an environment to an incompatible platform version. For more information, see [the section called “Platform updates” \(p. 415\)](#).

Blue/green deployments require that your environment runs independently of your production database, if your application uses one. If your environment has an Amazon RDS DB instance attached to it, the data will not transfer over to your second environment, and will be lost if you terminate the original environment.

For details on configuring your application to connect to an external (not managed by Elastic Beanstalk) Amazon RDS instance, see [Using Elastic Beanstalk with Amazon RDS \(p. 820\)](#).


To perform a blue/green deployment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. [Clone your current environment \(p. 384\)](#), or launch a new environment running the platform version you want.
3. [Deploy the new application version \(p. 399\)](#) to the new environment.
4. Test the new version on the new environment.
5. On the environment overview page, choose **Environment actions**, and then choose **Swap environment URLs**.
6. For **Environment name**, select the current environment.

Elastic Beanstalk > Environments > GettingStartedApp-env

Swap environment URLs

When you swap an environment's URL with another environment's URL, you can deploy versions with no d

 Swapping the environment URL will modify the Route 53 DNS configuration, which may take a few m
will continue to run while the changes are propagated.

Environment details

Environment name:
staging-env

Environment URL:
staging-env.bx7dx222kw.us-east-2.elasticbeanstalk.com

Select an environment to swap

Environment name:
prod-env (e-2mvwbhpfcs) ▼

Environment URL:
prod-env.bx7dx222kw.us-east-2.elasticbeanstalk.com

7. Choose **Swap**.

Elastic Beanstalk swaps the CNAME records of the old and new environments, redirecting traffic from the old version to the new version and vice versa.

After Elastic Beanstalk completes the swap operation, verify that the new environment responds when you try to connect to the old environment URL. However, do not terminate your old environment until

the DNS changes are propagated and your old DNS records expire. DNS servers don't necessarily clear old records from their cache based on the time to live (TTL) you set on your DNS records.

Configuration changes

When you modify configuration option settings in the **Configuration** section of the [environment management console \(p. 353\)](#), AWS Elastic Beanstalk propagates the change to all affected resources. These resources include the load balancer that distributes traffic to the Amazon EC2 instances running your application, the Auto Scaling group that manages those instances, and the EC2 instances themselves.

Many configuration changes can be applied to a running environment without replacing existing instances. For example, setting a [health check URL \(p. 477\)](#) triggers an environment update to modify the load balancer settings, but doesn't cause any downtime because the instances running your application continue serving requests while the update is propagated.

Configuration changes that modify the [launch configuration \(p. 556\)](#) or [VPC settings \(p. 568\)](#) require terminating all instances in your environment and replacing them. For example, when you change the instance type or SSH key setting for your environment, the EC2 instances must be terminated and replaced. To prevent downtime during these processes, Elastic Beanstalk applies these configuration changes in batches, keeping a minimum number of instances running and serving traffic at all times. This process is known as a [rolling update \(p. 409\)](#).

[Immutable updates \(p. 412\)](#) are an alternative to rolling updates where a temporary Auto Scaling group is launched outside of your environment with a separate set of instances running on the new configuration, which are placed behind your environment's load balancer. Old and new instances both serve traffic until the new instances pass health checks, at which time the new instances are moved into your environment's Auto Scaling group and the temporary group and old instances are terminated.

Warning

Some policies replace all instances during the deployment or update. This causes all accumulated [Amazon EC2 burst balances](#) to be lost. It happens in the following cases:

- Managed platform updates with instance replacement enabled
- Immutable updates
- Deployments with immutable updates or traffic splitting enabled

Supported update types

Rolling update setting	Load balanced environments	Single-instance environments	Legacy Windows server environments†
Disabled	✓	✓	✓
Rolling Based on Health	✓	×	✓
Rolling Based on Time	✓	×	✓
Immutable	✓	✓	×

† For the purpose of this table, a *Legacy Windows Server Environment* is an environment based on a [Windows Server platform configuration](#) that use an IIS version earlier than IIS 8.5.

Topics

- [Elastic Beanstalk rolling environment configuration updates \(p. 409\)](#)
- [Immutable environment updates \(p. 412\)](#)

Elastic Beanstalk rolling environment configuration updates

When a [configuration change requires replacing instances \(p. 408\)](#), Elastic Beanstalk can perform the update in batches to avoid downtime while the change is propagated. During a rolling update, capacity is only reduced by the size of a single batch, which you can configure. Elastic Beanstalk takes one batch of instances out of service, terminates them, and then launches a batch with the new configuration. After the new batch starts serving requests, Elastic Beanstalk moves on to the next batch.

Rolling configuration update batches can be processed periodically (time-based), with a delay between each batch, or based on health. For time-based rolling updates, you can configure the amount of time that Elastic Beanstalk waits after completing the launch of a batch of instances before moving on to the next batch. This pause time allows your application to bootstrap and start serving requests.

With health-based rolling updates, Elastic Beanstalk waits until instances in a batch pass health checks before moving on to the next batch. The health of an instance is determined by the health reporting system, which can be basic or enhanced. With [basic health \(p. 688\)](#), a batch is considered healthy as soon as all instances in it pass Elastic Load Balancing (ELB) health checks.

With [enhanced health reporting \(p. 691\)](#), all of the instances in a batch must pass multiple consecutive health checks before Elastic Beanstalk will move on to the next batch. In addition to ELB health checks, which check only your instances, enhanced health monitors application logs and the state of your environment's other resources. In a web server environment with enhanced health, all instances must pass 12 health checks over the course of two minutes (18 checks over three minutes for worker environments). If any instance fails one health check, the count resets.

If a batch doesn't become healthy within the rolling update timeout (default is 30 minutes), the update is canceled. Rolling update timeout is a [configuration option \(p. 536\)](#) that is available in the `aws:autoscaling:updatepolicy:rollingupdate (p. 412)` namespace. If your application doesn't pass health checks with `Ok` status but is stable at a different level, you can set the `HealthCheckSuccessThreshold` option in the `aws:elasticbeanstalk:healthreporting:system (p. 577)` namespace to change the level at which Elastic Beanstalk considers an instance to be healthy.

If the rolling update process fails, Elastic Beanstalk starts another rolling update to roll back to the previous configuration. A rolling update can fail due to failed health checks or if launching new instances causes you to exceed the quotas on your account. If you hit a quota on the number of Amazon EC2 instances, for example, the rolling update can fail when it attempts to provision a batch of new instances. In this case, the rollback fails as well.

A failed rollback ends the update process and leaves your environment in an unhealthy state. Unprocessed batches are still running instances with the old configuration, while any batches that completed successfully have the new configuration. To fix an environment after a failed rollback, first resolve the underlying issue that caused the update to fail, and then initiate another environment update.

An alternative method is to deploy the new version of your application to a different environment and then perform a CNAME swap to redirect traffic with zero downtime. See [Blue/Green deployments with Elastic Beanstalk \(p. 405\)](#) for more information.

Rolling updates versus rolling deployments

Rolling updates occur when you change settings that require new Amazon EC2 instances to be provisioned for your environment. This includes changes to the Auto Scaling group configuration, such as instance type and key-pair settings, and changes to VPC settings. In a rolling update, each batch of instances is terminated before a new batch is provisioned to replace it.

[Rolling deployments \(p. 400\)](#) occur whenever you deploy your application and can typically be performed without replacing instances in your environment. Elastic Beanstalk takes each batch out of service, deploys the new application version, and then places it back in service.

The exception to this is if you change settings that require instance replacement at the same time you deploy a new application version. For example, if you change the [key name \(p. 556\)](#) settings in a [configuration file \(p. 600\)](#) in your source bundle and deploy it to your environment, you trigger a rolling update. Instead of deploying your new application version to each batch of existing instances, a new batch of instances is provisioned with the new configuration. In this case, a separate deployment doesn't occur because the new instances are brought up with the new application version.

Anytime new instances are provisioned as part of an environment update, there is a deployment phase where your application's source code is deployed to the new instances and any configuration settings that modify the operating system or software on the instances are applied. [Deployment health check settings \(p. 401\)](#) (**Ignore health check**, **Healthy threshold**, and **Command timeout**) also apply to health-based rolling updates and immutable updates during the deployment phase.

Configuring rolling updates

You can enable and configure rolling updates in the Elastic Beanstalk console.

To enable rolling updates

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Rolling updates and deployments** configuration category, choose **Edit**.
5. In the **Configuration updates** section, for **Rolling update type**, select one of the **Rolling** options.

Configuration updates

Changes to virtual machine settings and VPC configuration trigger rolling updates to replace the instances in your environment. [Learn more](#)

Rolling update type
Rolling based on Health

Batch size
1
The maximum number of instances to replace in each phase of the update.

Minimum capacity
1
The minimum number of instances to keep in service at all times.

Pause time
hh:mm:ss
Pause the update for up to an hour between each batch.

6. Choose **Batch size**, **Minimum capacity**, and **Pause time** settings.
7. Choose **Apply**.

The **Configuration updates** section of the **Rolling updates and deployments** page has the following options for rolling updates:

- **Rolling update type** – Elastic Beanstalk waits after it finishes updating a batch of instances before moving on to the next batch, to allow those instances to finish bootstrapping and start serving traffic. Choose from the following options:
 - **Rolling based on Health** – Wait until instances in the current batch are healthy before placing instances in service and starting the next batch.
 - **Rolling based on Time** – Specify an amount of time to wait between launching new instances and placing them in service before starting the next batch.
 - **Immutable** – Apply the configuration change to a fresh group of instances by performing an [immutable update](#) (p. 412).
- **Batch size** – The number of instances to replace in each batch, between **1** and **10000**. By default, this value is one-third of the minimum size of the Auto Scaling group, rounded up to a whole number.
- **Minimum capacity** – The minimum number of instances to keep running while other instances are updated, between **0** and **9999**. The default value is either the minimum size of the Auto Scaling group or one less than the maximum size of the Auto Scaling group, whichever number is lower.
- **Pause time** (time-based only) – The amount of time to wait after a batch is updated before moving on to the next batch, to allow your application to start receiving traffic. Between 0 seconds and one hour.

The `aws:autoscaling:updatepolicy:rollingupdate` namespace

You can also use the [configuration options \(p. 536\)](#) in the `aws:autoscaling:updatepolicy:rollingupdate` (p. 564) namespace to configure rolling updates.

Use the `RollingUpdateEnabled` option to enable rolling updates, and `RollingUpdateType` to choose the update type. The following values are supported for `RollingUpdateType`:

- `Health` – Wait until instances in the current batch are healthy before placing instances in service and starting the next batch.
- `Time` – Specify an amount of time to wait between launching new instances and placing them in service before starting the next batch.
- `Immutable` – Apply the configuration change to a fresh group of instances by performing an [immutable update \(p. 412\)](#).

When you enable rolling updates, set the `MaxBatchSize` and `MinInstancesInService` options to configure the size of each batch. For time-based and health-based rolling updates, you can also configure a `PauseTime` and `Timeout`, respectively.

For example, to launch up to five instances at a time, while maintaining at least two instances in service, and wait five minutes and 30 seconds between batches, specify the following options and values.

Example `.ebextensions/timebased.config`

```
option_settings:
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateEnabled: true
    MaxBatchSize: 5
    MinInstancesInService: 2
    RollingUpdateType: Time
    PauseTime: PT5M30S
```

To enable health-based rolling updates, with a 45-minute timeout for each batch, specify the following options and values.

Example `.ebextensions/healthbased.config`

```
option_settings:
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateEnabled: true
    MaxBatchSize: 5
    MinInstancesInService: 2
    RollingUpdateType: Health
    Timeout: PT45M
```

`Timeout` and `PauseTime` values must be specified in [ISO8601 duration](#): `PT#H#M#S`, where each # is the number of hours, minutes, or seconds, respectively.

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See [Recommended values \(p. 537\)](#) for details.

Immutable environment updates

Immutable environment updates are an alternative to [rolling updates \(p. 409\)](#). Immutable environment updates ensure that configuration changes that require replacing instances are applied efficiently and

safely. If an immutable environment update fails, the rollback process requires only terminating an Auto Scaling group. A failed rolling update, on the other hand, requires performing an additional rolling update to roll back the changes.

To perform an immutable environment update, Elastic Beanstalk creates a second, temporary Auto Scaling group behind your environment's load balancer to contain the new instances. First, Elastic Beanstalk launches a single instance with the new configuration in the new group. This instance serves traffic alongside all of the instances in the original Auto Scaling group that are running the previous configuration.

When the first instance passes health checks, Elastic Beanstalk launches additional instances with the new configuration, matching the number of instances running in the original Auto Scaling group. When all of the new instances pass health checks, Elastic Beanstalk transfers them to the original Auto Scaling group, and terminates the temporary Auto Scaling group and old instances.

Note

During an immutable environment update, the capacity of your environment doubles for a short time when the instances in the new Auto Scaling group start serving requests and before the original Auto Scaling group's instances are terminated. If your environment has many instances, or you have a low [on-demand instance quota](#), ensure that you have enough capacity to perform an immutable environment update. If you are near the quota, consider using rolling updates instead.

Immutable updates require [enhanced health reporting \(p. 691\)](#) to evaluate your environment's health during the update. Enhanced health reporting combines standard load balancer health checks with instance monitoring to ensure that the instances running the new configuration are [serving requests successfully \(p. 694\)](#).

You can also use immutable updates to deploy new versions of your application, as an alternative to rolling deployments. When you [configure Elastic Beanstalk to use immutable updates for application deployments \(p. 400\)](#), it replaces all instances in your environment every time you deploy a new version of your application. If an immutable application deployment fails, Elastic Beanstalk reverts the changes immediately by terminating the new Auto Scaling group. This can prevent partial fleet deployments, which can occur when a rolling deployment fails after some batches have already completed.

Warning

Some policies replace all instances during the deployment or update. This causes all accumulated [Amazon EC2 burst balances](#) to be lost. It happens in the following cases:

- Managed platform updates with instance replacement enabled
- Immutable updates
- Deployments with immutable updates or traffic splitting enabled

If an immutable update fails, the new instances upload [bundle logs \(p. 732\)](#) to Amazon S3 before Elastic Beanstalk terminates them. Elastic Beanstalk leaves logs from a failed immutable update in Amazon S3 for one hour before deleting them, instead of the standard 15 minutes for bundle and tail logs.

Note

If you use immutable updates for application version deployments, but not for configuration, you might encounter an error if you attempt to deploy an application version that contains configuration changes that would normally trigger a rolling update (for example, configurations that change instance type). To avoid this, make the configuration change in a separate update, or configure immutable updates for both deployments and configuration changes.

You can't perform an immutable update in concert with resource configuration changes. For example, you can't change [settings that require instance replacement \(p. 408\)](#) while also updating other settings, or perform an immutable deployment with configuration files that change configuration settings or

additional resources in your source code. If you attempt to change resource settings (for example, load balancer settings) and concurrently perform an immutable update, Elastic Beanstalk returns an error.

If your resource configuration changes aren't dependent on your source code change or on instance configuration, perform them in two updates. If they are dependent, perform a [blue/green deployment](#) (p. 405) instead.

Configuring immutable updates

You can enable and configure immutable updates in the Elastic Beanstalk console.

To enable immutable updates (console)

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Rolling updates and deployments** configuration category, choose **Edit**.
5. In the **Configuration Updates** section, set **Rolling update type** to **Immutable**.

Configuration updates
Changes to virtual machine settings and VPC configuration trigger rolling updates to replace the instances in your environment. [Learn more](#)

Rolling update type
Immutable

Batch size
1
The maximum number of instances to replace in each phase of the update.

Minimum capacity
1
The minimum number of instances to keep in service at all times.

Pause time
hh:mm:ss
Pause the update for up to an hour between each batch.

6. Choose **Apply**.

The `aws:autoscaling:updatepolicy:rollingupdate` namespace

You can also use the options in the `aws:autoscaling:updatepolicy:rollingupdate` namespace to configure immutable updates. The following example [configuration file \(p. 600\)](#) enables immutable updates for configuration changes.

Example `.ebextensions/immutable-updates.config`

```
option_settings:
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateType: Immutable
```

The following example enables immutable updates for both configuration changes and deployments.

Example `.ebextensions/immutable-all.config`

```
option_settings:
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateType: Immutable
  aws:elasticbeanstalk:command:
    DeploymentPolicy: Immutable
```

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See [Recommended values \(p. 537\)](#) for details.

Updating your Elastic Beanstalk environment's platform version

Elastic Beanstalk regularly releases new platform versions to update all Linux-based and Windows Server-based [platforms \(p. 29\)](#). New platform versions provide updates to existing software components and support for new features and configuration options. To learn about platforms and platform versions, see [Elastic Beanstalk platforms glossary \(p. 25\)](#).

You can use the Elastic Beanstalk console or the EB CLI to update your environment's platform version. Depending on the platform version you'd like to update to, Elastic Beanstalk recommends one of two methods for performing platform updates.

- [Method 1 – Update your environment's platform version \(p. 418\)](#). We recommend this method when you're updating to the latest platform version within a platform branch—with the same runtime, web server, application server, and operating system, and without a change in the major platform version. This is the most common and routine platform update.
- [Method 2 – Perform a Blue/Green deployment \(p. 419\)](#). We recommend this method when you're updating to a platform version in a different platform branch—with a different runtime, web server, application server, or operating system, or to a different runtime major platform version. This is a good approach when you want to take advantage of new runtime capabilities or the latest Elastic Beanstalk functionality, or when you want to move off of a deprecated or retired platform branch.

[Migrating from a legacy platform version \(p. 425\)](#) requires a blue/green deployment, because these platform versions are incompatible with currently supported versions.

[Migrating a Linux application to Amazon Linux 2 \(p. 426\)](#) requires a blue/green deployment, because Amazon Linux 2 platform versions are incompatible with previous Amazon Linux AMI platform versions.

For more help with choosing the best platform update method, expand the section for your environment's platform.

Single Container Docker

Use [Method 1 \(p. 418\)](#) to perform platform updates.

Multicontainer Docker

Use [Method 1 \(p. 418\)](#) to perform platform updates.

Preconfigured Docker

Consider the following cases:

- If you're migrating your application to another platform, for example from *Go 1.4 (Docker)* to *Go 1.11* or from *Python 3.4 (Docker)* to *Python 3.6*, use [Method 2 \(p. 419\)](#).
- If you're migrating your application to a different Docker container version, for example from *Glassfish 4.1 (Docker)* to *Glassfish 5.0 (Docker)*, use [Method 2 \(p. 419\)](#).
- If you're updating to a latest platform version with no change in container version or major version, use [Method 1 \(p. 418\)](#).

Go

Use [Method 1 \(p. 418\)](#) to perform platform updates.

Java SE

Consider the following cases:

- If you're migrating your application to a different Java runtime version, for example from *Java 7* to *Java 8*, use [Method 2 \(p. 419\)](#).
- If you're updating to a latest platform version with no change in runtime version, use [Method 1 \(p. 418\)](#).

Java with Tomcat

Consider the following cases:

- If you're migrating your application to a different Java runtime version or Tomcat application server version, for example from *Java 7 with Tomcat 7* to *Java 8 with Tomcat 8.5*, use [Method 2 \(p. 419\)](#).
- If you're migrating your application across major Java with Tomcat platform versions (v1.x.x, v2.x.x, and v3.x.x), use [Method 2 \(p. 419\)](#).
- If you're updating to a latest platform version with no change in runtime version, application server version, or major version, use [Method 1 \(p. 418\)](#).

.NET on Windows server with IIS

Consider the following cases:

- If you're migrating your application to a different Windows operating system version, for example from *Windows Server 2008 R2* to *Windows Server 2016*, use [Method 2 \(p. 419\)](#).

- If you're migrating your application across major Windows Server platform versions, see [Migrating from earlier major versions of the Windows server platform \(p. 144\)](#), and use [Method 2 \(p. 419\)](#).
- If your application is currently running on a Windows Server platform V2.x.x and you're updating to a latest platform version, use [Method 1 \(p. 418\)](#).

Note

[Windows Server platform versions](#) earlier than v2 aren't semantically versioned. You can only launch the latest version of each of these Windows Server major platform versions and can't roll back after an upgrade.

Node.js

Use [Method 2 \(p. 419\)](#) to perform platform updates.

PHP

Consider the following cases:

- If you're migrating your application to a different PHP runtime version, for example from *PHP 5.6* to *PHP 7.2*, use [Method 2 \(p. 419\)](#).
- If you're migrating your application across major PHP platform versions (v1.x.x and v2.x.x), use [Method 2 \(p. 419\)](#).
- If you're updating to a latest platform version with no change in runtime version or major version, use [Method 1 \(p. 418\)](#).

Python

Consider the following cases:

- If you're migrating your application to a different Python runtime version, for example from *Python 2.7* to *Python 3.6*, use [Method 2 \(p. 419\)](#).
- If you're migrating your application across major Python platform versions (v1.x.x and v2.x.x), use [Method 2 \(p. 419\)](#).
- If you're updating to a latest platform version with no change in runtime version or major version, use [Method 1 \(p. 418\)](#).

Ruby

Consider the following cases:

- If you're migrating your application to a different Ruby runtime version or application server version, for example from *Ruby 2.3 with Puma* to *Ruby 2.6 with Puma*, use [Method 2 \(p. 419\)](#).
- If you're migrating your application across major Ruby platform versions (v1.x.x and v2.x.x), use [Method 2 \(p. 419\)](#).
- If you're updating to a latest platform version with no change in runtime version, application server version, or major version, use [Method 1 \(p. 418\)](#).

Method 1 – Update your environment's platform version

Use this method to update to the latest version of your environment's platform branch. If you've previously created an environment using an older platform version, or upgraded your environment from an older version, you can also use this method to revert to a previous platform version, provided that it's in the same platform branch.

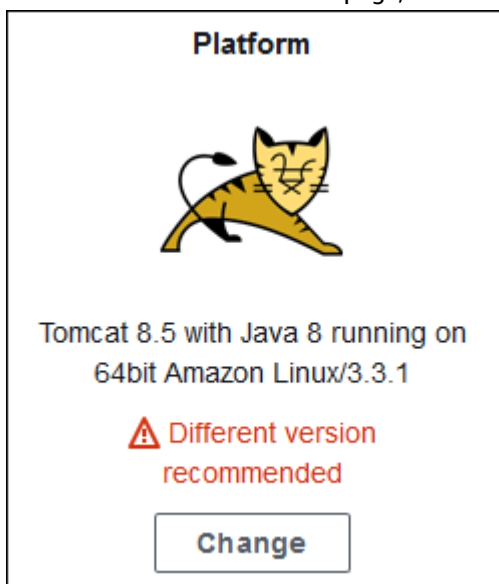
To update your environment's platform version

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note


If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, under **Platform**, choose **Change**.



4. On the **Update platform version** dialog, select a platform version. The newest (recommended) platform version in the branch is selected automatically. You can update to any version that you've used in the past.

Update platform version

 **Availability warning**

This operation replaces your instances; your application is unavailable during the update. To keep at least one instance in service during the update, enable rolling updates. Another option is to clone the environment, which creates a newer version of the platform, and then swap the CNAME of the environment when you are ready to deploy the clone. Learn more at [Updating AWS Elastic Beanstalk Environment Platform Version](#), [Rolling Updates](#) and [Deploying Version with Zero Downtime](#).

Platform branch
Tomcat 8.5 with Java 8 running on 64bit Amazon Linux

Current platform version
3.3.1

New platform version

5. Choose **Save**.

To further simplify platform updates, Elastic Beanstalk can manage them for you. You can configure your environment to apply minor and patch version updates automatically during a configurable weekly maintenance window. Elastic Beanstalk applies managed updates with no downtime or reduction in capacity, and cancels the update immediately if instances running your application on the new version fail health checks. For details, see [Managed platform updates \(p. 420\)](#).

Method 2 – Perform a Blue/Green deployment

Use this method to update to a different platform branch—with a different runtime, web server, application server, or operating system, or to a different major platform version. This is typically necessary when you want to take advantage of new runtime capabilities or the latest Elastic Beanstalk functionality. It's also required when you're migrating off of a deprecated or retired platform branch.

When you migrate across major platform versions or to platform versions with major component updates, there's a greater likelihood that your application, or some aspects of it, might not function as expected on the new platform version, and might require changes.

Before performing the migration, update your local development machine to the newer runtime versions and other components of the platform you plan on migrating to. Verify that your application still works

as expected, and make any necessary code fixes and changes. Then use the following best practice procedure to safely migrate your environment to the new platform version.

To migrate your environment to a platform version with major updates

1. [Create a new environment \(p. 363\)](#), using the new target platform version, and deploy your application code to it. The new environment should be in the Elastic Beanstalk application that contains the environment you're migrating. Don't terminate the existing environment yet.
2. Use the new environment to migrate your application. In particular:
 - Find and fix any application compatibility issues that you couldn't discover during the development phase.
 - Ensure that any customizations that your application makes using [configuration files \(p. 600\)](#) work correctly in the new environment. These might include option settings, additional installed packages, custom security policies, and script or configuration files installed on environment instances.
 - If your application uses a custom Amazon Machine Image (AMI), create a new custom AMI based on the AMI of the new platform version. To learn more, see [Using a custom Amazon machine image \(AMI\) \(p. 647\)](#). Specifically, this is required if your application uses the Windows Server platform with a custom AMI, and you're migrating to a Windows Server V2 platform version. In this case, see also [Migrating from earlier major versions of the Windows server platform \(p. 144\)](#).

Iterate on testing and deploying your fixes until you're satisfied with the application on the new environment.

3. Turn the new environment into your production environment by swapping its CNAME with the existing production environment's CNAME. For details, see [Blue/Green deployments with Elastic Beanstalk \(p. 405\)](#).
4. When you're satisfied with the state of your new environment in production, terminate the old environment. For details, see [Terminate an Elastic Beanstalk environment \(p. 386\)](#).

Managed platform updates

AWS Elastic Beanstalk regularly releases [platform updates \(p. 415\)](#) to provide fixes, software updates, and new features. With managed platform updates, you can configure your environment to automatically upgrade to the latest version of a platform during a scheduled [maintenance window \(p. 423\)](#). Your application remains in service during the update process with no reduction in capacity. Managed updates are available on both single-instance and load balanced environments.

Note

This feature isn't available on [Windows Server platform versions](#) earlier than version 2 (v2).

You can configure your environment to automatically apply [patch version updates \(p. 423\)](#), or both patch and minor version updates. Managed platform updates don't support updates across platform branches (updates to different major versions of platform components such as operating system, runtime, or Elastic Beanstalk components), because these can introduce changes that are backward incompatible.

You can also configure Elastic Beanstalk to replace all instances in your environment during the maintenance window, even if a platform update isn't available. Replacing all instances in your environment is helpful if your application encounters bugs or memory issues when running for a long period.

On environments created on November 25, 2019 or later using the Elastic Beanstalk console, managed updates are enabled by default whenever possible. Managed updates require [enhanced health \(p. 691\)](#)

to be enabled. Enhanced health is enabled by default when you select one of the [configuration presets \(p. 370\)](#), and disabled when you select **Custom configuration**. The console can't enable managed updates for older platform versions that don't support enhanced health, or when enhanced health is disabled. When the console enables managed updates for a new environment, the **Weekly update window** is set to a random day of the week at a random time. **Update level** is set to **Minor and patch**, and **Instance replacement** is disabled. You can disable or reconfigure managed updates before the final environment creation step.

For an existing environment, use the Elastic Beanstalk console anytime to configure managed platform updates.

To configure managed platform updates

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Managed updates** category, choose **Edit**.
5. Disable or enable **Managed updates**.
6. If managed updates are enabled, select a maintenance window, and then select an **Update level**.
7. (Optional) Select **Instance replacement** to enable weekly instance replacement.

The screenshot shows the 'Modify managed updates' configuration page in the AWS Management Console. The breadcrumb navigation at the top reads: 'Elastic Beanstalk > Environments > GettingStartedApp-env > Configuration'. The main heading is 'Modify managed updates'. Below this, there is a section titled 'Managed platform updates' with a descriptive paragraph: 'Enable managed platform updates to apply platform updates automatically during a weekly maintenance window. Your application stays available during the update process.' Underneath, the 'Managed updates' checkbox is checked and labeled 'Enabled'. The 'Weekly update window' is set to 'Tuesday' at '12' : '00' UTC. A note below states: 'Any available managed updates will run between Tuesday, 4:00 AM and Tuesday, 6:00 AM (-0800 GMT)'. The 'Update level' is set to 'Minor and patch'. The 'Instance replacement' checkbox is unchecked and labeled 'Enabled'. At the bottom right, there is a 'Cancel' button and a partially visible 'Apply' button.

8. Choose **Apply**.

Managed platform updates depend on [enhanced health reporting \(p. 691\)](#) to determine that your application is healthy enough to consider the platform update successful. See [Enabling Elastic Beanstalk enhanced health reporting \(p. 698\)](#) for instructions.

Sections

- [Permissions required to perform managed platform updates \(p. 423\)](#)
- [Managed update maintenance window \(p. 423\)](#)
- [Minor and patch version updates \(p. 423\)](#)
- [Immutable environment updates \(p. 423\)](#)
- [Managing managed updates \(p. 424\)](#)
- [Managed action option namespaces \(p. 425\)](#)

Permissions required to perform managed platform updates

Elastic Beanstalk needs permission to initiate a platform update on your behalf. When you use the default [service role \(p. 20\)](#) for your environment, the console adds the required permissions when you enable managed platform updates. If you aren't using the default service role, or you're managing your environments with a different client, add the [AWSElasticBeanstalkService \(p. 768\)](#) managed policy to your service role.

If you're using your account's [monitoring service-linked role \(p. 770\)](#) for your environment, you can't enable managed platform updates. The service-linked role doesn't have the required permissions. Select a different role, and be sure it has the [AWSElasticBeanstalkService \(p. 768\)](#) managed policy.

Note

If you use [configuration files \(p. 600\)](#) to extend your environment to include additional resources, you might need to add permissions to your environment's service role. Typically you need to add permissions when you reference these resources by name in other sections or files.

If an update fails, you can find the reason for the failure on the [Managed Updates \(p. 424\)](#) page.

Managed update maintenance window

When AWS releases a new version of your environment's platform, Elastic Beanstalk schedules a managed platform update during the next weekly maintenance window. Maintenance windows are two hours long. Elastic Beanstalk starts a scheduled update during the maintenance window. The update might not complete until after the window ends.

Note

In most cases, Elastic Beanstalk schedules your managed update to occur during your coming weekly maintenance window. The system considers various aspects of update safety and service availability when scheduling managed updates. In rare cases, an update might not be scheduled for the first coming maintenance window. If this happens, the system tries again during the next maintenance window. To manually apply the managed update, choose **Apply now** as explained in [Managing managed updates \(p. 424\)](#) on this page.

Minor and patch version updates

You can enable managed platform updates to apply patch version updates only, or for both minor and patch version updates. Patch version updates provide bug fixes and performance improvements, and can include minor configuration changes to the on-instance software, scripts, and configuration options. Minor version updates provide support for new Elastic Beanstalk features. You can't apply major version updates, which might make changes that are backward incompatible, with managed platform updates.

In a platform version number, the second number is the minor update version, and the third number is the patch version. For example, a version 2.0.7 platform version has a minor version of 0 and a patch version of 7.

Immutable environment updates

Managed platform updates perform [immutable environment updates \(p. 412\)](#) to upgrade your environment to a new platform version. Immutable updates update your environment without taking any instances out of service or modifying your environment, before confirming that instances running the new version pass health checks.

In an immutable update, Elastic Beanstalk deploys as many instances as are currently running with the new platform version. The new instances begin to take requests alongside those running the old version. If the new set of instances passes all health checks, Elastic Beanstalk terminates the old set of instances, leaving only instances with the new version.

Managed platform updates always perform immutable updates, even when you apply them outside of the maintenance window. If you change the platform version from the **Dashboard**, Elastic Beanstalk applies the update policy that you've chosen for configuration updates.

Warning

Some policies replace all instances during the deployment or update. This causes all accumulated [Amazon EC2 burst balances](#) to be lost. It happens in the following cases:

- Managed platform updates with instance replacement enabled
- Immutable updates
- Deployments with immutable updates or traffic splitting enabled

Managing managed updates

The Elastic Beanstalk console shows detailed information about managed updates on the **Managed updates overview** page.

To view information about managed updates (console)

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Managed updates**.

The **Managed Updates Overview** section provides information about scheduled and pending managed updates. The **History** section lists successful updates and failed attempts.

You can choose to apply a scheduled update immediately, instead of waiting until the maintenance window.

To apply a managed platform update immediately (console)

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Managed updates**.
4. Choose **Apply now**.
5. Verify the update details, and then choose **Apply**.

When you apply a managed platform update outside of the maintenance window, Elastic Beanstalk performs an immutable update. If you update the environment's platform from the [Dashboard \(p. 355\)](#), or by using a different client, Elastic Beanstalk uses the update type that you selected for [configuration changes \(p. 408\)](#).

If you don't have a managed update scheduled, your environment might already be running the latest version. Other reasons for not having an update scheduled include:

- A [minor version \(p. 423\)](#) update is available, but your environment is configured to automatically apply only patch version updates.

- Your environment hasn't been scanned since the update was released. Elastic Beanstalk typically checks for updates every hour.
- An update is pending or already in progress.

When your maintenance window starts or when you choose **Apply now**, scheduled updates go into pending status before execution.

Managed action option namespaces

You can use [configuration options \(p. 536\)](#) in the `aws:elasticbeanstalk:managedactions` (p. 578) and `aws:elasticbeanstalk:managedactions:platformupdate` (p. 578) namespaces to enable and configure managed platform updates.

The `ManagedActionsEnabled` option turns on managed platform updates. Set this option to `true` to enable managed platform updates, and use the other options to configure update behavior.

Use `PreferredStartTime` to configure the beginning of the weekly maintenance window in `day:hour:minute` format.

Set `UpdateLevel` to `minor` or `patch` to apply both minor and patch version updates, or just patch version updates, respectively.

When managed platform updates are enabled, you can enable instance replacement by setting the `InstanceRefreshEnabled` option to `true`. When this setting is enabled, Elastic Beanstalk runs an immutable update on your environment every week, regardless of whether there is a new platform version available.

The following example [configuration file \(p. 600\)](#) enables managed platform updates for patch version updates with a maintenance window starting at 9:00 AM UTC each Tuesday.

Example `.ebextensions/managed-platform-update.config`

```
option_settings:
  aws:elasticbeanstalk:managedactions:
    ManagedActionsEnabled: true
    PreferredStartTime: "Tue:09:00"
  aws:elasticbeanstalk:managedactions:platformupdate:
    UpdateLevel: patch
    InstanceRefreshEnabled: true
```

Migrating your application from a legacy platform version

If you have deployed an Elastic Beanstalk application that uses a legacy platform version, you should migrate your application to a new environment using a non-legacy platform version so that you can get access to new features. If you are unsure whether you are running your application using a legacy platform version, you can check in the Elastic Beanstalk console. For instructions, see [To check if you are using a legacy platform version \(p. 426\)](#).

What new features are legacy platform versions missing?

Legacy platforms do not support the following features:

- Configuration files, as described in the [Advanced environment customization with configuration files \(.ebextensions\) \(p. 600\)](#) topic
- ELB health checks, as described in the [Basic health reporting \(p. 688\)](#) topic

- Instance Profiles, as described in the [Managing Elastic Beanstalk instance profiles \(p. 760\)](#) topic
- VPCs, as described in the [Using Elastic Beanstalk with Amazon VPC \(p. 834\)](#) topic
- Data Tiers, as described in the [Adding a database to your Elastic Beanstalk environment \(p. 506\)](#) topic
- Worker Tiers, as described in the [Worker environments \(p. 15\)](#) topic
- Single Instance Environments, as described in the [Environment types \(p. 435\)](#) topic
- Tags, as described in the [Tagging resources in your Elastic Beanstalk environments \(p. 513\)](#) topic
- Rolling Updates, as described in the [Elastic Beanstalk rolling environment configuration updates \(p. 409\)](#) topic

Why are some platform versions marked legacy?

Some older platform versions do not support the latest Elastic Beanstalk features. These versions are marked **(legacy)** on the environment overview page in the Elastic Beanstalk console.

To check if you are using a legacy platform version

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, view the **Platform** name.

Your application is using a legacy platform version if you see **(legacy)** next to the platform's name.

To migrate your application

1. Deploy your application to a new environment. For instructions, go to [Creating an Elastic Beanstalk environment \(p. 363\)](#).
2. If you have an Amazon RDS DB Instance, update your database security group to allow access to your EC2 security group for your new environment. For instructions on how to find the name of your EC2 security group using the AWS Management Console, see [Security groups \(p. 455\)](#). For more information about configuring your EC2 security group, go to the "Authorizing Network Access to an Amazon EC2 Security Group" section of [Working with DB Security Groups](#) in the *Amazon Relational Database Service User Guide*.
3. Swap your environment URL. For instructions, go to [Blue/Green deployments with Elastic Beanstalk \(p. 405\)](#).
4. Terminate your old environment. For instructions, go to [Terminate an Elastic Beanstalk environment \(p. 386\)](#).

Note

If you use AWS Identity and Access Management (IAM) then you will need to update your policies to include AWS CloudFormation and Amazon RDS (if applicable). For more information, see [Using Elastic Beanstalk with AWS Identity and Access Management \(p. 759\)](#).

Migrating your Elastic Beanstalk Linux application to Amazon Linux 2

AWS provides two versions of Amazon Linux: [Amazon Linux 2](#) and [Amazon Linux AMI](#). AWS Elastic Beanstalk maintains platform branches with both Amazon Linux versions. For details about Linux platforms, see [the section called "Linux platforms" \(p. 30\)](#).

If your Elastic Beanstalk application is based on an Amazon Linux AMI platform branch, and the platform has newer generation Amazon Linux 2 platform branch(es), use this page to learn how to migrate your application's environments to Amazon Linux 2. The two platform generations aren't guaranteed to be backward compatible with your existing application. Furthermore, even if your application code successfully deploys to the new platform version, it might behave or perform differently due to operating system and run time differences. Although Amazon Linux AMI and Amazon Linux 2 share the same Linux kernel, they differ in their initialization system, `libc` versions, the compiler tool chain, and various packages. We've also updated platform specific versions of runtime, build tools, and other dependencies. Therefore we recommend that you take your time, test your application thoroughly in a development environment, and make any necessary adjustments.

When you're ready to go to production, Elastic Beanstalk requires a blue/green deployment to perform the upgrade. For details about platform update strategies, see [the section called "Platform updates" \(p. 415\)](#).

Important

If you're using an Amazon Linux 2 platform version that is in beta for your evaluation, *do not go to production*. Wait until we release a supported platform version. Beta platform versions aren't final, and we may change some naming and implementation details before we fully support these platforms.

Considerations for all Linux platforms

The following table discusses considerations you should be aware of when planning an application migration to Amazon Linux 2. These considerations apply to any of the Elastic Beanstalk Linux platforms, regardless of specific programming languages or application servers.

Area	Changes and information
Configuration Files	<p>On Amazon Linux 2 platforms, you can use configuration files (p. 600) as before, and all sections work the same way. However, specific settings might not work the same as they did on previous Amazon Linux AMI platforms. For example:</p> <ul style="list-style-type: none"> Some software packages that you install using a configuration file might not be available on Amazon Linux 2, or their names might have changed. Some platform specific configuration options have moved from their platform specific namespaces to different, platform agnostic namespaces. Proxy configuration files provided in the <code>.ebextensions/nginx</code> directory should move to the <code>.platform/nginx</code> platform hooks directory. For details, expand the <i>Reverse Proxy Configuration</i> section in the section called "Extending Linux platforms" (p. 31). <p>We recommend using platform hooks to run custom code on your environment instances. You can still use commands and container commands in <code>.ebextensions</code> configuration files, but they aren't as easy to work with. For example, writing command scripts inside a YAML file can be cumbersome and difficult to test.</p> <p>You still need to use <code>.ebextensions</code> configuration files for any script that needs a reference to an AWS CloudFormation resource.</p>
Platform hooks	<p>Amazon Linux 2 platforms introduce a new way to extend your environment's platform by adding executable files to hook directories on the environment's instances. With previous Linux platform versions, you might have used custom platform hooks (p. 44). These hooks weren't designed for managed platforms and weren't supported, but could work in useful ways in some cases. With Amazon Linux 2 platform versions, custom platform hooks don't work. You should migrate any hooks to the new platform hooks.</p>

Area	Changes and information
	For details, expand the <i>Platform Hooks</i> section in the section called “Extending Linux platforms” (p. 31) .
Supported proxy servers	Amazon Linux 2 platforms no longer support Apache HTTPD. They only support the nginx proxy server. If you have any HTTPD custom configuration, either files in the <code>.ebextensions/httpd</code> directory or commands in the <code>commands:</code> or <code>container_commands:</code> sections of an <code>.ebextensions</code> configuration file, modify them to use nginx instead. For information about nginx proxy configuration, expand the <i>Reverse Proxy Configuration</i> section in the section called “Extending Linux platforms” (p. 31) .
Instance profile	Amazon Linux 2 platforms require an instance profile to be configured. Environment creation might temporarily succeed without one, but the environment might show errors soon after creation when actions requiring an instance profile start failing. For details, see the section called “Instance profiles” (p. 760) .
Enhanced health	Amazon Linux 2 platform versions enable enhanced health by default. This is a change if you don't use the Elastic Beanstalk console to create your environments. The console enables enhanced health by default whenever possible, regardless of platform version. For details, see the section called “Enhanced health reporting and monitoring” (p. 691) .
Custom AMI	If your environment uses a custom AMI (p. 647) , create a new AMI based on Amazon Linux 2 for your new environment using an Elastic Beanstalk Amazon Linux 2 platform.
Custom platforms	The managed AMIs of Amazon Linux 2 platform versions don't support custom platforms (p. 38) .

Platform specific considerations

This section discusses migration considerations specific to particular Elastic Beanstalk Linux platforms.

Docker

The following table lists migration information for the Amazon Linux 2 platform versions in the [Docker platform \(p. 50\)](#).

Area	Changes and information
Storage	<p>Elastic Beanstalk configures Docker to use storage drivers to store Docker images and container data. On Amazon Linux AMI, Elastic Beanstalk used the Device Mapper storage driver. To improve performance, Elastic Beanstalk provisioned an extra Amazon EBS volume. On Amazon Linux 2 Docker platform versions, Elastic Beanstalk uses the OverlayFS storage driver, and achieves even better performance while not requiring a separate volume anymore.</p> <p>With Amazon Linux AMI, if you used the <code>BlockDeviceMappings</code> option of the <code>aws:autoscaling:launchconfiguration</code> namespace to add custom storage volumes to a Docker environment, we advised you to also add the <code>/dev/xvdcz</code> Amazon EBS volume that Elastic Beanstalk provisions. Elastic Beanstalk doesn't provision this volume anymore, so you should remove it from your configuration files. For details, see the section called “Docker configuration on Amazon Linux AMI (preceding Amazon Linux 2)” (p. 78).</p>

Area	Changes and information
Private Repository Authentication	When you provide a Docker-generated authentication file to connect to a private repository, you no longer need to convert it to the older format that Amazon Linux AMI Docker platform versions required. Amazon Linux 2 Docker platform versions support the new format. For details, see the section called "Using images from a private repository" (p. 76) .

Go

The following table lists migration information for the Amazon Linux 2 platform versions in the [Go platform \(p. 86\)](#).

Area	Changes and information
Port passing	On Amazon Linux 2 platforms, Elastic Beanstalk doesn't pass a port value to your application process through the <code>PORT</code> environment variable. You can simulate this behavior for your process by configuring a <code>PORT</code> environment property yourself. However, if you have multiple processes, and you're counting on Elastic Beanstalk passing incremental port values to your processes (5000, 5100, 5200 etc.), you should modify your implementation. For details, expand the <i>Reverse proxy configuration</i> section in the section called "Extending Linux platforms" (p. 31) .

Amazon Corretto

The following table lists migration information for the Corretto platform branches in the [Java SE platform \(p. 112\)](#).

Area	Changes and information
Corretto vs. OpenJDK	To implement the Java Platform, Standard Edition (Java SE), Amazon Linux 2 platform branches use Amazon Corretto , an AWS distribution of the Open Java Development Kit (OpenJDK). Prior Elastic Beanstalk Java SE platform branches use the OpenJDK packages included with Amazon Linux AMI.
Build tools	Amazon Linux 2 platforms have newer versions of the build tools: <code>gradle</code> , <code>maven</code> , and <code>ant</code> .
JAR file handling	On Amazon Linux 2 platforms, if your source bundle (ZIP file) contains a single JAR file and no other files, Elastic Beanstalk no longer renames the JAR file to <code>application.jar</code> . Renaming occurs only if you submit a JAR file on its own, not within a ZIP file.
Port passing	On Amazon Linux 2 platforms, Elastic Beanstalk doesn't pass a port value to your application process through the <code>PORT</code> environment variable. You can simulate this behavior for your process by configuring a <code>PORT</code> environment property yourself. However, if you have multiple processes, and you're counting on Elastic Beanstalk passing incremental port values to your processes (5000, 5100, 5200 etc.), you should modify your implementation. For details, expand the <i>Reverse proxy configuration</i> section in the section called "Extending Linux platforms" (p. 31) .
Java 7	Elastic Beanstalk doesn't support an Amazon Linux 2 Java 7 platform branch. If you have a Java 7 application, migrate it to Corretto 8 or Corretto 11.

Tomcat

The following table lists migration information for the Amazon Linux 2 platform versions in the [Tomcat platform \(p. 86\)](#).

Area	Changes and information						
Configuration options	<p>On Amazon Linux 2 platform versions, Elastic Beanstalk doesn't support the configuration options in the <code>aws:elasticbeanstalk:environment:proxy</code> namespace. Here's the migration information for each option.</p> <table border="1"> <thead> <tr> <th>Option</th> <th>Migration information</th> </tr> </thead> <tbody> <tr> <td><code>GzipCompression</code></td> <td>Isn't supported on Amazon Linux 2 platform versions.</td> </tr> <tr> <td><code>ProxyServer</code></td> <td> <p>Amazon Linux 2 Tomcat platform versions only support nginx. The Apache proxy server isn't supported.</p> <p>On Amazon Linux AMI platform versions the default proxy was Apache 2.4. If you were using the default proxy, and you added custom configuration, you now have to migrate your proxy configuration to nginx. If, however, you were already using nginx through an option setting, your proxy configuration should still work on Amazon Linux 2.</p> </td> </tr> </tbody> </table> <p>The <code>xx:MaxPermSize</code> option in the <code>aws:elasticbeanstalk:container:tomcat:jvmoptions</code> namespace isn't supported on Amazon Linux 2 platform versions. The JVM setting to modify the size of the permanent generation applies only to Java 7 and earlier, and is therefore not applicable to Amazon Linux 2 platform versions.</p>	Option	Migration information	<code>GzipCompression</code>	Isn't supported on Amazon Linux 2 platform versions.	<code>ProxyServer</code>	<p>Amazon Linux 2 Tomcat platform versions only support nginx. The Apache proxy server isn't supported.</p> <p>On Amazon Linux AMI platform versions the default proxy was Apache 2.4. If you were using the default proxy, and you added custom configuration, you now have to migrate your proxy configuration to nginx. If, however, you were already using nginx through an option setting, your proxy configuration should still work on Amazon Linux 2.</p>
Option	Migration information						
<code>GzipCompression</code>	Isn't supported on Amazon Linux 2 platform versions.						
<code>ProxyServer</code>	<p>Amazon Linux 2 Tomcat platform versions only support nginx. The Apache proxy server isn't supported.</p> <p>On Amazon Linux AMI platform versions the default proxy was Apache 2.4. If you were using the default proxy, and you added custom configuration, you now have to migrate your proxy configuration to nginx. If, however, you were already using nginx through an option setting, your proxy configuration should still work on Amazon Linux 2.</p>						
Application path	<p>On Amazon Linux 2 platforms, the path to the application's directory on Amazon EC2 instances of your environment is <code>/var/app/current</code>. It was <code>/var/lib/tomcat8/webapps</code> on Amazon Linux AMI platforms.</p>						

Node.js

The following table lists migration information for the Amazon Linux 2 platform versions in the [Node.js platform \(p. 195\)](#).

Area	Changes and information
Installed Node.js versions	<p>On Amazon Linux 2 platforms, Elastic Beanstalk maintains several Node.js platform branches, and only installs the latest version of the Node.js major version corresponding with the platform branch on each platform version. For example, each platform version in the Node.js 12 platform branch only has Node.js 12.x.y installed by default. On Amazon Linux AMI platform versions, we installed the multiple versions of multiple Node.js versions on each platform version, and only maintained a single platform branch.</p> <p>Choose the Node.js platform branch that corresponds with the Node.js major version that your application needs.</p>
Configuration options	<p>On Amazon Linux 2 platforms, Elastic Beanstalk doesn't support the configuration options in the <code>aws:elasticbeanstalk:container:nodejs</code> namespace. Some of the options have alternatives. Here's the migration information for each option.</p>

Area	Changes and information										
	<table border="1"> <thead> <tr> <th>Option</th> <th>Migration information</th> </tr> </thead> <tbody> <tr> <td>NodeCommand</td> <td>Use a Procfile or the <code>scripts</code> keyword in a <code>package.json</code> file to specify the start script.</td> </tr> <tr> <td>NodeVersion</td> <td>Use the <code>engines</code> keyword in a <code>package.json</code> file to specify the Node.js version. Be aware that you can only specify a Node.js version that corresponds with your platform branch. For example, if you're using the Node.js 12 platform branch, you can specify only a 12.x.y Node.js version. For details, see the section called "Specifying Node.js dependencies with a package.json file" (p. 200).</td> </tr> <tr> <td>GzipCompression</td> <td>Not supported on Amazon Linux 2 platform versions.</td> </tr> <tr> <td>ProxyServer</td> <td> <p>Amazon Linux 2 Node.js platform versions only support nginx. The Apache proxy server isn't supported.</p> <p>In addition, these platform versions don't support standalone applications that don't run behind a proxy server. This used to be possible through the <code>none</code> value of the <code>ProxyServer</code> option. If your environment runs a standalone application, update your code to listen to the port that nginx forwards traffic to.</p> <pre>var port = process.env.PORT 8080; app.listen(port, function() { console.log('Server running at http://127.0.0.1:%s', port); });</pre> </td> </tr> </tbody> </table>	Option	Migration information	NodeCommand	Use a Procfile or the <code>scripts</code> keyword in a <code>package.json</code> file to specify the start script.	NodeVersion	Use the <code>engines</code> keyword in a <code>package.json</code> file to specify the Node.js version. Be aware that you can only specify a Node.js version that corresponds with your platform branch. For example, if you're using the Node.js 12 platform branch, you can specify only a 12.x.y Node.js version. For details, see the section called "Specifying Node.js dependencies with a package.json file" (p. 200).	GzipCompression	Not supported on Amazon Linux 2 platform versions.	ProxyServer	<p>Amazon Linux 2 Node.js platform versions only support nginx. The Apache proxy server isn't supported.</p> <p>In addition, these platform versions don't support standalone applications that don't run behind a proxy server. This used to be possible through the <code>none</code> value of the <code>ProxyServer</code> option. If your environment runs a standalone application, update your code to listen to the port that nginx forwards traffic to.</p> <pre>var port = process.env.PORT 8080; app.listen(port, function() { console.log('Server running at http://127.0.0.1:%s', port); });</pre>
Option	Migration information										
NodeCommand	Use a Procfile or the <code>scripts</code> keyword in a <code>package.json</code> file to specify the start script.										
NodeVersion	Use the <code>engines</code> keyword in a <code>package.json</code> file to specify the Node.js version. Be aware that you can only specify a Node.js version that corresponds with your platform branch. For example, if you're using the Node.js 12 platform branch, you can specify only a 12.x.y Node.js version. For details, see the section called "Specifying Node.js dependencies with a package.json file" (p. 200).										
GzipCompression	Not supported on Amazon Linux 2 platform versions.										
ProxyServer	<p>Amazon Linux 2 Node.js platform versions only support nginx. The Apache proxy server isn't supported.</p> <p>In addition, these platform versions don't support standalone applications that don't run behind a proxy server. This used to be possible through the <code>none</code> value of the <code>ProxyServer</code> option. If your environment runs a standalone application, update your code to listen to the port that nginx forwards traffic to.</p> <pre>var port = process.env.PORT 8080; app.listen(port, function() { console.log('Server running at http://127.0.0.1:%s', port); });</pre>										

PHP

The following table lists migration information for the Amazon Linux 2 platform versions in the [PHP platform](#) (p. 230).

Area	Changes and information
PHP file processing	On Amazon Linux 2 platforms, PHP files are processed using PHP-FPM (a CGI process manager). On Amazon Linux AMI platforms we used <code>mod_php</code> (an Apache module).

Python

The following table lists migration information for the Amazon Linux 2 platform versions in the [Python platform](#) (p. 290).

Area	Changes and information
WSGI server	On Amazon Linux 2 platforms, Gunicorn is the default WSGI server. If you're setting the <code>WSGIPath</code> option of the <code>aws:elasticbeanstalk:container:python</code> (p. 597) namespace, replace the value with Gunicorn's syntax. For details, see the section called "Python configuration namespaces" (p. 292).

Area	Changes and information
	Alternatively, you can use a <code>Procfile</code> to specify and configure the WSGI server. For details, see the section called “Procfile” (p. 293) .
Application path	On Amazon Linux 2 platforms, the path to the application's directory on Amazon EC2 instances of your environment is <code>/var/app/current</code> . It was <code>/opt/python/current/app</code> on Amazon Linux AMI platforms.

Ruby

The following table lists migration information for the Amazon Linux 2 platform versions in the [Ruby platform \(p. 316\)](#).

Area	Changes and information
Installed Ruby versions	<p>On Amazon Linux 2 platforms, Elastic Beanstalk only installs the latest version of a single Ruby version, corresponding with the platform branch, on each platform version. For example, each platform version in the Ruby 2.6 platform branch only has Ruby 2.6.x installed. On Amazon Linux AMI platform versions, we installed the latest versions of multiple Ruby versions, for example, 2.4.x, 2.5.x, and 2.6.x.</p> <p>If your application uses a Ruby version that doesn't correspond to the platform branch you're using, we recommend that you switch to a platform branch that has the correct Ruby version for your application.</p>
Application server	<p>On Amazon Linux 2 platforms, Elastic Beanstalk only installs the Puma application server on all Ruby platform versions. You can use a <code>Procfile</code> to start a different application server, and a <code>Gemfile</code> to install it.</p> <p>On the Amazon Linux AMI platform, we supported two flavors of platform branches for each Ruby version—one with the Puma application server and the other with the Passenger application server. If your application uses Passenger, you can configure your Ruby environment to install and use Passenger.</p> <p>For more information and examples, see the section called “The Ruby platform” (p. 316).</p>

Canceling environment configuration updates and application deployments

You can cancel in-progress updates that are triggered by environment configuration changes. You can also cancel the deployment of a new application version in progress. For example, you might want to cancel an update if you decide you want to continue using the existing environment configuration instead of applying new environment configuration settings. Or, you might realize that the new application version that you are deploying has problems that will cause it to not start or not run properly. By canceling an environment update or application version update, you can avoid waiting until the update or deployment process is done before you begin a new attempt to update the environment or application version.

Note

During the cleanup phase in which old resources that are no longer needed are removed, after the last batch of instances has been updated, you can no longer cancel the update.

Elastic Beanstalk performs the rollback the same way that it performed the last successful update. For example, if you have time-based rolling updates enabled in your environment, then Elastic Beanstalk will wait the specified pause time between rolling back changes on one batch of instances before rolling back changes on the next batch. Or, if you recently turned on rolling updates, but the last time you successfully updated your environment configuration settings was without rolling updates, Elastic Beanstalk will perform the rollback on all instances simultaneously.

You cannot stop Elastic Beanstalk from rolling back to the previous environment configuration once it begins to cancel the update. The rollback process continues until all instances in the environment have the previous environment configuration or until the rollback process fails. For application version deployments, canceling the deployment simply stops the deployment; some instances will have the new application version and others will continue to run the existing application version. You can deploy the same or another application version later.

For more information about rolling updates, see [Elastic Beanstalk rolling environment configuration updates \(p. 409\)](#). For more information about batched application version deployments, see [Deployment policies and settings \(p. 400\)](#).

To cancel an update

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Environment actions**, and then choose **Abort current operation**.

Rebuilding Elastic Beanstalk environments

Your AWS Elastic Beanstalk environment can become unusable if you don't use Elastic Beanstalk functionality to modify or terminate the environment's underlying AWS resources. If this happens, you can **rebuild** the environment to attempt to restore it to a working state. Rebuilding an environment terminates all of its resources and replaces them with new resources with the same configuration.

You can also rebuild terminated environments within six weeks (42 days) of their termination. When you rebuild, Elastic Beanstalk attempts to create a new environment with the same name, ID, and configuration.

Rebuilding a running environment

You can rebuild an environment through the Elastic Beanstalk console or by using the `RebuildEnvironment` API.

To rebuild a running environment (console)

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Rebuild environment**.
4. Choose **Rebuild**.

Rebuilding a running environment creates new resources that have the same configuration as the old resources; however, the resource IDs are different, and any data on the old resources is not restored. For example, rebuilding an environment with an Amazon RDS database instance creates a new database with the same configuration, but does not apply a snapshot to the new database.

To rebuild a running environment with the Elastic Beanstalk API, use the `RebuildEnvironment` action with the AWS CLI or the AWS SDK.

```
$ aws elasticbeanstalk rebuild-environment --environment-id e-vdnftxbwq
```

Rebuilding a terminated environment

You can rebuild and restore a terminated environment by using the Elastic Beanstalk console, the EB CLI, or the `RebuildEnvironment` API.

Note

Unless you are using your own custom domain name with your terminated environment, the environment uses a subdomain of `elasticbeanstalk.com`. These subdomains are shared within an Elastic Beanstalk region. Therefore, they can be used by any environment created by any customer in the same region. While your environment was terminated, another environment could use its subdomain. In this case, the rebuild would fail.

You can avoid this issue by using a custom domain. See [Your Elastic Beanstalk environment's Domain name \(p. 534\)](#) for details.

Recently terminated environments appear in the application overview for up to an hour. During this time, you can view events for the environment in its [dashboard \(p. 353\)](#), and use the **Restore environment action (p. 356)** to rebuild it.

To rebuild an environment that is no longer visible, use the **Restore terminated environment** option from the application page.

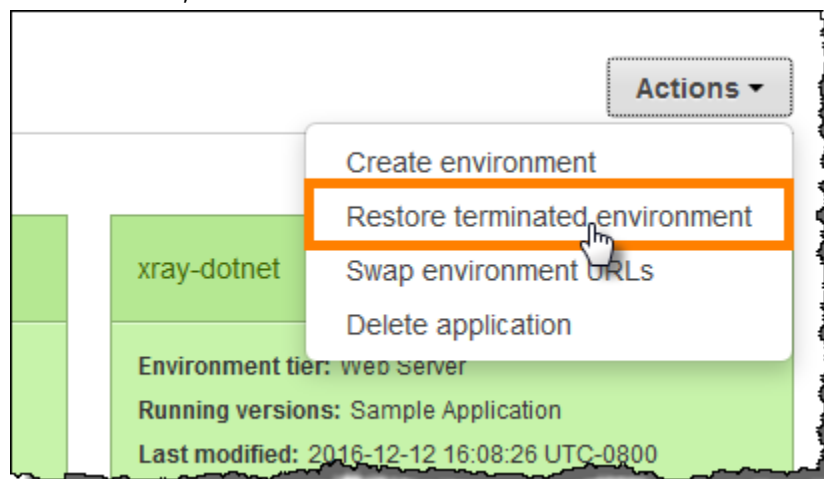
To rebuild a terminated environment (console)

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.

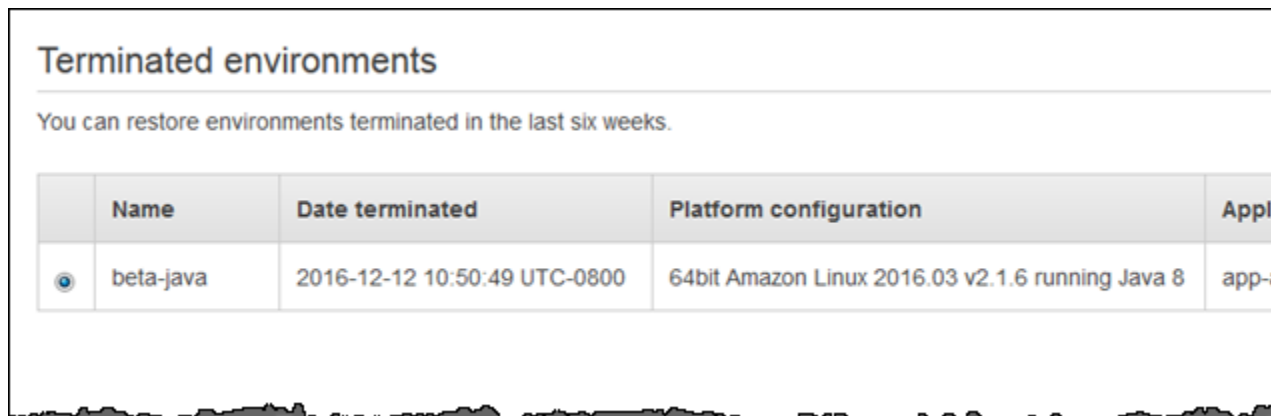
Note

If you have many applications, use the search bar to filter the application list.

3. Choose **Actions**, and then choose **Restore terminated environment**.



4. Choose a terminated environment.
5. Choose **Restore**.



Elastic Beanstalk attempts to create a new environment with the same name, ID, and configuration. If an environment with the same name or URL exists when you attempt to rebuild, the rebuild fails. Deleting the application version that was deployed to the environment will also cause the rebuild to fail.

If you use the EB CLI to manage your environment, use the **eb restore** command to rebuild a terminated environment.

```
$ eb restore e-vdnftxubwq
```

See [eb restore](#) (p. 924) for more information.

To rebuild a terminated environment with the Elastic Beanstalk API, use the [RebuildEnvironment](#) action with the AWS CLI or the AWS SDK.

```
$ aws elasticbeanstalk rebuild-environment --environment-id e-vdnftxubwq
```

Environment types

In AWS Elastic Beanstalk, you can create a load-balancing, autoscaling environment or a single-instance environment. The type of environment that you require depends on the application that you deploy. For example, you can develop and test an application in a single-instance environment to save costs and then upgrade that environment to a load-balancing, autoscaling environment when the application is ready for production.

Note

A worker environment tier for a web application that processes background tasks doesn't include a load balancer. However, a worker environment does effectively scale out by adding instances to the Auto Scaling group to process data from the Amazon SQS queue when the load necessitates it.

Load-balancing, autoscaling environment

A load-balancing and autoscaling environment uses the Elastic Load Balancing and Amazon EC2 Auto Scaling services to provision the Amazon EC2 instances that are required for your deployed application.

Amazon EC2 Auto Scaling automatically starts additional instances to accommodate increasing load on your application. If the load on your application decreases, Amazon EC2 Auto Scaling stops instances but always leaves your specified minimum number of instances running. If your application requires scalability with the option of running in multiple Availability Zones, use a load-balancing, autoscaling environment. If you're not sure which environment type to select, you can pick one and, if required, switch the environment type later.

Single-instance environment

A single-instance environment contains one Amazon EC2 instance with an Elastic IP address. A single-instance environment doesn't have a load balancer, which can help you reduce costs compared to a load-balancing, autoscaling environment. Although a single-instance environment does use the Amazon EC2 Auto Scaling service, settings for the minimum number of instances, maximum number of instances, and desired capacity are all set to 1. Consequently, new instances are not started to accommodate increasing load on your application.

Use a single-instance environment if you expect your production application to have low traffic or if you are doing remote development. If you're not sure which environment type to select, you can pick one and, if required, you can switch the environment type later. For more information, see [Changing environment type](#) (p. 436).

Changing environment type

You can change your environment type to a single-instance or load-balancing, autoscaling environment by editing your environment's configuration. In some cases, you might want to change your environment type from one type to another. For example, let's say that you developed and tested an application in a single-instance environment to save costs. When your application is ready for production, you can change the environment type to a load-balancing, autoscaling environment so that it can scale to meet the demands of your customers.

To change an environment's type

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** category, choose **Edit**.
5. From the **Environment Type** list, select the type of environment that you want.

The screenshot shows the 'Modify capacity' configuration page in the AWS Elastic Beanstalk console. The breadcrumb navigation at the top reads 'Elastic Beanstalk > Environments > GettingStartedApp-env > Configuration'. The main heading is 'Modify capacity', followed by a sub-heading 'Auto Scaling Group'. Below this, there are several configuration sections: 'Environment type' is set to 'Load balanced'; 'Instances' are configured with a 'Min' of 1 and a 'Max' of 2; 'Fleet composition' has 'On-Demand instances' selected with a radio button; and 'Maximum spot price' is present with a descriptive note. The page is partially cut off on the right side.

6. Choose **Save**.

It can take several minutes for the environment to update while Elastic Beanstalk provisions AWS resources.

If your environment is in a VPC, select subnets to place Elastic Load Balancing and Amazon EC2 instances in. Each Availability Zone that your application runs in must have both. See [Using Elastic Beanstalk with Amazon VPC \(p. 834\)](#) for details.

Elastic Beanstalk worker environments

If your AWS Elastic Beanstalk application performs operations or workflows that take a long time to complete, you can offload those tasks to a dedicated *worker environment*. Decoupling your web application front end from a process that performs blocking operations is a common way to ensure that your application stays responsive under load.

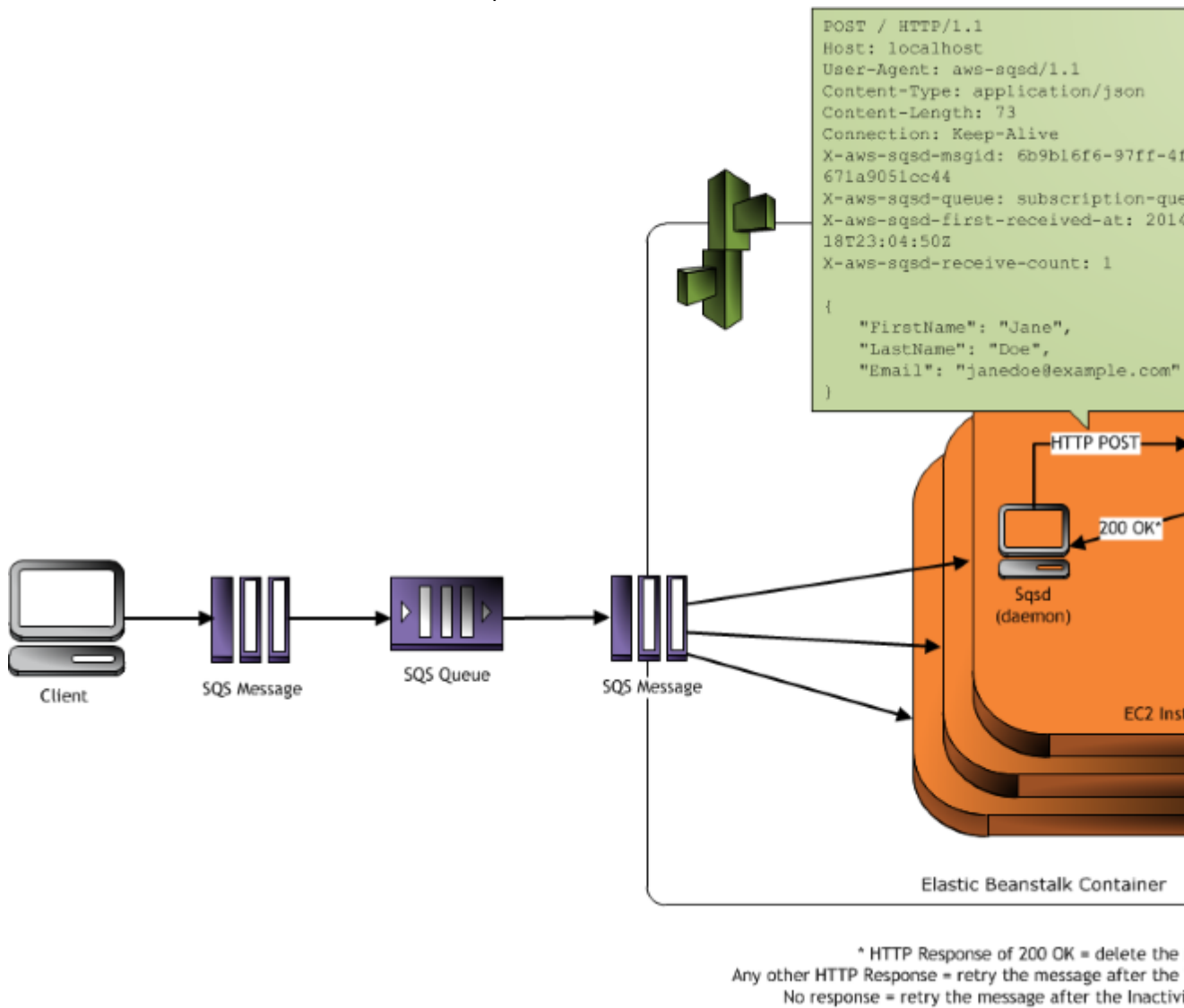
A long-running task is anything that substantially increases the time it takes to complete a request, such as processing images or videos, sending email, or generating a ZIP archive. These operations can take only a second or two to complete, but a delay of a few seconds is a lot for a web request that would otherwise complete in less than 500 ms.

One option is to spawn a worker process locally, return success, and process the task asynchronously. This works if your instance can keep up with all of the tasks sent to it. Under high load, however, an

instance can become overwhelmed with background tasks and become unresponsive to higher priority requests. If individual users can generate multiple tasks, the increase in load might not correspond to an increase in users, making it hard to scale out your web server tier effectively.

To avoid running long-running tasks locally, you can use the AWS SDK for your programming language to send them to an Amazon Simple Queue Service (Amazon SQS) queue, and run the process that performs them on a separate set of instances. You then design these worker instances to take items from the queue only when they have capacity to run them, preventing them from becoming overwhelmed.

Elastic Beanstalk worker environments simplify this process by managing the Amazon SQS queue and running a [daemon process \(p. 439\)](#) on each instance that reads from the queue for you. When the daemon pulls an item from the queue, it sends an HTTP POST request locally to `http://localhost/` on port 80 with the contents of the queue message in the body. All that your application needs to do is perform the long-running task in response to the POST. You can [configure the daemon \(p. 441\)](#) to post to a different path, use a MIME type other than `application/JSON`, connect to an existing queue, or customize connections (maximum concurrent requests), timeouts, and retries.



With [periodic tasks \(p. 440\)](#), you can also configure the worker daemon to queue messages based on a cron schedule. Each periodic task can POST to a different path. Enable periodic tasks by including a YAML file in your source code that defines the schedule and path for each task.

Note

The [.NET on Windows Server platform \(p. 137\)](#) doesn't support worker environments.

Sections

- [The worker environment SQS daemon \(p. 439\)](#)
- [Dead-letter queues \(p. 440\)](#)
- [Periodic tasks \(p. 440\)](#)
- [Use Amazon CloudWatch for automatic scaling in worker environment tiers \(p. 441\)](#)
- [Configuring worker environments \(p. 441\)](#)

The worker environment SQS daemon

Worker environments run a daemon process provided by Elastic Beanstalk. This daemon is updated regularly to add features and fix bugs. To get the latest version of the daemon, update to the latest [platform version \(p. 29\)](#).

When the application in the worker environment returns a 200 OK response to acknowledge that it has received and successfully processed the request, the daemon sends a `DeleteMessage` call to the Amazon SQS queue to delete the message from the queue. If the application returns any response other than 200 OK, Elastic Beanstalk waits to put the message back in the queue after the configured `ErrorVisibilityTimeout` period. If there is no response, Elastic Beanstalk waits to put the message back in the queue after the `InactivityTimeout` period so that the message is available for another attempt at processing.

Note

The properties of Amazon SQS queues (message order, at-least-once delivery, and message sampling) can affect how you design a web application for a worker environment. For more information, see [Properties of Distributed Queues](#) in the [Amazon Simple Queue Service Developer Guide](#).

Amazon SQS automatically deletes messages that have been in a queue for longer than the configured `RetentionPeriod`.

The daemon sets the following HTTP headers.

HTTP headers	
Name	Value
User-Agent	aws-sqs-d aws-sqs-d/1.11
X-Aws-Sqs-Msgid	SQS message ID, used to detect message storms (an unusually high number of new messages).
X-Aws-Sqs-Queue	Name of the SQS queue.
X-Aws-Sqs-First-Received-At	UTC time, in ISO 8601 format , when the message was first received.
X-Aws-Sqs-Receive-Count	SQS message receive count.
X-Aws-Sqs-Attr- <i>message-attribute-name</i>	Custom message attributes assigned to the message being processed. The <code>message-attribute-name</code> is the actual message attribute name. All string

HTTP headers	
	and number message attributes are added to the header. Binary attributes are discarded and not included in the header.
Content-Type	Mime type configuration; by default, <code>application/json</code> .

Dead-letter queues

Elastic Beanstalk worker environments support Amazon Simple Queue Service (Amazon SQS) dead-letter queues. A dead-letter queue is a queue where other (source) queues can send messages that for some reason could not be successfully processed. A primary benefit of using a dead-letter queue is the ability to sideline and isolate the unsuccessfully processed messages. You can then analyze any messages sent to the dead-letter queue to try to determine why they were not successfully processed.

If you specify an autogenerated Amazon SQS queue at the time you create your worker environment tier, a dead-letter queue is enabled by default for a worker environment. If you choose an existing SQS queue for your worker environment, you must use SQS to configure a dead-letter queue independently. For information about how to use SQS to configure a dead-letter queue, see [Using Amazon SQS Dead Letter Queues](#).

You cannot disable dead-letter queues. Messages that cannot be delivered are always eventually sent to a dead-letter queue. You can, however, effectively disable this feature by setting the `MaxRetries` option to the maximum valid value of 100.

If a dead-letter queue isn't configured for your worker environment's Amazon SQS queue, Amazon SQS keeps messages on the queue until the retention period expires. For details about configuring the retention period, see [the section called "Configuring worker environments" \(p. 441\)](#).

Note

The Elastic Beanstalk `MaxRetries` option is equivalent to the SQS `MaxReceiveCount` option. If your worker environment doesn't use an autogenerated SQS queue, use the `MaxReceiveCount` option in SQS to effectively disable your dead-letter queue. For more information, see [Using Amazon SQS Dead Letter Queues](#).

For more information about the lifecycle of an SQS message, go to [Message Lifecycle](#).

Periodic tasks

You can define periodic tasks in a file named `cron.yaml` in your source bundle to add jobs to your worker environment's queue automatically at a regular interval.

For example, the following `cron.yaml` file creates two periodic tasks. The first one runs every 12 hours and the second one runs at 11 PM UTC every day.

Example `cron.yaml`

```
version: 1
cron:
  - name: "backup-job"
    url: "/backup"
    schedule: "0 */12 * * *"
  - name: "audit"
    url: "/audit"
    schedule: "0 23 * * *"
```

The **name** must be unique for each task. The URL is the path to which the POST request is sent to trigger the job. The schedule is a [CRON expression](#) that determines when the task runs.

When a task runs, the daemon posts a message to the environment's SQS queue with a header indicating the job that needs to be performed. Any instance in the environment can pick up the message and process the job.

Note

If you configure your worker environment with an existing SQS queue and choose an [Amazon SQS FIFO queue](#), periodic tasks aren't supported.

Elastic Beanstalk uses leader election to determine which instance in your worker environment queues the periodic task. Each instance attempts to become leader by writing to an Amazon DynamoDB table. The first instance that succeeds is the leader, and must continue to write to the table to maintain leader status. If the leader goes out of service, another instance quickly takes its place.

For periodic tasks, the worker daemon sets the following additional headers.

HTTP headers	
Name	Value
X-Aws-Sqsd-Taskname	For periodic tasks, the name of the task to perform.
X-Aws-Sqsd-Scheduled-At	Time at which the periodic task was scheduled
X-Aws-Sqsd-Sender-Id	AWS account number of the sender of the message

Use Amazon CloudWatch for automatic scaling in worker environment tiers

Together, Amazon EC2 Auto Scaling and CloudWatch monitor the CPU utilization of the running instances in the worker environment. How you configure the automatic scaling limit for CPU capacity determines how many instances the Auto Scaling group runs to appropriately manage the throughput of messages in the Amazon SQS queue. Each EC2 instance publishes its CPU utilization metrics to CloudWatch. Amazon EC2 Auto Scaling retrieves from CloudWatch the average CPU usage across all instances in the worker environment. You configure the upper and lower threshold as well as how many instances to add or terminate according to CPU capacity. When Amazon EC2 Auto Scaling detects that you have reached the specified upper threshold on CPU capacity, Elastic Beanstalk creates new instances in the worker environment. The instances are deleted when the CPU load drops back below the threshold.

Note

Messages that have not been processed at the time an instance is terminated are returned to the queue where they can be processed by another daemon on an instance that is still running.

You can also set other CloudWatch alarms, as needed, by using the Elastic Beanstalk console, CLI, or the options file. For more information, see [Using Elastic Beanstalk with Amazon CloudWatch \(p. 742\)](#) and [Create an Auto Scaling group with Step Scaling Policies](#).

Configuring worker environments

You can manage a worker environment's configuration by editing the **Worker** category on the **Configuration** page in the [environment management console \(p. 353\)](#).

Modify worker

You can create a new Amazon SQS queue for your worker application or pull work items from an existing queue. The worker daemon instances in your environment pulls an item from the queue and relays it in the body of a POST request to a local HTTP path relative to the

Queue

Worker queue

Autogenerated queue



SQS queue from which to read work items.

Messages

HTTP path

/

The daemon pulls items from the Amazon SQS queue and posts them locally to this path.

MIME type

application/json

Change the MIME type of the POST requests that the worker daemon sends to your application.

HTTP connections

50

Maximum number of concurrent connections to the application.

Visibility timeout

300

seconds

The amount of time to lock an incoming message for processing before returning it to the queue.

Error visibility timeout

seconds

The amount of time to wait before resending a message after an error response from the application.

▼ Advanced options

The following settings control advanced behavior of the worker tier daemon. [Learn more](#)

Max retries

10

442

Maximum number of retries after which the message is discarded.

Connection timeout

Note

You can configure the URL path for posting worker queue messages, but you can't configure the IP port. Elastic Beanstalk always posts worker queue messages on port 80. The worker environment application or its proxy must listen to port 80.

To configure the worker daemon

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Worker** configuration category, choose **Edit**.

The **Modify worker** configuration page has the following options.

In the **Queue** section:

- **Worker queue** – Specify the Amazon SQS queue from which the daemon reads. If you have one, you can choose an existing queue. If you choose **Autogenerated queue**, Elastic Beanstalk creates a new Amazon SQS queue and a corresponding **Worker queue URL**.

Note

When you choose **Autogenerated queue**, the queue that Elastic Beanstalk creates is a [standard](#) Amazon SQS queue. When you choose an existing queue, you can provide either a standard or a [FIFO](#) Amazon SQS queue. Be aware that if you provide a FIFO queue, [periodic tasks](#) (p. 440) aren't supported.

- **Worker queue URL** – If you choose an existing **Worker queue**, this setting displays the URL associated with that Amazon SQS queue.

In the **Messages** section:

- **HTTP path** – Specify the relative path to the application that will receive the data from the Amazon SQS queue. The data is inserted into the message body of an HTTP POST message. The default value is `/`.
- **MIME type** – Indicate the MIME type that the HTTP POST message uses. The default value is `application/json`. However, any value is valid because you can create and then specify your own MIME type.
- **HTTP connections** – Specify the maximum number of concurrent connections that the daemon can make to any application within an Amazon EC2 instance. The default is **50**. You can specify **1** to **100**.
- **Visibility timeout** – Indicate the amount of time, in seconds, an incoming message from the Amazon SQS queue is locked for processing. After the configured amount of time has passed, the message is again made visible in the queue for another daemon to read. Choose a value that is longer than you expect your application requires to process messages, up to **43200** seconds.
- **Error visibility timeout** – Indicate the amount of time, in seconds, that elapses before Elastic Beanstalk returns a message to the Amazon SQS queue after an attempt to process it fails with an explicit error. You can specify **0** to **43200** seconds.

In the **Advanced options** section:

- **Max retries** – Specify the maximum number of times Elastic Beanstalk attempts to send the message to the Amazon SQS queue before moving the message to the [dead-letter queue](#) (p. 440). The default value is **10**. You can specify **1** to **100**.

Note

The **Max retries** option doesn't apply if your worker environment's Amazon SQS queues doesn't have a configured dead-letter queue. In such a case, Amazon SQS retains messages in the queue and processes them until the period specified by the **Retention period** option expires.

- **Connection timeout** – Indicate the amount of time, in seconds, to wait for successful connections to an application. The default value is **5**. You can specify **1** to **60** seconds.
- **Inactivity timeout** – Indicate the amount of time, in seconds, to wait for a response on an existing connection to an application. The default value is **180**. You can specify **1** to **36000** seconds.
- **Retention period** – Indicate the amount of time, in seconds, a message is valid and is actively processed. The default value is **345600**. You can specify **60** to **1209600** seconds.

If you use an existing Amazon SQS queue, the settings that you configure when you create a worker environment can conflict with settings you configured directly in Amazon SQS. For example, if you configure a worker environment with a `RetentionPeriod` value that is higher than the `MessageRetentionPeriod` value you set in Amazon SQS, Amazon SQS deletes the message when it exceeds the `MessageRetentionPeriod`.

Conversely, if the `RetentionPeriod` value you configure in the worker environment settings is lower than the `MessageRetentionPeriod` value you set in Amazon SQS, the daemon deletes the message before Amazon SQS can. For `VisibilityTimeout`, the value that you configure for the daemon in the worker environment settings overrides the Amazon SQS `VisibilityTimeout` setting. Ensure that messages are deleted appropriately by comparing your Elastic Beanstalk settings to your Amazon SQS settings.

Creating links between Elastic Beanstalk environments

As your application grows in size and complexity, you may want to split it into components that have different development and operational lifecycles. By running smaller services that interact with each other over a well defined interface, teams can work independently and deployments can be lower risk. AWS Elastic Beanstalk lets you link your environments to share information between components that depend on one another.

Note

Elastic Beanstalk currently supports environment links for all platforms except Multicontainer Docker.

With environment links, you can specify the connections between your application's component environments as named references. When you create an environment that defines a link, Elastic Beanstalk sets an environment variable with the same name as the link. The value of the variable is the endpoint that you can use to connect to the other component, which can be a web server or worker environment.

For example, if your application consists of a frontend that collects email addresses and a worker that sends a welcome email to the email addresses collected by the frontend, you can create a link to the worker in your frontend and have the frontend automatically discover the endpoint (queue URL) for your worker.

Define links to other environments in an [environment manifest \(p. 645\)](#), a YAML formatted file named `env.yaml` in the root of your application source. The following manifest defines a link to an environment named worker:

```
~/workspace/my-app/frontend/env.yaml
```

```
AWSConfigurationTemplateVersion: 1.1.0.0
EnvironmentLinks:
  "WORKERQUEUE": "worker"
```

When you create an environment with an application version that includes the above environment manifest, Elastic Beanstalk looks for an environment named `worker` that belongs to the same application. If that environment exists, Elastic Beanstalk creates an environment property named `WORKERQUEUE`. The value of `WORKERQUEUE` is the Amazon SQS queue URL. The frontend application can read this property in the same manner as an environment variable. See [Environment manifest \(env.yaml\)](#) (p. 645) for details.

To use environment links, add an environment manifest to your application source and upload it with the EB CLI, AWS CLI or an SDK. If you use the AWS CLI or an SDK, set the `process` flag when you call `CreateApplicationVersion`:

```
$ aws elasticbeanstalk create-application-version --process --application-name my-app --
version-label frontend-v1 --source-bundle S3Bucket="my-bucket",S3Key="front-v1.zip"
```

This option tells Elastic Beanstalk to validate the environment manifest and configuration files in your source bundle when you create the application version. The EB CLI sets this flag automatically when you have an environment manifest in your project directory.

Create your environments normally using any client. When you need to terminate environments, terminate the environment with the link first. If an environment is linked to by another environment, Elastic Beanstalk will prevent the linked environment from being terminated. To override this protection, use the `ForceTerminate` flag. This parameter is available in the AWS CLI as `--force-terminate`:

```
$ aws elasticbeanstalk terminate-environment --force-terminate --environment-name worker
```

Configuring Elastic Beanstalk environments

AWS Elastic Beanstalk provides a wide range of options for customizing the resources in your environment, and Elastic Beanstalk behavior and platform settings. When you create a web server environment, Elastic Beanstalk creates several resources to support the operation of your application.

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or nginx as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the [AWS CloudFormation console](#).
- **Domain name** – A domain name that routes to your web app in the form `subdomain.region.elasticbeanstalk.com`.

This topic focuses on the resource configuration options available in the Elastic Beanstalk console. The following topics show how to configure your environment in the console. They also describe the underlying namespaces that correspond to the console options for use with configuration files or API configuration options. To learn about advanced configuration methods, see [Configuring environments \(advanced\)](#) (p. 536).

Topics

- [Environment configuration using the Elastic Beanstalk console](#) (p. 447)
- [Your Elastic Beanstalk environment's Amazon EC2 instances](#) (p. 451)

- [Auto Scaling group for your Elastic Beanstalk environment \(p. 457\)](#)
- [Load balancer for your Elastic Beanstalk environment \(p. 470\)](#)
- [Adding a database to your Elastic Beanstalk environment \(p. 506\)](#)
- [Your AWS Elastic Beanstalk environment security \(p. 510\)](#)
- [Tagging resources in your Elastic Beanstalk environments \(p. 513\)](#)
- [Environment properties and other software settings \(p. 516\)](#)
- [Elastic Beanstalk environment notifications with Amazon SNS \(p. 526\)](#)
- [Configuring Amazon Virtual Private Cloud \(Amazon VPC\) with Elastic Beanstalk \(p. 530\)](#)
- [Your Elastic Beanstalk environment's Domain name \(p. 534\)](#)

Environment configuration using the Elastic Beanstalk console

You can use the Elastic Beanstalk console to view and modify many [configuration options \(p. 536\)](#) of your environment and its resources. You can customize how the environment behaves during deployments, enable additional features, and modify the instance type and other settings that you chose during environment creation.

To view a summary of your environment's configuration

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.

Configuration overview page

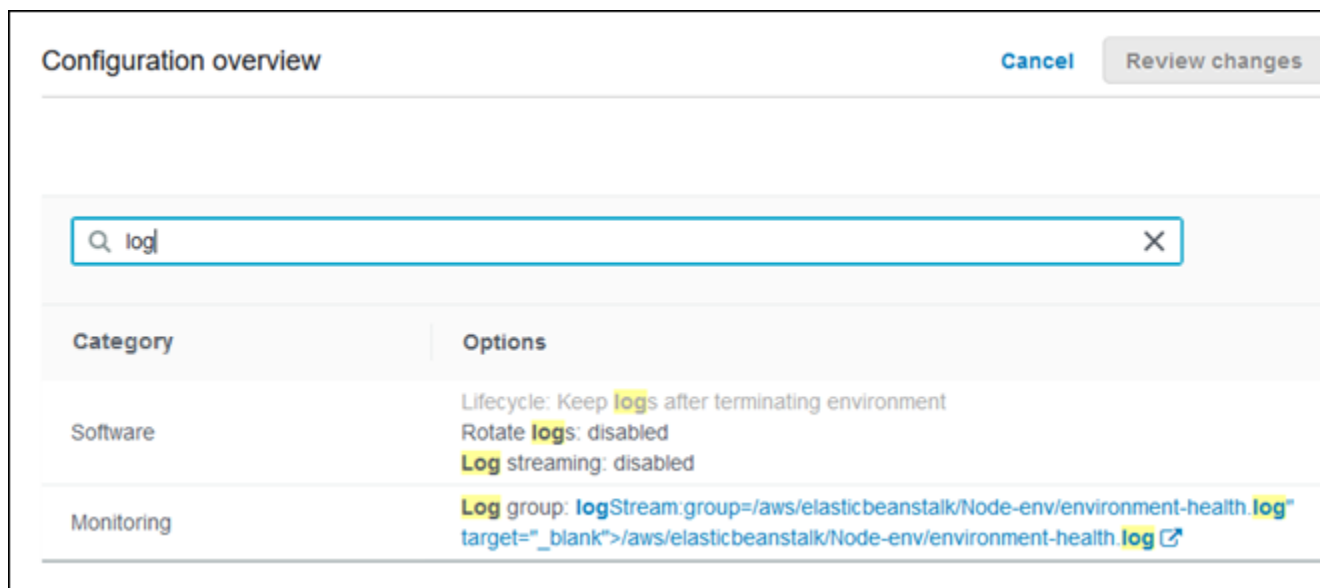
The **Configuration overview** page shows a set of configuration categories. Each configuration category summarizes the current state of a group of related options.

You can choose two views for this page. Turn on **Table View** to see a list of options grouped by category.

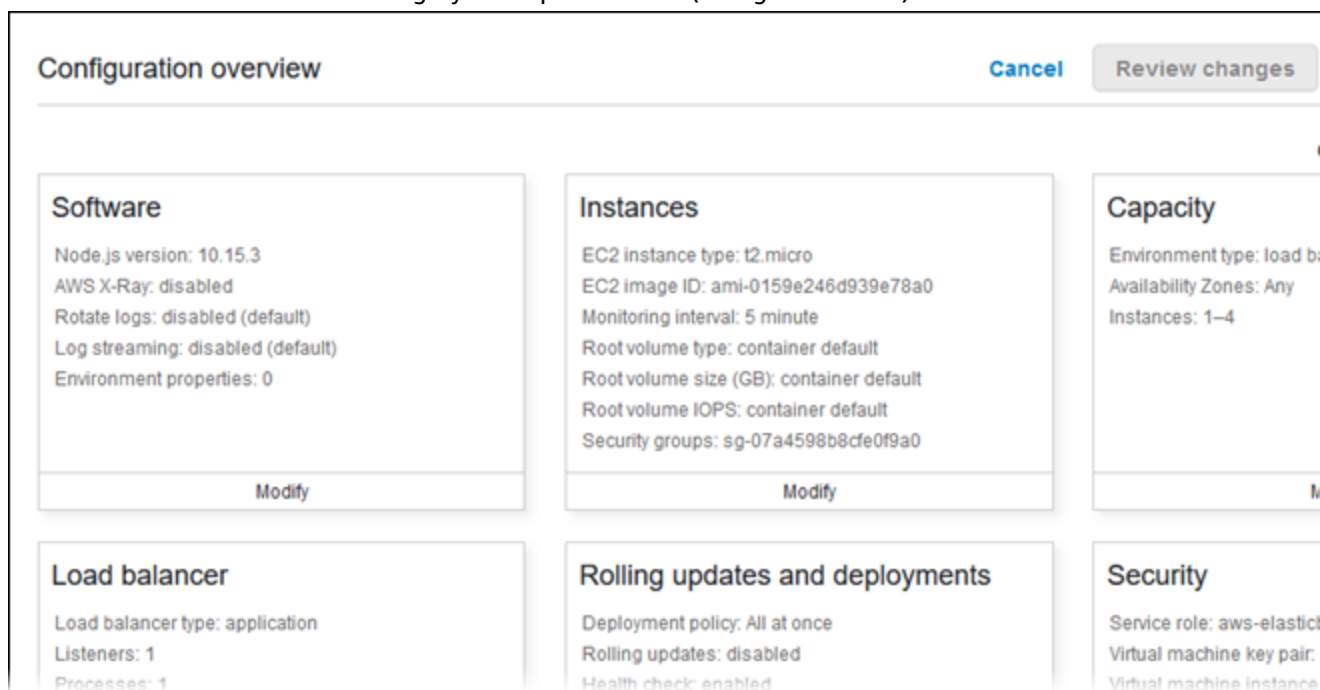
Configuration overview Cancel Review changes

Category	Options
Software	Rotate logs: disabled Node command: Gzip compression: true Proxy server: nginx Node.js version: 10.15.3 Log streaming: disabled X-Ray daemon: disabled
Instances	AMI ID: ami-0159e246d939e78a0 Instance type: t2.micro Monitoring interval: 5 minute IOPS: container default Size: container default Root volume type: container default EC2 security groups: load balancing, auto scaling
Capacity	Max: 4 Min: 1 Scale down increment: -1 Lower threshold: 2000000 Period: 5 Scale up increment: 1 Upper threshold: 6000000 Availability Zones: Any Breach duration: 5

You can search for an option by its name or value by entering search terms into a search box. As you type, the list gets shorter and shows only options that match your search terms.



Turn off **Table View** to see each category in a separate frame (configuration card).



Choose **Edit** in a configuration category to get to a related configuration page, where you can see full option values and make changes. When you're done viewing and modifying options, you can choose one of the following actions:

- **Cancel** – Go back to the environment's dashboard without applying your configuration changes. When you choose **Cancel**, the console loses any pending changes you made on any configuration category.

You can also cancel your configuration changes by choosing another console page, like **Dashboard** or **Logs**. In this case, if there are any pending configuration changes, the console prompts you to confirm that you agree to losing them.

- **Review changes** – Get a summary of all the pending changes you made in any of the configuration categories. For details, see [Review changes page \(p. 450\)](#).
- **Apply configuration** – Apply the changes you made in any of the configuration categories to your environment. In some cases you're prompted to confirm a consequence of one of your configuration decisions.

Review changes page

The **Review Changes** page displays a table showing all the pending option changes you made in any of the configuration categories and haven't applied to your environment yet. If you removed any options (for example, a custom environment property), a second table shows the removed options.

Both tables list each option as a combination of the **Namespace** and **Option Name** with which Elastic Beanstalk identifies it. For details, see [Configuration options \(p. 536\)](#).

For example, you might make these configuration changes:

- In the **Capacity** category: Change **Instances (Min)** from 1 to 2, and **Instances (Max)** from 2 to 4. This change corresponds to two changes in the `aws:autoscaling:asg` namespace on the **Changed Options** list.
- In the **Software** category:
 - Enable the **Rotate logs** option. This change corresponds to a change in the `aws:elasticbeanstalk:hostmanager` namespace on the **Changed Options** list.
 - Remove the `MY_ENV_PROPERTY` environment property. This change corresponds to a single entry for the `aws:elasticbeanstalk:application:environment` namespace on the **Removed Options** list.
- In the **Managed updates** category: Enable the **Managed updates** option. This single configuration change corresponds to three option changes across two namespaces—the last three items on the **Changed Options** list.

The following image shows the lists of your configuration changes on the **Review Changes** page.

Review Changes Continue			
Changed Options			
Namespace ^	Option Name	Old Value	New Value
aws:autoscaling:asg	MaxSize	2	4
aws:autoscaling:asg	MinSize	1	2
aws:elasticbeanstalk:hostmanager	LogPublicationControl	false	true
aws:elasticbeanstalk:managedactions	ManagedActionsEnabled	false	true
aws:elasticbeanstalk:managedactions	PreferredStartTime		WED:12:07
aws:elasticbeanstalk:managedactions :platformupdate	UpdateLevel		minor
Removed Options			
Namespace	Option Name		
aws:elasticbeanstalk:application:environment	MY_ENV_PROPERTY		

When you're done reviewing your changes, you can choose one of the following actions:

- **Continue** – Go back to the **Configuration overview** page. You can then continue making changes or apply pending ones.
- **Apply configuration** – Apply the changes you made in any of the configuration categories to your environment. In some cases you're prompted to confirm a consequence of one of your configuration decisions.

Your Elastic Beanstalk environment's Amazon EC2 instances

When you create a web server environment, AWS Elastic Beanstalk creates one or more Amazon Elastic Compute Cloud (Amazon EC2) virtual machines, known as *Instances*.

The instances in your environment are configured to run web apps on the platform that you choose. You can make changes to various properties and behaviors of your environment's instances during environment creation, after creation on a running environment, or as part of the source code that you deploy to the environment. For details, see [the section called "Configuration options" \(p. 536\)](#).

Note

The [Auto Scaling group \(p. 457\)](#) in your environment manages the Amazon EC2 instances that run your application. When you make configuration changes described on this page, the

launch configuration (either an Amazon EC2 launch template or an Auto Scaling group launch configuration resource) changes. This change requires [replacement of all instances \(p. 408\)](#) and trigger a [rolling update \(p. 409\)](#) or [immutable update \(p. 412\)](#), depending on which one is configured.

During environment creation you choose an [instance type](#) to determine the hardware of the host computer used to run your instances. Elastic Beanstalk supports new instance types soon after Amazon EC2 introduces them, typically by the next [platform update \(p. 415\)](#).

Elastic Beanstalk supports several Amazon EC2 [instance purchasing options](#): *On-Demand Instances*, *Reserved Instances*, and *Spot Instances*. An On-Demand Instance is a pay-as-you-go resource—there is no long-term commitment required when you use it. A Reserved Instance is a pre-purchased billing discount applied automatically to matching On-Demand instances in your environment. A Spot Instance is an unused Amazon EC2 instance that is available for less than the On-Demand price. You can enable Spot Instances in your environment by setting a single option. You can configure Spot Instance usage, including the mix of On-Demand and Spot Instances, using additional options. For more information, see [Auto Scaling group \(p. 457\)](#).

Sections

- [Configuring your environment's Amazon EC2 instances \(p. 452\)](#)
- [The aws:autoscaling:launchconfiguration namespace \(p. 456\)](#)

Configuring your environment's Amazon EC2 instances

You can modify your Elastic Beanstalk environment's Amazon EC2 instance configuration in the Elastic Beanstalk console.

To configure Amazon EC2 instances in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Instances** configuration category, choose **Edit**. Make changes to settings in this category, and then choose **Apply**. For setting descriptions, see the section [the section called "Instances category settings" \(p. 452\)](#) on this page.
5. In the **Capacity** configuration category, choose **Edit**. Make changes to settings in this category, and then choose **Continue**. For setting descriptions, see the section [the section called "Capacity category settings" \(p. 455\)](#) on this page.

Instances category settings

The following settings related to Amazon EC2 instances are available in the **Instances** configuration category.

Options

- [Monitoring interval \(p. 455\)](#)
- [Root volume \(boot device\) \(p. 455\)](#)

- [Security groups \(p. 455\)](#)

Elastic Beanstalk > Environments > GettingStartedApp-env > Configuration

Modify instances

Amazon CloudWatch monitoring

The time interval between when metrics are reported from the EC2 instances.

Monitoring interval

5 minute ▼

Root volume (boot device)

Root volume type

(Container default) ▼

Size

The number of gigabytes of the root volume attached to each instance.

GB

IOPS

Input/output operations per second for a provisioned IOPS (SSD) volume.

100 IOPS

EC2 security groups

	Group name	Group ID	Name
<input type="checkbox"/>	awseb-e-2mwvwhpfcstack-AWSEBLoadBalancerSecurityGroup-1DY3U98Q97M6V	sg-09d293ed9dde904a2	GettingS
<input type="checkbox"/>	awseb-e-2mwvwhpfcstack-AWSEBSecurityGroup-119YW6B19SFNK	sg-06e3ba42f5667c3a5	GettingS
<input type="checkbox"/>	awseb-e-cubmdjm6ga-stack-AWSEBLoadBalancerSecurityGroup-1GCJKWYQZ8MHY	sg-0ae9c16e639f7494e	GettingS
<input checked="" type="checkbox"/>	awseb-e-cubmdjm6ga-stack-AWSEBSecurityGroup-1HV9R4OGAAAXK	sg-0ca3299707080312a	GettingS
<input type="checkbox"/>	default	sg-1d9e5075	

Cancel

Monitoring interval

By default, the instances in your environment publish [basic health metrics \(p. 688\)](#) to CloudWatch at five-minute intervals at no additional cost.

For more detailed reporting, you can set the **Monitoring interval** to **1 minute** to increase the frequency with which the resources in your environment publish [basic health metrics \(p. 690\)](#) to CloudWatch. Amazon CloudWatch service charges apply for one-minute interval metrics. See [Amazon CloudWatch](#) for more information.

Root volume (boot device)

Each instance in your environment is configured with a root volume. The root volume is the Amazon EBS block device attached to the instance to store the operating system, libraries, scripts, and your application source code. By default, all platforms use general-purpose SSD block devices for storage.

You can modify **Root volume type** to use magnetic storage or provisioned IOPS SSD volume types and, if needed, increase the volume size. For provisioned IOPS volumes, you must also select the number of IOPS to provision. Select the volume type that meets your performance and price requirements.

For more information, see [Amazon EBS Volume Types](#) and [Amazon EBS Product Details](#).

Security groups

The security groups attached to your instances determine which traffic is allowed to reach the instances (ingress), and which traffic is allowed to leave the instances (egress). Elastic Beanstalk creates a security group that allows traffic from the load balancer on the standard ports for HTTP (80) and HTTPS (443).

You can specify additional security groups that you have created to allow traffic on other ports or from other sources. For example, you can create a security group for SSH access that allows ingress on port 22 from a restricted IP address range or, for additional security, from a bastion host to which only you have access.

Note

To allow traffic between environment A's instances and environment B's instances, you can add a rule to the security group that Elastic Beanstalk attached to environment B, and specify the security group that Elastic Beanstalk attached to environment A. This allows ingress from, or egress to, environment A's instances. However, doing so creates a dependency between the two security groups. If you later try to terminate environment A, Elastic Beanstalk will not be able to delete the environment's security group, because environment B's security group is dependent on it.

A safer approach would be to create a separate security group, attach it to environment A, and specify it in a rule of environment B's security group.

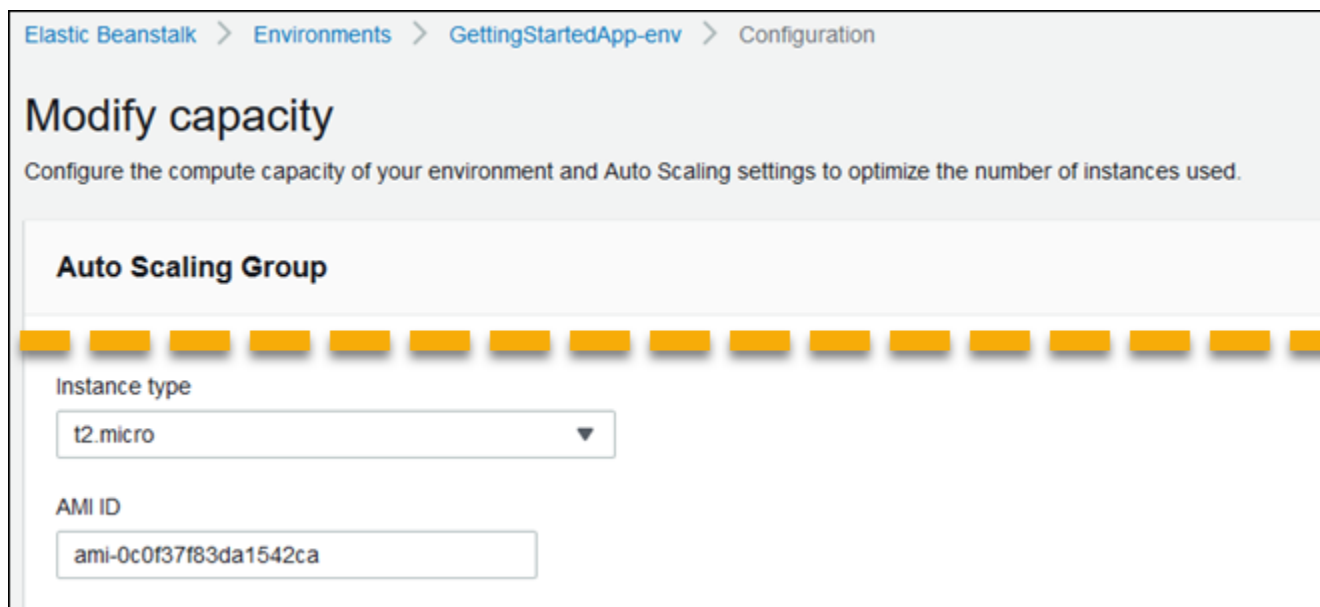
For more information on Amazon Amazon EC2 security groups, see [Amazon EC2 Security Groups](#) in the *Amazon EC2 User Guide for Linux Instances*.

Capacity category settings

The following settings related to Amazon EC2 instances are available in the **Capacity** configuration category.

Options

- [Instance type \(p. 456\)](#)
- [AMI ID \(p. 456\)](#)



Instance type

The **Instance type** setting determines the type of Amazon EC2 instance launched to run your application. Choose an instance that is powerful enough to run your application under load, but not so powerful that it's idle most of the time. For development purposes, the t2 family of instances provides a moderate amount of power with the ability to burst for short periods of time.

For large-scale, high-availability applications, use a pool of instances to ensure that capacity is not greatly affected if any single instance goes down. Start with an instance type that allows you to run five instances under moderate load during normal hours. If any instance fails, the rest of the instances can absorb the rest of the traffic. The capacity buffer also allows time for the environment to scale up as traffic begins to rise during peak hours.

For more information about Amazon EC2 instance families and types, see [Instance Types](#) in the *Amazon EC2 User Guide for Linux Instances*.

When you enable Spot Instance requests for your environment, this configuration page shows a list of **Instance types** instead of a single setting. You can select one or more instance types for your Spot Instances. For details, see [the section called "Spot instance support" \(p. 458\)](#).

AMI ID

The Amazon Machine Image (AMI) is the Amazon Linux or Windows Server machine image that Elastic Beanstalk uses to launch Amazon EC2 instances in your environment. Elastic Beanstalk provides machine images that contain the tools and resources required to run your application.

Elastic Beanstalk selects a default AMI for your environment based on the region, platform, and instance type that you choose. If you have created a [custom AMI \(p. 647\)](#), replace the default AMI ID with yours.

The aws:autoscaling:launchconfiguration namespace

You can use the [configuration options \(p. 536\)](#) in the [aws:autoscaling:launchconfiguration \(p. 556\)](#) namespace to configure your environment's instances, including additional options that are not available in the console.

The following [configuration file \(p. 600\)](#) example configures the basic options shown in this topic, the options `EC2KeyName` and `IamInstanceProfile` discussed in [Security \(p. 510\)](#), and an additional option, `BlockDeviceMappings`, which isn't available in the console.

```
option_settings:
  aws:autoscaling:launchconfiguration:
    InstanceType: m1.small
    SecurityGroups: my-securitygroup
    EC2KeyName: my-keypair
    MonitoringInterval: "1 minute"
    IamInstanceProfile: "aws-elasticbeanstalk-ec2-role"
    BlockDeviceMappings: "/dev/sdj=:100,/dev/sdh=snap-51eef269,/dev/sdb=ephemeral0"
```

`BlockDeviceMappings` lets you configure additional block devices for your instances. For more information, see [Block Device Mapping](#) in the *Amazon EC2 User Guide for Linux Instances*.

Note

The `InstanceType` option is obsolete. It's replaced by the newer and more powerful `InstanceTypes` option in the `aws:ec2:instances (p. 567)` namespace. The new option allows you to specify a list of one or more instance types for your environment. The first value on that list is equivalent to the value of the `InstanceType` option included in the `aws:autoscaling:launchconfiguration` namespace described here. The recommended way to specify instance types is by using the new option. If specified, the new option takes precedence over the old one. For more information, see [the section called "The aws:ec2:instances namespace" \(p. 463\)](#).

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See [Recommended values \(p. 537\)](#) for details.

Auto Scaling group for your Elastic Beanstalk environment

Your AWS Elastic Beanstalk environment includes an *Auto Scaling group* that manages the [Amazon EC2 instances \(p. 451\)](#) in your environment. In a single-instance environment, the Auto Scaling group ensures that there is always one instance running. In a load balanced environment, you configure the group with a range of instances to run, and Auto Scaling adds or removes instances as needed, based on load.

The Auto Scaling group also applies the launch configuration for the instances in your environment. You can [modify the launch configuration \(p. 451\)](#) to change the instance type, key pair, Amazon Elastic Block Store (Amazon EBS) storage, and other settings that can only be configured when you launch an instance.

The Auto Scaling group uses two Amazon CloudWatch alarms to trigger scaling operations. The default triggers scale when the average outbound network traffic from each instance is higher than 6 MiB or lower than 2 MiB over a period of five minutes. To use Auto Scaling effectively, [configure triggers \(p. 464\)](#) that are appropriate for your application, instance type, and service requirements. You can scale based on several statistics including latency, disk I/O, CPU utilization, and request count.

To optimize your environment's use of Amazon EC2 instances through predictable periods of peak traffic, [configure your Auto Scaling group to change its instance count on a schedule \(p. 466\)](#). You can schedule changes to your group's configuration that recur daily or weekly, or schedule one-time changes to prepare for marketing events that will drive a lot of traffic to your site.

As an option, Elastic Beanstalk can combine On-Demand and [Spot \(p. 458\)](#) Instances for your environment.

Auto Scaling monitors the health of each Amazon EC2 instance that it launches. If any instance terminates unexpectedly, Auto Scaling detects the termination and launches a replacement instance. To configure the group to use the load balancer's health check mechanism, see [Auto Scaling health check setting](#) (p. 469).

You can configure Auto Scaling for your environment using the [Elastic Beanstalk console](#) (p. 460), the [EB CLI](#) (p. 463), or [configuration options](#) (p. 463).

Topics

- [Spot instance support](#) (p. 458)
- [Auto Scaling group configuration using the Elastic Beanstalk console](#) (p. 460)
- [Auto Scaling group configuration using the EB CLI](#) (p. 463)
- [Configuration options](#) (p. 463)
- [Auto Scaling triggers](#) (p. 464)
- [Scheduled Auto Scaling actions](#) (p. 466)
- [Auto Scaling health check setting](#) (p. 469)

Spot instance support

To take advantage of Amazon EC2 [Spot Instances](#), you can enable a Spot option for your environment. Your environment's Auto Scaling group then combines Amazon EC2 purchase options and maintains a mix of On-Demand and Spot Instances.

Important

Demand for Spot Instances can vary significantly from moment to moment, and the availability of Spot Instances can also vary significantly depending on how many unused Amazon EC2 instances are available. It's always possible that your Spot Instance might be interrupted. Therefore, you must ensure that your application is prepared for a Spot Instance interruption. For more information, see [Spot Instance Interruptions](#) in the *Amazon EC2 User Guide for Linux Instances*.

Elastic Beanstalk provides several configuration options to support the Spot feature. They're discussed in the following sections about configuring your Auto Scaling Group. Two of these options deserve special attention: `SpotFleetOnDemandBase` and `SpotFleetOnDemandAboveBasePercentage` (both in the `aws:ec2:instances` namespace). Consider how they relate to the `MinSize` option (in the `aws:autoscaling:asg` namespace). Only `MinSize` determines your environment's initial capacity—the number of instances you want running at a minimum. `SpotFleetOnDemandBase` doesn't affect initial capacity; when Spot is enabled, this option only determines how many On-Demand Instances are provisioned before any Spot Instances are considered. When `SpotFleetOnDemandBase` is less than `MinSize`, you still get exactly `MinSize` instances as initial capacity; at least `SpotFleetOnDemandBase` of them must be On-Demand Instances. When `SpotFleetOnDemandBase` is greater than `MinSize`, then as your environment scales out, you're guaranteed to get at least an additional `SpotFleetOnDemandBase - MinSize` Instances that are On-Demand before satisfying the `SpotFleetOnDemandBase` requirement.

In production environments, Spot Instances are particularly useful as part of an Auto Scaling (load balanced) environment. We don't recommend using Spot in a single instance environment. If Spot Instances aren't available, you might lose the entire capacity (a single instance) of your environment. You may still wish to use a Spot Instance in a single instance environment for development or testing. When you do, be sure to set both `SpotFleetOnDemandBase` and `SpotFleetOnDemandAboveBasePercentage` to zero. Any other settings result in an On-Demand Instance.

Note

Some older AWS accounts might provide Elastic Beanstalk with default instance types that don't support Spot Instances (for example, `t1.micro`). If you enable Spot Instance requests and you see

the error None of the instance types you specified supports Spot, be sure to configure instance types that support Spot. To choose Spot Instance types, use the [Spot Instance Advisor](#).

The following examples demonstrate different scenarios of setting the various scaling options. All examples assume a load balanced environment with Spot Instance requests enabled.

Example 1: On-Demand and Spot as part of initial capacity

Option settings

Option	Namespace	Value
MinSize	aws:autoscaling:asg	10
MaxSize	aws:autoscaling:asg	24
SpotFleetOnDemandBase	aws:ec2:instances	4
SpotFleetOnDemandAboveBasePercentage	aws:ec2:instances	50

In this example, the environment starts with ten instances, of which seven are On-Demand (four base, and 50% of the six above base) and three are Spot. The environment can scale out up to 24 instances. As it scales out, the portion of On-Demand in the part of the fleet above the four base On-Demand instances is kept at 50%, up to a maximum of 24 instances overall, of which 14 are On-Demand (four base, and 50% of the 20 above base) and ten are Spot.

Example 2: All On-Demand initial capacity

Option settings

Option	Namespace	Value
MinSize	aws:autoscaling:asg	4
MaxSize	aws:autoscaling:asg	24
SpotFleetOnDemandBase	aws:ec2:instances	4
SpotFleetOnDemandAboveBasePercentage	aws:ec2:instances	50

In this example, the environment starts with four instances, all of which are On-Demand. The environment can scale out up to 24 instances. As it scales out, the portion of On-Demand in the part of the fleet above the four base On-Demand instances is kept at 50%, up to a maximum of 24 instances overall, of which 14 are On-Demand (four base, and 50% of the 20 above base) and ten are Spot.

Example 3: Additional On-Demand base beyond initial capacity

Option settings

Option	Namespace	Value
MinSize	aws:autoscaling:asg	3
MaxSize	aws:autoscaling:asg	24
SpotFleetOnDemandBase	aws:ec2:instances	4
SpotFleetOnDemandAboveBasePercentage	aws:ec2:instances	50

In this example, the environment starts with three instances, all of which are On-Demand. The environment can scale out up to 24 instances. The first additional instance above the initial three is On-Demand, to complete the four base On-Demand instances. As it scales out further, the portion of On-Demand in the part of the fleet above the four base On-Demand instances is kept at 50%, up to a maximum of 24 instances overall, of which 14 are On-Demand (four base, and 50% of the 20 above base) and ten are Spot.

Auto Scaling group configuration using the Elastic Beanstalk console

You can configure how Auto Scaling works by editing **Capacity** on the environment's **Configuration** page in the [Elastic Beanstalk console](#) (p. 353).

To configure the Auto Scaling group in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Auto Scaling group** section, configure the following settings.
 - **Environment type** – Select **Load balanced**.
 - **Min instances** – The minimum number of EC2 instances that the group should contain at any time. The group starts with the minimum count and adds instances when the scale-up trigger condition is met.
 - **Max instances** – The maximum number of EC2 instances that the group should contain at any time.

Note

If you use rolling updates, be sure that the maximum instance count is higher than the [Minimum instances in service setting](#) (p. 410) for rolling updates.

- **Fleet composition** – Select **Combined purchase options and instance types** to enable Spot Instance requests.

For this example, select **On-Demand Instances**.

- **Instance type** – The type of Amazon EC2 instance launched to run your application. For details, see [the section called "Instance type"](#) (p. 456).
- **AMI ID** – The machine image that Elastic Beanstalk uses to launch Amazon EC2 instances in your environment. For details, see [the section called "AMI ID"](#) (p. 456).
- **Availability Zones** – Choose the number of Availability Zones to spread your environment's instances across. By default, the Auto Scaling group launches instances evenly across all usable zones. To concentrate your instances in fewer zones, choose the number of zones to use. For production environments, use at least two zones to ensure that your application is available in case one Availability Zone goes out.
- **Placement** (optional) – Choose the Availability Zones to use. Use this setting if your instances need to connect to resources in specific zones, or if you have purchased [reserved instances](#), which are zone-specific. If you set `Cooldown: '720'` Custom Availability Zones: `'us-west-2a,us-west-2b'` `MaxSize: '4'` so set the number of zones, you must choose at least that many custom zones.

If you launch your environment in a custom VPC, you cannot configure this option. In a custom VPC, you choose Availability Zones for the subnets that you assign to your environment.

- **Scaling cooldown** – The amount of time, in seconds, to wait for instances to launch or terminate after scaling, before continuing to evaluate triggers. For more information, see [Scaling Cooldowns](#).

Modify capacity

Configure the compute capacity of your environment and Auto Scaling settings to optimize the number of instances used.

Auto Scaling Group

Environment type

Load balanced

Instances

Min 1

Max 2

Fleet composition

Choose a mix of On-Demand and Spot Instances with multiple instance types. Spot Instances are automatically launched at the lowest available price.

- On-Demand instances
- Combine purchase options and instances

Maximum spot price

The maximum price per instance-hour, in USD, that you're willing to pay for a Spot Instance. Setting a custom price limits your chances to purchase Spot instances.

- Default - the On-Demand price for each instance type (recommended)
- Set your maximum price

On-Demand base

The minimum number of On-Demand Instances that your Auto Scaling group provisions before considering Spot Instances as your environment scales out.

0

On-Demand above base

The percentage of On-Demand Instances as part of any additional capacity that your Auto Scaling group provisions beyond the On-Demand base instances.

70 %

Instance type

t2.micro

AMI ID

ami-0c0f37f83da1542ca

Availability Zones

Number of Availability Zones (AZs) to use.

Any 462

Placement

Specify Availability Zones (AZs) to use.

-- Choose Availability Zones (AZs) --

6. Choose **Apply**.

Auto Scaling group configuration using the EB CLI

When you create an environment using the [eb create](#) (p. 894) command, you can specify a few options related to your environment's Auto Scaling group. These options help you control the environment's capacity.

`--single`

An option to create the environment with a single Amazon EC2 instance and without a load balancer. Without this option, a load balanced environment is created.

`--instance-types`

A list of Amazon EC2 instance types you want your environment to use.

`--enable-spot`, `--spot-max-price`

Options related to enabling and configuring Spot Instance requests.

The following example creates an environment and configures the Auto Scaling group to enable Spot Instance requests for the new environment, with three possible instance types to use .

```
$ eb create --enable-spot --instance-types "t2.micro,t3.micro,t3.small"
```

Configuration options

Elastic Beanstalk provides [configuration options](#) (p. 536) for Auto Scaling settings in two namespaces: [aws:autoscaling:asg](#) (p. 555) and [aws:ec2:instances](#).

The `aws:autoscaling:asg` namespace

The [aws:autoscaling:asg](#) (p. 555) namespace provides options for overall scale and availability.

The following [configuration file](#) (p. 600) example configures the Auto Scaling group to use two to four instances, specific availability zones, and a cooldown period of 12 minutes (720 seconds).

```
option_settings:
  aws:autoscaling:asg:
    Availability Zones: Any
    Cooldown: '720'
    Custom Availability Zones: 'us-west-2a,us-west-2b'
    MaxSize: '4'
    MinSize: '2'
```

The `aws:ec2:instances` namespace

The [aws:ec2:instances](#) (p. 567) namespace provides options related to your environment's instances, including Spot Instance management. It complements [aws:autoscaling:launchconfiguration](#) (p. 556) and [aws:autoscaling:asg](#) (p. 555).

When you update your environment configuration and remove one or more instance types from the `InstanceTypes` option, Elastic Beanstalk terminates any Amazon EC2 instances running on any of

the removed instance types. Your environment's Auto Scaling group then launches new instances, as necessary to complete the desired capacity, using your current specified instance types.

The following [configuration file \(p. 600\)](#) example configures the Auto Scaling group to enable Spot Instance requests for your environment, with three possible instance types to use, at least one On-Demand Instance as baseline capacity, and a sustained 33% of On-Demand Instances for any additional capacity.

```
option_settings:
  aws:ec2:instances:
    EnableSpot: true
    InstanceTypes: 't2.micro,t3.micro,t3.small'
    SpotFleetOnDemandBase: '1'
    SpotFleetOnDemandAboveBasePercentage: '33'
```

To choose Spot Instance types, use the [Spot Instance Advisor](#).

Auto Scaling triggers

The Auto Scaling group in your Elastic Beanstalk environment uses two Amazon CloudWatch alarms to trigger scaling operations. The default triggers scale when the average outbound network traffic from each instance is higher than 6 MB or lower than 2 MB over a period of five minutes. To use Amazon EC2 Auto Scaling effectively, configure triggers that are appropriate for your application, instance type, and service requirements. You can scale based on several statistics including latency, disk I/O, CPU utilization, and request count.

For more information about CloudWatch metrics and alarms, see [Amazon CloudWatch Concepts](#) in the *Amazon CloudWatch User Guide*.

Configuring Auto Scaling triggers

You can configure the triggers that adjust the number of instances in your environment's Auto Scaling group in the Elastic Beanstalk console.

To configure triggers in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Scaling triggers** section, configure the following settings:
 - **Metric** – Metric used for your Auto Scaling trigger.
 - **Statistic** – Statistic calculation the trigger should use, such as *Average*.
 - **Unit** – Unit for the trigger metric, such as **Bytes**.
 - **Period** – Specifies how frequently Amazon CloudWatch measures the metrics for your trigger.
 - **Breach duration** – Amount of time, in minutes, a metric can be outside of the upper and lower thresholds before triggering a scaling operation.
 - **Upper threshold** – If the metric exceeds this number for the breach duration, a scaling operation is triggered.
 - **Scale up increment** – The number of Amazon EC2 instances to add when performing a scaling activity.

- **Lower threshold** – If the metric falls below this number for the breach duration, a scaling operation is triggered.
- **Scale down increment** – The number of Amazon EC2 instances to remove when performing a scaling activity.

Scaling triggers

Metric
Change the metric that is monitored to determine if the environment's capacity is too low or too high.

NetworkOut ▼

Statistic
Choose how the metric is interpreted.

Average ▼

Unit

Bytes ▼

Period
The period between metric evaluations.

5 Min

Breach duration
The amount of time a metric can exceed a threshold before triggering a scaling operation.

5 Min

Upper threshold

6000000 Bytes

Scale up increment

1 EC2 instances

Lower threshold

2000000 Bytes

Scale down increment

-1 EC2 instances

6. Choose **Apply**.

The `aws:autoscaling:trigger` namespace

Elastic Beanstalk provides [configuration options \(p. 536\)](#) for Auto Scaling settings in the `aws:autoscaling:trigger` (p. 563) namespace. Settings in this namespace are organized by the resource that they apply to.

```
option_settings:
  AWSEBAutoScalingScaleDownPolicy.aws:autoscaling:trigger:
    LowerBreachScaleIncrement: '-1'
  AWSEBAutoScalingScaleUpPolicy.aws:autoscaling:trigger:
    UpperBreachScaleIncrement: '1'
  AWSEBCloudwatchAlarmHigh.aws:autoscaling:trigger:
    UpperThreshold: '6000000'
  AWSEBCloudwatchAlarmLow.aws:autoscaling:trigger:
    BreachDuration: '5'
    EvaluationPeriods: '1'
    LowerThreshold: '2000000'
    MeasureName: NetworkOut
    Period: '5'
    Statistic: Average
    Unit: Bytes
```

Scheduled Auto Scaling actions

To optimize your environment's use of Amazon EC2 instances through predictable periods of peak traffic, configure your Amazon EC2 Auto Scaling group to change its instance count on a schedule. You can configure your environment with a recurring action to scale up each day in the morning, and scale down at night when traffic is low. For example, if you have a marketing event that will drive traffic to your site for a limited period of time, you can schedule a one-time event to scale up when it starts, and another to scale down when it ends.

You can define up to 120 active scheduled actions per environment. Elastic Beanstalk also retains up to 150 expired scheduled actions, which you can reuse by updating their settings.

Configuring scheduled actions

You can create scheduled actions for your environment's Auto Scaling group in the Elastic Beanstalk console.

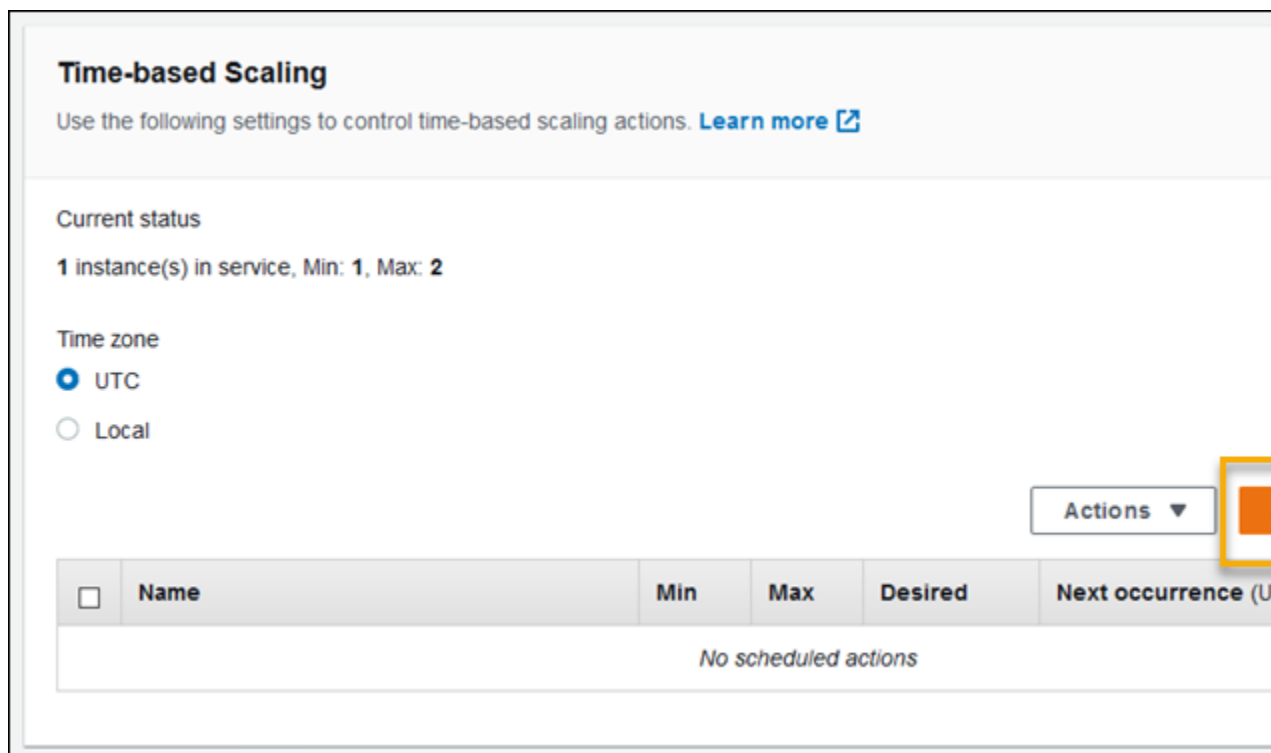
To configure scheduled actions in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Time-based scaling** section, choose **Add scheduled action**.



6. Fill in the following scheduled action settings:

- **Name** – Specify a unique name of up to 255 alphanumeric characters, with no spaces.
- **Instances** – Choose the minimum and maximum instance count to apply to the Auto Scaling group.
- **Desired capacity** (optional) – Set the initial desired capacity for the Auto Scaling group. After the scheduled action is applied, triggers adjust the desired capacity based on their settings.
- **Occurrence** – Choose **Recurring** to repeat the scaling action on a schedule.
- **Start time** – For one-time actions, choose the date and time to run the action.

For recurrent actions, a start time is optional. Specify it to choose the earliest time the action is performed. After this time, the action recurs according to the **Recurrence** expression.

- **Recurrence** – Use a [Cron](#) expression to specify the frequency with which you want the scheduled action to occur. For example, `30 6 * * 2` runs the action every Tuesday at 6:30 AM UTC.
- **End time** (optional) – Optional for recurrent actions. If specified, the action recurs according to the **Recurrence** expression, and is not performed again after this time.

When a scheduled action ends, Auto Scaling doesn't automatically go back to its previous settings. Configure a second scheduled action to return Auto Scaling to the original settings as needed.

7. Choose **Add**.

8. Choose **Apply**.

Note

Scheduled actions will not be saved until applied.

The `aws:autoscaling:scheduledaction` namespace

If you need to configure a large number of scheduled actions, you can use [configuration files \(p. 600\)](#) or [the Elastic Beanstalk API \(p. 554\)](#) to apply the configuration option changes from a YAML or JSON file.

These methods also let you access the [Suspend option \(p. 562\)](#) to temporarily deactivate a recurrent scheduled action.

Note

When working with scheduled action configuration options outside of the console, use ISO 8601 time format to specify start and end times in UTC. For example, 2015-04-28T04:07:02Z. For more information about ISO 8601 time format, see [Date and Time Formats](#). The dates must be unique across all scheduled actions.

Elastic Beanstalk provides configuration options for scheduled action settings in the [aws:autoscaling:scheduledaction \(p. 562\)](#) namespace. Use the `resource_name` field to specify the name of the scheduled action.

Example Scheduled-scale-up-specific-time-long.config

This configuration file instructs Elastic Beanstalk to scale out from five instances to 10 instances at 2015-12-12T00:00:00Z.

```
option_settings:
- namespace: aws:autoscaling:scheduledaction
  resource_name: ScheduledScaleUpSpecificTime
  option_name: MinSize
  value: '5'
- namespace: aws:autoscaling:scheduledaction
  resource_name: ScheduledScaleUpSpecificTime
  option_name: MaxSize
  value: '10'
- namespace: aws:autoscaling:scheduledaction
  resource_name: ScheduledScaleUpSpecificTime
  option_name: DesiredCapacity
  value: '5'
- namespace: aws:autoscaling:scheduledaction
  resource_name: ScheduledScaleUpSpecificTime
  option_name: StartTime
  value: '2015-12-12T00:00:00Z'
```

Example Scheduled-scale-up-specific-time.config

To use the shorthand syntax with the EB CLI or configuration files, prepend the resource name to the namespace.

```
option_settings:
  ScheduledScaleUpSpecificTime.aws:autoscaling:scheduledaction:
    MinSize: '5'
    MaxSize: '10'
    DesiredCapacity: '5'
    StartTime: '2015-12-12T00:00:00Z'
```

Example Scheduled-scale-down-specific-time.config

This configuration file instructs Elastic Beanstalk to scale in at 2015-12-12T07:00:00Z.

```
option_settings:
  ScheduledScaleDownSpecificTime.aws:autoscaling:scheduledaction:
    MinSize: '1'
    MaxSize: '1'
    DesiredCapacity: '1'
    StartTime: '2015-12-12T07:00:00Z'
```

Example Scheduled-periodic-scale-up.config

This configuration file instructs Elastic Beanstalk to scale out every day at 9AM. The action is scheduled to begin May 14, 2015 and end January 12, 2016.

```
option_settings:
  ScheduledPeriodicScaleUp.aws:autoscaling:scheduledaction:
    MinSize: '5'
    MaxSize: '10'
    DesiredCapacity: '5'
    StartTime: '2015-05-14T07:00:00Z'
    EndTime: '2016-01-12T07:00:00Z'
    Recurrence: 0 9 * * *
```

Example Scheduled-periodic-scale-down.config

This configuration file instructs Elastic Beanstalk to scale in to no running instance every day at 6PM. If you know that your application is mostly idle outside of business hours, you can create a similar scheduled action. If your application must be down outside of business hours, change `MaxSize` to 0.

```
option_settings:
  ScheduledPeriodicScaleDown.aws:autoscaling:scheduledaction:
    MinSize: '0'
    MaxSize: '1'
    DesiredCapacity: '0'
    StartTime: '2015-05-14T07:00:00Z'
    EndTime: '2016-01-12T07:00:00Z'
    Recurrence: 0 18 * * *
```

Example Scheduled-weekend-scale-down.config

This configuration file instructs Elastic Beanstalk to scale in every Friday at 6PM. If you know that your application doesn't receive as much traffic over the weekend, you can create a similar scheduled action.

```
option_settings:
  ScheduledWeekendScaleDown.aws:autoscaling:scheduledaction:
    MinSize: '1'
    MaxSize: '4'
    DesiredCapacity: '1'
    StartTime: '2015-12-12T07:00:00Z'
    EndTime: '2016-01-12T07:00:00Z'
    Recurrence: 0 18 * * 5
```

Auto Scaling health check setting

Amazon EC2 Auto Scaling monitors the health of each Amazon Elastic Compute Cloud (Amazon EC2) instance that it launches. If any instance terminates unexpectedly, Auto Scaling detects the termination and launches a replacement instance. By default, the Auto Scaling group created for your environment uses [Amazon EC2 status checks](#). If an instance in your environment fails an Amazon EC2 status check, Auto Scaling takes it down and replaces it.

Amazon EC2 status checks only cover an instance's health, not the health of your application, server, or any Docker containers running on the instance. If your application crashes, but the instance that it runs on is still healthy, it may be kicked out of the load balancer, but Auto Scaling won't replace it automatically. The default behavior is good for troubleshooting. If Auto Scaling replaced the instance as soon as the application crashed, you might not realize that anything went wrong, even if it crashed quickly after starting up.

If you want Auto Scaling to replace instances whose application has stopped responding, you can use a [configuration file \(p. 600\)](#) to configure the Auto Scaling group to use Elastic Load Balancing health checks. The following example sets the group to use the load balancer's health checks, in addition to the Amazon EC2 status check, to determine an instance's health.

Example `.ebextensions/autoscaling.config`

```
Resources:
  AWSEBAutoScalingGroup:
    Type: "AWS::AutoScaling::AutoScalingGroup"
    Properties:
      HealthCheckType: ELB
      HealthCheckGracePeriod: 300
```

For more information about the `HealthCheckType` and `HealthCheckGracePeriod` properties, see [AWS::AutoScaling::AutoScalingGroup](#) in the *AWS CloudFormation User Guide* and [Health Checks for Auto Scaling Instances](#) in the *Amazon EC2 Auto Scaling User Guide*.

By default, the Elastic Load Balancing health check is configured to attempt a TCP connection to your instance over port 80. This confirms that the web server running on the instance is accepting connections. However, you might want to [customize the load balancer health check \(p. 470\)](#) to ensure that your application, and not just the web server, is in a good state. The grace period setting sets the number of seconds that an instance can fail the health check without being terminated and replaced. Instances can recover after being kicked out of the load balancer, so give the instance an amount of time that is appropriate for your application.

Load balancer for your Elastic Beanstalk environment

When you [enable load balancing \(p. 436\)](#), AWS Elastic Beanstalk creates an [Elastic Load Balancing](#) load balancer for your environment. The load balancer distributes traffic among your environment's instances.

Elastic Beanstalk supports these load balancer types:

- [Classic Load Balancer](#) – The Elastic Load Balancing previous-generation load balancer. Routes HTTP, HTTPS, or TCP request traffic to different ports on environment instances.
- [Application Load Balancer](#) – An application layer load balancer. Routes HTTP or HTTPS request traffic to different ports on environment instances based on the request path.
- [Network Load Balancer](#) – A network layer load balancer. Routes TCP request traffic to different ports on environment instances. Supports both active and passive health checks.

By default, Elastic Beanstalk creates an Application Load Balancer for your environment when you enable load balancing with the Elastic Beanstalk console or the EB CLI. It configures the load balancer to listen for HTTP traffic on port 80 and forward this traffic to instances on the same port. You can choose the type of load balancer that your environment uses only during environment creation. Later, you can change settings to manage the behavior of your running environment's load balancer, but you can't change its type.

Note

Your environment must be in a VPC with subnets in at least two Availability Zones to create an Application Load Balancer. All new AWS accounts include default VPCs that meet this requirement. If your environment is in a VPC with subnets in only one Availability Zone, it defaults to a Classic Load Balancer. If you don't have any subnets, you can't enable load balancing.

You can create and manage environments with all load balancer types using the Elastic Beanstalk console, the EB CLI [eb create](#) (p. 894) command, the Elastic Beanstalk APIs, or configuration files ([Ebextensions](#) (p. 600)).

See the following topics to learn about each Elastic Beanstalk-supported load balancer type, its functionality, and how to configure and manage it in an Elastic Beanstalk environment, and how to configure a load balancer to [upload access logs](#) (p. 505) to Amazon S3.

Topics

- [Configuring a Classic Load Balancer](#) (p. 471)
- [Configuring an Application Load Balancer](#) (p. 479)
- [Configuring a Network Load Balancer](#) (p. 496)
- [Configuring access logs](#) (p. 505)

Configuring a Classic Load Balancer

When you [enable load balancing](#) (p. 436), your AWS Elastic Beanstalk environment is equipped with an Elastic Load Balancing load balancer to distribute traffic among the instances in your environment. Elastic Load Balancing supports several load balancer types. To learn about them, see the [Elastic Load Balancing User Guide](#).

This topic describes the configuration of a [Classic Load Balancer](#). For information about configuring all the load balancer types that Elastic Beanstalk supports, see [Load balancer for your Elastic Beanstalk environment](#) (p. 470).

Note

You can choose the type of load balancer that your environment uses only during environment creation. Later, you can change settings to manage the behavior of your running environment's load balancer, but you can't change its type.

Introduction

A [Classic Load Balancer](#) is the Elastic Load Balancing previous-generation load balancer. It supports routing HTTP, HTTPS, or TCP request traffic to different ports on environment instances.

When your environment uses a Classic Load Balancer, Elastic Beanstalk configures it by default to [listen](#) for HTTP traffic on port 80 and forward it to instances on the same port. To support secure connections, you can configure your load balancer with a listener on port 443 and a TLS certificate.

The load balancer uses a [health check](#) to determine whether the Amazon EC2 instances running your application are healthy. The health check makes a request to a specified URL at a set interval. If the URL returns an error message, or fails to return within a specified timeout period, the health check fails.

If your application performs better by serving multiple requests from the same client on a single server, you can configure your load balancer to use [sticky sessions](#). With sticky sessions, the load balancer adds a cookie to HTTP responses that identifies the Amazon EC2 instance that served the request. When a subsequent request is received from the same client, the load balancer uses the cookie to send the request to the same instance.

With [cross-zone load balancing](#), each load balancer node for your Classic Load Balancer distributes requests evenly across the registered instances in all enabled Availability Zones. If cross-zone load balancing is disabled, each load balancer node distributes requests evenly across the registered instances in its Availability Zone only.

When an instance is removed from the load balancer because it has become unhealthy or the environment is scaling down, [connection draining](#) gives the instance time to complete requests before

closing the connection between the instance and the load balancer. You can change the amount of time given to instances to send a response, or disable connection draining completely.

Note

Connection draining is enabled by default when you create an environment with the Elastic Beanstalk console or the EB CLI. For other clients, you can enable it with [configuration options \(p. 479\)](#).

You can use advanced load balancer settings to configure listeners on arbitrary ports, modify additional sticky session settings, and configure the load balancer to connect to EC2 instances securely. These settings are available through [configuration options \(p. 479\)](#) that you can set by using configuration files in your source code, or directly on an environment by using the Elastic Beanstalk API. Many of these settings are also available in the Elastic Beanstalk console. In addition, you can configure a load balancer to [upload access logs \(p. 505\)](#) to Amazon S3.

Configuring a Classic Load Balancer using the Elastic Beanstalk console

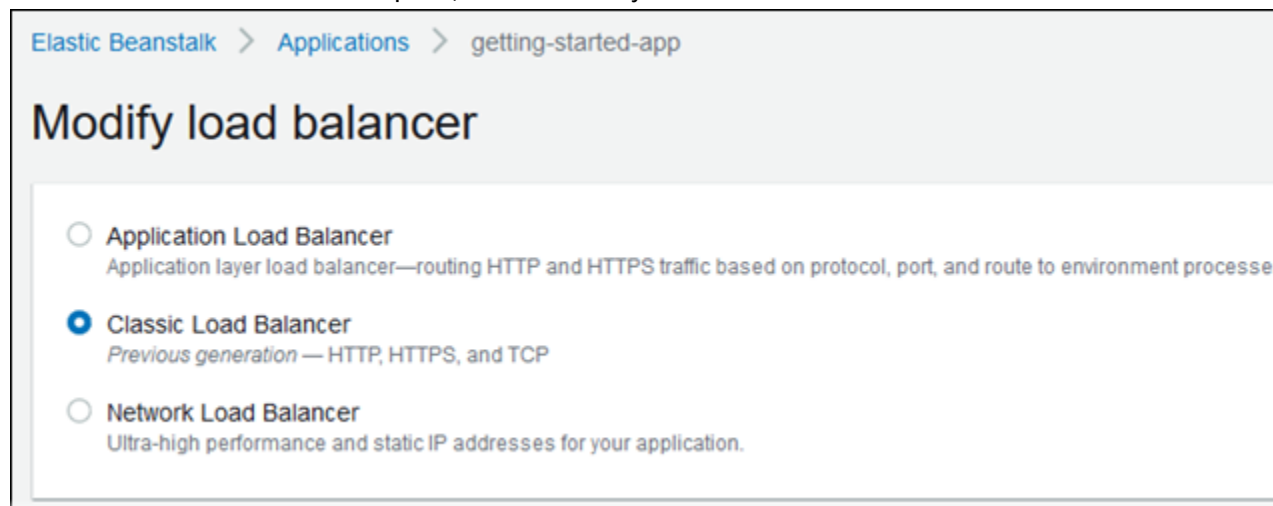
You can use the Elastic Beanstalk console to configure a Classic Load Balancer's ports, HTTPS certificate, and other settings, during environment creation or later when your environment is running.

To configure a Classic Load Balancer in the Elastic Beanstalk console during environment creation

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**.
3. Choose [Create a new environment \(p. 365\)](#) to start creating your environment.
4. On the wizard's main page, before choosing **Create environment**, choose **Configure more options**.
5. Choose the **High availability** configuration preset.

Alternatively, in the **Capacity** configuration category, configure a **Load balanced** environment type. For details, see [Capacity \(p. 373\)](#).

6. In the **Load balancer** configuration category, choose **Edit**.
7. Select the **Classic Load Balancer** option, if it isn't already selected.



8. Make any Classic Load Balancer configuration changes that your environment requires.
9. Choose **Save**, and then make any other configuration changes that your environment requires.

10. Choose **Create environment**.

To configure a running environment's Classic Load Balancer in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Load balancer** configuration category, choose **Edit**.

Note

If the **Load balancer** configuration category doesn't have an **Edit** button, your environment doesn't have a load balancer. To learn how to set one up, see [Changing environment type](#) (p. 436).

5. Make the Classic Load Balancer configuration changes that your environment requires.
6. Choose **Apply**.

Classic Load Balancer settings

- [Listeners](#) (p. 473)
- [Sessions](#) (p. 476)
- [Cross-zone load balancing](#) (p. 476)
- [Connection draining](#) (p. 476)
- [Health check](#) (p. 477)

Listeners

Use this list to specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port using a specified protocol to your instances. Initially, the list shows the default listener, which routes incoming HTTP traffic on port 80 to your environment's instance servers that are listening to HTTP traffic on port 80.

Classic Load Balancer

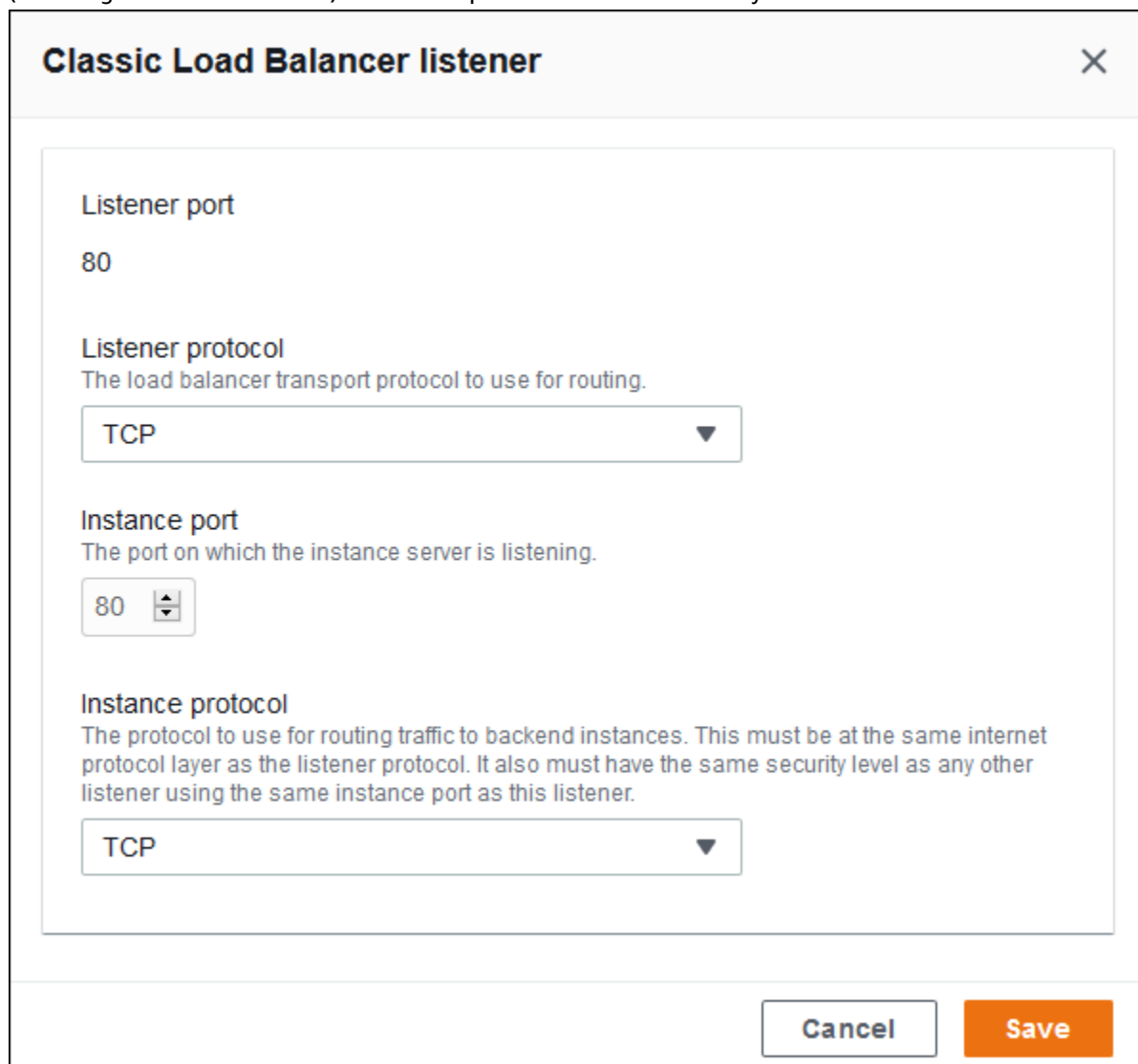
You can specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port using a specified protocol to your instances. By default, we've configured your load balancer with a standard web server on port 80.

<input type="checkbox"/>	Port	Protocol	Instance port	Instance protocol	SSL cert
<input type="checkbox"/>	80	HTTP	80	HTTP	--

To configure an existing listener

1. Select the check box next to its table entry, choose **Actions**, and then choose the action you want.
2. If you chose **Edit**, use the **Classic Load Balancer listener** dialog box to edit settings, and then choose **Save**.

For example, you can edit the default listener and change the **Protocol** from **HTTP** to **TCP** if you want the load balancer to forward a request as is. This prevents the load balancer from rewriting headers (including `X-Forwarded-For`). The technique doesn't work with sticky sessions.



The screenshot shows a dialog box titled "Classic Load Balancer listener" with a close button (X) in the top right corner. The dialog contains four configuration sections:

- Listener port:** A text input field containing the number "80".
- Listener protocol:** A dropdown menu with the text "The load balancer transport protocol to use for routing." and the value "TCP" selected.
- Instance port:** A spinner control with the number "80" and up/down arrows.
- Instance protocol:** A dropdown menu with the text "The protocol to use for routing traffic to backend instances. This must be at the same internet protocol layer as the listener protocol. It also must have the same security level as any other listener using the same instance port as this listener." and the value "TCP" selected.

At the bottom right of the dialog, there are two buttons: "Cancel" (white with a grey border) and "Save" (orange).

To add a listener

1. Choose **Add listener**.
2. In the **Classic Load Balancer listener** dialog box, configure the settings you want, and then choose **Add**.

Adding a secure listener is a common use case. The example in the following image adds a listener for HTTPS traffic on port 443. This listener routes the incoming traffic to environment instance servers listening to HTTPS traffic on port 443.

Before you can configure an HTTPS listener, ensure that you have a valid SSL certificate. Do one of the following:

- If AWS Certificate Manager (ACM) is [available in your AWS Region](#), create or import a certificate using ACM. For more information about requesting an ACM certificate, see [Request a Certificate](#) in the *AWS*

Certificate Manager User Guide. For more information about importing third-party certificates into ACM, see [Importing Certificates](#) in the *AWS Certificate Manager User Guide*.

- If ACM isn't [available in your AWS Region](#), upload your existing certificate and key to IAM. For more information about creating and uploading certificates to IAM, see [Working with Server Certificates](#) in the *IAM User Guide*.

For more detail on configuring HTTPS and working with certificates in Elastic Beanstalk, see [Configuring HTTPS for your Elastic Beanstalk environment](#) (p. 652).

For **SSL certificate**, choose the ARN of your SSL certificate. For example, `arn:aws:iam::123456789012:server-certificate/abc/certs/build`, or `arn:aws:acm:us-east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678`.


Classic Load Balancer listener ✕

Listener port
443

Listener protocol
The load balancer transport protocol to use for routing.
HTTPS

Instance port
The port on which the instance server is listening.
443

Instance protocol
The protocol to use for routing traffic to backend instances. This must be at the same internet protocol layer as the listener protocol. It also must have the same security level as any other listener using the same instance port as this listener.
HTTPS

SSL certificate
arn:aws:acm:us-east-2:123456789012:certific... 

Cancel Add

For details about configuring HTTPS and working with certificates in Elastic Beanstalk, see [Configuring HTTPS for your Elastic Beanstalk environment](#) (p. 652).

Sessions

Select or clear the **Session stickiness enabled** box to enable or disable sticky sessions. Use **Cookie duration** to configure a sticky session's duration, up to **1000000** seconds. On the **Load balancer ports** list, select listener ports that the default policy (`AWSEB-ELB-StickinessPolicy`) applies to.

Sessions

The following settings let you control whether the load balancer routes requests for the same session to the Amazon EC2 instance consistently to the same instance.

Session stickiness enabled

Cookie duration
Lifetime of the sticky session cookie between an Amazon EC2 instance and the load balancer.

0 seconds

Load balancer ports
List of the listener ports that the default policy (`AWSEB-ELB-StickinessPolicy`) applies to.

Choose load balancer ports

- 80
- 443

Cross-zone load balancing

Select or clear the **Load balancing across multiple Availability Zones enabled** box to enable or disable cross-zone load balancing.

Cross-zone load balancing

Load balancing across multiple Availability Zones enabled

Connection draining

Select or clear the **Connection draining enabled** box to enable or disable connection draining. Set the **Draining timeout**, up to **3600** seconds.

Connection draining

Connection draining enabled

Draining timeout

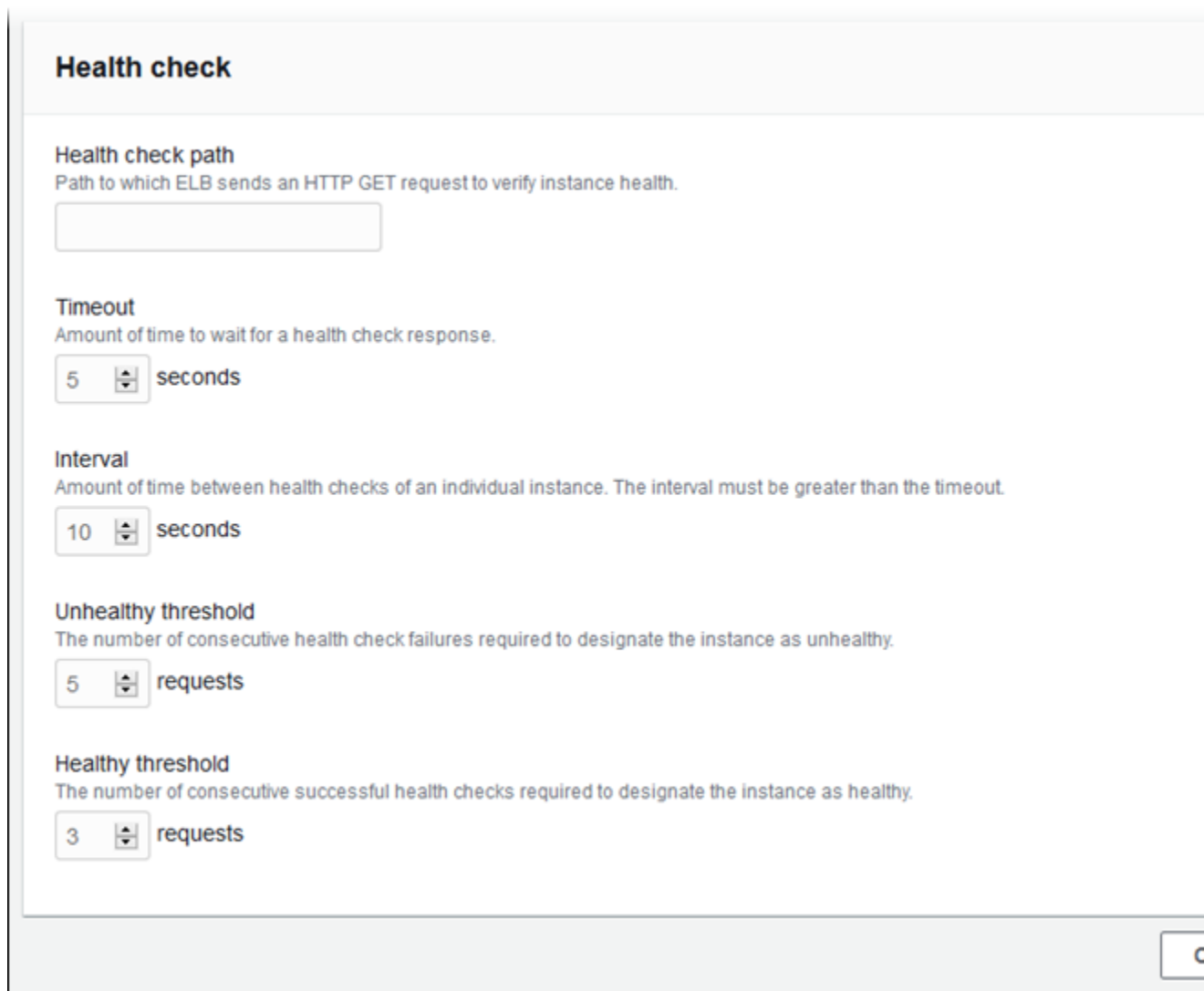
Maximum time that the load balancer maintains connections to an Amazon EC2 instance before forcibly closing connections.

20 seconds

Health check

Use the following settings to configure load balancer health checks:

- **Health check path** – The path to which the load balancer sends health check requests. If you don't set the path, the load balancer attempts to make a TCP connection on port 80 to verify health.
- **Timeout** – The amount of time, in seconds, to wait for a health check response.
- **Interval** – The amount of time, in seconds, between health checks of an individual instance. The interval must be greater than the timeout.
- **Unhealthy threshold, Healthy threshold** – The number of health checks that must fail or pass, respectively, before Elastic Load Balancing changes an instance's health state.



Health check

Health check path
Path to which ELB sends an HTTP GET request to verify instance health.

Timeout
Amount of time to wait for a health check response.

5 seconds

Interval
Amount of time between health checks of an individual instance. The interval must be greater than the timeout.

10 seconds

Unhealthy threshold
The number of consecutive health check failures required to designate the instance as unhealthy.

5 requests

Healthy threshold
The number of consecutive successful health checks required to designate the instance as healthy.

3 requests

Note

The Elastic Load Balancing health check doesn't affect the health check behavior of an environment's Auto Scaling group. Instances that fail an Elastic Load Balancing health check are not automatically replaced by Amazon EC2 Auto Scaling unless you manually configure Amazon EC2 Auto Scaling to do so. See [Auto Scaling health check setting \(p. 469\)](#) for details.

For more information about health checks and how they influence your environment's overall health, see [Basic health reporting \(p. 688\)](#).

Configuring a Classic Load Balancer using the EB CLI

The EB CLI prompts you to choose a load balancer type when you run [eb create \(p. 894\)](#).

```
$ eb create
Enter Environment Name
(default is my-app): test-env
Enter DNS CNAME prefix
(default is my-app): test-env-DLW24ED23SF

Select a load balancer type
```

```
1) classic
2) application
3) network
(default is 1):
```

Press **Enter** to select `classic`.

You can also specify a load balancer type by using the `--elb-type` option.

```
$ eb create test-env --elb-type classic
```

Classic Load Balancer configuration namespaces

You can find settings related to Classic Load Balancers in the following namespaces:

- [aws:elb:healthcheck \(p. 582\)](#) – Configure the thresholds, check interval, and timeout for load balancer health checks.
- [aws:elasticbeanstalk:application \(p. 569\)](#) – Configure the health check URL.
- [aws:elb:loadbalancer \(p. 583\)](#) – Enable cross-zone load balancing. Assign security groups to the load balancer and override the default security group that Elastic Beanstalk creates. This namespace also includes deprecated options for configuring the standard and secure listeners that have been replaced by options in the `aws:elb:listener` namespace.
- [aws:elb:listener \(p. 584\)](#) – Configure the default listener on port 80, a secure listener on port 443, or additional listeners for any protocol on any port. If you specify `aws:elb:listener` as the namespace, settings apply to the default listener on port 80. If you specify a port (for example, `aws:elb:listener:443`), a listener is configured on that port.
- [aws:elb:policies \(p. 585\)](#) – Configure additional settings for your load balancer. Use options in this namespace to configure listeners on arbitrary ports, modify additional sticky session settings, and configure the load balancer to connect to Amazon EC2 instances securely.

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See [Recommended values \(p. 537\)](#) for details.

Example `.ebextensions/loadbalancer-terminatehttps.config`

The following example configuration file creates an HTTPS listener on port 443, assigns a certificate that the load balancer uses to terminate the secure connection, and disables the default listener on port 80. The load balancer forwards the decrypted requests to the EC2 instances in your environment on HTTP:80.

```
option_settings:
  aws:elb:listener:443:
    ListenerProtocol: HTTPS
    SSLCertificateId: arn:aws:acm:us-
east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678
    InstancePort: 80
    InstanceProtocol: HTTP
  aws:elb:listener:
    ListenerEnabled: false
```

Configuring an Application Load Balancer

When you [enable load balancing \(p. 436\)](#), your AWS Elastic Beanstalk environment is equipped with an Elastic Load Balancing load balancer to distribute traffic among the instances in your environment.

Elastic Load Balancing supports several load balancer types. To learn about them, see the [Elastic Load Balancing User Guide](#).

This topic describes the configuration of an [Application Load Balancer](#). For information about configuring all the load balancer types that Elastic Beanstalk supports, see [Load balancer for your Elastic Beanstalk environment \(p. 470\)](#).

Note

You can choose the type of load balancer that your environment uses only during environment creation. You can change settings to manage the behavior of your running environment's load balancer, but you can't change its type.

Introduction

An Application Load Balancer inspects traffic at the application network protocol layer to identify the request's path so that it can direct requests for different paths to different destinations.

When your environment uses an Application Load Balancer, Elastic Beanstalk configures it by default to perform the same function as a Classic Load Balancer. The default listener accepts HTTP requests on port 80 and distributes them to the instances in your environment. You can add a secure listener on port 443 with a certificate to decrypt HTTPS traffic, configure health check behavior, and push access logs from the load balancer to an Amazon Simple Storage Service (Amazon S3) bucket.

Note

Unlike a Classic Load Balancer or a Network Load Balancer, an Application Load Balancer can't have transport layer (layer 4) TCP or SSL/TLS listeners. It supports only HTTP and HTTPS listeners. Additionally, it can't use backend authentication to authenticate HTTPS connections between the load balancer and backend instances.

In an Elastic Beanstalk environment, you can use an Application Load Balancer to direct traffic for certain paths to a different port on your web server instances. With a Classic Load Balancer, all traffic to a listener is routed to a single port on the backend instances. With an Application Load Balancer, you can configure multiple *rules* on the listener to route requests to certain paths to different backend ports.

For example, you could run a login process separately from your main application. While the main application on your environment's instances accepts the majority of requests and listens on port 80, your login process listens on port 5000 and accepts requests to the `/login` path. All incoming requests from clients come in on port 80. With an Application Load Balancer, you can configure a single listener for incoming traffic on port 80, with two rules that route traffic to two separate processes, depending on the path in the request. One rule routes traffic to `/login` to the login process listening on port 5000. The default rule routes all other traffic to the main application process listening on port 80.

An Application Load Balancer rule maps a request to a *target group*. In Elastic Beanstalk, a target group is represented by a *process*. You can configure a process with a protocol, port, and health check settings. The process represents the process running on the instances in your environment. The default process is a listener on port 80 of the reverse proxy (nginx or Apache) that runs in front of your application.

Note

Outside of Elastic Beanstalk, a target group maps to a group of instances. A listener can use rules and target groups to route traffic to different instances based on the path. Within Elastic Beanstalk, all of your instances in your environment are identical, so the distinction is made between processes listening on different ports.

A Classic Load Balancer uses a single health check path for the entire environment. With an Application Load Balancer, each process has a separate health check path that is monitored by the load balancer and Elastic Beanstalk-enhanced health monitoring.

To use an Application Load Balancer, your environment must be in a default or custom VPC, and must have a service role with the standard set of permissions. If you have an older service role, you might need to [update the permissions \(p. 763\)](#) on it to include `elasticloadbalancing:DescribeTargetHealth` and

`elasticloadbalancing:DescribeLoadBalancers`. For more information about Application Load Balancers, see [What is an Application Load Balancer?](#).

Note

The Application Load Balancer health check doesn't use the Elastic Beanstalk health check path. Instead, it uses the specific path configured for each process separately.

Configuring an Application Load Balancer using the Elastic Beanstalk console

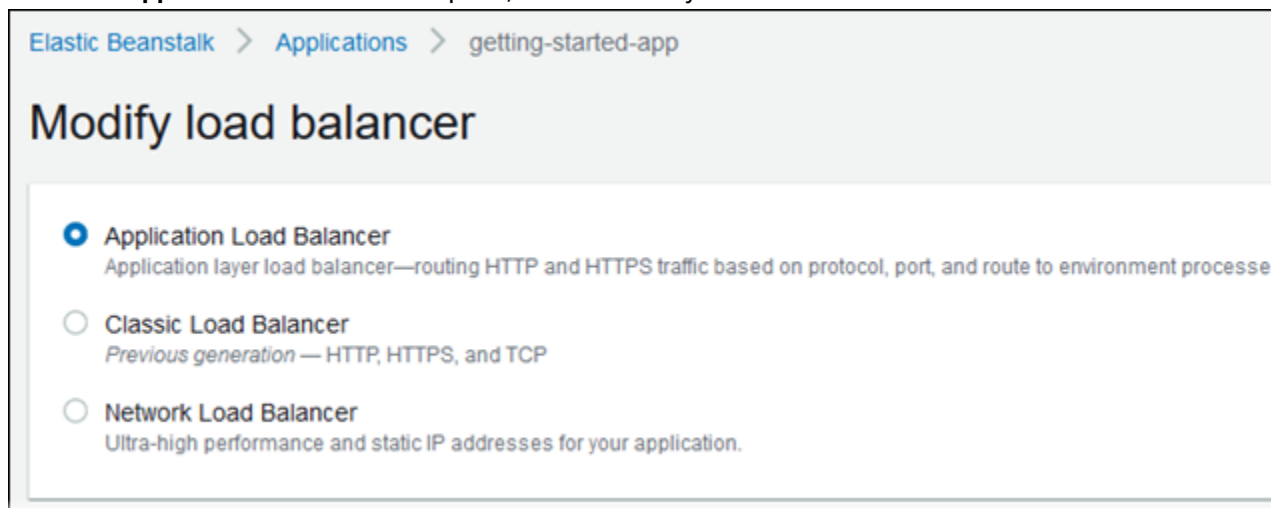
You can use the Elastic Beanstalk console to configure an Application Load Balancer's listeners, processes, and rules, during environment creation or later when your environment is running.

To configure an Application Load Balancer in the Elastic Beanstalk console during environment creation

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**.
3. Choose [Create a new environment \(p. 365\)](#) to start creating your environment.
4. On the wizard's main page, before choosing **Create environment**, choose **Configure more options**.
5. Choose the **High availability** configuration preset.

Alternatively, in the **Capacity** configuration category, configure a **Load balanced** environment type. For details, see [Capacity \(p. 373\)](#).

6. In the **Load balancer** configuration category, choose **Edit**.
7. Select the **Application Load Balancer** option, if it isn't already selected.



8. Make any Application Load Balancer configuration changes that your environment requires.
9. Choose **Save**, and then make any other configuration changes that your environment requires.
10. Choose **Create environment**.

To configure a running environment's Application Load Balancer in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Load balancer** configuration category, choose **Edit**.

Note

If the **Load balancer** configuration category doesn't have an **Edit** button, your environment doesn't have a load balancer. To learn how to set one up, see [Changing environment type \(p. 436\)](#).

5. Make the Application Load Balancer configuration changes that your environment requires.
6. Choose **Apply**.

Application Load Balancer settings

- [Listeners \(p. 482\)](#)
- [Processes \(p. 483\)](#)
- [Rules \(p. 487\)](#)
- [Access log capture \(p. 489\)](#)
- [Example: Application Load Balancer with a secure listener and two processes \(p. 490\)](#)

Listeners

Use this list to specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port using a specified protocol to one or more processes on your instances. Initially, the list shows the default listener, which routes incoming HTTP traffic on port 80 to a process named **default**, which listens to HTTP port 80.

Application Load Balancer

You can specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port using a specified protocol to one or more processes. By default, we've configured your load balancer with a standard web server on port 80.

<input type="checkbox"/>	Port	Protocol	SSL certificate
<input type="checkbox"/>	80	HTTP	--

Actions ▼

To configure an existing listener

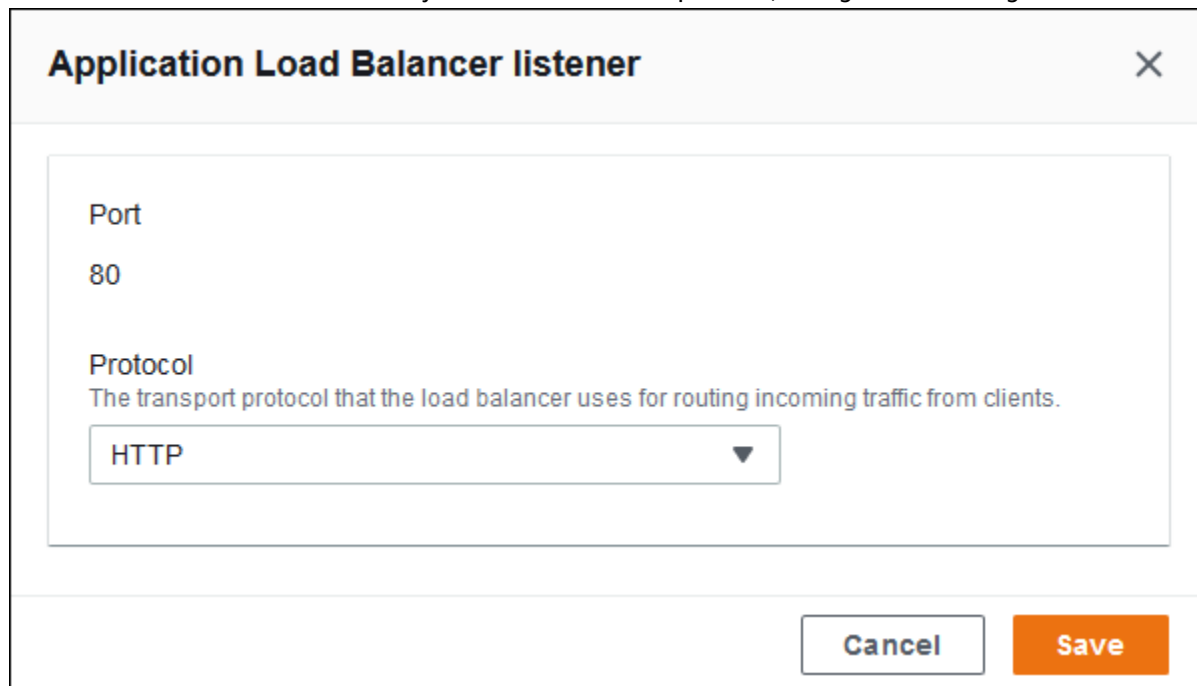
1. Select the check box next to its table entry, and then choose **Actions, Edit**.
2. Use the **Application Load Balancer listener** dialog box to edit settings, and then choose **Save**.

To add a listener

1. Choose **Add listener**.

2. In the **Application Load Balancer listener** dialog box, configure settings you want, and then choose **Add**.

Use the **Application Load Balancer listener** dialog box settings to choose the port and protocol on which the listener listens to traffic. If you choose the HTTPS protocol, configure SSL settings.



The screenshot shows a dialog box titled "Application Load Balancer listener". It contains two main configuration fields: "Port" with the value "80" and "Protocol" with a dropdown menu set to "HTTP". Below the protocol dropdown is a descriptive text: "The transport protocol that the load balancer uses for routing incoming traffic from clients." At the bottom right of the dialog are "Cancel" and "Save" buttons.

Before you can configure an HTTPS listener, ensure that you have a valid SSL certificate. Do one of the following:

- If AWS Certificate Manager (ACM) is [available in your AWS Region](#), create or import a certificate using ACM. For more information about requesting an ACM certificate, see [Request a Certificate](#) in the *AWS Certificate Manager User Guide*. For more information about importing third-party certificates into ACM, see [Importing Certificates](#) in the *AWS Certificate Manager User Guide*.
- If ACM isn't [available in your AWS Region](#), upload your existing certificate and key to IAM. For more information about creating and uploading certificates to IAM, see [Working with Server Certificates](#) in the *IAM User Guide*.

For more detail on configuring HTTPS and working with certificates in Elastic Beanstalk, see [Configuring HTTPS for your Elastic Beanstalk environment \(p. 652\)](#).

Processes

Use this list to specify processes for your load balancer. A process is a target for listeners to route traffic to. Each listener routes incoming client traffic on a specified port using a specified protocol to one or more processes on your instances. Initially, the list shows the default process, which listens to incoming HTTP traffic on port 80.

Processes

For each environment process, you can specify the protocol and port that the load balancer uses to route requests to the process. You can also specify the health check path that the load balancer performs process health checks.

Actions ▼

<input type="checkbox"/>	Name	Port	Protocol	HTTP code	Health check path
<input type="checkbox"/>	default	80	HTTP		/

You can edit the settings of an existing process, or add a new process. To start editing a process on the list or adding a process to it, use the same steps listed for the [listener list \(p. 482\)](#). The **Environment process** dialog box opens.

Application Load Balancer's environment process dialog box settings

- [Definition \(p. 484\)](#)
- [Health check \(p. 485\)](#)
- [Sessions \(p. 487\)](#)

Definition

Use these settings to define the process: its **Name**, and the **Port** and **Protocol** on which it listens to requests.

Environment process [X]

Name
default

Port
80

Protocol
HTTP

Health check

Use the following settings to configure process health checks:

- **HTTP code** – The HTTP status code designating a healthy process.
- **Path** – The health check request path for the process.
- **Timeout** – The amount of time, in seconds, to wait for a health check response.
- **Interval** – The amount of time, in seconds, between health checks of an individual instance. The interval must be greater than the timeout.
- **Unhealthy threshold, Healthy threshold** – The number of health checks that must fail or pass, respectively, before Elastic Load Balancing changes an instance's health state.
- **Deregistration delay** – The amount of time, in seconds, to wait for active requests to complete before deregistering an instance.

Health check

HTTP code

HTTP status code of a healthy instance in your environment.

Path

Path to which the load balancer sends HTTP health check requests.

Timeout

Amount of time to wait for a health check response.

 seconds

Interval

Amount of time between health checks of an individual instance. The interval must be greater than the timeout.

 seconds

Unhealthy threshold

The number of consecutive health check failures required to designate the instance as unhealthy.

 requests

Healthy threshold

The number of consecutive successful health checks required to designate the instance as healthy.

 requests

Deregistration delay

Amount of time to wait for active requests to complete before deregistering.

 seconds

Note

The Elastic Load Balancing health check doesn't affect the health check behavior of an environment's Auto Scaling group. Instances that fail an Elastic Load Balancing health check are not automatically replaced by Amazon EC2 Auto Scaling unless you manually configure Amazon EC2 Auto Scaling to do so. See [Auto Scaling health check setting \(p. 469\)](#) for details.

For more information about health checks and how they influence your environment's overall health, see [Basic health reporting \(p. 688\)](#).

Sessions

Select or clear the **Stickiness policy enabled** box to enable or disable sticky sessions. Use **Cookie duration** to configure a sticky session's duration, up to **604800** seconds.

Sessions

The following settings let you control whether the load balancer routes requests for the same session to the Amazon EC2 instance with the smallest load, or consistently to the same instance.

Stickiness policy enabled

Stickiness policy enabled

Cookie duration

Lifetime of the sticky session cookie between an Amazon EC2 instance and the load balancer.

86400

Cancel Save

Rules

Use this list to specify listener rules for your load balancer. A rule maps requests that the listener receives on a specific path pattern to a target process. Each listener can have multiple rules, routing requests on different paths to different processes on your instances.

Rules have numeric priorities that determine the precedence in which they are applied to incoming requests. For each new listener you add, Elastic Beanstalk adds a default rule that routes all the listener's traffic to the default process. The default rule's precedence is the lowest; it's applied if no other rule for the same listener matches the incoming request. Initially, the list shows the default rule of the default HTTP port 80 listener.

Rules

Your load balancer routes requests to environment processes based on rules. Rules with lower priority numbers have higher precedence. If no rule matches any rule's pattern, the request is routed to the process associated with the default rule.

Actions ▼

<input type="checkbox"/>	Listener port	Name	Priority	Path pattern	Process
<input type="checkbox"/>	80	default	--	/*	default

You can edit the settings of an existing rule, or add a new rule. To start editing a rule on the list or adding a rule to it, use the same steps listed for the [listener list \(p. 482\)](#). The **Listener rule** dialog box opens, with the following settings:

- **Name** – The rule's name.
- **Listener port** – The port of the listener that the rule applies to.
- **Priority** – The rule's priority. A lower priority number has higher precedence. Priorities of a listener's rules must be unique.
- **Path pattern** – A pattern defining the request paths that the rule applies to.
- **Process** – The process to which the load balancer routes requests that match the rule.

When editing any existing rule, you can't change its **Name** and **Listener port**. When editing a default rule, **Process** is the only setting you can change.

Listener rule ✕

Name

Listener port

Priority
A lower priority number has higher precedence. Rule priorities must be unique.

Path pattern
Example: /img/*

Process

Access log capture

Use these settings to configure Elastic Load Balancing to capture logs with detailed information about requests sent to your Application Load Balancer. Access log capture is disabled by default. When **Store logs** is enabled, Elastic Load Balancing stores the logs in the **S3 bucket** that you configure. The **Prefix** setting specifies a top-level folder in the bucket for the logs. Elastic Load Balancing places the logs in a folder named `AWLogs` under your prefix. If you don't specify a prefix, Elastic Load Balancing places its folder at the root level of the bucket.

Access log files
Configure Elastic Load Balancing to capture logs with detailed information about requests sent to your Load Balancer. Logs are stored [more](#)

Store logs
(Standard Amazon S3 charges apply.)
 Enabled

S3 bucket
(You must first configure bucket permissions. [Learn more](#))
-- Choose an Amazon S3 bucket --
Choose a bucket.

Prefix
Logical hierarchy in the bucket. If you don't specify a prefix, Elastic Load Balancing stores access logs at the bucket's root.

Example: Application Load Balancer with a secure listener and two processes

In this example, your application requires end-to-end traffic encryption and a separate process for handling administrative requests.

To configure your environment's Application Load Balancer to meet these requirements, you remove the default listener, add an HTTPS listener, indicate that the default process listens to port 443 on HTTPS, and add a process and a rule for admin traffic on a different path.

To configure the load balancer for this example

1. *Add a secure listener.* For **Port**, type 443. For **Protocol**, choose HTTPS. For **SSL certificate**, choose the ARN of your SSL certificate. For example, `arn:aws:iam::123456789012:server-certificate/abc/certs/build`, or `arn:aws:acm:us-east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678`.

Application Load Balancer listener ✕

Port

443

Protocol
The transport protocol that the load balancer uses for routing incoming traffic from clients.

HTTPS

Security settings

SSL certificate

arn:aws:acm:us-east-2:123456789012:certific... ↻

SSL policy
The Secure Sockets Layer (SSL) negotiation configuration, known as a security policy, that this load balancer uses to negotiate SSL connections with clients.

ELBSecurityPolicy-2016-08

Cancel Add

You can now see your additional listener on the list.

<input type="checkbox"/>	Port	Protocol	SSL certificate
<input type="checkbox"/>	80	HTTP	--
<input type="checkbox"/>	443	HTTPS	arn:aws:acm:us-east-2:123456789012:certificate/12345678-34cd-56ef-12345678

2. *Disable the default port 80 HTTP listener.* For the default listener, turn off the **Enabled** option.

<input type="checkbox"/>	Port	Protocol	SSL certificate
<input type="checkbox"/>	80	HTTP	--
<input type="checkbox"/>	443	HTTPS	arn:aws:acm:us-east-2:123456789012:certificate/12345678-34cd-56ef-12345678

3. *Configure the default process to HTTPS.* Select the default process, and then for **Actions**, choose **Edit**. For **Port**, type 443. For **Protocol**, choose HTTPS.

Environment process [X]

Name
default

Port
443

Protocol
HTTPS

4. *Add an admin process.* For **Name**, type admin. For **Port**, type 443. For **Protocol**, choose HTTPS. Under **Health check**, for **Path** type /admin.

Environment process [X]

Name
admin

Port
443

Protocol
HTTPS

Health check

HTTP code
HTTP status code of a healthy instance in your environment.
200

Path
Path to which the load balancer sends HTTP health check requests.
/admin

5. Add a rule for admin traffic. For **Name**, type admin. For **Listener port**, type 443. For **Path pattern**, type /admin/*. For **Process**, choose admin.

Listener rule [X]

Name
admin

Listener port
443 ▼

Priority
A lower priority number has higher precedence. Rule priorities must be unique.
1 ▼

Path pattern
Example: /img/*
/admin/*

Process
admin ▼

Cancel Add

Configuring an Application Load Balancer using the EB CLI

The EB CLI prompts you to choose a load balancer type when you run `eb create` (p. 894).

```
$ eb create
Enter Environment Name
(default is my-app): test-env
Enter DNS CNAME prefix
(default is my-app): test-env-DLW24ED23SF

Select a load balancer type
1) classic
2) application
3) network
(default is 1): 2
```

You can also specify a load balancer type with the `--elb-type` option.

```
$ eb create test-env --elb-type application
```

Application Load Balancer namespaces

You can find settings related to Application Load Balancers in the following namespaces:

- `aws:elasticbeanstalk:environment` (p. 572) – Choose the load balancer type for the environment. The value for an Application Load Balancer is `application`.
- `aws:elbv2:loadbalancer` (p. 590) – Configure access logs and other settings that apply to the Application Load Balancer as a whole.
- `aws:elbv2:listener` (p. 588) – Configure listeners on the Application Load Balancer. These settings map to the settings in `aws:elb:listener` for Classic Load Balancers.
- `aws:elbv2:listenerrule` (p. 589) – Configure rules that route traffic to different processes, depending on the request path. Rules are unique to Application Load Balancers.
- `aws:elasticbeanstalk:environment:process` (p. 573) – Configure health checks and specify the port and protocol for the processes that run on your environment's instances. The port and protocol settings map to the instance port and instance protocol settings in `aws:elb:listener` for a listener on a Classic Load Balancer. Health check settings map to the settings in the `aws:elb:healthcheck` and `aws:elasticbeanstalk:application` namespaces.

Example `.ebextensions/application-load-balancer.config`

To get started with an Application Load Balancer, use a [configuration file \(p. 600\)](#) to set the load balancer type to `application`.

```
option_settings:  
  aws:elasticbeanstalk:environment:  
    LoadBalancerType: application
```

Note

You can set the load balancer type only during environment creation.

Example `.ebextensions/alb-access-logs.config`

The following configuration file enables access log uploads for an environment with an Application Load Balancer.

```
option_settings:  
  aws:elbv2:loadbalancer:  
    AccessLogsS3Bucket: my-bucket  
    AccessLogsS3Enabled: 'true'  
    AccessLogsS3Prefix: beanstalk-alb
```

Example `.ebextensions/alb-default-process.config`

The following configuration file modifies health check and stickiness settings on the default process.

```
option_settings:  
  aws:elasticbeanstalk:environment:process:default:  
    DeregistrationDelay: '20'  
    HealthCheckInterval: '15'  
    HealthCheckPath: /  
    HealthCheckTimeout: '5'  
    HealthyThresholdCount: '3'  
    UnhealthyThresholdCount: '5'
```

```
Port: '80'  
Protocol: HTTP  
StickinessEnabled: 'true'  
StickinessLBCookieDuration: '43200'
```

Example `.ebextensions/alb-secure-listener.config`

The following configuration file adds a secure listener and a matching process on port 443.

```
option_settings:  
  aws:elbv2:listener:443:  
    DefaultProcess: https  
    ListenerEnabled: 'true'  
    Protocol: HTTPS  
    SSLCertificateArns: arn:aws:acm:us-east-2:123456789012:certificate/21324896-0fa4-412b-  
bf6f-f362d6eb6dd7  
  aws:elasticbeanstalk:environment:process:https:  
    Port: '443'  
    Protocol: HTTPS
```

Example `.ebextensions/alb-admin-rule.config`

The following configuration file adds a secure listener with a rule that routes traffic with a request path of `/admin` to a process named `admin` that listens on port 4443.

```
option_settings:  
  aws:elbv2:listener:443:  
    DefaultProcess: https  
    ListenerEnabled: 'true'  
    Protocol: HTTPS  
    Rules: admin  
    SSLCertificateArns: arn:aws:acm:us-east-2:123456789012:certificate/21324896-0fa4-412b-  
bf6f-f362d6eb6dd7  
  aws:elasticbeanstalk:environment:process:https:  
    Port: '443'  
    Protocol: HTTPS  
  aws:elasticbeanstalk:environment:process:admin:  
    HealthCheckPath: /admin  
    Port: '4443'  
    Protocol: HTTPS  
  aws:elbv2:listenerrule:admin:  
    PathPatterns: /admin/*  
    Priority: 1  
    Process: admin
```

Configuring a Network Load Balancer

When you [enable load balancing \(p. 436\)](#), your AWS Elastic Beanstalk environment is equipped with an Elastic Load Balancing load balancer to distribute traffic among the instances in your environment. Elastic Load Balancing supports several load balancer types. To learn about them, see the [Elastic Load Balancing User Guide](#).

This topic describes the configuration of a [Network Load Balancer](#). For information about configuring all the load balancer types that Elastic Beanstalk supports, see [Load balancer for your Elastic Beanstalk environment \(p. 470\)](#).

Note

You can choose the type of load balancer that your environment uses only during environment creation. You can change settings to manage the behavior of your running environment's load balancer, but you can't change its type.

Introduction

With a Network Load Balancer, the default listener accepts TCP requests on port 80 and distributes them to the instances in your environment. You can configure health check behavior, configure the listener port, or add a listener on another port.

Note

Unlike a Classic Load Balancer or an Application Load Balancer, a Network Load Balancer can't have application layer (layer 7) HTTP or HTTPS listeners. It only supports transport layer (layer 4) TCP listeners. HTTP and HTTPS traffic can be routed to your environment over TCP. To establish secure HTTPS connections between web clients and your environment, install a [self-signed certificate \(p. 653\)](#) on the environment's instances, and configure the instances to listen on the appropriate port (typically 443) and terminate HTTPS connections. The configuration varies per platform. See [Configuring your application to terminate HTTPS connections at the instance \(p. 658\)](#) for instructions. Then configure your Network Load Balancer to add a listener that maps to a process listening on the appropriate port.

A Network Load Balancer supports active health checks. These checks are based on messages to the root (/) path. In addition, a Network Load Balancer supports passive health checks. It automatically detects faulty backend instances and routes traffic only to healthy instances.

Configuring a Network Load Balancer using the Elastic Beanstalk console

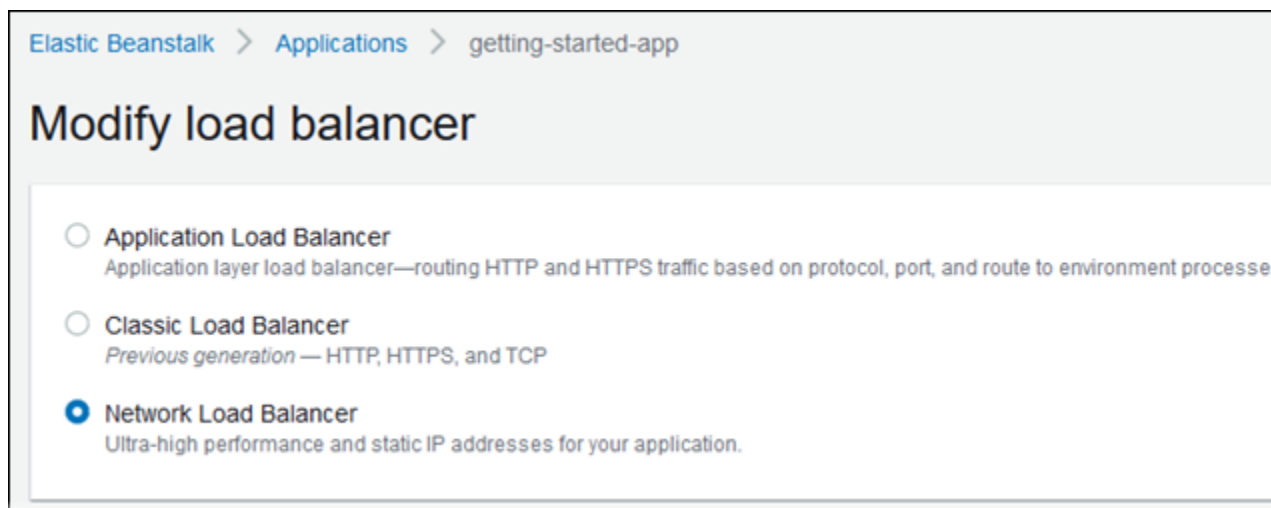
You can use the Elastic Beanstalk console to configure a Network Load Balancer's listeners and processes during environment creation, or later when your environment is running.

To configure a Network Load Balancer in the Elastic Beanstalk console during environment creation

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**.
3. Choose [Create a new environment \(p. 365\)](#) to start creating your environment.
4. On the wizard's main page, before choosing **Create environment**, choose **Configure more options**.
5. Choose the **High availability** configuration preset.

Alternatively, in the **Capacity** configuration category, configure a **Load balanced** environment type. For details, see [Capacity \(p. 373\)](#).

6. In the **Load balancer** configuration category, choose **Edit**.
7. Select the **Network Load Balancer** option, if it isn't already selected.



8. Make any Network Load Balancer configuration changes that your environment requires.
9. Choose **Save**, and then make any other configuration changes that your environment requires.
10. Choose **Create environment**.

To configure a running environment's Network Load Balancer in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Load balancer** configuration category, choose **Edit**.

Note

If the **Load balancer** configuration category doesn't have an **Edit** button, your environment doesn't have a load balancer. To learn how to set one up, see [Changing environment type](#) (p. 436).

5. Make the Network Load Balancer configuration changes that your environment requires.
6. Choose **Apply**.

Network Load Balancer settings

- [Listeners](#) (p. 498)
- [Processes](#) (p. 500)
- [Example: Network Load Balancer for an environment with end-to-end encryption](#) (p. 502)

Listeners

Use this list to specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port to a process on your instances. Initially, the list shows the default listener, which routes incoming traffic on port 80 to a process named **default**, which listens to port 80.

Network Load Balancer

You can specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port using TCP to an environment (by the port that the process listens on). By default, we've configured your load balancer with a listener on port 80 that routes traffic to a process on port 80.

Actions ▼

<input type="checkbox"/>	Listener port	Process port	Protocol	Enabled
<input type="checkbox"/>	80	80	TCP	<input checked="" type="checkbox"/>

To configure an existing listener

1. Select the check box next to its table entry, and then choose **Actions, Edit**.
2. Use the **Network Load Balancer listener** dialog box to edit settings, and then choose **Save**.

To add a listener

1. Choose **Add listener**.
2. In the **Network Load Balancer listener** dialog box, configure the required settings, and then choose **Add**.

Use the **Network Load Balancer listener** dialog box to configure the port on which the listener listens to traffic, and to choose the process to which you want to route traffic (specified by the port that the process listens to).

Network Load Balancer listener ✕

Listener port
80

Protocol
The transport protocol that the load balancer uses for routing incoming traffic from clients.
TCP ▼

Process port
The port to which this listener routes traffic. It determines the environment process that receives traffic from the listener.
80 ▼

Cancel Save

Processes

Use this list to specify processes for your load balancer. A process is a target for listeners to route traffic to. Each listener routes incoming client traffic on a specified port to a process on your instances. Initially, the list shows the default process, which listens to incoming traffic on port 80.

Processes

For each environment process, you can specify the port that the load balancer uses to route requests to the process. You can also specify the interval and healthy threshold that the load balancer performs process health checks.

Actions ▼

<input type="checkbox"/>	Process name	Process port	Interval	Healthy threshold	Unhealthy threshold
<input type="checkbox"/>	default	80	10	5	5

Cancel

You can edit the settings of an existing process, or add a new process. To start editing a process on the list or adding a process to it, use the same steps listed for the [listener list \(p. 482\)](#). The **Environment process** dialog box opens.

Network Load Balancer's environment process dialog box settings

- [Definition \(p. 501\)](#)
- [Health check \(p. 501\)](#)

Definition

Use these settings to define the process: its **Name** and the **Process port** on which it listens to requests.



The screenshot shows a dialog box titled "Environment process" with a close button (X) in the top right corner. The dialog contains two input fields. The first field is labeled "Name" and has the text "default" entered. The second field is labeled "Process port" and has a dropdown menu with "80" selected. The dialog is enclosed in a thin black border.

Health check

Use the following settings to configure process health checks:

- **Interval** – The amount of time, in seconds, between health checks of an individual instance.
- **Healthy threshold** – The number of health checks that must pass before Elastic Load Balancing changes an instance's health state. (For Network Load Balancer, **Unhealthy threshold** is a read-only setting that is always equal to the healthy threshold value.)
- **Deregistration delay** – The amount of time, in seconds, to wait for active requests to complete before deregistering an instance.

Health check

Interval
Amount of time between health checks of an individual instance.

10

seconds

Healthy threshold
The number of consecutive successful health checks required to designate the instance as healthy.

5 requests

Unhealthy threshold
The number of consecutive health check failures required to designate the instance as unhealthy.

5 requests

Deregistration delay
Amount of time to wait for active requests to complete before deregistering.

20 seconds

Note

The Elastic Load Balancing health check doesn't affect the health check behavior of an environment's Auto Scaling group. Instances that fail an Elastic Load Balancing health check will not automatically be replaced by Amazon EC2 Auto Scaling unless you manually configure Amazon EC2 Auto Scaling to do so. See [Auto Scaling health check setting \(p. 469\)](#) for details.

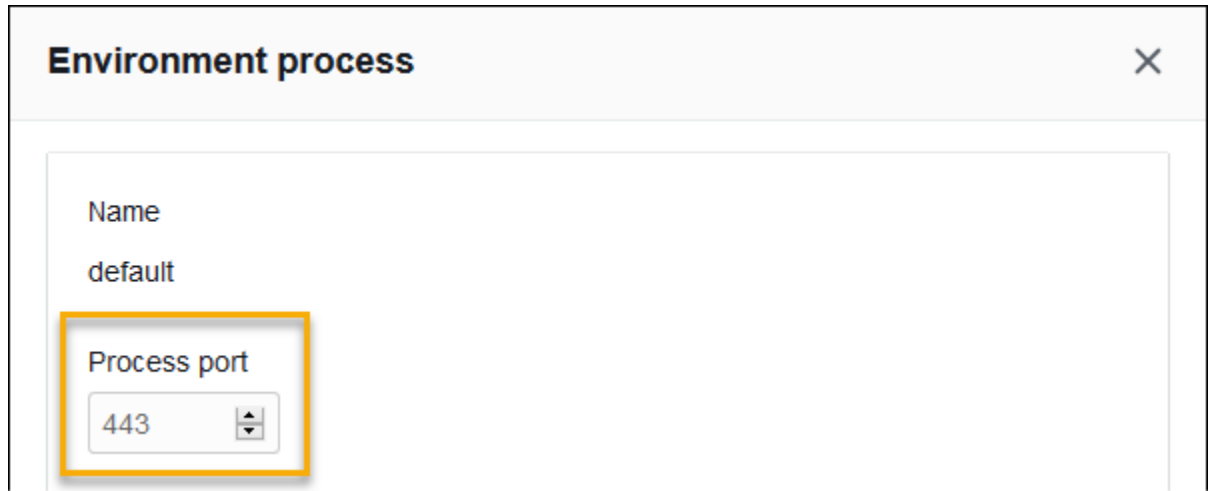
For more information about health checks and how they influence your environment's overall health, see [Basic health reporting \(p. 688\)](#).

Example: Network Load Balancer for an environment with end-to-end encryption

In this example, your application requires end-to-end traffic encryption. To configure your environment's Network Load Balancer to meet these requirements, you configure the default process to listen to port 443, add a listener to port 443 that routes traffic to the default process, and disable the default listener.

To configure the load balancer for this example

1. *Configure the default process.* Select the default process, and then, for **Actions**, choose **Edit**. For **Process port**, type 443.

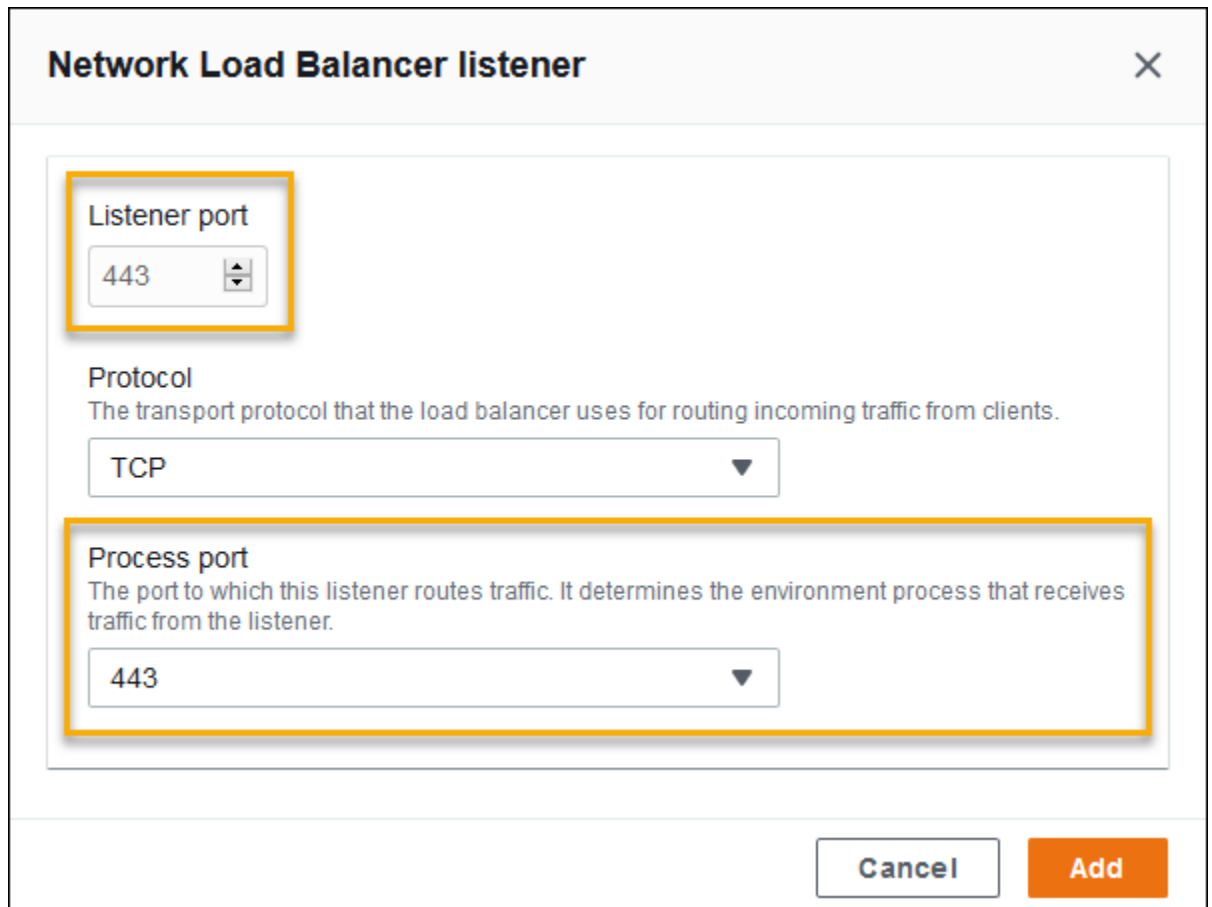


Environment process [X]

Name
default

Process port
443

2. *Add a port 443 listener.* Add a new listener. For **Listener port**, type 443. For **Process port**, make sure that 443 is selected.



Network Load Balancer listener [X]

Listener port
443

Protocol
The transport protocol that the load balancer uses for routing incoming traffic from clients.
TCP

Process port
The port to which this listener routes traffic. It determines the environment process that receives traffic from the listener.
443

Cancel Add

You can now see your additional listener on the list.

<input type="checkbox"/>	Listener port	Process port	Protocol	Ena
<input type="checkbox"/>	80	443	TCP	<input checked="" type="checkbox"/>
<input type="checkbox"/>	443	443	TCP	<input checked="" type="checkbox"/>

3. *Disable the default port 80 listener.* For the default listener, turn off the **Enabled** option.

<input type="checkbox"/>	Listener port	Process port	Protocol	Ena
<input type="checkbox"/>	80	443	TCP	<input type="checkbox"/>
<input type="checkbox"/>	443	443	TCP	<input checked="" type="checkbox"/>

Configuring a Network Load Balancer using the EB CLI

The EB CLI prompts you to choose a load balancer type when you run `eb create` (p. 894).

```
$ eb create
Enter Environment Name
(default is my-app): test-env
Enter DNS CNAME prefix
(default is my-app): test-env-DLW24ED23SF

Select a load balancer type
1) classic
2) application
3) network
(default is 1): 3
```

You can also specify a load balancer type with the `--elb-type` option.

```
$ eb create test-env --elb-type network
```

Network Load Balancer namespaces

You can find settings related to Network Load Balancers in the following namespaces:

- [aws:elasticbeanstalk:environment](#) (p. 572) – Choose the load balancer type for the environment. The value for a Network Load Balancer is `network`.
- [aws:elbv2:listener](#) (p. 588) – Configure listeners on the Network Load Balancer. These settings map to the settings in `aws:elb:listener` for Classic Load Balancers.
- [aws:elasticbeanstalk:environment:process](#) (p. 573) – Configure health checks and specify the port and protocol for the processes that run on your environment's instances. The port and protocol settings map to the instance port and instance protocol settings in `aws:elb:listener` for a listener on a Classic Load Balancer. Health check settings map to the settings in the `aws:elb:healthcheck` and `aws:elasticbeanstalk:application` namespaces.

Example `.ebextensions/network-load-balancer.config`

To get started with a Network Load Balancer, use a [configuration file \(p. 600\)](#) to set the load balancer type to network.

```
option_settings:
  aws:elasticbeanstalk:environment:
    LoadBalancerType: network
```

Note

You can set the load balancer type only during environment creation.

Example `.ebextensions/nlb-default-process.config`

The following configuration file modifies health check settings on the default process.

```
option_settings:
  aws:elasticbeanstalk:environment:process:default:
    DeregistrationDelay: '20'
    HealthCheckInterval: '10'
    HealthyThresholdCount: '5'
    UnhealthyThresholdCount: '5'
    Port: '80'
    Protocol: TCP
```

Example `.ebextensions/nlb-secure-listener.config`

The following configuration file adds a listener for secure traffic on port 443 and a matching target process that listens to port 443.

```
option_settings:
  aws:elbv2:listener:443:
    DefaultProcess: https
    ListenerEnabled: 'true'
  aws:elasticbeanstalk:environment:process:https:
    Port: '443'
```

The `DefaultProcess` option is named this way because of Application Load Balancers, which can have non-default listeners on the same port for traffic to specific paths (see [Application Load Balancer \(p. 479\)](#) for details). For a Network Load Balancer the option specifies the only target process for this listener.

In this example, we named the process `https` because it listens to secure (HTTPS) traffic. The listener sends traffic to the process on the designated port using the TCP protocol, because a Network Load Balancer works only with TCP. This is okay, because network traffic for HTTP and HTTPS is implemented on top of TCP.

Configuring access logs

You can use [configuration files \(p. 600\)](#) to configure your environment's load balancer to upload access logs to an Amazon S3 bucket. See the following example configuration files on GitHub for instructions:

- [loadbalancer-accesslogs-existingbucket.config](#) – Configure the load balancer to upload access logs to an existing Amazon S3 bucket.
- [loadbalancer-accesslogs-newbucket.config](#) – Configure the load balancer to upload access logs to a new bucket.

Adding a database to your Elastic Beanstalk environment

Elastic Beanstalk provides integration with [Amazon Relational Database Service \(Amazon RDS\)](#) to help you add a database instance to your Elastic Beanstalk environment. You can use Elastic Beanstalk to add a MySQL, PostgreSQL, Oracle, or SQL Server database to your environment during or after environment creation. When you add a database instance to your environment, Elastic Beanstalk provides connection information to your application by setting environment properties for the database hostname, port, user name, password, and database name.

A database instance that is part of your environment is tied to the lifecycle of your environment. You can't remove it from your environment once added. If you terminate the environment, the database instance is terminated as well. You can configure Elastic Beanstalk to save a snapshot of the database when you terminate your environment, and restore a database from a snapshot when you add a DB instance to an environment. You might incur charges for storing database snapshots. For more information, see the *Backup Storage* section of [Amazon RDS Pricing](#).

For a production environment, you can [launch a database instance outside of your environment \(p. 820\)](#) and configure your application to connect to it outside of the functionality provided by Elastic Beanstalk. Using a database instance that is external to your environment requires additional security group and connection string configuration. However, it also lets you connect to the database from multiple environments, use database types not supported with integrated databases, perform blue/green deployments, and tear down your environment without affecting the database instance.

Sections

- [Adding an Amazon RDS DB instance to your environment \(p. 506\)](#)
- [Connecting to the database \(p. 509\)](#)
- [Configuring an integrated RDS DB Instance using the console \(p. 509\)](#)
- [Configuring an integrated RDS DB Instance using configuration files \(p. 510\)](#)

Adding an Amazon RDS DB instance to your environment

You can add a DB instance to your environment by using the Elastic Beanstalk console.

To add a DB instance to your environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Choose a DB engine, and enter a user name and password.
6. Choose **Apply**.

You can configure the following options:

- **Snapshot** – Choose an existing database snapshot. Elastic Beanstalk restores the snapshot and adds it to your environment. The default value is **None**, which lets you configure a new database using the other settings on this page.
- **Engine** – Choose a database engine.
- **Engine version** – Choose a specific version of the database engine.
- **Instance class** – Choose the DB instance class. For information about the DB instance classes, see <https://aws.amazon.com/rds/>.
- **Storage** – Choose the amount of storage to provision for your database. You can increase allocated storage later, but you cannot decrease it. For information about storage allocation, see [Features](#).
- **Username** – Enter a user name of your choice using alphanumeric characters.
- **Password** – Enter a password of your choice containing 8–16 printable ASCII characters (excluding /, \, and @).
- **Retention** – Choose **Create snapshot** to create a snapshot of the database when you terminate your environment.
- **Availability** – Choose **High (Multi-AZ)** to run a warm backup in a second Availability Zone for high availability.

Note

Elastic Beanstalk creates a master user for the database using the user name and password you provide. To learn more about the master user and its privileges, see [Master User Account Privileges](#).

Elastic Beanstalk > Environments > GettingStartedApp-env > Configuration

Modify database

Add an Amazon RDS SQL database to your environment for development and testing. AWS Elastic Beanstalk provides information to your instances by setting environment properties for the database hostname, username, password, and so on. When you add a database to your environment, its lifecycle is tied to your environment's. For production environments, you can configure your instances to connect to a database. [Learn more](#)

Restore a snapshot

Restore an existing snapshot in your account, or create a new database.

Snapshot

None

Database settings

Choose an engine and instance type for your environment's database.

Engine

mysql

Engine version

5.6.41

Instance class

db.t2.micro

Storage

Choose a number between 5 GB and 1024 GB.

5

Username

Password

Retention

Create snapshot

Adding a DB instance takes about 10 minutes. When the environment update is complete, the DB instance's hostname and other connection information are available to your application through the following environment properties:

Property name	Description	Property value
RDS_HOSTNAME	The hostname of the DB instance.	On the Connectivity & security tab on the Amazon RDS console: Endpoint .
RDS_PORT	The port on which the DB instance accepts connections. The default value varies among DB engines.	On the Connectivity & security tab on the Amazon RDS console: Port .
RDS_DB_NAME	The database name, ebdb .	On the Configuration tab on the Amazon RDS console: DB Name .
RDS_USERNAME	The username that you configured for your database.	On the Configuration tab on the Amazon RDS console: Master username .
RDS_PASSWORD	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

Connecting to the database

Use the connectivity information to connect to your DB from inside your application through environment variables. For more information about using Amazon RDS with your applications, see the following topics.

- Java SE – [Connecting to a database \(Java SE platforms\) \(p. 119\)](#)
- Java with Tomcat – [Connecting to a database \(Tomcat platforms\) \(p. 120\)](#)
- Node.js – [Connecting to a database \(p. 227\)](#)
- .NET – [Connecting to a database \(p. 165\)](#)
- PHP – [Connecting to a database with a PDO or MySQLi \(p. 287\)](#)
- Python – [Connecting to a database \(p. 313\)](#)
- Ruby – [Connecting to a database \(p. 331\)](#)

Configuring an integrated RDS DB Instance using the console

You can view and modify configuration settings for your DB instance in the **Database** section on the environment's **Configuration** page in the [Elastic Beanstalk console \(p. 353\)](#).

To configure your environment's DB instance in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.

You can modify the **Instance class**, **Storage**, **Password**, **Retention**, and **Availability** settings after database creation. If you change the instance class, Elastic Beanstalk re-provisions the DB instance.

Warning

Don't modify settings on the DB instance outside of the functionality provided by Elastic Beanstalk (for example, in the Amazon RDS console). If you do, your Amazon RDS DB configuration might be out of sync with your environment's definition. When you update or restart your environment, the settings specified in the environment override any settings you made outside of Elastic Beanstalk.

If you need to modify settings that Elastic Beanstalk doesn't directly support, use Elastic Beanstalk [configuration files](#) (p. 510).

Configuring an integrated RDS DB Instance using configuration files

You can configure your environment's DB instance using [configuration files](#) (p. 600). Use the options in the `aws:rds:dbinstance` (p. 591) namespace. The following example modifies the allocated database storage size to 100 GB.

Example `.ebextensions/db-instance-options.config`

```
option_settings:  
  aws:rds:dbinstance:  
    DBAllocatedStorage: 100
```

If you need to configure DB instance properties that Elastic Beanstalk doesn't support, you can still use a configuration file, and specify your settings using the `resources` key. The following example sets values to the `StorageType` and `Iops` Amazon RDS properties.

Example `.ebextensions/db-instance-properties.config`

```
Resources:  
  AWSEBRDSDatabase:  
    Type: AWS::RDS::DBInstance  
    Properties:  
      StorageType: io1  
      Iops: 1000
```

Your AWS Elastic Beanstalk environment security

Elastic Beanstalk provides several options that control the security of your environment and of the Amazon EC2 instances in it. This topic discusses the configuration of these options.

Sections

- [Configuring your environment security](#) (p. 511)
- [Environment security configuration namespaces](#) (p. 513)

Configuring your environment security

You can modify your Elastic Beanstalk environment security configuration in the Elastic Beanstalk console.

To configure environment security in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Security** configuration category, choose **Edit**.

The following settings are available.

Settings

- [Service role \(p. 512\)](#)
- [EC2 key pair \(p. 512\)](#)
- [IAM instance profile \(p. 513\)](#)

Elastic Beanstalk > Environments > GettingStartedApp-env > Configuration

Modify security

Service role

Service role

aws-elasticbeanstalk-service-role ▼ ↻

Virtual machine permissions

EC2 key pair

-- Choose a key pair -- ▼ ↻

IAM instance profile

aws-elasticbeanstalk-ec2-role ▼ ↻

Cancel Continue

Service role

Select a [service role](#) (p. 764) to associate with your Elastic Beanstalk environment. Elastic Beanstalk assumes the service role when it accesses other AWS services on your behalf. For details, see [Managing Elastic Beanstalk service roles](#) (p. 764).

EC2 key pair

You can securely log in to the Amazon Elastic Compute Cloud (Amazon EC2) instances provisioned for your Elastic Beanstalk application with an Amazon EC2 key pair. For instructions on creating a key pair, see [Creating a Key Pair Using Amazon EC2](#) in the *Amazon EC2 User Guide for Linux Instances*.

Note

When you create a key pair, Amazon EC2 stores a copy of your public key. If you no longer need to use it to connect to any environment instances, you can delete it from Amazon EC2. For details, see [Deleting Your Key Pair](#) in the *Amazon EC2 User Guide for Linux Instances*.

Choose an **EC2 key pair** from the drop-down menu to assign it to your environment's instances. When you assign a key pair, the public key is stored on the instance to authenticate the private key, which you store locally. The private key is never stored on AWS.

For more information about connecting to Amazon EC2 instances, see [Connect to Your Instance](#) and [Connecting to Linux/UNIX Instances from Windows using PuTTY](#) in the *Amazon EC2 User Guide for Linux Instances*.

IAM instance profile

An [instance profile](#) (p. 21) is an IAM role that is applied to instances launched in your Elastic Beanstalk environment. Amazon EC2 instances assume the instance profile role to sign requests to AWS and access APIs, for example, to upload logs to Amazon S3.

The first time you create an environment in the Elastic Beanstalk console, Elastic Beanstalk prompts you to create an instance profile with a default set of permissions. You can add permissions to this profile to provide your instances access to other AWS services. For details, see [Managing Elastic Beanstalk instance profiles](#) (p. 760).

Environment security configuration namespaces

Elastic Beanstalk provides [configuration options](#) (p. 536) in the following namespaces to enable you to customize the security of your environment:

- [aws:elasticbeanstalk:environment](#) (p. 572) – Configure the environment's service role using the `ServiceRole` option.
- [aws:autoscaling:launchconfiguration](#) (p. 556) – Configure permissions for the environment's Amazon EC2 instances using the `EC2KeyName` and `IamInstanceProfile` options.

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See [Recommended values](#) (p. 537) for details.

Tagging resources in your Elastic Beanstalk environments

You can apply tags to your AWS Elastic Beanstalk environments. Tags are key-value pairs associated with AWS resources. For information about Elastic Beanstalk resource tagging, use cases, tag key and value constraints, and supported resource types, see [Tagging Elastic Beanstalk application resources](#) (p. 349).

You can also use tags to manage permissions at the specific resource level within an environment. For more information, see [Tagging Your Amazon EC2 Resources](#) in the *Amazon EC2 User Guide for Linux Instances*.

By default, Elastic Beanstalk applies three tags to your environment:

- `elasticbeanstalk:environment-name` – The name of the environment.
- `elasticbeanstalk:environment-id` – The environment ID.
- `Name` – Also the name of the environment. `Name` is used in the Amazon EC2 dashboard to identify and sort resources.

You can't edit these default tags.

You can specify tags when you create the Elastic Beanstalk environment. In an existing environment, you can add or remove tags, and update the values of existing tags. In addition to the default tags, you can add up to 47 additional tags to each environment.

Adding tags during environment creation

When you use the Elastic Beanstalk console to create an environment, you can specify tag keys and values on the **Modify tags** configuration page of the [Create New Environment wizard](#) (p. 365).

Elastic Beanstalk > Applications > getting-started-app

Modify tags

Apply up to 50 tags to the resources in your environment in addition to the default tags.

Key	Value	
mytag1	value1	Remove

Add tag

49 remaining

If you use the EB CLI to create an environment, use the `--tags` option with [eb create](#) (p. 894) to add tags.

```
~/workspace/my-app$ eb create --tags mytag1=value1,mytag2=value2
```

With the AWS CLI or other API-based clients, use the `--tags` parameter on the [create-environment](#) command.

```
$ aws elasticbeanstalk create-environment \  
  --tags Key=mytag1,Value=value1 Key=mytag2,Value=value2 \  
  --application-name my-app --environment-name my-env --cname-prefix my-app --version-  
label v1 --template-name my-saved-config
```

[Saved configurations](#) (p. 540) include user-defined tags. When you apply a saved configuration that contains tags during environment creation, those tags are applied to the new environment, as long as you don't specify any new tags. If you add tags to an environment using one of the preceding methods, any tags defined in the saved configuration are discarded.

Managing tags of an existing environment

You can add, update, and delete tags in an existing Elastic Beanstalk environment. Elastic Beanstalk applies the changes to your environment's resources.

However, you can't edit the default tags that Elastic Beanstalk applies to your environment.

To manage an environment's tags in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Tags**.

The tag management page shows the list of tags that currently exist in the environment.

The screenshot shows the 'Tags for GettingStartedApp-env' page in the AWS Elastic Beanstalk console. The breadcrumb navigation is 'Elastic Beanstalk > Environments > GettingStartedApp-env > Tags'. Below the title, there is a descriptive text: 'Apply up to 47 tags in addition to the default tags to the resources in your environment. You can use tags to group environments. A tag is a key-value pair. The key must be unique within the environment and is case-sensitive.' The main content area is a table with two columns: 'Key' and 'Value'. The first two rows show pre-filled tags: 'elasticbeanstalk:environment-id' with value 'e-cubmdjm6ga', and 'elasticbeanstalk:environment-name' with value 'GettingStartedApp-env'. The third row shows 'Name' with value 'GettingStartedApp-env'. Below these are two empty rows for adding new tags, with 'mytag1' and 'value1' in the first, and 'mytag2' and 'value2' in the second. At the bottom left, there is an 'Add tag' button and a counter showing '45 remaining'.

Key	Value
elasticbeanstalk:environment-id	e-cubmdjm6ga
elasticbeanstalk:environment-name	GettingStartedApp-env
Name	GettingStartedApp-env
mytag1	value1
mytag2	value2

Add tag
45 remaining

4. Add, update, or delete tags:
 - To add a tag, enter it into the empty boxes at the bottom of the list. To add another tag, choose **Add tag** and Elastic Beanstalk adds another pair of empty boxes.
 - To update a tag's key or value, edit the respective box in the tag's row.
 - To delete a tag, choose **Remove** next to the tag's value box.
5. Choose **Apply**.

If you use the EB CLI to update your environment, use [eb tags \(p. 930\)](#) to add, update, delete, or list tags.

For example, the following command lists the tags in your default environment.

```
~/workspace/my-app$ eb tags --list
```

The following command updates the tag `mytag1` and deletes the tag `mytag2`.

```
~/workspace/my-app$ eb tags --update mytag1=newvalue --delete mytag2
```

For a complete list of options and more examples, see [eb tags](#) (p. 930).

With the AWS CLI or other API-based clients, use the [list-tags-for-resource](#) command to list the tags of an environment.

```
$ aws elasticbeanstalk list-tags-for-resource --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-id:environment/my-app/my-env"
```

Use the [update-tags-for-resource](#) command to add, update, or delete tags in an environment.

```
$ aws elasticbeanstalk update-tags-for-resource \
  --tags-to-add Key=mytag1,Value=newvalue --tags-to-remove mytag2 \
  --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-id:environment/my-app/my-env"
```

Specify both tags to add and tags to update in the `--tags-to-add` parameter of [update-tags-for-resource](#). A nonexistent tag is added, and an existing tag's value is updated.

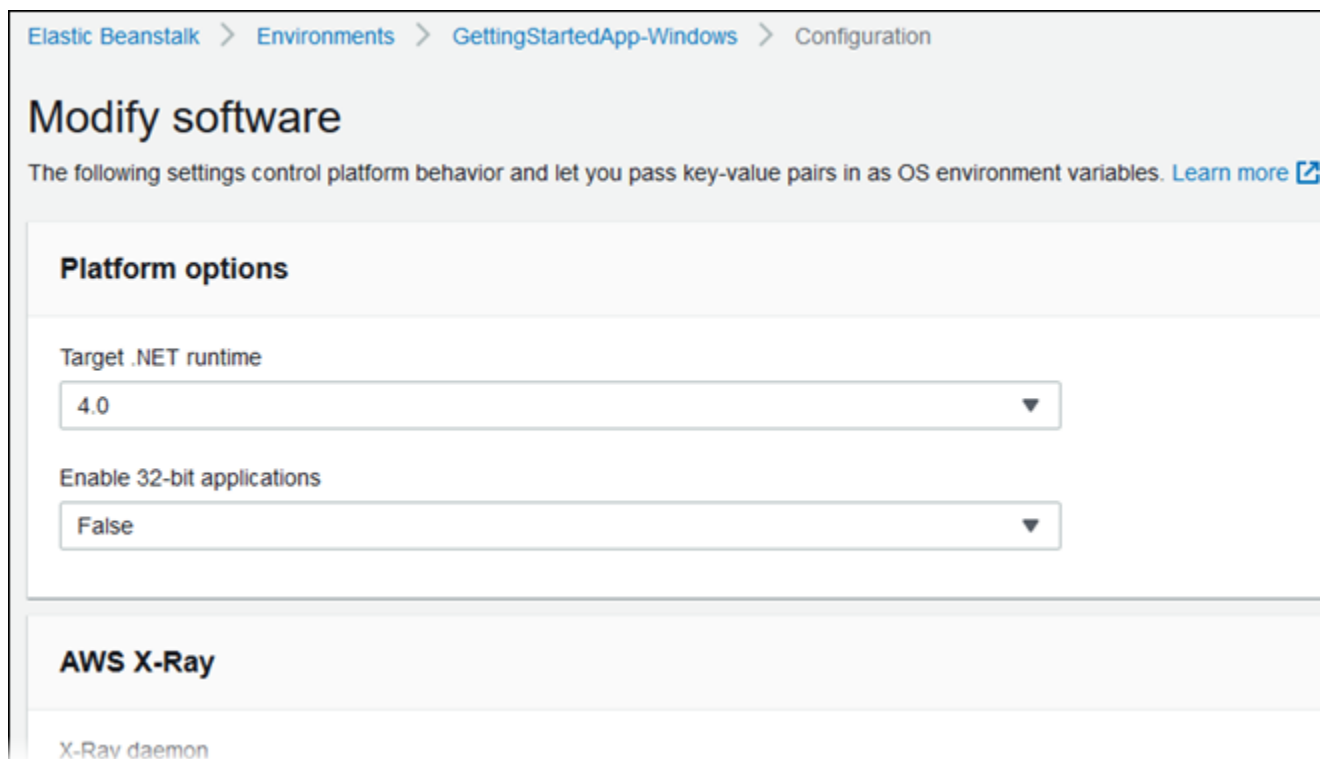
Note

To use these two AWS CLI commands with an Elastic Beanstalk environment, you need the environment's ARN. You can retrieve the ARN by using the following command.

```
$ aws elasticbeanstalk describe-environments
```

Environment properties and other software settings

The **Modify software** configuration page lets you configure the software on the Amazon Elastic Compute Cloud (Amazon EC2) instances that run your application. You can configure environment properties, AWS X-Ray debugging, instance log storing and streaming, and platform-specific settings.



Topics

- [Configure platform-specific settings \(p. 517\)](#)
- [Configuring environment properties \(p. 518\)](#)
- [Software setting namespaces \(p. 519\)](#)
- [Accessing environment properties \(p. 521\)](#)
- [Configuring AWS X-Ray debugging \(p. 521\)](#)
- [Viewing your Elastic Beanstalk environment logs \(p. 524\)](#)

Configure platform-specific settings

In addition to the standard set of options available for all environments, most Elastic Beanstalk platforms let you specify language-specific or framework-specific settings. These appear in the **Platform options** section of the **Modify software** page, and can take the following forms.

- **Preset environment properties** – The Ruby platform uses environment properties for framework settings, such as `RACK_ENV` and `BUNDLE_WITHOUT`.
- **Placeholder environment properties** – The Tomcat platform defines an environment property named `JDBC_CONNECTION_STRING` that is not set to any value. This type of setting was more common on older platform versions.
- **Configuration options** – Most platforms define [configuration options \(p. 536\)](#) in platform-specific or shared namespaces, such as `aws:elasticbeanstalk:xray` or `aws:elasticbeanstalk:container:python`.

To configure platform-specific settings in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. Under **Platform options**, make necessary option setting changes.
6. Choose **Apply**.

For information about platform-specific options, and about getting environment property values in your code, see the platform topic for your language or framework:

- Docker – [the section called “Environment configuration” \(p. 74\)](#)
- Go – [Using the Elastic Beanstalk Go platform \(p. 86\)](#)
- Java SE – [Using the Elastic Beanstalk Java SE platform \(p. 112\)](#)
- Tomcat – [Using the Elastic Beanstalk Tomcat platform \(p. 101\)](#)
- .NET – [Using the Elastic Beanstalk .NET platform \(p. 141\)](#)
- Node.js – [Using the Elastic Beanstalk Node.js platform \(p. 195\)](#)
- PHP – [Using the Elastic Beanstalk PHP platform \(p. 230\)](#)
- Python – [Using the Elastic Beanstalk Python platform \(p. 290\)](#)
- Ruby – [Using the Elastic Beanstalk Ruby platform \(p. 316\)](#)

Configuring environment properties

You can use **environment properties** to pass secrets, endpoints, debug settings, and other information to your application. Environment properties help you run your application in multiple environments for different purposes, such as development, testing, staging, and production.

In addition, when you [add a database to your environment \(p. 506\)](#), Elastic Beanstalk sets environment properties, such as `RDS_HOSTNAME`, that you can read in your application code to construct a connection object or string.

Environment variables

In most cases, environment properties are passed to your application as *environment variables*, but the behavior is platform dependent. For example, [the Java SE platform \(p. 112\)](#) sets environment variables that you retrieve with `System.getenv`, while [the Tomcat platform \(p. 101\)](#) sets Java system properties that you retrieve with `System.getProperty`. In general, properties are *not* visible if you connect to an instance and run `env`.

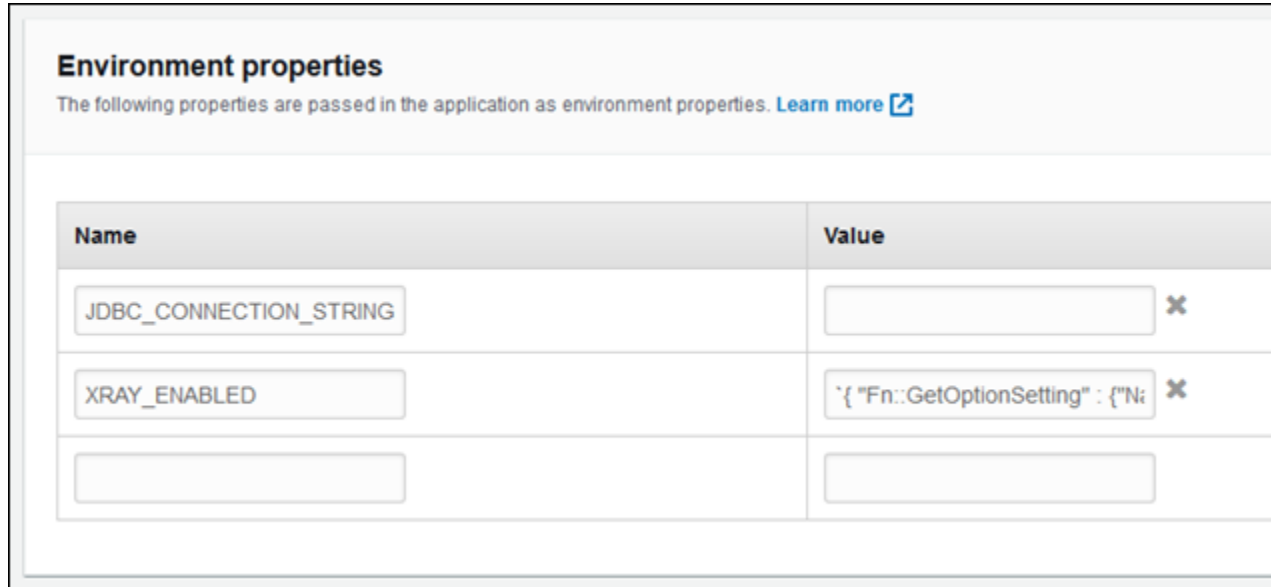
To configure environment properties in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. Under **Environment properties**, enter key-value pairs.



6. Choose **Apply**.

Environment property limits

- **Keys** can contain any alphanumeric characters and the following symbols: `_ . : / + \ - @`

The symbols listed are valid for environment property keys, but might not be valid for environment variable names on your environment's platform. For compatibility with all platforms, limit environment properties to the following pattern: `[A-Z_][A-Z0-9_]*`

- **Values** can contain any alphanumeric characters, white space, and the following symbols: `_ . : / = + \ - @ ' "`

Note

Single and double quotation marks in values must be escaped.

- **Keys** can contain up to 128 characters. **Values** can contain up to 256 characters.
- **Keys** and **values** are case sensitive.
- The combined size of all environment properties cannot exceed 4,096 bytes when stored as strings with the format `key=value`.

Software setting namespaces

You can use a [configuration file \(p. 600\)](#) to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be defined by the Elastic Beanstalk service or the platform that you use and are organized into *namespaces*.

You can use Elastic Beanstalk [configuration files \(p. 600\)](#) to set environment properties and configuration options in your source code. Use the [aws:elasticbeanstalk:application:environment namespace \(p. 570\)](#) to define environment properties.

Example .ebextensions/options.config

```
option_settings:  
  aws:elasticbeanstalk:application:environment:
```

```
API_ENDPOINT: www.example.com/api
```

If you use configuration files or AWS CloudFormation templates to create [custom resources \(p. 622\)](#), you can use an AWS CloudFormation function to get information about the resource and assign it to an environment property dynamically during deployment. The following example from the [elastic-beanstalk-samples](#) GitHub repository uses the [Ref function \(p. 626\)](#) to get the ARN of an Amazon SNS topic that it creates, and assigns it to an environment property named `NOTIFICATION_TOPIC`.

Notes

- If you use an AWS CloudFormation function to define an environment property, the Elastic Beanstalk console displays the value of the property before the function is evaluated. You can use the [get-config platform script \(p. 45\)](#) to confirm the values of environment properties that are available to your application.
- The [Multicontainer Docker \(p. 58\)](#) platform doesn't use AWS CloudFormation to create container resources. As a result, this platform doesn't support defining environment properties using AWS CloudFormation functions.

Example `.Ebextensions/sns-topic.config`

```
Resources:
  NotificationTopic:
    Type: AWS::SNS::Topic

option_settings:
  aws:elasticbeanstalk:application:environment:
    NOTIFICATION_TOPIC: `{"Ref" : "NotificationTopic"}`
```

You can also use this feature to propagate information from [AWS CloudFormation pseudo parameters](#). This example gets the current region and assigns it to a property named `AWS_REGION`.

Example `.Ebextensions/env-regionname.config`

```
option_settings:
  aws:elasticbeanstalk:application:environment:
    AWS_REGION: `{"Ref" : "AWS::Region"}`
```

Most Elastic Beanstalk platforms define additional namespaces with options for configuring software that runs on the instance, such as the reverse proxy that relays requests to your application. For more information about the namespaces available for your platform, see the following:

- Go – [Go configuration namespace \(p. 88\)](#)
- Java SE – [Java SE configuration namespace \(p. 114\)](#)
- Tomcat – [Tomcat configuration namespaces \(p. 104\)](#)
- .NET – [The `aws:elasticbeanstalk:container:dotnet:apppool` namespace \(p. 142\)](#)
- Node.js – [Node.js configuration namespace \(p. 197\)](#)
- PHP – [The `aws:elasticbeanstalk:container:php:phpini` namespace \(p. 232\)](#)
- Python – [Python configuration namespaces \(p. 292\)](#)
- Ruby – [Ruby configuration namespaces \(p. 318\)](#)

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See [Configuration options \(p. 536\)](#) for more information.

Accessing environment properties

In most cases, you access environment properties in your application code like an environment variable. In general, however, environment properties are passed only to the application and can't be viewed by connecting an instance in your environment and running `env`.

- [Go \(p. 88\)](#) – `os.Getenv`

```
endpoint := os.Getenv("API_ENDPOINT")
```

- [Java SE \(p. 113\)](#) – `System.getenv`

```
String endpoint = System.getenv("API_ENDPOINT");
```

- [Tomcat \(p. 103\)](#) – `System.getProperty`

```
String endpoint = System.getProperty("API_ENDPOINT");
```

- [.NET \(p. 142\)](#) – `appConfig`

```
NameValueCollection appConfig = ConfigurationManager.AppSettings;  
string endpoint = appConfig["API_ENDPOINT"];
```

Note

Elastic Beanstalk doesn't support passing environment variables to .NET Core applications and multiple-application IIS deployments that use a [deployment manifest \(p. 145\)](#).

- [Node.js \(p. 197\)](#) – `process.env`

```
var endpoint = process.env.API_ENDPOINT
```

- [PHP \(p. 232\)](#) – `$_SERVER`

```
$endpoint = $_SERVER['API_ENDPOINT'];
```

- [Python \(p. 292\)](#) – `os.environ`

```
import os  
endpoint = os.environ['API_ENDPOINT']
```

- [Ruby \(p. 318\)](#) – `ENV`

```
endpoint = ENV['API_ENDPOINT']
```

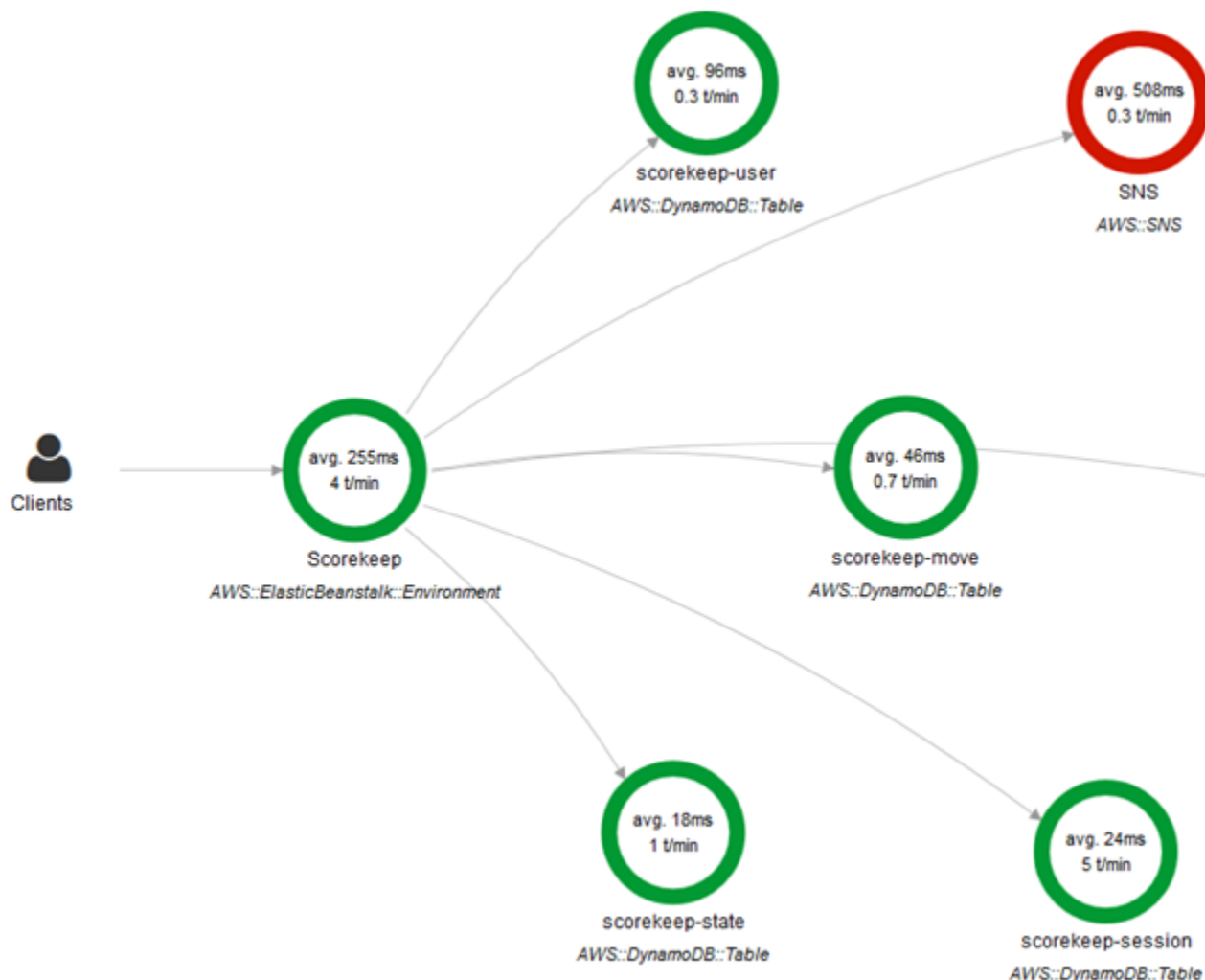
Outside of application code, such as in a script that runs during deployment, you can access environment properties with the [get-config platform script \(p. 45\)](#). See the [elastic-beanstalk-samples](#) GitHub repository for example configurations that use `get-config`.

Configuring AWS X-Ray debugging

You can use the AWS Elastic Beanstalk console or a configuration file to run the AWS X-Ray daemon on the instances in your environment. X-Ray is an AWS service that gathers data about the requests that your application serves, and uses it to construct a service map that you can use to identify issues with your application and opportunities for optimization.

Note

Some regions don't offer X-Ray. If you create an environment in one of these regions, you can't run the X-Ray daemon on the instances in your environment. For information about the AWS services offered in each region, see [Region Table](#).



X-Ray provides an SDK that you can use to instrument your application code, and a daemon application that relays debugging information from the SDK to the X-Ray API.

Supported platforms

You can use the X-Ray SDK with the following Elastic Beanstalk platforms:

- **Go** - version 2.9.1 and later
- **Java 8** - version 2.3.0 and later
- **Java 8 with Tomcat 8** - version 2.4.0 and later
- **Node.js** - version 3.2.0 and later
- **Windows Server** - all platform versions released on or after December 18th, 2016
- **Python** - version 2.5.0 and later

On supported platforms, you can use a configuration option to run the X-Ray daemon on the instances in your environment. You can enable the daemon in the [Elastic Beanstalk console \(p. 523\)](#) or by using a [configuration file \(p. 523\)](#).

To upload data to X-Ray, the X-Ray daemon requires IAM permissions in the **AWSXrayWriteOnlyAccess** managed policy. These permissions are included in [the Elastic Beanstalk instance profile \(p. 21\)](#). If you don't use the default instance profile, see [Giving the Daemon Permission to Send Data to X-Ray](#) in the *AWS X-Ray Developer Guide*.

Debugging with X-Ray requires the use of the X-Ray SDK. See the [Getting Started with AWS X-Ray](#) in the *AWS X-Ray Developer Guide* for instructions and sample applications.

If you use a platform version that doesn't include the daemon, you can still run it with a script in a configuration file. For more information, see [Downloading and Running the X-Ray Daemon Manually \(Advanced\)](#) in the *AWS X-Ray Developer Guide*.

Sections

- [Configuring debugging \(p. 523\)](#)
- [The aws:elasticbeanstalk:xray namespace \(p. 523\)](#)

Configuring debugging

You can enable the X-Ray daemon on a running environment in the Elastic Beanstalk console.

To enable debugging in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. In the **AWS X-Ray** section, select **X-Ray daemon**.
6. Choose **Apply**.

You can also enable this option during environment creation. For more information, see [The create new environment wizard \(p. 365\)](#).

The aws:elasticbeanstalk:xray namespace

You can use the `XRayEnabled` option in the `aws:elasticbeanstalk:xray` namespace to enable debugging.

To enable debugging automatically when you deploy your application, set the option in a [configuration file \(p. 600\)](#) in your source code, as follows.

Example .ebextensions/debugging.config

```
option_settings:
  aws:elasticbeanstalk:xray:
    XRayEnabled: true
```

Viewing your Elastic Beanstalk environment logs

AWS Elastic Beanstalk provides two ways to regularly view logs from the Amazon EC2 instances that run your application:

- Configure your Elastic Beanstalk environment to upload rotated instance logs to the environment's Amazon S3 bucket.
- Configure the environment to stream instance logs to Amazon CloudWatch Logs.

When you configure instance log streaming to CloudWatch Logs, Elastic Beanstalk creates CloudWatch Logs log groups for proxy and deployment logs on the Amazon EC2 instances, and transfers these log files to CloudWatch Logs in real time. For more information about instance logs, see [Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment](#) (p. 732).

In addition to instance logs, if you enable [enhanced health](#) (p. 691) for your environment, you can configure the environment to stream health information to CloudWatch Logs. When the environment's health status changes, Elastic Beanstalk adds a record to a health log group, with the new status and a description of the cause of the change. For information about environment health streaming, see [Streaming Elastic Beanstalk environment health information to Amazon CloudWatch Logs](#) (p. 750).

Configuring instance log viewing

To view instance logs, you can enable instance log rotation and log streaming in the Elastic Beanstalk console.

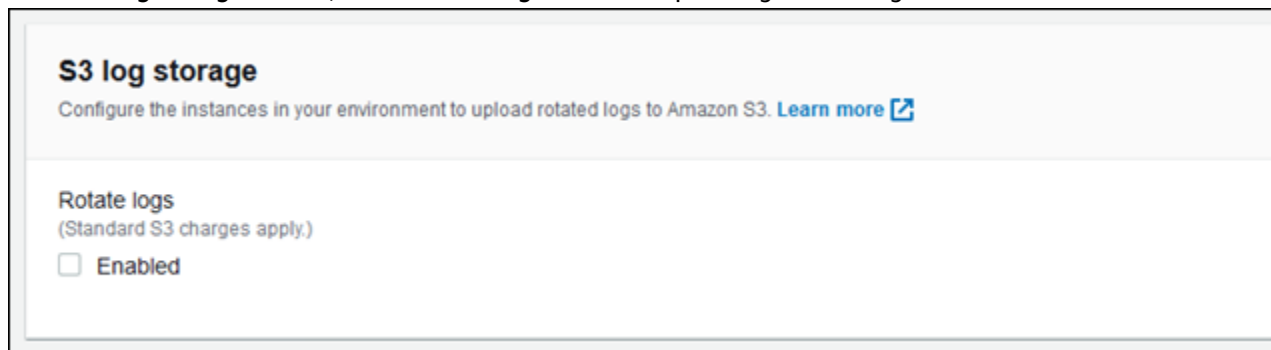
To configure instance log rotation and log streaming in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. In the **S3 log storage** section, select **Rotate logs** to enable uploading rotated logs to Amazon S3.



6. in the **Instance log streaming to CloudWatch Logs** section, configure the following settings:
 - **Log streaming** – Select to enable log streaming.
 - **Retention** – Specify the number of days to retain logs in CloudWatch Logs.
 - **Lifecycle** – Set to **Delete logs upon termination** to delete logs from CloudWatch Logs immediately if the environment is terminated, instead of waiting for them to expire.

7. Choose **Apply**.

After you enable log streaming, you can return to the **Software** configuration category or page and find the **Log Groups** link. Click this link to see your instance logs in the CloudWatch console.

Instance log streaming to CloudWatch Logs
Configure the instances in your environment to stream logs to CloudWatch Logs. You can set the retention to up to ten years and configure Elastic logs when you terminate your environment.

Log groups
[/aws/elasticbeanstalk/GettingStartedApp-env](#)

Log streaming
(Standard CloudWatch charges apply.)
 Enabled

Retention
7 days

Lifecycle
Keep logs after terminating environment

Configuring environment health log viewing

To view environment health logs, you can enable environment health log streaming in the Elastic Beanstalk console.

To configure environment health log streaming in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Monitoring** configuration category, choose **Edit**.
5. Under **Health event streaming to CloudWatch Logs**, configure the following settings:
 - **Log streaming** – Choose to enable log streaming.
 - **Retention** – Specify the number of days to retain logs in CloudWatch Logs.
 - **Lifecycle** – Set to **Delete logs upon termination** to delete logs from CloudWatch Logs immediately if the environment is terminated, instead of waiting for them to expire.
6. Choose **Apply**.

After you enable log streaming, you can return to the **Monitoring** configuration category or page and find the **Log Group** link. Click this link to see your environment health logs in the CloudWatch console.

Health event streaming to CloudWatch Logs

Configure Elastic Beanstalk to stream environment health events to CloudWatch Logs. You can set the retention period to a maximum of ten years and configure Elastic Beanstalk to delete the logs when you terminate your environment.

Log group [/aws/elasticbeanstalk/GettingStartedApp-env/environment-health.log](#)

Log streaming Enabled (Standard CloudWatch charges apply.)

Retention 7 days

Lifecycle Keep logs after terminating environment

Log viewing namespaces

The following namespaces contain settings for log viewing:

- [aws:elasticbeanstalk:hostmanager](#) (p. 578) – Configure uploading rotated logs to Amazon S3.
- [aws:elasticbeanstalk:cloudwatch:logs](#) (p. 570) – Configure instance log streaming to CloudWatch.
- [aws:elasticbeanstalk:cloudwatch:logs:health](#) (p. 570) – Configure environment health streaming to CloudWatch.

Elastic Beanstalk environment notifications with Amazon SNS

You can configure your AWS Elastic Beanstalk environment to use Amazon Simple Notification Service (Amazon SNS) to notify you of important events that affect your application. Specify an email address during or after environment creation to receive emails from AWS when an error occurs, or when your environment's health changes.

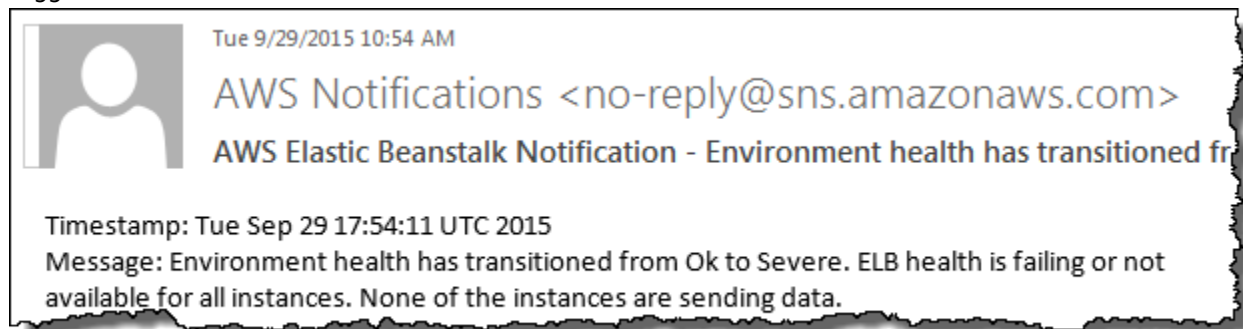
Note

Elastic Beanstalk uses Amazon SNS for notifications. For details about Amazon SNS pricing, see <https://aws.amazon.com/sns/pricing/>.

When you configure notifications for your environment, Elastic Beanstalk creates an Amazon SNS topic for your environment. To send messages to an Amazon SNS topic, Elastic Beanstalk must have the required permission. For details, see [Configuring permissions to send notifications](#) (p. 529).

When a notable [event](#) (p. 728) occurs, Elastic Beanstalk sends a message to the topic. Amazon SNS relays messages it receives to the topic's subscribers. Notable events include environment creation errors and all changes in [environment and instance health](#) (p. 691). Events for Amazon EC2 Auto Scaling

operations (adding and removing instances from the environment) and other informational events don't trigger notifications.



The Elastic Beanstalk console lets you enter an email address during or after environment creation to create an Amazon SNS topic and subscribe to it. Elastic Beanstalk manages the lifecycle of the topic, and deletes it when your environment is terminated or when you remove your email address in the [environment management console](#) (p. 353).

The `aws:elasticbeanstalk:sns:topics` namespace provides options for configuring an Amazon SNS topic by using configuration files, or by using a CLI or SDK. These methods let you configure the type of subscriber and the endpoint, so that the subscriber can be an Amazon SQS queue or HTTP URL.

You can only turn Amazon SNS notifications on or off. Depending on the size and composition of your environment, the frequency of notifications sent to the topic can be high. For notifications that are sent only under specific circumstances, you can [configure your environment to publish custom metrics](#) (p. 714) and [set Amazon CloudWatch alarms](#) (p. 725) to notify you when those metrics reach an unhealthy threshold.

Configuring notifications using the Elastic Beanstalk console

The Elastic Beanstalk console lets you enter an email address to create an Amazon SNS topic for your environment.

To configure notifications in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Notifications** configuration category, choose **Edit**.
5. Enter an email address.

Elastic Beanstalk > Environments > GettingStartedApp-env > Configuration

Modify notifications

Email notifications

Enter an email address to receive email notifications for important events from your environment.

Email

Cancel

6. Choose **Apply**.

When you enter an email address for notifications, Elastic Beanstalk creates an Amazon SNS topic for your environment and adds a subscription. Amazon SNS sends an email to the subscribed address to confirm the subscription. You must click the link in the confirmation email to activate the subscription and receive notifications.

Configuring notifications using configuration options

Use the options in the [aws:elasticbeanstalk:sns:topics namespace \(p. 579\)](#) to configure Amazon SNS notifications for your environment. You can set these options by using [configuration files \(p. 600\)](#), a CLI, or an SDK.

Notification Endpoint – Email address, Amazon SQS queue, or URL to send notifications to. Setting this option creates an SQS queue and a subscription for the specified endpoint. If the endpoint is not an email address, you must also set the **Notification Protocol** option. SNS validates the value of **Notification Endpoint** based on the value of **Notification Protocol**. Setting this option multiple times creates additional subscriptions to the topic. Removing this option deletes the topic.

Notification Protocol – The protocol used to send notifications to the **Notification Endpoint**. This option defaults to `email`. Set this option to `email-json` to send JSON-formatted emails, `http` or `https` to post JSON-formatted notifications to an HTTP endpoint, or `sq` to send notifications to an SQS queue.

Note

AWS Lambda notifications are not supported.

Notification Topic ARN – After setting a notification endpoint for your environment, read this setting to get the ARN of the SNS topic. You can also set this option to use an existing SNS topic for notifications. A topic that you attach to your environment by using this option is not deleted when you change this option or terminate the environment.

`Notification Topic Name` – Set this option to customize the name of the Amazon SNS topic used for environment notifications. If a topic with the same name already exists, Elastic Beanstalk attaches that topic to the environment.

Warning

If you attach an existing SNS topic to an environment with `Notification Topic Name`, Elastic Beanstalk will delete the topic if you terminate the environment or change this setting.

Changing this option also changes the `Notification Topic ARN`. If a topic is already attached to the environment, Elastic Beanstalk deletes the old topic, and then creates a new topic and subscription.

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See [Recommended values \(p. 537\)](#) for details.

Configuring permissions to send notifications

This section discusses security considerations related to notifications using Amazon SNS. There are two distinct cases: using the default Amazon SNS topic that Elastic Beanstalk creates for your environment, and providing an external Amazon SNS topic through configuration options.

Permissions for a default topic

When you configure notifications for your environment, Elastic Beanstalk creates an Amazon SNS topic for your environment. To send messages to an Amazon SNS topic, Elastic Beanstalk must have the required permission. If your environment uses the [service role \(p. 764\)](#) that the Elastic Beanstalk console or the EB CLI generated for it, or your account's [monitoring service-linked role \(p. 770\)](#), you don't need to do anything else. These managed roles include the necessary permission.

However, if you provided a custom service role when you created your environment, be sure that this custom service role includes the following policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sns:Publish"
      ],
      "Resource": [
        "arn:aws:sns:us-east-2:123456789012:ElasticBeanstalkNotifications*"
      ]
    }
  ]
}
```

Permissions for an external topic

In [Configuring notifications using configuration options \(p. 528\)](#), we explain how you can replace the Amazon SNS topic that Elastic Beanstalk provides with another Amazon SNS topic. If you did that, Elastic Beanstalk needs to verify that you have permission to publish to this SNS topic in order to allow you to associate the SNS topic with the environment. You should have the same permission that the service role has, `sns:Publish`. To verify that, Elastic Beanstalk sends a test notification to SNS as part of your action to create or update the environment. If this test fails, then your attempt to create or update the environment fails, and Elastic Beanstalk shows a message explaining the failure.

If you provide a custom service role for your environment, be sure that your custom service role includes the following policy. Replace `sns_topic_name` with the name of the Amazon SNS topic you provided in the configuration options.


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sns:Publish"
      ],
      "Resource": [
        "arn:aws:sns:us-east-2:123456789012:sns_topic_name"
      ]
    }
  ]
}
```

Configuring Amazon Virtual Private Cloud (Amazon VPC) with Elastic Beanstalk

[Amazon Virtual Private Cloud](#) (Amazon VPC) is the networking service that routes traffic securely to the EC2 instances that run your application in Elastic Beanstalk. If you don't configure a VPC when you launch your environment, Elastic Beanstalk uses the default VPC.

You can launch your environment in a custom VPC to customize networking and security settings. Elastic Beanstalk lets you choose which subnets to use for your resources, and how to configure IP addresses for the instances and load balancer in your environment. An environment is locked to a VPC when you create it, but you can change subnet and IP address settings on a running environment.

For instructions on creating a VPC for use with Elastic Beanstalk, see [Using Elastic Beanstalk with Amazon VPC](#) (p. 834).

Configuring VPC settings in the Elastic Beanstalk console

If you chose a custom VPC when you created your environment, you can modify its VPC settings in the Elastic Beanstalk console.

To configure your environment's VPC settings

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Network** configuration category, choose **Edit**.

The following settings are available.

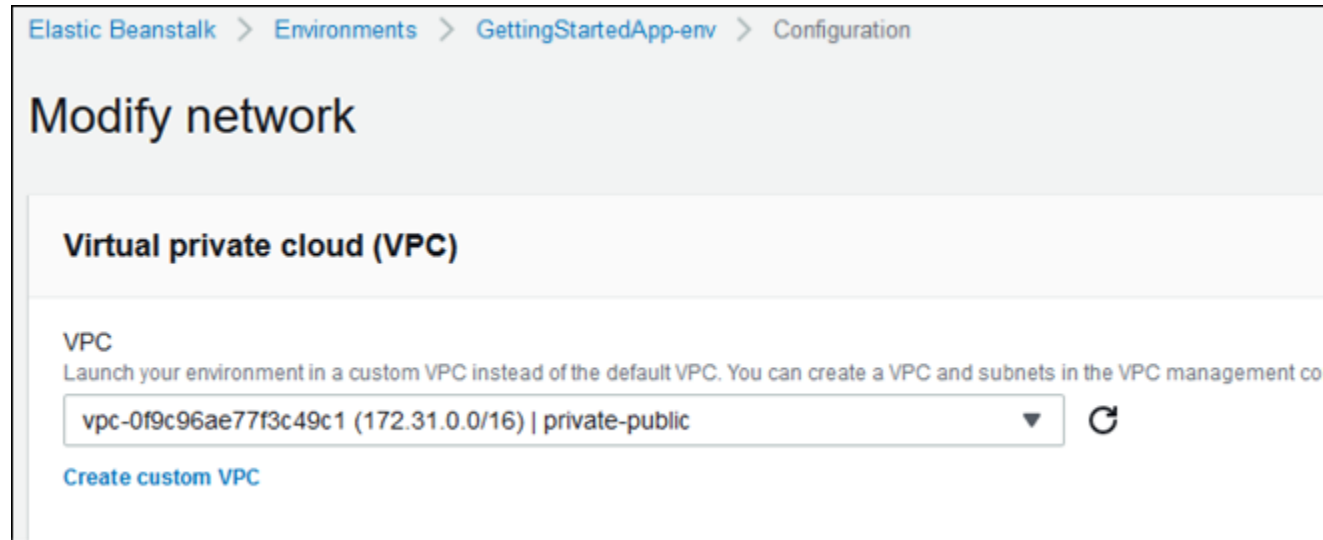
Options

- [VPC](#) (p. 531)

- [Load balancer visibility](#) (p. 531)
- [Load balancer subnets](#) (p. 531)
- [Instance public IP address](#) (p. 532)
- [Instance subnets](#) (p. 532)
- [Database subnets](#) (p. 533)

VPC

Choose a VPC for your environment. You can only change this setting during environment creation.



Load balancer visibility

For a load balanced environment, choose the load balancer scheme. By default, the load balancer is public, with a public IP address and domain name. If your application only serves traffic from within your VPC or a connected VPN, deselect this option and choose private subnets for your load balancer to make the load balancer internal and disable access from the Internet.

Load balancer subnets

For a load balanced environment, choose the subnets that your load balancer uses to serve traffic. For a public application, choose public subnets. Use subnets in multiple availability zones for high availability. For an internal application, choose private subnets and disable load balancer visibility.

Load balancer settings

Assign your load balancer to a subnet in each Availability Zone (AZ) in which your application runs. For a publicly accessible application, choose public subnets.

Visibility
Make your load balancer internal if your application serves requests only from connected VPCs. Public load balancers serve requests from the Internet.

Public

Load balancer subnets

<input type="checkbox"/>	Availability Zone	Subnet	CIDR	Name
<input checked="" type="checkbox"/>	us-east-2a	subnet-04a707767b8ca8023	172.31.0.0/24	public-a
<input type="checkbox"/>	us-east-2a	subnet-0c559eeeb1a89adb4	172.31.3.0/24	private-a
<input checked="" type="checkbox"/>	us-east-2b	subnet-034a813125cd2077a	172.31.2.0/24	private-b
<input type="checkbox"/>	us-east-2b	subnet-09a24e24e7f7359fa	172.31.1.0/24	public-b

Instance public IP address

If you choose public subnets for your application instances, enable public IP addresses to make them routable from the Internet.

Instance subnets

Choose subnets for your application instances. Choose at least one subnet for each availability zone that your load balancer uses. If you choose private subnets for your instances, your VPC must have a NAT gateway in a public subnet that the instances can use to access the Internet.

Instance settings

Choose a subnet in each AZ for the instances that run your application. To avoid exposing your instances to the Internet, run your instances and load balancer in public subnets. To run your load balancer and instances in the same public subnets, assign public IP addresses.

Public IP address
Assign a public IP address to the Amazon EC2 instances in your environment.

Instance subnets

<input type="checkbox"/>	Availability Zone	Subnet	CIDR	Name
<input type="checkbox"/>	us-east-2a	subnet-04a707767b8ca8023	172.31.0.0/24	public-a
<input checked="" type="checkbox"/>	us-east-2a	subnet-0c559eeeb1a89adb4	172.31.3.0/24	private-a
<input type="checkbox"/>	us-east-2b	subnet-034a813125cd2077a	172.31.2.0/24	private-b
<input checked="" type="checkbox"/>	us-east-2b	subnet-09a24e24e7f7359fa	172.31.1.0/24	public-b

Cancel

Database subnets

When you run an Amazon RDS database attached to your Elastic Beanstalk environment, choose subnets for your database instances. For high availability, make the database multi-AZ and choose a subnet for each availability zone. To ensure that your application can connect to your database, run both in the same subnets.

The aws:ec2:vpc namespace

You can use the configuration options in the `aws:ec2:vpc` (p. 568) namespace to configure your environment's network settings.

The following [configuration file](#) (p. 600) uses options in this namespace to set the environment's VPC and subnets for a public-private configuration. In order to set the VPC ID in a configuration file, the file must be included in the application source bundle during environment creation. See [Setting configuration options during environment creation](#) (p. 543) for other methods of configuring these settings during environment creation.

Example .ebextensions/vpc.config – Public-private

```
option_settings:
  aws:ec2:vpc:
    VPCId: vpc-087a68c03b9c50c84
```

```
AssociatePublicIpAddress: 'false'  
ELBScheme: public  
ELBSubnets: subnet-0fe6b36bcb0ffc462,subnet-032fe3068297ac5b2  
Subnets: subnet-026c6117b178a9c45,subnet-0839e902f656e8bd1
```

This example shows a public-public configuration, where the load balancer and EC2 instances run in the same public subnets.

Example .ebextensions/vpc.config – Public-public

```
option_settings:  
  aws:ec2:vpc:  
    VPCId: vpc-087a68c03b9c50c84  
    AssociatePublicIpAddress: 'true'  
    ELBScheme: public  
    ELBSubnets: subnet-0fe6b36bcb0ffc462,subnet-032fe3068297ac5b2  
    Subnets: subnet-0fe6b36bcb0ffc462,subnet-032fe3068297ac5b2
```

Your Elastic Beanstalk environment's Domain name

By default, your environment is available to users at a subdomain of `elasticbeanstalk.com`. When you [create an environment \(p. 363\)](#), you can choose a hostname for your application. The subdomain and domain are autopopulated to `region.elasticbeanstalk.com`.

To route users to your environment, Elastic Beanstalk registers a CNAME record that points to your environment's load balancer. You can see URL of your environment's application with the current value of the CNAME in the [environment overview \(p. 355\)](#) page of the Elastic Beanstalk console.

Elastic Beanstalk > Environments > GettingStartedApp-env

GettingStartedApp-env

Getting StartedApp-env.bx7dx222kw.us-east-2.elasticbeanstalk.com

Health
Ok

Running version
Sample Application-3
Upload and deploy

Choose the URL on the overview page, or choose **Go to environment** on the navigation pane, to navigate to your application's web page.

You can change the CNAME on your environment by swapping it with the CNAME of another environment. For instructions, see [Blue/Green deployments with Elastic Beanstalk \(p. 405\)](#).

If you own a domain name, you can use Amazon Route 53 to resolve it to your environment. You can purchase a domain name with Amazon Route 53, or use one that you purchase from another provider.

To purchase a domain name with Route 53, see [Registering a New Domain](#) in the *Amazon Route 53 Developer Guide*.

To learn more about using a custom domain, see [Routing Traffic to an AWS Elastic Beanstalk Environment](#) in the *Amazon Route 53 Developer Guide*.

Important

If you terminate an environment, you must also delete any CNAME mappings you created, as other customers can reuse an available hostname.

Configuring Elastic Beanstalk environments (advanced)

When you create an AWS Elastic Beanstalk environment, Elastic Beanstalk provisions and configures all of the AWS resources required to run and support your application. In addition to configuring your environment's metadata and update behavior, you can customize these resources by providing values for [configuration options](#) (p. 536). For example, you may want to add an Amazon SQS queue and an alarm on queue depth, or you might want to add an Amazon ElastiCache cluster.

Most of the configuration options have default values that are applied automatically by Elastic Beanstalk. You can override these defaults with configuration files, saved configurations, command line options, or by directly calling the Elastic Beanstalk API. The EB CLI and Elastic Beanstalk console also apply recommended values for some options.

You can easily customize your environment at the same time that you deploy your application version by including a configuration file with your source bundle. When customizing the software on your instance, it is more advantageous to use a configuration file than to create a custom AMI because you do not need to maintain a set of AMIs.

When deploying your applications, you may want to customize and configure the software that your application depends on. These files could be either dependencies required by the application—for example, additional packages from the yum repository—or they could be configuration files such as a replacement for `httpd.conf` to override specific settings that are defaulted by AWS Elastic Beanstalk.

Topics

- [Configuration options](#) (p. 536)
- [Advanced environment customization with configuration files \(.ebextensions\)](#) (p. 600)
- [Using Elastic Beanstalk saved configurations](#) (p. 640)
- [Environment manifest \(env.yaml\)](#) (p. 645)
- [Using a custom Amazon machine image \(AMI\)](#) (p. 647)
- [Serving static files](#) (p. 650)
- [Configuring HTTPS for your Elastic Beanstalk environment](#) (p. 652)

Configuration options

Elastic Beanstalk defines a large number of configuration options that you can use to configure your environment's behavior and the resources that it contains. Configuration options are organized into namespaces like `aws:autoscaling:asg`, which defines options for an environment's Auto Scaling group.

The Elastic Beanstalk console and EB CLI set configuration options when you create an environment, including options that you set explicitly, and [recommended values](#) (p. 537) defined by the client. You can also set configuration options in saved configurations and configuration files. If the same option is set in multiple locations, the value used is determined by the [order of precedence](#) (p. 537).

Configuration option settings can be composed in text format and saved prior to environment creation, applied during environment creation using any supported client, and added, modified or removed after environment creation. For a detailed breakdown of all of the available methods for working with configuration options at each of these three stages, read the following topics:

- [Setting configuration options before environment creation \(p. 539\)](#)
- [Setting configuration options during environment creation \(p. 543\)](#)
- [Setting configuration options after environment creation \(p. 547\)](#)

For a complete list of namespaces and options, including default and supported values for each, see [General options for all environments \(p. 555\)](#) and [Platform specific options \(p. 592\)](#).

Precedence

During environment creation, configuration options are applied from multiple sources with the following precedence, from highest to lowest:

- **Settings applied directly to the environment** – Settings specified during a create environment or update environment operation on the Elastic Beanstalk API by any client, including the Elastic Beanstalk console, EB CLI, AWS CLI, and SDKs. The Elastic Beanstalk console and EB CLI also apply [recommended values \(p. 537\)](#) for some options that apply at this level unless overridden.
- **Saved Configurations** – Settings for any options that are not applied directly to the environment are loaded from a saved configuration, if specified.
- **Configuration Files (.ebextensions)** – Settings for any options that are not applied directly to the environment, and also not specified in a saved configuration, are loaded from configuration files in the `.ebextensions` folder at the root of the application source bundle.

Configuration files are executed in alphabetical order. For example, `.ebextensions/01run.config` is executed before `.ebextensions/02do.config`.

- **Default Values** – If a configuration option has a default value, it only applies when the option is not set at any of the above levels.

If the same configuration option is defined in more than one location, the setting with the highest precedence is applied. When a setting is applied from a saved configuration or settings applied directly to the environment, the setting is stored as part of the environment's configuration. These settings can be removed [with the AWS CLI \(p. 554\)](#) or [with the EB CLI \(p. 552\)](#).

Settings in configuration files are not applied directly to the environment and cannot be removed without modifying the configuration files and deploying a new application version. If a setting applied with one of the other methods is removed, the same setting will be loaded from configuration files in the source bundle.

For example, say you set the minimum number of instances in your environment to 5 during environment creation, using either the Elastic Beanstalk console, a command line option, or a saved configuration. The source bundle for your application also includes a configuration file that sets the minimum number of instances to 2.

When you create the environment, Elastic Beanstalk sets the `MinSize` option in the `aws:autoscaling:asg` namespace to 5. If you then remove the option from the environment configuration, the value in the configuration file is loaded, and the minimum number of instances is set to 2. If you then remove the configuration file from the source bundle and redeploy, Elastic Beanstalk uses the default setting of 1.

Recommended values

The Elastic Beanstalk Command Line Interface (EB CLI) and Elastic Beanstalk console provide recommended values for some configuration options. These values can be different from the default values and are set at the API level when your environment is created. Recommended values allow Elastic Beanstalk to improve the default environment configuration without making backwards incompatible changes to the API.

For example, both the EB CLI and Elastic Beanstalk console set the configuration option for EC2 instance type (`InstanceType` in the `aws:autoscaling:launchconfiguration` namespace). Each client provides a different way of overriding the default setting. In the console you can choose a different instance type from a drop down menu on the **Configuration Details** page of the **Create New Environment** wizard. With the EB CLI, you can use the `--instance_type` parameter for [`eb create`](#) (p. 894).

Because the recommended values are set at the API level, they will override values for the same options that you set in configuration files or saved configurations. The following options are set:

Elastic Beanstalk console

- Namespace: `aws:autoscaling:launchconfiguration`
Option Names: `IamInstanceProfile`, `EC2KeyName`, `InstanceType`
- Namespace: `aws:autoscaling:updatepolicy:rollingupdate`
Option Names: `RollingUpdateType` and `RollingUpdateEnabled`
- Namespace: `aws:elasticbeanstalk:application`
Option Name: `Application Healthcheck URL`
- Namespace: `aws:elasticbeanstalk:command`
Option Name: `DeploymentPolicy`, `BatchSize` and `BatchSizeType`
- Namespace: `aws:elasticbeanstalk:environment`
Option Name: `ServiceRole`
- Namespace: `aws:elasticbeanstalk:healthreporting:system`
Option Name: `SystemType` and `HealthCheckSuccessThreshold`
- Namespace: `aws:elasticbeanstalk:sns:topics`
Option Name: `Notification Endpoint`
- Namespace: `aws:elasticbeanstalk:sqs`
Option Name: `HttpConnections`
- Namespace: `aws:elb:loadbalancer`
Option Name: `CrossZone`
- Namespace: `aws:elb:policies`
Option Names: `ConnectionDrainingTimeout` and `ConnectionDrainingEnabled`

EB CLI

- Namespace: `aws:autoscaling:launchconfiguration`
Option Names: `IamInstanceProfile`, `InstanceType`
- Namespace: `aws:autoscaling:updatepolicy:rollingupdate`
Option Names: `RollingUpdateType` and `RollingUpdateEnabled`
- Namespace: `aws:elasticbeanstalk:command`
Option Name: `BatchSize` and `BatchSizeType`
- Namespace: `aws:elasticbeanstalk:environment`

Option Name: `ServiceRole`

- Namespace: `aws:elasticbeanstalk:healthreporting:system`

Option Name: `SystemType`

- Namespace: `aws:elb:loadbalancer`

Option Name: `CrossZone`

- Namespace: `aws:elb:policies`

Option Names: `ConnectionDrainingEnabled`

Setting configuration options before environment creation

AWS Elastic Beanstalk supports a large number of [configuration options \(p. 536\)](#) that let you modify the settings that are applied to resources in your environment. Several of these options have default values that can be overridden to customize your environment. Other options can be configured to enable additional features.

Elastic Beanstalk supports two methods of saving configuration option settings. Configuration files in YAML or JSON format can be included in your application's source code in a directory named `.ebextensions` and deployed as part of your application source bundle. You create and manage configuration files locally.

Saved configurations are templates that you create from a running environment or JSON options file and store in Elastic Beanstalk. Existing saved configurations can also be extended to create a new configuration.

Note

Settings defined in configuration files and saved configurations have lower precedence than settings configured during or after environment creation, including recommended values applied by the Elastic Beanstalk console and [EB CLI \(p. 852\)](#). See [Precedence \(p. 537\)](#) for details.

Options can also be specified in a JSON document and provided directly to Elastic Beanstalk when you create or update an environment with the EB CLI or AWS CLI. Options provided directly to Elastic Beanstalk in this manner override all other methods.

For a full list of available options, see [Configuration options \(p. 536\)](#).

Methods

- [Configuration files \(.ebextensions\) \(p. 539\)](#)
- [Saved configurations \(p. 540\)](#)
- [JSON document \(p. 542\)](#)
- [EB CLI configuration \(p. 542\)](#)

Configuration files (`.ebextensions`)

Use `.ebextensions` to configure options that are required to make your application work, and provide default values for other options that can be overridden at a higher level of [precedence \(p. 537\)](#). Options specified in `.ebextensions` have the lowest level of precedence and are overridden by settings at any other level.

To use configuration files, create a folder named `.ebextensions` at the top level of your project's source code. Add a file with the extension `.config` and specify options in the following manner:

```
option_settings:
- namespace: namespace
  option_name: option name
  value: option value
- namespace: namespace
  option_name: option name
  value: option value
```

For example, the following configuration file sets the application's health check url to `/health`:

`healthcheckurl.config`

```
option_settings:
- namespace: aws:elasticbeanstalk:application
  option_name: Application Healthcheck URL
  value: /health
```

In JSON:

```
{
  "option_settings" :
  [
    {
      "namespace" : "aws:elasticbeanstalk:application",
      "option_name" : "Application Healthcheck URL",
      "value" : "/health"
    }
  ]
}
```

This configures the Elastic Load Balancing load balancer in your Elastic Beanstalk environment to make an HTTP request to the path `/health` to each EC2 instance to determine if it is healthy or not.

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

Include the `.ebextensions` directory in your [Application Source Bundle \(p. 342\)](#) and deploy it to a new or existing Elastic Beanstalk environment.

Configuration files support several sections in addition to `option_settings` for customizing the software and files that run on the servers in your environment. For more information, see [.Ebextensions \(p. 600\)](#).

Saved configurations

Create a saved configuration to save settings that you have applied to an existing environment during or after environment creation by using the Elastic Beanstalk console, EB CLI, or AWS CLI. Saved configurations belong to an application and can be applied to new or existing environments for that application.

Clients

- [Elastic Beanstalk console \(p. 541\)](#)
- [EB CLI \(p. 541\)](#)
- [AWS CLI \(p. 541\)](#)

Elastic Beanstalk console

To create a saved configuration (Elastic Beanstalk console)

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Save configuration**.
4. Use the on-screen dialog box to complete the action.

Saved configurations are stored in the Elastic Beanstalk S3 bucket in a folder named after your application. For example, configurations for an application named `my-app` in the `us-west-2` region for account number `123456789012` can be found at `s3://elasticbeanstalk-us-west-2-123456789012/resources/templates/my-app`.

EB CLI

The [EB CLI \(p. 852\)](#) also provides subcommands for interacting with saved configurations under [eb config \(p. 891\)](#):

To create a saved configuration (EB CLI)

1. Save the attached environment's current configuration:

```
~/project$ eb config save --cfg my-app-v1
```

The EB CLI saves the configuration to `~/project/.elasticbeanstalk/saved_configs/my-app-v1.cfg.yml`

2. Modify the saved configuration locally if needed.
3. Upload the saved configuration to S3:

```
~/project$ eb config put my-app-v1
```

AWS CLI

Create a saved configuration from a running environment with `aws elasticbeanstalk create-configuration-template`

To create a saved configuration (AWS CLI)

1. Identify your Elastic Beanstalk environment's environment ID with `describe-environments`:

```
$ aws elasticbeanstalk describe-environments --environment-name my-env
{
  "Environments": [
    {
      "ApplicationName": "my-env",
      "EnvironmentName": "my-env",
      "VersionLabel": "89df",
      "Status": "Ready",
      "Description": "Environment created from the EB CLI using \"eb create\"",
      "EnvironmentId": "e-vcghmm2zwk",
    }
  ]
}
```

```
        "EndpointURL": "awseb-e-v-AWSEBLoa-1JUM8159RA11M-43V6ZI1194.us-  
west-2.elb.amazonaws.com",  
        "SolutionStackName": "64bit Amazon Linux 2015.03 v2.0.2 running Multi-  
container Docker 1.7.1 (Generic)",  
        "CNAME": "my-env-nfptuqaper.elasticbeanstalk.com",  
        "Health": "Green",  
        "AbortableOperationInProgress": false,  
        "Tier": {  
            "Version": " ",  
            "Type": "Standard",  
            "Name": "WebServer"  
        },  
        "HealthStatus": "Ok",  
        "DateUpdated": "2015-10-01T00:24:04.045Z",  
        "DateCreated": "2015-09-30T23:27:55.768Z"  
    }  
]  
}
```

2. Save the environment's current configuration with `create-configuration-template`:

```
$ aws elasticbeanstalk create-configuration-template --environment-id e-vcghmm2zwk --  
application-name my-app --template-name v1
```

Elastic Beanstalk saves the configuration to your Elastic Beanstalk bucket in Amazon S3.

JSON document

If you use the AWS CLI to create and update environments, you can also provide configuration options in JSON format. A library of configuration files in JSON is useful if you use the AWS CLI to create and manage environments.

For example, the following JSON document sets the application's health check url to `/health`:

`~/ebconfigs/healthcheckurl.json`

```
[  
  {  
    "Namespace": "aws:elasticbeanstalk:application",  
    "OptionName": "Application Healthcheck URL",  
    "Value": "/health"  
  }  
]
```

EB CLI configuration

In addition to supporting saved configurations and direct environment configuration with `eb config` commands, the EB CLI has a configuration file with an option named `default_ec2_keyname` that you can use to specify an Amazon EC2 key pair for SSH access to the instances in your environment. The EB CLI uses this option to set the `EC2KeyName` configuration option in the `aws:autoscaling:launchconfiguration` namespace.

`~/workspace/my-app/.elasticbeanstalk/config.yml`

```
branch-defaults:  
  master:  
    environment: my-env  
  develop:  
    environment: my-env-dev
```

```
deploy:
  artifact: ROOT.war
global:
  application_name: my-app
  default_ec2_keyname: my-keypair
  default_platform: Tomcat 8 Java 8
  default_region: us-west-2
  profile: null
  sc: git
```

Setting configuration options during environment creation

When you create an AWS Elastic Beanstalk environment by using the Elastic Beanstalk console, EB CLI, AWS CLI, an SDK, or the Elastic Beanstalk API, you can provide values for configuration options to customize your environment and the AWS resources that are launched within it.

For anything other than a one-off configuration change, you can [store configuration files \(p. 539\)](#) locally, in your source bundle, or in Amazon S3.

This topic includes procedures for all of the methods to set configuration options during environment creation.

Clients

- [In the Elastic Beanstalk console \(p. 543\)](#)
- [Using the EB CLI \(p. 544\)](#)
- [Using the AWS CLI \(p. 546\)](#)

In the Elastic Beanstalk console

When you create an Elastic Beanstalk environment in the Elastic Beanstalk console, you can provide configuration options using configuration files, saved configurations, and forms in the **Create New Environment** wizard.

Methods

- [Using configuration files \(.ebextensions\) \(p. 543\)](#)
- [Using a saved configuration \(p. 544\)](#)
- [Using the new environment wizard \(p. 544\)](#)

Using configuration files (.ebextensions)

Include `.config` files in your [application source bundle \(p. 342\)](#) in a folder named `.ebextensions`.

For details about configuration files, see [.Ebextensions \(p. 600\)](#).

```
~/workspace/my-app-v1.zip
|-- .ebextensions
|   |-- environmentvariables.config
|   |-- healthcheckurl.config
|-- index.php
`-- styles.css
```

Upload the source bundle to Elastic Beanstalk normally, during [environment creation \(p. 363\)](#).

The Elastic Beanstalk console applies [recommended values \(p. 537\)](#) for some configuration options and has form fields for others. Options configured by the Elastic Beanstalk console are applied directly to the environment and override settings in configuration files.

Using a saved configuration

When you create a new environment using the Elastic Beanstalk console, one of the first steps is to choose a configuration. The configuration can be a [predefined configuration \(p. 29\)](#), typically the latest version of a platform such as **PHP** or **Tomcat**, or it can be a **saved configuration**.

To apply a saved configuration during environment creation (Elastic Beanstalk console)

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.

Note

If you have many applications, use the search bar to filter the application list.

3. In the navigation pane, find your application's name and choose **Saved configurations**.
4. Select the saved configuration you want to apply, and then choose **Launch environment**.
5. Proceed through the wizard to create your environment.

Saved configurations are application-specific. See [Saved configurations \(p. 540\)](#) for details on creating saved configurations.

Using the new environment wizard

Most of the standard configuration options are presented on the **Configure more options** page of the [Create New Environment wizard \(p. 365\)](#). If you create an Amazon RDS database or configure a VPC for your environment, additional configuration options are available for those resources.

To set configuration options during environment creation (Elastic Beanstalk console)

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**.
3. Choose or [create \(p. 334\)](#) an application.
4. Choose **Actions**, and then choose **Create environment**.
5. Proceed through the wizard, and choose **Configure more options**.
6. Choose any of the **configuration presets**, and then choose **Edit** in one or more of the configuration categories to change a group of related configuration options.
7. When you are done making option selections, choose **Create environment**.

Any options that you set in the new environment wizard are set directly on the environment and override any option settings in saved configurations or configuration files (`.ebextensions`) that you apply. You can remove settings after the environment is created using the [EB CLI \(p. 550\)](#) or [AWS CLI \(p. 552\)](#) to allow the settings in saved configurations or configuration files to surface.

For details about the new environment wizard, see [The create new environment wizard \(p. 365\)](#).

Using the EB CLI

Methods

- [Using configuration files \(.ebextensions\) \(p. 545\)](#)
- [Using saved configurations \(p. 545\)](#)
- [Using command line options \(p. 545\)](#)

Using configuration files (.ebextensions)

Include `.config` files in your project folder under `.ebextensions` to deploy them with your application code.

For details about configuration files, see [.Ebextensions \(p. 600\)](#).

```
~/workspace/my-app/  
|-- .ebextensions  
|   |-- environmentvariables.config  
|   |-- healthcheckurl.config  
|-- .elasticbeanstalk  
|   |-- config.yml  
|-- index.php  
|-- styles.css
```

Create your environment and deploy your source code to it with **eb create**.

```
~/workspace/my-app$ eb create my-env
```

Using saved configurations

To apply a saved configuration when you create an environment with **eb create** ([p. 894](#)), use the `--cfg` option.

```
~/workspace/my-app$ eb create --cfg savedconfig
```

You can store the saved configuration in your project folder or in your Elastic Beanstalk storage location on Amazon S3. In the previous example, the EB CLI first looks for a saved configuration file named `savedconfig.cfg.yml` in the folder `.elasticbeanstalk/saved_configs/`. Do not include the file name extensions (`.cfg.yml`) when applying a saved configuration with `--cfg`.

```
~/workspace/my-app/  
|-- .ebextensions  
|   |-- healthcheckurl.config  
|-- .elasticbeanstalk  
|   |-- saved_configs  
|       |-- savedconfig.cfg.yml  
|       |-- config.yml  
|-- index.php  
|-- styles.css
```

If the EB CLI does not find the configuration locally, it looks in the Elastic Beanstalk storage location in Amazon S3. For details on creating, editing, and uploading saved configurations, see [Saved configurations \(p. 540\)](#).

Using command line options

The EB CLI **eb create** command has several [options \(p. 895\)](#) that you can use to set configuration options during environment creation. You can use these options to add an RDS database to your environment, configure a VPC, or override [recommended values \(p. 537\)](#).

For example, the EB CLI uses the `t2.micro` instance type by default. To choose a different instance type, use the `--instance_type` option.

```
$ eb create my-env --instance_type t2.medium
```


To create an Amazon RDS database instance and attach it to your environment, use the `--database` options.

```
$ eb create --database.engine postgres --database.username dbuser
```

If you leave out the environment name, database password, or any other parameters that are required to create your environment, the EB CLI prompts you to enter them.

See [eb create \(p. 894\)](#) for a full list of available options and usage examples.

Using the AWS CLI

When you use the `create-environment` command to create an Elastic Beanstalk environment with the AWS CLI, the AWS CLI does not apply any [recommended values \(p. 537\)](#). All configuration options are defined in configuration files in the source bundle that you specify.

Methods

- [Using configuration files \(.ebextensions\) \(p. 546\)](#)
- [Using a saved configuration \(p. 547\)](#)
- [Using command line options \(p. 547\)](#)

Using configuration files (.ebextensions)

To apply configuration files to an environment that you create with the AWS CLI, include them in the application source bundle that you upload to Amazon S3.

For details about configuration files, see [.Ebextensions \(p. 600\)](#).

```
~/workspace/my-app-v1.zip
|-- .ebextensions
|   |-- environmentvariables.config
|   |-- healthcheckurl.config
|-- index.php
|-- styles.css
```

To upload an application source bundle and create an environment with the AWS CLI

1. If you don't already have an Elastic Beanstalk bucket in Amazon S3, create one with `create-storage-location`.

```
$ aws elasticbeanstalk create-storage-location
{
  "S3Bucket": "elasticbeanstalk-us-west-2-123456789012"
}
```

2. Upload your application source bundle to Amazon S3.

```
$ aws s3 cp sourcebundle.zip s3://elasticbeanstalk-us-west-2-123456789012/my-app/
sourcebundle.zip
```

3. Create the application version.

```
$ aws elasticbeanstalk create-application-version --application-name my-app --
version-label v1 --description MyAppv1 --source-bundle S3Bucket="elasticbeanstalk-us-
west-2-123456789012",S3Key="my-app/sourcebundle.zip" --auto-create-application
```

4. Create the environment.

```
$ aws elasticbeanstalk create-environment --application-name my-app --environment-name my-env --version-label v1 --solution-stack-name "64bit Amazon Linux 2015.03 v2.0.0 running Tomcat 8 Java 8"
```

Using a saved configuration

To apply a saved configuration to an environment during creation, use the `--template-name` parameter.

```
$ aws elasticbeanstalk create-environment --application-name my-app --environment-name my-env --template-name savedconfig --version-label v1
```

When you specify a saved configuration, do not also specify a solution stack name. Saved configurations already specify a solution stack and Elastic Beanstalk will return an error if you try to use both options.

Using command line options

Use the `--option-settings` parameter to specify configuration options in JSON format.

```
$ aws elasticbeanstalk create-environment --application-name my-app --environment-name my-env --version-label v1 --template-name savedconfig --option-settings '[
  {
    "Namespace": "aws:elasticbeanstalk:application",
    "OptionName": "Application Healthcheck URL",
    "Value": "/health"
  }
]
```

To load the JSON from a file, use the `file://` prefix.

```
$ aws elasticbeanstalk create-environment --application-name my-app --environment-name my-env --version-label v1 --template-name savedconfig --option-settings file://healthcheckurl.json
```

Elastic Beanstalk applies option settings that you specify with the `--option-settings` option directly to your environment. If the same options are specified in a saved configuration or configuration file, `--option-settings` overrides those values.

Setting configuration options after environment creation

You can modify the option settings on a running environment by applying saved configurations, uploading a new source bundle with configuration files (`.ebextensions`), or using a JSON document. The EB CLI and Elastic Beanstalk console also have client-specific functionality for setting and updating configuration options.

When you set or change a configuration option, you can trigger a full environment update, depending on the severity of the change. For example, changes to options in the [aws:autoscaling:launchconfiguration](#) (p. 556), such as `InstanceType`, require that the Amazon EC2 instances in your environment are reprovisioned. This triggers a [rolling update](#) (p. 409). Other configuration changes can be applied without any interruption or reprovisioning.

You can remove option settings from an environment with EB CLI or AWS CLI commands. Removing an option that has been set directly on an environment at an API level allows settings in configuration files, which are otherwise masked by settings applied directly to an environment, to surface and take effect.

Settings in saved configurations and configuration files can be overridden by setting the same option directly on the environment with one of the other configuration methods. However, these can only be removed completely by applying an updated saved configuration or configuration file. When an option is not set in a saved configuration, in a configuration file, or directly on an environment, the default value applies, if there is one. See [Precedence \(p. 537\)](#) for details.

Clients

- [The Elastic Beanstalk console \(p. 548\)](#)
- [The EB CLI \(p. 550\)](#)
- [The AWS CLI \(p. 552\)](#)

The Elastic Beanstalk console

You can update configuration option settings in the Elastic Beanstalk console by deploying an application source bundle that contains configuration files, applying a saved configuration, or modifying the environment directly with the **Configuration** page in the environment management console.

Methods

- [Using configuration files \(.ebextensions\) \(p. 548\)](#)
- [Using a saved configuration \(p. 548\)](#)
- [Using the Elastic Beanstalk console \(p. 549\)](#)

Using configuration files (.ebextensions)

Update configuration files in your source directory, create a new source bundle, and deploy the new version to your Elastic Beanstalk environment to apply the changes.

For details about configuration files, see [.Ebextensions \(p. 600\)](#).

To deploy a source bundle

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Changes made to configuration files will not override option settings in saved configurations or settings applied directly to the environment at the API level. See [Precedence \(p. 537\)](#) for details.

Using a saved configuration

Apply a saved configuration to a running environment to apply option settings that it defines.

To apply a saved configuration to a running environment (Elastic Beanstalk console)

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.

Note

If you have many applications, use the search bar to filter the application list.

3. In the navigation pane, find your application's name and choose **Saved configurations**.
4. Select the saved configuration you want to apply, and then choose **Load**.
5. Select an environment, and then choose **Load**.

Settings defined in a saved configuration override settings in configuration files, and are overridden by settings configured using the environment management console.

See [Saved configurations \(p. 540\)](#) for details on creating saved configurations.

Using the Elastic Beanstalk console

The Elastic Beanstalk console presents many configuration options on the **Configuration** page for each environment.

To change configuration options on a running environment (Elastic Beanstalk console)

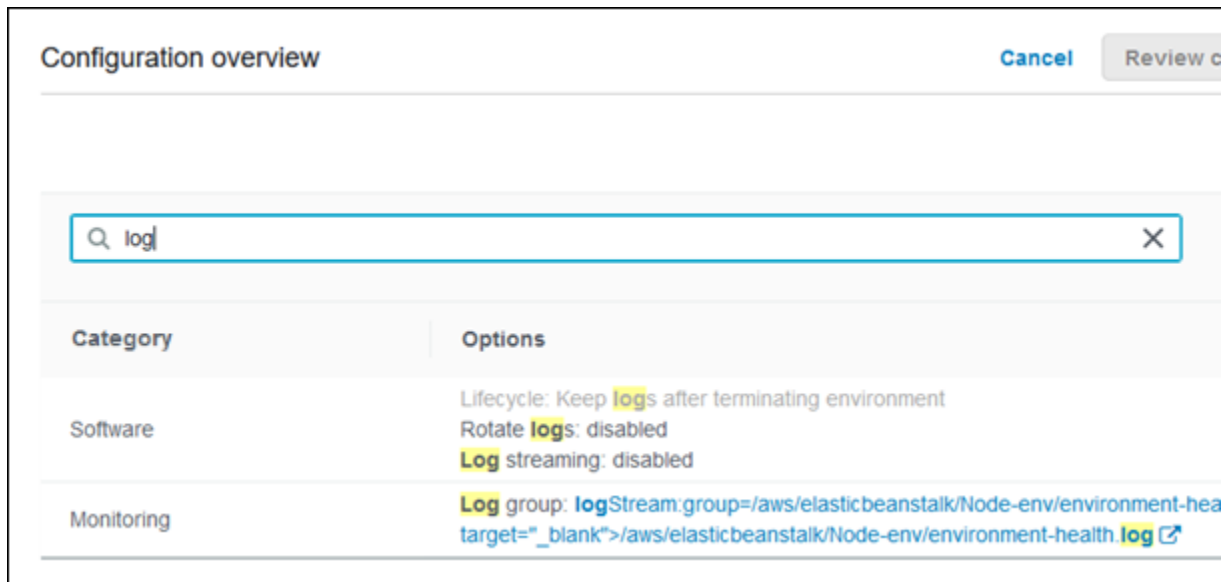
1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. Find the configuration page you want to edit:
 - If you see the option you're interested in, or you know which configuration category it's in, choose **Edit** in the configuration category for it.
 - To look for an option, turn on **Table View**, and then enter search terms into the search box. As you type, the list gets shorter and shows only options that match your search terms.

When you see the option you're looking for, choose **Edit** in the configuration category that contains it.



5. Change settings, and then choose **Save**.
6. Repeat the previous two steps in additional configuration categories, as needed.
7. Choose **Apply**.

Changes made to configuration options in the environment management console are applied directly to the environment. These changes override settings for the same options in configuration files or saved configurations. For details, see [Precedence \(p. 537\)](#).

For details about changing configuration options on a running environment using the Elastic Beanstalk console, see the topics under [Configuring Elastic Beanstalk environments \(p. 446\)](#).

The EB CLI

You can update configuration option settings with the EB CLI by deploying source code that contains configuration files, applying settings from a saved configuration, or modifying the environment configuration directly with the **eb config** command.

Methods

- [Using configuration files \(.ebextensions\) \(p. 550\)](#)
- [Using a saved configuration \(p. 550\)](#)
- [Using eb config \(p. 551\)](#)
- [Using eb setenv \(p. 552\)](#)

Using configuration files (.ebextensions)

Include `.config` files in your project folder under `.ebextensions` to deploy them with your application code.

For details about configuration files, see [.Ebextensions \(p. 600\)](#).

```
~/workspace/my-app/  
|-- .ebextensions  
|   |-- environmentvariables.config  
|   |-- healthcheckurl.config  
|-- .elasticbeanstalk  
|   |-- config.yml  
|-- index.php  
|-- styles.css
```

Deploy your source code with **eb deploy**.

```
~/workspace/my-app$ eb deploy
```

Using a saved configuration

You can use the **eb config** command to apply a saved configuration to a running environment. Use the `--cfg` option with the name of the saved configuration to apply its settings to your environment.

```
$ eb config --cfg v1
```

In this example, `v1` is the name of a [previously created and saved configuration file \(p. 540\)](#).

Settings applied to an environment with this command override settings that were applied during environment creation, and settings defined in configuration files in your application source bundle.

Using `eb config`

The EB CLI's `eb config` command lets you set and remove option settings directly on an environment by using a text editor.

When you run `eb config`, the EB CLI shows settings applied to your environment from all sources, including configuration files, saved configurations, recommended values, options set directly on the environment, and API defaults.

Note

`eb config` does not show environment properties. To set environment properties that you can read from within your application, use [`eb setenv`](#) (p. 552).

The following example shows settings applied in the `aws:autoscaling:launchconfiguration` namespace. These settings include:

- Two recommended values, for `IamInstanceProfile` and `InstanceType`, applied by the EB CLI during environment creation.
- The option `EC2KeyName`, set directly on the environment during creation based on repository configuration.
- API default values for the other options.

```
ApplicationName: tomcat
DateUpdated: 2015-09-30 22:51:07+00:00
EnvironmentName: tomcat
SolutionStackName: 64bit Amazon Linux 2015.03 v2.0.1 running Tomcat 8 Java 8
settings:
...
aws:autoscaling:launchconfiguration:
  BlockDeviceMappings: null
  EC2KeyName: my-key
  IamInstanceProfile: aws-elasticbeanstalk-ec2-role
  ImageId: ami-1f316660
  InstanceType: t2.micro
...
```

To set or change configuration options with `eb config`

1. Run `eb config` to view your environment's configuration.

```
~/workspace/my-app/$ eb config
```

2. Change any of the setting values using the default text editor.

```
aws:autoscaling:launchconfiguration:
  BlockDeviceMappings: null
  EC2KeyName: my-key
  IamInstanceProfile: aws-elasticbeanstalk-ec2-role
  ImageId: ami-1f316660
  InstanceType: t2.medium
```

3. Save the temporary configuration file and exit.
4. The EB CLI updates your environment configuration.

Setting configuration options with `eb config` overrides settings from all other sources.

You can also remove options from your environment with **eb config**.

To remove configuration options (EB CLI)

1. Run **eb config** to view your environment's configuration.

```
~/workspace/my-app/$ eb config
```

2. Replace any value shown with the string `null`. You can also delete the entire line containing the option that you want to remove.

```
aws:autoscaling:launchconfiguration:
  BlockDeviceMappings: null
  EC2KeyName: my-key
  IamInstanceProfile: aws-elasticbeanstalk-ec2-role
  ImageId: ami-1f316660
  InstanceType: null
```

3. Save the temporary configuration file and exit.
4. The EB CLI updates your environment configuration.

Removing options from your environment with **eb config** allows settings for the same options to surface from configuration files in your application source bundle. See [Precedence \(p. 537\)](#) for details.

Using **eb setenv**

To set environment properties with the EB CLI, use **eb setenv**.

```
~/workspace/my-app/$ eb setenv ENVVAR=TEST
INFO: Environment update is starting.
INFO: Updating environment my-env's configuration settings.
INFO: Environment health has transitioned from Ok to Info. Command is executing on all
instances.
INFO: Successfully deployed new configuration to environment.
```

This command sets environment properties in the [aws:elasticbeanstalk:application:environment namespace \(p. 570\)](#). Environment properties set with **eb setenv** are available to your application after a short update process.

View environment properties set on your environment with **eb printenv**.

```
~/workspace/my-app/$ eb printenv
Environment Variables:
  ENVVAR = TEST
```

The AWS CLI

You can update configuration option settings with the AWS CLI by deploying a source bundle that contains configuration files, applying a remotely stored saved configuration, or modifying the environment directly with the `aws elasticbeanstalk update-environment` command.

Methods

- [Using configuration files \(.ebextensions\) \(p. 553\)](#)
- [Using a saved configuration \(p. 553\)](#)
- [Using command line options \(p. 554\)](#)

Using configuration files (.ebextensions)

To apply configuration files to a running environment with the AWS CLI, include them in the application source bundle that you upload to Amazon S3.

For details about configuration files, see [.Ebextensions \(p. 600\)](#).

```
~/workspace/my-app-v1.zip
|-- .ebextensions
|   |-- environmentvariables.config
|   |-- healthcheckurl.config
|-- index.php
|-- styles.css
```

To upload an application source bundle and apply it to a running environment (AWS CLI)

1. If you don't already have an Elastic Beanstalk bucket in Amazon S3, create one with `create-storage-location`:

```
$ aws elasticbeanstalk create-storage-location
{
  "S3Bucket": "elasticbeanstalk-us-west-2-123456789012"
}
```

2. Upload your application source bundle to Amazon S3.

```
$ aws s3 cp sourcebundlev2.zip s3://elasticbeanstalk-us-west-2-123456789012/my-app/sourcebundlev2.zip
```

3. Create the application version.

```
$ aws elasticbeanstalk create-application-version --application-name my-app --version-label v2 --description MyAppv2 --source-bundle S3Bucket=elasticbeanstalk-us-west-2-123456789012,S3Key=my-app/sourcebundlev2.zip
```

4. Update the environment.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --version-label v2
```

Using a saved configuration

You can apply a saved configuration to a running environment with the `--template-name` option on the `aws elasticbeanstalk update-environment` command.

The saved configuration must be in your Elastic Beanstalk bucket in a path named after your application under `resources/templates`. For example, the `v1` template for the `my-app` application in the US West (Oregon) Region (`us-west-2`) for account `123456789012` is located at `s3://elasticbeanstalk-us-west-2-123456789012/resources/templates/my-app/v1`

To apply a saved configuration to a running environment (AWS CLI)

- Specify the saved configuration in an `update-environment` call with the `--template-name` option.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --template-name v1
```


Elastic Beanstalk places saved configurations in this location when you create them with `aws elasticbeanstalk create-configuration-template`. You can also modify saved configurations locally and place them in this location yourself.

Using command line options

To change configuration options with a JSON document (AWS CLI)

1. Define your option settings in JSON format in a local file.
2. Run `update-environment` with the `--option-settings` option.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --option-settings  
file://-/ebconfigs/as-zero.json
```

In this example, `as-zero.json` defines options that configure the environment with a minimum and maximum of zero instances. This stops the instances in the environment without terminating the environment.

`-/ebconfigs/as-zero.json`

```
[  
  {  
    "Namespace": "aws:autoscaling:asg",  
    "OptionName": "MinSize",  
    "Value": "0"  
  },  
  {  
    "Namespace": "aws:autoscaling:asg",  
    "OptionName": "MaxSize",  
    "Value": "0"  
  },  
  {  
    "Namespace": "aws:autoscaling:updatepolicy:rollingupdate",  
    "OptionName": "RollingUpdateEnabled",  
    "Value": "false"  
  }  
]
```

Note

Setting configuration options with `update-environment` overrides settings from all other sources.

You can also remove options from your environment with `update-environment`.

To remove configuration options (AWS CLI)

- Run the `update-environment` command with the `--settings-to-remove` option.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --options-to-remove  
Namespace=aws:autoscaling:launchconfiguration,OptionName=InstanceType
```

Removing options from your environment with `update-environment` allows settings for the same options to surface from configuration files in your application source bundle. If an option isn't configured using any of these methods, the API default value applies, if one exists. See [Precedence \(p. 537\)](#) for details.

General options for all environments

Namespaces

- [aws:autoscaling:asg](#) (p. 555)
- [aws:autoscaling:launchconfiguration](#) (p. 556)
- [aws:autoscaling:scheduledaction](#) (p. 562)
- [aws:autoscaling:trigger](#) (p. 563)
- [aws:autoscaling:updatepolicy:rollingupdate](#) (p. 564)
- [aws:ec2:instances](#) (p. 567)
- [aws:ec2:vpc](#) (p. 568)
- [aws:elasticbeanstalk:application](#) (p. 569)
- [aws:elasticbeanstalk:application:environment](#) (p. 570)
- [aws:elasticbeanstalk:cloudwatch:logs](#) (p. 570)
- [aws:elasticbeanstalk:cloudwatch:logs:health](#) (p. 570)
- [aws:elasticbeanstalk:command](#) (p. 571)
- [aws:elasticbeanstalk:environment](#) (p. 572)
- [aws:elasticbeanstalk:environment:process:default](#) (p. 573)
- [aws:elasticbeanstalk:environment:process:process_name](#) (p. 575)
- [aws:elasticbeanstalk:environment:proxy:staticfiles](#) (p. 577)
- [aws:elasticbeanstalk:healthreporting:system](#) (p. 577)
- [aws:elasticbeanstalk:hostmanager](#) (p. 578)
- [aws:elasticbeanstalk:managedactions](#) (p. 578)
- [aws:elasticbeanstalk:managedactions:platformupdate](#) (p. 578)
- [aws:elasticbeanstalk:monitoring](#) (p. 579)
- [aws:elasticbeanstalk:sns:topics](#) (p. 579)
- [aws:elasticbeanstalk:sqs](#) (p. 580)
- [aws:elasticbeanstalk:trafficsplitting](#) (p. 581)
- [aws:elasticbeanstalk:xray](#) (p. 582)
- [aws:elb:healthcheck](#) (p. 582)
- [aws:elb:loadbalancer](#) (p. 583)
- [aws:elb:listener](#) (p. 584)
- [aws:elb:listener:listener_port](#) (p. 584)
- [aws:elb:policies](#) (p. 585)
- [aws:elb:policies:policy_name](#) (p. 586)
- [aws:elbv2:listener:default](#) (p. 587)
- [aws:elbv2:listener:listener_port](#) (p. 588)
- [aws:elbv2:listenerrule:rule_name](#) (p. 589)
- [aws:elbv2:loadbalancer](#) (p. 590)
- [aws:rds:dbinstance](#) (p. 591)

aws:autoscaling:asg

Configure your environment's Auto Scaling group.

Namespace: `aws:autoscaling:asg`

Name	Description	Default	Valid values
Availability Zones	Availability Zones (AZs) are distinct locations within a region that are engineered to be isolated from failures in other AZs and provide inexpensive, low-latency network connectivity to other AZs in the same region. Choose the number of AZs for your instances.	Any	Any Any 1 Any 2 Any 3
Cooldown	Cooldown periods help prevent Amazon EC2 Auto Scaling from initiating additional scaling activities before the effects of previous activities are visible. A cooldown period is the amount of time, in seconds, after a scaling activity completes before another scaling activity can start.	360	0 to 10000
Custom Availability Zones	Define the AZs for your instances.	None	us-east-1a us-east-1b us-east-1c us-east-1d us-east-1e eu-central-1
MinSize	Minimum number of instances you want in your Auto Scaling group.	1	1 to 10000
MaxSize	Maximum number of instances you want in your Auto Scaling group.	4	1 to 10000

aws:autoscaling:launchconfiguration

Configure your environment's Amazon Elastic Compute Cloud (Amazon EC2) instances.

Your environment's instances are created using either an Amazon EC2 launch template or an Auto Scaling group launch configuration resource. These options work with both of these resource types.

Namespace: `aws:autoscaling:launchconfiguration`

Name	Description	Default	Valid values
EC2KeyName	A key pair enables you to securely log into your EC2 instance. Note If you use the Elastic Beanstalk console to create an environment, you can't set this option in a configuration file (p. 600). The console	None	

Name	Description	Default	Valid values
	<p>overrides this option with a recommended value (p. 537).</p>		
iamInstanceProfile	<p>The instance profile enables AWS Identity and Access Management (IAM) users and AWS services to access temporary security credentials to make AWS API calls. Specify the instance profile's name or its ARN.</p> <p>Examples:</p> <ul style="list-style-type: none"> • aws-elasticbeanstalk-ec2-role • arn:aws:iam::123456789012:instance-profile/aws-elasticbeanstalk-ec2-role <p>Note If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file (p. 600). The console and EB CLI override this option with a recommended value (p. 537).</p>	None	Instance profile name or ARN
ImageId	<p>You can override the default Amazon Machine Image (AMI) by specifying your own custom AMI ID.</p> <p>Example: ami-1f316660</p>	None	

Name	Description	Default	Valid values
InstanceType	<p>The instance type used to run your application in an Elastic Beanstalk environment.</p> <p>Important The <code>InstanceType</code> option is obsolete. It's replaced by the newer and more powerful <code>InstanceTypes</code> option in the aws:ec2:instances (p. 567) namespace. The new option allows you to specify a list of one or more instance types for your environment. The first value on that list is equivalent to the value of the <code>InstanceType</code> option included in the <code>aws:autoscaling:launchconfiguration</code> namespace described here. The recommended way to specify instance types is by using the new option. If specified, the new option takes precedence over the old one. For more information, see the section called "The aws:ec2:instances namespace" (p. 463).</p> <p>The instance types available depend on platform, solution stack (configuration) and region. To get a list of available instance types for your solution stack of choice, use the DescribeConfigurationOptions action in the API, describe-configuration-options command in the AWS CLI.</p> <p>For example, the following command lists the available instance types for version 1.4.3 of the PHP 5.6 stack in the current region:</p> <pre>\$ aws elasticbeanstalk describe-configuration-options --options Namespace=aws:autoscaling:launchconfiguration,OptionName=InstanceType --solution-stack-name "64bit Amazon Linux 2015.03 v1.4.3 running PHP 5.6"</pre> <p>Note If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file (p. 600). The console and</p>	Region dependent	

Name	Description	Default	Valid values
	<p>EB CLI override this option with a recommended value (p. 537).</p>		
MonitoringInterval	<p>Interval at which you want Amazon CloudWatch metrics returned.</p>	5 minute	<p>1 minute 5 minute</p>
SecurityGroups	<p>Lists the Amazon EC2 security groups to assign to the EC2 instances in the Auto Scaling group in order to define firewall rules for the instances.</p> <p>You can provide a single string of comma-separated values that contain the name of existing Amazon EC2 security groups or references to AWS::EC2::SecurityGroup resources created in the template. Security group names are case sensitive.</p> <p>If you use Amazon Virtual Private Cloud (Amazon VPC) with Elastic Beanstalk so that your instances are launched within a virtual private cloud (VPC), specify security group IDs instead of security group names.</p>	elasticbeanstalk-default	

Name	Description	Default	Valid values
SSHSourceRestriction	<p>Used to lock down SSH access to an environment. For instance, you can lock down SSH access to the EC2 instances so that only a bastion host can access the instances in the private subnet.</p> <p>This string takes the following form:</p> <pre>protocol, fromPort, toPort, source_restriction</pre> <p><i>protocol</i></p> <p>The protocol for the ingress rule.</p> <p><i>fromPort</i></p> <p>The starting port number.</p> <p><i>toPort</i></p> <p>The ending port number.</p> <p><i>source_restriction</i></p> <p>The CIDR range or the name of a security group to allow traffic from. To specify a security group from another account (EC2-Classical only, must be in the same region), include the account ID prior to the security group name (e.g., <i>other_account_id/security_group_name</i>). If you use Amazon Virtual Private Cloud (Amazon VPC) with Elastic Beanstalk so that your instances are launched within a virtual private cloud (VPC), specify a security group ID instead of a security group name.</p> <p>Example: tcp, 22, 22, 54.240.196.185/32</p> <p>Example: tcp, 22, 22, my-security-group</p> <p>Example (EC2-Classical): tcp, 22, 22, 123456789012/their-security-group</p> <p>Example (VPC): tcp, 22, 22, sg-903004f8</p>	None	

Name	Description	Default	Valid values
BlockDeviceMappings	<p>Attaches additional Amazon EBS volumes or instance store volumes on all of the instances in the autoscaling group.</p> <p>When you map instance store volumes you only map the device name to a volume name; when you map Amazon EBS volumes, you can specify the following fields, separated by a colon:</p> <ul style="list-style-type: none"> • snapshot ID • size, in GB • delete on terminate (<code>true</code> or <code>false</code>) • storage type (<code>gp2</code>, <code>standard</code>, <code>st1</code>, <code>sc1</code>, or <code>io1</code>) • IOPS (only for <code>io1</code> volumes). <p>The following example attaches three Amazon EBS volumes, one blank 100GB <code>gp2</code> volume and one snapshot, one blank 20GB <code>io1</code> volume with 2000 provisioned IOPS, and an instance store volume <code>ephemeral0</code>. Multiple instance store volumes can be attached if the instance type supports it.</p> <pre>/dev/sdj=:100:true:gp2, / dev/sdh=snap-51eef269, /dev/ sdi=:20:true:io1:2000, /dev/ sdb=ephemeral0</pre>	None	
RootVolumeType	Volume type (magnetic, general purpose SSD or provisioned IOPS SSD) to use for the root Amazon EBS volume attached to your environment's EC2 instances.	Varies per platform.	<p><code>standard</code> for magnetic storage</p> <p><code>gp2</code> for general purpose SSD</p> <p><code>io1</code> for provisioned IOPS SSD</p>
RootVolumeSize	<p>Storage capacity of the root Amazon EBS volume in whole GB.</p> <p>Required if you set <code>RootVolumeType</code> to provisioned IOPS SSD.</p> <p>For example, "64".</p>	Varies per platform for magnetic storage and general purpose SSD. None for provisioned IOPS SSD.	<p>10 to 16384 GB for general purpose and provisioned IOPS SSD.</p> <p>8 to 1024 GB for magnetic.</p>

Name	Description	Default	Valid values
RootVolumeIOPS	Desired input/output operations per second (IOPS) for a provisioned IOPS SSD root volume. The maximum ratio of IOPS to volume size is 30 to 1. For example, a volume with 3000 IOPS must be at least 100 GB.	None	100 to 20000

aws:autoscaling:scheduledaction

Configure [scheduled actions](#) (p. 466) for your environment's Auto Scaling group. For each action, specify a `resource_name` in addition to the option name, namespace, and value for each setting. See [The aws:autoscaling:scheduledaction namespace](#) (p. 467) for examples.

Namespace: `aws:autoscaling:scheduledaction`

Name	Description	Default	Valid values
StartTime	For one-time actions, choose the date and time to run the action. For recurrent actions, choose when to activate the action.	None	A ISO-8601 timestamp unique across all scheduled scaling actions.
EndTime	A date and time in the future (in the UTC/GMT time zone) when you want the scheduled scaling action to stop repeating. If you don't specify an EndTime , the action recurs according to the <code>Recurrence</code> expression. Example: <code>2015-04-28T04:07:2Z</code> When a scheduled action ends, Amazon EC2 Auto Scaling does not automatically go back to its previous settings. Configure a second scheduled action to return to the original settings as needed.	None	A ISO-8601 timestamp unique across all scheduled scaling actions.
MaxSize	The maximum instance count to apply when the action runs.	None	0 to 10000
MinSize	The minimum instance count to apply when the action runs.	None	0 to 10000
DesiredCapacity	Set the initial desired capacity for the Auto Scaling group. After the scheduled action is applied, triggers will adjust the desired capacity based on their settings.	None	0 to 10000
Recurrence	The frequency with which you want the scheduled action to occur. If you do not specify a recurrence, then the scaling action will occur only once, as specified by the <code>StartTime</code> .	None	A Cron expression.

Name	Description	Default	Valid values
Suspend	Set to <code>true</code> to deactivate a recurrent scheduled action temporarily.	<code>false</code>	<code>true</code> <code>false</code>

aws:autoscaling:trigger

Configure scaling triggers for your environment's Auto Scaling group.

Note

Three options in this namespace determine how long a trigger's metric can remain beyond its defined limits before the trigger fires. These options are related as follows:

$BreachDuration = Period * EvaluationPeriods$

The default values for these options (5, 5, and 1, respectively) satisfy this equation. If you specify inconsistent values, Elastic Beanstalk might modify one of the values so that the equation is still satisfied.

Namespace: `aws:autoscaling:trigger`

Name	Description	Default	Valid values
BreachDuration	Amount of time, in minutes, a metric can be beyond its defined limit (as specified in the <code>UpperThreshold</code> and <code>LowerThreshold</code>) before the trigger fires.	5	1 to 600
LowerBreachScaleIncrement	How many Amazon EC2 instances to remove when performing a scaling activity.	-1	
LowerThreshold	If the measurement falls below this number for the breach duration, a trigger is fired.	2000000	0 to 20000000
MeasureName	Metric used for your Auto Scaling trigger.	<code>NetworkOut</code>	<code>CPUUtilization</code> <code>NetworkIn</code> <code>NetworkOut</code> <code>DiskWriteOps</code> <code>DiskReadBytes</code> <code>DiskReadOps</code> <code>DiskWriteBytes</code> <code>Latency</code> <code>RequestCount</code> <code>HealthyHostCount</code> <code>UnhealthyHostCount</code> <code>TargetResponseTime</code>

Name	Description	Default	Valid values
Period	Specifies how frequently Amazon CloudWatch measures the metrics for your trigger. The value is the number of minutes between two consecutive periods.	5	1 to 600
EvaluationPeriods	The number of consecutive evaluation periods used to determine if a breach is occurring.	1	1 to 600
Statistic	Statistic the trigger should use, such as Average.	Average	Minimum Maximum Sum Average
Unit	Unit for the trigger measurement, such as Bytes.	Bytes	Seconds Percent Bytes Bits Count Bytes/Second Bits/Second Count/Second None
UpperBreachScaleIncrement	How many Amazon EC2 instances to add when performing a scaling activity.	1	
UpperThreshold	If the measurement is higher than this number for the breach duration, a trigger is fired.	6000000	0 to 20000000

aws:autoscaling:updatepolicy:rollingupdate

Configure rolling updates your environment's Auto Scaling group.

Namespace: `aws:autoscaling:updatepolicy:rollingupdate`

Name	Description	Default	Valid values
MaxBatchSize	The number of instances included in each batch of the rolling update.	One-third of the minimum size of the autoscaling group, rounded to the next highest integer.	1 to 10000

Name	Description	Default	Valid values
MinInstancesInService	The minimum number of instances that must be in service within the autoscaling group while other instances are terminated.	The minimum size of the AutoScaling group or one less than the maximum size of the autoscaling group, whichever is lower.	0 to 9999
RollingUpdateEnabled	<p>If <code>true</code>, enables rolling updates for an environment. Rolling updates are useful when you need to make small, frequent updates to your Elastic Beanstalk software application and you want to avoid application downtime.</p> <p>Setting this value to <code>true</code> automatically enables the <code>MaxBatchSize</code>, <code>MinInstancesInService</code>, and <code>PauseTime</code> options. Setting any of those options also automatically sets the <code>RollingUpdateEnabled</code> option value to <code>true</code>. Setting this option to <code>false</code> disables rolling updates.</p> <p>Note If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file (p. 600). The console and EB CLI override this option with a recommended value (p. 537).</p>	<code>false</code>	<code>true</code> <code>false</code>

Name	Description	Default	Valid values
RollingUpdateType	<p>Time-based rolling updates apply a <code>PauseTime</code> between batches. Health-based rolling updates wait for new instances to pass health checks before moving on to the next batch. Immutable updates (p. 412) launch a full set of instances in a new AutoScaling group.</p> <p>Note If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file (p. 600). The console and EB CLI override this option with a recommended value (p. 537).</p>	Time	<p>Time</p> <p>Health</p> <p>Immutable</p>
PauseTime	The amount of time the Elastic Beanstalk service will wait after it has completed updates to one batch of instances before it continues on to the next batch.	Automatically computed based on instance type and container.	PT0S* (0 seconds) to PT1H (1 hour)
Timeout	Maximum amount of time to wait for all instances in a batch of instances to pass health checks before canceling the update.	PT30M (30 minutes)	<p>PT5M* (5 minutes) to PT1H (1 hour)</p> <p>*ISO8601 duration format: <code>PT#H#M#S</code> where each # is the number of hours, minutes, and/or seconds, respectively.</p>

aws:ec2:instances

Configure your environment's instances, including Spot options. This namespace complements [aws:autoscaling:launchconfiguration](#) (p. 556) and [aws:autoscaling:asg](#) (p. 555). For more information, see [the section called "Auto Scaling group"](#) (p. 457).

Namespace: aws:ec2:instances

Name	Description	Default	Valid values
EnableSpot	Enable Spot Instance requests for your environment. When <code>false</code> , some options in this namespace don't take effect.	<code>false</code>	<code>true</code> <code>false</code>
InstanceTypes	<p>A comma-separated list of instance types you want your environment to use. For example: <code>t2.micro,t3.micro</code></p> <p>When Spot Instances are disabled (<code>EnableSpot</code> is <code>false</code>), only the first instance type on the list is used.</p> <p>The first instance type on the list in this option is equivalent to the value of the <code>InstanceType</code> option in the aws:autoscaling:launchconfiguration (p. 556) namespace. The latter is obsolete, and we don't recommend using it. If you specify both, the first instance type on the list in the <code>InstanceTypes</code> option is used, and <code>InstanceType</code> is ignored.</p> <p>Note Some older AWS accounts might provide Elastic Beanstalk with default instance types that don't support Spot Instances (for example, <code>t1.micro</code>). If you enable Spot Instance requests and you get an error about an instance type that doesn't support Spot, be sure to configure instance types that support Spot. To choose Spot Instance types, use the Spot Instance Advisor.</p> <p>When you update your environment configuration and remove one or more instance types from the <code>InstanceTypes</code> option, Elastic Beanstalk terminates any Amazon EC2 instances running on any of the removed instance types. Your environment's Auto Scaling group then launches new instances, as necessary to complete</p>	A list of two instance types. Varies by account, region, and platform.	1 to 10 EC2 instance types (at least two recommended)

Name	Description	Default	Valid values
	the desired capacity, using your current specified instance types.		
SpotFleetOnDemandBaseInstances	The minimum number of On-Demand Instances that your Auto Scaling group provisions before considering Spot Instances as your environment scales up. This option is relevant only when <code>EnableSpot</code> is <code>true</code> .	0	0 to <code>MaxSize</code> option in <code>aws:autoscaling:asg</code> (p. 555) namespace
SpotFleetOnDemandAboveBasePercentage	The Base Percentage of On-Demand Instances as part of additional capacity that your Auto Scaling group provisions beyond the <code>SpotOnDemandBase</code> instances. This option is relevant only when <code>EnableSpot</code> is <code>true</code> .	0 for a Single Instance environment 70 for a Load Balanced environment	0 to 100
SpotMaxPrice	The maximum price per unit hour, in US \$, that you're willing to pay for a Spot Instance. This option is relevant only when <code>EnableSpot</code> is <code>true</code> .	On-Demand price, per instance type. The option's value in this case is <code>null</code> .	0.001 to 20.0 <code>null</code>

aws:ec2:vpc

Configure your environment to launch resources in a custom [Amazon Virtual Private Cloud](#) (Amazon VPC). If you don't configure settings in this namespace, Elastic Beanstalk launches resources in the default VPC.

Namespace: `aws:ec2:vpc`

Name	Description	Default	Valid values
VPCId	The ID for your Amazon VPC.	None	
Subnets	The IDs of the Auto Scaling group subnet or subnets. If you have multiple subnets, specify the value as a single comma-delimited string of subnet IDs (for example, "subnet-11111111, subnet-22222222").	None	

Name	Description	Default	Valid values
ELBSubnets	The IDs of the subnet or subnets for the elastic load balancer. If you have multiple subnets, specify the value as a single comma-delimited string of subnet IDs (for example, "subnet-11111111, subnet-22222222").	None	
ELBScheme	Specify <code>internal</code> if you want to create an internal load balancer in your Amazon VPC so that your Elastic Beanstalk application cannot be accessed from outside your Amazon VPC. If you specify a value other than <code>public</code> or <code>internal</code> , Elastic Beanstalk will ignore the value.	<code>public</code>	<code>public</code> <code>internal</code>
DBSubnets	Contains the IDs of the database subnets. This is only used if you want to add an Amazon RDS DB Instance as part of your application. If you have multiple subnets, specify the value as a single comma-delimited string of subnet IDs (for example, "subnet-11111111, subnet-22222222").	None	
AssociatePublicIP	Specifies whether to launch instances with public IP addresses in your Amazon VPC. Instances with public IP addresses do not require a NAT device to communicate with the Internet. You must set the value to <code>true</code> if you want to include your load balancer and instances in a single public subnet. This option has no effect on a single-instance environment, which always has a single Amazon EC2 instance with an Elastic IP address. The option is relevant to load-balancing, autoscaling environments.	None	<code>true</code> <code>false</code>

aws:elasticbeanstalk:application

Configure a health check path for your application. For more information, see [Basic health reporting \(p. 688\)](#).

Namespace: `aws:elasticbeanstalk:application`

Name	Description	Default	Valid values
Application Healthcheck URL	The path to which to send health check requests. If not set, the load balancer attempts to make a TCP connection on port 80 to verify health. Set to a path starting with <code>/</code> to send an HTTP GET request to that path. You can also include a protocol (HTTP, HTTPS, TCP, or SSL) and port prior to the path to check HTTPS connectivity or use a non-default port. Note If you use the Elastic Beanstalk console to create	None	<code>/</code> (HTTP GET to root path) <code>/health</code> <code>HTTPS:443/</code> <code>HTTPS:443/health</code> etc

Name	Description	Default	Valid values
	an environment, you can't set this option in a configuration file (p. 600) . The console overrides this option with a recommended value (p. 537) .		

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See [Recommended values \(p. 537\)](#) for details.

aws:elasticbeanstalk:application:environment

Configure environment properties for your application.

Namespace: `aws:elasticbeanstalk:application:environment`

Name	Description	Default	Valid values
Any environment variable name.	Pass in key-value pairs.	None	Any environment variable value.

See [Environment properties and other software settings \(p. 516\)](#) for more information.

aws:elasticbeanstalk:cloudwatch:logs

Configure instance log streaming for your application.

Namespace: `aws:elasticbeanstalk:cloudwatch:logs`

Name	Description	Default	Valid values
StreamLogs	Whether to create groups in CloudWatch Logs for proxy and deployment logs, and stream logs from each instance in your environment.	false	true false
DeleteOnTermination	Whether to delete the log groups when the environment is terminated. If false, the logs are kept RetentionInDays days.	false	true false
RetentionInDays	The number of days to keep log events before they expire.	7	1, 3, 5, 7, 14, 30, 60, 90, 120, 150, 180, 365, 400, 545, 731, 1827, 3653

aws:elasticbeanstalk:cloudwatch:logs:health

Configure environment health log streaming for your application.

Namespace: `aws:elasticbeanstalk:cloudwatch:logs:health`

Name	Description	Default	Valid values
HealthStreamingEnabled	For environments with enhanced health reporting enabled, whether to create a group in CloudWatch Logs for environment health and archive Elastic Beanstalk environment health data. For information about enabling enhanced health, see aws:elasticbeanstalk:healthreporting:system (p. 577).	false	true false
DeleteOnTermination	Whether to delete the log group when the environment is terminated. If false, the health data is kept <code>RetentionInDays</code> days.	false	true false
RetentionInDays	The number of days to keep the archived health data before it expires.	7	1, 3, 5, 7, 14, 30, 60, 90, 120, 150, 180, 365, 400, 545, 731, 1827, 3653

aws:elasticbeanstalk:command

Configure the deployment policy for your application code. For more information, see [the section called "Deployment options"](#) (p. 400).

Namespace: `aws:elasticbeanstalk:command`

Name	Description	Default	Valid values
DeploymentPolicy	Choose a deployment policy (p. 400) for application version deployments. Note If you use the Elastic Beanstalk console to create an environment, you can't set this option in a configuration file (p. 600). The console overrides this option with a recommended value (p. 537).	AllAtOnce	AllAtOnce Rolling RollingWithAddition Immutable TrafficSplitting
Timeout	Number of seconds to wait for an instance to complete executing commands. Elastic Beanstalk internally adds 240 seconds (four minutes) to the <code>Timeout</code> value. For example, the effective timeout by default is 840 seconds (600 + 240), or 14 minutes.	600	1 to 3600

Name	Description	Default	Valid values
BatchSizeType	The type of number that is specified in BatchSize . Note If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file (p. 600) . The console and EB CLI override this option with a recommended value (p. 537) .	Percentage	Percentage Fixed
BatchSize	Percentage or fixed number of Amazon EC2 instances in the Auto Scaling group on which to simultaneously perform deployments. Valid values vary per BatchSizeType setting. Note If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file (p. 600) . The console and EB CLI override this option with a recommended value (p. 537) .	100	1 to 100 (Percentage). 1 to aws:autoscaling:asg::Max (Fixed)
IgnoreHealthCheck	Do not cancel a deployment due to failed health checks.	false	true false

aws:elasticbeanstalk:environment

Configure your environment's architecture and service role.

Namespace: `aws:elasticbeanstalk:environment`

Name	Description	Default	Valid values
EnvironmentType	Set to <code>SingleInstance</code> to launch one EC2 instance with no load balancer.	LoadBalanced	SingleInstance LoadBalanced
ServiceRole	The name of an IAM role that Elastic Beanstalk uses to manage resources for the environment. Specify a role name (optionally prefixed with a custom path) or its ARN. Examples: <ul style="list-style-type: none">• <code>aws-elasticbeanstalk-service-role</code>• <code>custom-path/custom-role</code>• <code>arn:aws:iam::123456789012:role/aws-elasticbeanstalk-service-role</code>	None	IAM role name, path/name, or ARN

Name	Description	Default	Valid values
	<p>Note If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file (p. 600). The console and EB CLI override this option with a recommended value (p. 537).</p>		
LoadBalancerType	The type of load balancer for your environment.	classic	classic application network

aws:elasticbeanstalk:environment:process:default

Configure your environment's default process.

Namespace: `aws:elasticbeanstalk:environment:process:default`

Name	Description	Default	Valid values
DeregistrationDelay	Time, in seconds, to wait for active requests to complete before deregistering.	20	0 to 3600
HealthCheckInterval	The interval, in seconds, at which Elastic Load Balancing will check the health of your application's Amazon EC2 instances.	With classic or application load balancer: 15 With network load balancer: 30	With classic or application load balancer: 5 to 300 With network load balancer: 10, 30
HealthCheckPath	Path to which to send HTTP requests for health checks.	/	A routable path.
HealthCheckTimeout	Time, in seconds, to wait for a response during a health check. This option is only applicable to environments with an application load balancer.	5	1 to 60
HealthyThresholdCount	Consecutive successful requests before Elastic Load Balancing changes the instance health status.	With classic or application load balancer: 3 With network load balancer: 5	2 to 10

Name	Description	Default	Valid values
MatcherHTTPCode	<p>A comma-separated list of HTTP code(s) that indicate that an instance is healthy.</p> <p>This option is only applicable to environments with a network or application load balancer.</p>	200	<p>With application load balancer: 200 to 499</p> <p>With network load balancer: 200 to 399</p>
Port	Port on which the process listens.	80	1 to 65535
Protocol	<p>Protocol that the process uses.</p> <p>With an application load balancer, you can only set this option to <code>HTTP</code> or <code>HTTPS</code>.</p> <p>With a network load balancer, you can only set this option to <code>TCP</code>.</p>	<p>With classic or application load balancer: <code>HTTP</code></p> <p>With network load balancer: <code>TCP</code></p>	<p><code>TCP</code></p> <p><code>HTTP</code></p> <p><code>HTTPS</code></p>
StickinessEnabled	<p>Set to true to enable sticky sessions.</p> <p>This option is only applicable to environments with an application load balancer.</p>	'false'	'false' 'true'
StickinessLBCookieDuration	<p>Lifetime, in seconds, of the sticky session cookie.</p> <p>This option is only applicable to environments with an application load balancer.</p>	86400 (one day)	1 to 604800
StickinessType	<p>Set to <code>lb_cookie</code> to use cookies for sticky sessions.</p> <p>This option is only applicable to environments with an application load balancer.</p>	<code>lb_cookie</code>	<code>lb_cookie</code>

Name	Description	Default	Valid values
UnhealthyThresholdCount	Consecutive unsuccessful requests before Elastic Load Balancing changes the instance health status.	5	2 to 10

aws:elasticbeanstalk:environment:process:process_name

Configure additional processes for your environment.

Namespace: `aws:elasticbeanstalk:environment:process:process_name`

Name	Description	Default	Valid values
DeregistrationDelay	Time, in seconds, to wait for active requests to complete before deregistering.	20	0 to 3600
HealthCheckInterval	The interval, in seconds, at which Elastic Load Balancing will check the health of your application's Amazon EC2 instances.	With classic or application load balancer: 15 With network load balancer: 30	With classic or application load balancer: 5 to 300 With network load balancer: 10, 30
HealthCheckPath	Path to which to send HTTP requests for health checks.	/	A routable path.
HealthCheckTimeout	Time, in seconds, to wait for a response during a health check. This option is only applicable to environments with an application load balancer.	5	1 to 60
HealthyThresholdCount	Consecutive successful requests before Elastic Load Balancing changes the instance health status.	With classic or application load balancer: 3 With network load balancer: 5	2 to 10
MatcherHTTPCode	A comma-separated list of HTTP code(s) that indicate that an instance is healthy. This option is only applicable to environments with a	200	With application load balancer: 200 to 499 With network load balancer: 200 to 399

Name	Description	Default	Valid values
	network or application load balancer.		
Port	Port on which the process listens.	80	1 to 65535
Protocol	<p>Protocol that the process uses.</p> <p>With an application load balancer, you can only set this option to <code>HTTP</code> or <code>HTTPS</code>.</p> <p>With a network load balancer, you can only set this option to <code>TCP</code>.</p>	<p>With classic or application load balancer: <code>HTTP</code></p> <p>With network load balancer: <code>TCP</code></p>	<p><code>TCP</code></p> <p><code>HTTP</code></p> <p><code>HTTPS</code></p>
StickinessEnabled	<p>Set to <code>true</code> to enable sticky sessions.</p> <p>This option is only applicable to environments with an application load balancer.</p>	'false'	'false' 'true'
StickinessLBCookieDuration	<p>Lifetime, in seconds, of the sticky session cookie.</p> <p>This option is only applicable to environments with an application load balancer.</p>	86400 (one day)	1 to 604800
StickinessType	<p>Set to <code>lb_cookie</code> to use cookies for sticky sessions.</p> <p>This option is only applicable to environments with an application load balancer.</p>	<code>lb_cookie</code>	<code>lb_cookie</code>
UnhealthyThresholdCount	Consecutive unsuccessful requests before Elastic Load Balancing changes the instance health status.	5	2 to 10

aws:elasticbeanstalk:environment:proxy:staticfiles

You can use the following namespace to configure the proxy server to serve static files. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application. This reduces the number of requests that your application has to process.

Map a path served by the proxy server to a folder in your source code that contains static assets. Each option that you define in this namespace maps a different path.

Note

This namespace applies to platform branches based on Amazon Linux 2. If your environment uses a platform version based on Amazon Linux AMI (preceding Amazon Linux 2), refer to [the section called "Platform specific options" \(p. 592\)](#) for platform-specific static file namespaces.

Namespace: aws:elasticbeanstalk:environment:proxy:staticfiles

Name	Value
<p>Path where the proxy server will serve the files. Start the value with <code>/</code>.</p> <p>Example: Specify <code>/images</code> to serve files at <code>subdomain.elasticbeanstalk.com/images</code>.</p>	<p>Name of the folder containing the files.</p> <p>Example: Specify <code>staticimages</code> to serve files from a folder named <code>staticimages</code> at the top level of your source bundle.</p>

aws:elasticbeanstalk:healthreporting:system

Configure enhanced health reporting for your environment.

Namespace: aws:elasticbeanstalk:healthreporting:system

Name	Description	Default	Valid values
SystemType	<p>Health reporting system (basic (p. 688) or enhanced (p. 691)). Enhanced health reporting requires a service role (p. 20) and a version 2 or newer platform version (p. 29).</p> <p>Note If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file (p. 600). The console and EB CLI override this option with a recommended value (p. 537).</p>	basic	<p>basic</p> <p>enhanced</p>
ConfigDocument	A JSON document describing the environment and instance metrics to publish to CloudWatch.	None	
HealthCheckSuccessThreshold	<p>The number of the threshold for instances to pass health checks.</p> <p>Note If you use the Elastic Beanstalk console to create an environment, you can't set this option in a configuration file (p. 600). The console overrides this option with a recommended value (p. 537).</p>	Ok	<p>Ok</p> <p>Warning</p> <p>Degraded</p> <p>Severe</p>

aws:elasticbeanstalk:hostmanager

Configure the EC2 instances in your environment to upload rotated logs to Amazon S3.

Namespace: `aws:elasticbeanstalk:hostmanager`

Name	Description	Default	proxy:staticfiles Valid values
LogPublicationControl	Copy the log files for your application's Amazon EC2 instances to the Amazon S3 bucket associated with your application.	false	true false

aws:elasticbeanstalk:managedactions

Configure managed platform updates for your environment.

Namespace: `aws:elasticbeanstalk:managedactions`

Name	Description	Default	Valid values
ManagedActionsEnabled	Enable managed platform updates (p. 425). When you set this to <code>true</code> , you must also specify a <code>PreferredStartTime</code> and <code>UpdateLevel</code> (p. 578).	true	true false
PreferredStartTime	Configure a maintenance window for managed actions in UTC. For example, " <code>Tue:09:00</code> ".	None	Day and time in <code>day:hour:minute</code> format.

aws:elasticbeanstalk:managedactions:platformupdate

Configure managed platform updates for your environment.

Namespace: `aws:elasticbeanstalk:managedactions:platformupdate`

Name	Description	Default	Valid values
UpdateLevel	The highest level of update to apply with managed platform updates. Platforms are versioned <code>major.minor.patch</code> . For example, 2.0.8 has a major version of 2, a minor version of 0, and a patch version of 8.	None	patch for patch version updates only. minor for both minor and patch version updates.
InstanceRefreshEnabled	Enable weekly instance replacement.	false	true false

Name	Description	Default	Valid values
	Requires <code>ManagedActionsEnabled</code> to be set to <code>true</code> .		

aws:elasticbeanstalk:monitoring

Configure your environment to terminate EC2 instances that fail health checks.

Namespace: `aws:elasticbeanstalk:monitoring`

Name	Description	Default	Valid values
Automatically Terminate Unhealthy Instances	<p>Terminate an instance if it fails health checks.</p> <p>Note This option was only supported on legacy environments (p. 425). It determined the health of an instance based on being able to reach it and on other instance-based metrics. Elastic Beanstalk doesn't provide a way to automatically terminate instances based on application health.</p>	<code>true</code>	<code>true</code> <code>false</code>

aws:elasticbeanstalk:sns:topics

Configure notifications for your environment.

Namespace: `aws:elasticbeanstalk:sns:topics`

Name	Description	Default	Valid values
Notification Endpoint	<p>Endpoint where you want to be notified of important events affecting your application.</p> <p>Note If you use the Elastic Beanstalk console to create an environment, you can't set this option in a configuration file (p. 600). The console overrides this option with a recommended value (p. 537).</p>	<code>None</code>	
Notification Protocol	Protocol used to send notifications to your endpoint.	<code>email</code>	<code>http</code> <code>https</code>

Name	Description	Default	Valid values
			email email-json sqs
Notification Topic ARN	Amazon Resource Name for the topic you subscribed to.	None	
Notification Topic Name	Name of the topic you subscribed to.	None	

aws:elasticbeanstalk:sqsd

Configure the Amazon SQS queue for a worker environment.

Namespace: `aws:elasticbeanstalk:sqsd`

Name	Description	Default	Valid values
WorkerQueueURL	The URL of the queue from which the daemon in the worker environment tier reads messages Note When you don't specify a value, the queue that Elastic Beanstalk automatically creates is a standard Amazon SQS queue. When you provide a value, you can provide the URL of either a standard or a FIFO Amazon SQS queue. Be aware that if you provide a FIFO queue, periodic tasks (p. 440) aren't supported.	automatically generated	If you don't specify a value, then Elastic Beanstalk automatically creates a queue.
HttpPath	The relative path to the application to which HTTP POST messages are sent	/	
MimeType	The MIME type of the message sent in the HTTP POST request	application/json	application/json application/x-www-form-urlencoded application/xml text/plain Custom MIME type.
HttpConnections	The maximum number of concurrent connections to any application(s) within an Amazon EC2 instance	50	1 to 100

Name	Description	Default	Valid values
	<p>Note</p> <p>If you use the Elastic Beanstalk console to create an environment, you can't set this option in a configuration file (p. 600). The console overrides this option with a recommended value (p. 537).</p>		
ConnectTimeout	The amount of time, in seconds, to wait for successful connections to an application	5	1 to 60
InactivityTimeout	The amount of time, in seconds, to wait for a response on an existing connection to an application The message is reprocessed until the daemon receives a 200 OK response from the application in the worker environment tier or the <code>RetentionPeriod</code> expires.	299	1 to 36000
VisibilityTimeout	The amount of time, in seconds, an incoming message from the Amazon SQS queue is locked for processing. After the configured amount of time has passed, then the message is again made visible in the queue for any other daemon to read.	300	0 to 43200
ErrorVisibilityTimeout	The amount of time, in seconds, that elapses before Elastic Beanstalk returns a message to the Amazon SQS queue after a processing attempt fails with an explicit error.	2 seconds	0 to 43200 seconds
RetentionPeriod	The amount of time, in seconds, a message is valid and will be actively processed	345600	60 to 1209600
MaxRetries	The maximum number of attempts that Elastic Beanstalk attempts to send the message to the web application that will process it before moving the message to the dead-letter queue.	10	1 to 100

aws:elasticbeanstalk:trafficsplitting

Configure traffic-splitting deployments for your environment.

This namespace applies when you set the `DeploymentPolicy` option of the [aws:elasticbeanstalk:command \(p. 571\)](#) namespace to `TrafficSplitting`. For more information about deployment policies, see [the section called "Deployment options" \(p. 400\)](#).

Namespace: `aws:elasticbeanstalk:trafficsplitting`

Name	Description	Default	Valid values
<code>NewVersionPercent</code>	The initial percentage of incoming client traffic that Elastic Beanstalk shifts to environment instances running the new application version you're deploying.	10	1 to 100
<code>EvaluationTime</code>	The time period, in minutes, that Elastic Beanstalk waits after an initial healthy deployment before proceeding to shift all incoming client traffic to the new application version that you're deploying.	5	3 to 600

aws:elasticbeanstalk:xray

Run the AWS X-Ray daemon to relay trace information from your [X-Ray integrated \(p. 521\)](#) application.

Namespace: `aws:elasticbeanstalk:xray`

Name	Description	Default	Valid values
<code>XRayEnabled</code>	Set to <code>true</code> to run the X-Ray daemon on the instances in your environment.	<code>false</code>	<code>true</code> <code>false</code>

aws:elb:healthcheck

Configure healthchecks for a classic load balancer.

Namespace: `aws:elb:healthcheck`

Name	Description	Default	Valid values
<code>HealthyThreshold</code>	Consecutive successful requests before Elastic Load Balancing changes the instance health status.	3	2 to 10
<code>Interval</code>	The interval at which Elastic Load Balancing will check the health of your application's Amazon EC2 instances.	10	5 to 300
<code>Timeout</code>	Number of seconds Elastic Load Balancing will wait for a response before it considers the instance nonresponsive.	5	2 to 60
<code>UnhealthyThreshold</code>	Consecutive unsuccessful requests before Elastic Load Balancing changes the instance health status.	5	2 to 10
<code>(deprecated) Target</code>	Destination on backend instance to which to send health checks. Use <code>Application Healthcheck URL</code> in the aws:elasticbeanstalk:application (p. 569) namespace instead.	<code>TCP:80</code>	Target in the format <i>PROTOCOL:PORT/PATH</i>

aws:elb:loadbalancer

Configure your environment's classic load balancer.

Several of the options in this namespace have been deprecated in favor of listener-specific options in the [aws:elb:listener \(p. 584\)](#) namespace. The deprecated options only let you configure two listeners (one secure and one unsecure) on standard ports.

Namespace: aws:elb:loadbalancer

Name	Description	Default	Valid values
CrossZone	<p>Configure the load balancer to route traffic evenly across all instances in all Availability Zones rather than only within each zone.</p> <p>Note If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file (p. 600). The console and EB CLI override this option with a recommended value (p. 537).</p>	false	true false
SecurityGroups	Assign one or more security groups that you created to the load balancer.	None	One or more security group IDs.
ManagedSecurityGroup	<p>Assign an existing security group to your environment's load balancer, instead of creating a new one. To use this setting, update the <code>SecurityGroups</code> setting in this namespace to include your security group's ID, and remove the automatically created security group's ID, if one exists.</p> <p>To allow traffic from the load balancer to your environment's EC2 instances, Elastic Beanstalk adds a rule to the instances' security group that allows inbound traffic from the managed security group.</p>	None	A security group ID.
(deprecated) LoadBalancerHTTPPort	Port to listen on for the unsecure listener.	80	OFF 80
(deprecated) LoadBalancerPortProtocol	Protocol to use on the unsecure listener.	HTTP	HTTP TCP
(deprecated) LoadBalancerHTTPSPort	Port to listen on for the secure listener.	OFF	OFF 443 8443
(deprecated) LoadBalancerSSLPortProtocol	Protocol to use on the secure listener.	HTTPS	HTTPS SSL

Name	Description	Default	Valid values
(deprecated) SSLCertificateId	ARN of an SSL certificate to bind to the secure listener.	None	

aws:elb:listener

Configure the default listener (port 80) on a classic load balancer.

Namespace: `aws:elb:listener`

Name	Description	Default	Valid values
ListenerProtocol	The protocol used by the listener.	HTTP	HTTP TCP
InstancePort	The port that this listener uses to communicate with the EC2 instances.	80	1 to 65535
InstanceProtocol	The protocol that this listener uses to communicate with the EC2 instances. It must be at the same internet protocol layer as the <code>ListenerProtocol</code> . It also must have the same security level as any other listener using the same <code>InstancePort</code> as this listener. For example, if <code>ListenerProtocol</code> is <code>HTTPS</code> (application layer, using a secure connection), you can set <code>InstanceProtocol</code> to <code>HTTP</code> (also at the application layer, using an insecure connection). If, in addition, you set <code>InstancePort</code> to 80, you must set <code>InstanceProtocol</code> to <code>HTTP</code> in all other listeners with <code>InstancePort</code> set to 80.	HTTP when <code>ListenerProtocol</code> is HTTP TCP when <code>ListenerProtocol</code> is TCP	HTTP or HTTPS when <code>ListenerProtocol</code> is HTTP or HTTPS TCP or SSL when <code>ListenerProtocol</code> is TCP or SSL
PolicyNames	A comma-separated list of policy names to apply to the port for this listener. We suggest that you use the <code>LoadBalancerPorts</code> option of the aws:elb:policies (p. 585) namespace instead.	None	
ListenerEnabled	Specifies whether this listener is enabled. If you specify <code>false</code> , the listener is not included in the load balancer.	true	true false

aws:elb:listener:listener_port

Configure additional listeners on a classic load balancer.

Namespace: `aws:elb:listener:listener_port`

Name	Description	Default	Valid values
ListenerProtocol	The protocol used by the listener.	HTTP	HTTP HTTPS TCP SSL

Name	Description	Default	Valid values
InstancePort	The port that this listener uses to communicate with the EC2 instances.	The same as <code>listener_port</code> .	1 to 65535
InstanceProtocol	<p>The protocol that this listener uses to communicate with the EC2 instances.</p> <p>It must be at the same internet protocol layer as the <code>ListenerProtocol</code>. It also must have the same security level as any other listener using the same <code>InstancePort</code> as this listener.</p> <p>For example, if <code>ListenerProtocol</code> is <code>HTTPS</code> (application layer, using a secure connection), you can set <code>InstanceProtocol</code> to <code>HTTP</code> (also at the application layer, using an insecure connection). If, in addition, you set <code>InstancePort</code> to 80, you must set <code>InstanceProtocol</code> to <code>HTTP</code> in all other listeners with <code>InstancePort</code> set to 80.</p>	<p>HTTP when <code>ListenerProtocol</code> is HTTP or HTTPS</p> <p>TCP when <code>ListenerProtocol</code> is TCP or SSL</p>	<p>HTTP or HTTPS when <code>ListenerProtocol</code> is HTTP or HTTPS</p> <p>TCP or SSL when <code>ListenerProtocol</code> is TCP or SSL</p>
PolicyNames	A comma-separated list of policy names to apply to the port for this listener. We suggest that you use the <code>LoadBalancerPorts</code> option of the <code>aws:elb:policies</code> (p. 585) namespace instead.	None	
SSLCertificateId	ARN of an SSL certificate to bind to the listener.	None	
ListenerEnabled	Specifies whether this listener is enabled. If you specify <code>false</code> , the listener is not included in the load balancer.	<code>true</code> if any other option is set. <code>false</code> otherwise.	<code>true</code> <code>false</code>

aws:elb:policies

Modify the default stickiness and global load balancer policies for a classic load balancer.

Namespace: `aws:elb:policies`

Name	Description	Default	Valid values
ConnectionDrainingEnabled	<p>Specifies whether the load balancer maintains existing connections to instances that have become unhealthy or deregistered to complete in-progress requests.</p> <p>Note If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file (p. 600). The console and</p>	<code>false</code>	<code>true</code> <code>false</code>

Name	Description	Default	Valid values
	EB CLI override this option with a recommended value (p. 537) .		
ConnectionDrainingTimeout	The maximum number of seconds that the load balancer maintains existing connections to an instance during connection draining before forcibly closing the connections. Note If you use the Elastic Beanstalk console to create an environment, you can't set this option in a configuration file (p. 600) . The console overrides this option with a recommended value (p. 537) .	20	1 to 3600
ConnectionSettingIdleTimeout	Number of seconds that the load balancer waits for any data to be sent or received over the connection. If no data has been sent or received after this time period elapses, the load balancer closes the connection.	60	1 to 3600
LoadBalancerPorts	A comma-separated list of the listener ports that the default policy (<code>AWSEB-ELB-StickinessPolicy</code>) applies to.	None	You can use <code>:all</code> to indicate all listener ports
Stickiness Cookie Expiration	The amount of time, in seconds, that each cookie is valid. Uses the default policy (<code>AWSEB-ELB-StickinessPolicy</code>).	0	0 to 1000000
Stickiness Policy	Binds a user's session to a specific server instance so that all requests coming from the user during the session are sent to the same server instance. Uses the default policy (<code>AWSEB-ELB-StickinessPolicy</code>).	false	true false

aws:elb:policies:policy_name

Create additional load balancer policies for a classic load balancer.

Namespace: `aws:elb:policies:policy_name`

Name	Description	Default	Valid values
CookieName	The name of the application-generated cookie that controls the session lifetimes of a <code>AppCookieStickinessPolicyType</code> policy. This policy can be associated only with HTTP/HTTPS listeners.	None	
InstancePorts	A comma-separated list of the instance ports that this policy applies to.	None	A list of ports, or <code>:all</code>
LoadBalancerPorts	A comma-separated list of the listener ports that this policy applies to.	None	A list of ports, or <code>:all</code>

Name	Description	Default	Valid values
ProxyProtocol	For a <code>ProxyProtocolPolicyType</code> policy, specifies whether to include the IP address and port of the originating request for TCP messages. This policy can be associated only with TCP/SSL listeners.	None	true false
PublicKey	The contents of a public key for a <code>PublicKeyPolicyType</code> policy to use when authenticating the back-end server or servers. This policy cannot be applied directly to back-end servers or listeners; it must be part of a <code>BackendServerAuthenticationPolicyType</code> policy.	None	
PublicKeyPolicyNames	A comma-separated list of policy names (from the <code>PublicKeyPolicyType</code> policies) for a <code>BackendServerAuthenticationPolicyType</code> policy that controls authentication to a back-end server or servers. This policy can be associated only with back-end servers that are using HTTPS/SSL.	None	
SSLProtocols	A comma-separated list of SSL protocols to be enabled for a <code>SSLNegotiationPolicyType</code> policy that defines the ciphers and protocols that will be accepted by the load balancer. This policy can be associated only with HTTPS/SSL listeners.	None	
SSLReferencePolicy	The name of a predefined security policy that adheres to AWS security best practices and that you want to enable for a <code>SSLNegotiationPolicyType</code> policy that defines the ciphers and protocols that will be accepted by the load balancer. This policy can be associated only with HTTPS/SSL listeners.	None	
Stickiness Cookie Expiration	The amount of time, in seconds, that each cookie is valid.	0	0 to 1000000
Stickiness Policy	Binds a user's session to a specific server instance so that all requests coming from the user during the session are sent to the same server instance.	false	true false

aws:elbv2:listener:default

Configure the default listener (port 80) on an application load balancer or a network load balancer.

Namespace: `aws:elbv2:listener:default`

Name	Description	Default	Valid values
DefaultProcess	Name of the process (p. 573) to which to forward traffic when no rules match.	default	A process name.
ListenerEnabled	Set to <code>false</code> to disable the listener. You can use this option to disable the default listener on port 80.	true	true false
Protocol	Protocol of traffic to process.	With application load balancer: HTTP With network load balancer: TCP	With application load balancer: HTTP, HTTPS With network load balancer: TCP
Rules	List of rules (p. 589) to apply to the listener This option is only applicable to environments with an application load balancer.	None	Comma separated list of rule names.
SSLCertificateArns	The ARN of the SSL certificate to bind to the listener. This option is only applicable to environments with an application load balancer.	None	The ARN of a certificate stored in IAM or ACM.
SSLPolicy	Specify a security policy to apply to the listener. This option is only applicable to environments with an application load balancer.	None (ELB default)	The name of a load balancer security policy.

[aws:elbv2:listener:listener_port](#)

Configure additional listeners on an application load balancer or a network load balancer.

Namespace: `aws:elbv2:listener:listener_port`

Name	Description	Default	Valid values
DefaultProcess	Name of the process (p. 573) where traffic is forwarded when no rules match.	default	A process name.
ListenerEnabled	Set to <code>false</code> to disable the listener. You can use this option to disable the default listener on port 80.	true	true false
Protocol	Protocol of traffic to process.	With application load balancer: HTTP With network load balancer: TCP	With application load balancer: HTTP, HTTPS With network load balancer: TCP
Rules	List of rules (p. 589) to apply to the listener This option is only applicable to environments with an application load balancer.	None	Comma separated list of rule names.
SSLCertificateArns	The ARN of the SSL certificate to bind to the listener. This option is only applicable to environments with an application load balancer.	None	The ARN of a certificate stored in IAM or ACM.
SSLPolicy	Specify a security policy to apply to the listener. This option is only applicable to environments with an application load balancer.	None (ELB default)	The name of a load balancer security policy.

aws:elbv2:listenerrule:rule_name

Define listener rules for an application load balancer. If a request matches the host names or paths in a rule, the load balancer forwards it to the specified process. To use a rule, add it to a listener with the `Rules` option in the `aws:elbv2:listener:listener_port` (p. 588) namespace.

Note

This namespace isn't applicable to environments with a network load balancer.

Namespace: `aws:elbv2:listenerrule:rule_name`

Name	Description	Default	Valid values
HostHeaders	List of host names to match. For example, <code>my.example.com</code> .	None	Each name can be up to 128 characters (A-Z, a-z, 0-9, -) and up to three wildcard characters (* matches zero or more characters; ? matches exactly one character)
PathPatterns	A path pattern to match. For example, <code>/img/*</code> . This option is only applicable to environments with an application load balancer.	None	A pattern can be up to 128 characters (A-Z, a-z, 0-9, -) and can include up to three wildcard characters (* matches zero or more characters; ? matches exactly one character)
Priority	Precedence of this rule when multiple rules match. The lower number takes precedence. No two rules can have the same priority.	1	1 to 1000
Process	Name of the process (p. 573) to which to forward traffic when this rule matches the request.	<code>default</code>	A process name.

aws:elbv2:loadbalancer

Configure an application load balancer.

Note

This namespace isn't applicable to environments with a network load balancer.

Namespace: `aws:elbv2:loadbalancer`

Name	Description	Default	Valid values
AccessLogsS3Bucket	Amazon S3 bucket in which to store access logs. The bucket must be in the same region as the environment and allow the load balancer write access.	None	A bucket name.
AccessLogsS3Enabled	Enable access log storage.	<code>false</code>	<code>true</code> <code>false</code>
AccessLogsS3Prefix	Prefix to prepend to access log names. By default, the load balancer uploads logs to a directory named <code>AWSLogs</code> in the bucket you specify. Specify a prefix to place the <code>AWSLogs</code> directory inside another directory.	None	

Name	Description	Default	Valid values
IdleTimeout	Time to wait for a request to complete before closing connections to client and instance.	None	1 to 3600
ManagedSecurityGroup	Assign an existing security group to your environment's load balancer, instead of creating a new one. To use this setting, update the <code>SecurityGroups</code> setting in this namespace to include your security group's ID, and remove the automatically created security group's ID, if one exists. To allow traffic from the load balancer to your environment's EC2 instances, Elastic Beanstalk adds a rule to the instances' security group that allows inbound traffic from the managed security group.	The security group that Elastic Beanstalk creates for your load balancer.	A security group ID.
SecurityGroups	List of security groups to attach to the load balancer.	The security group that Elastic Beanstalk creates for your load balancer.	Comma separated list of security group IDs.

aws:rds:dbinstance

Configure an attached Amazon RDS DB instance.

Namespace: `aws:rds:dbinstance`

Name	Description	Default	Valid values
DBAllocatedStorage	The allocated database storage size, specified in gigabytes.	MySQL: 5 Oracle: 10 sqlserver-se: 200 sqlserver-ex: 30 sqlserver-web: 30	MySQL: 5-1024 Oracle: 10-1024 sqlserver: cannot be modified
DBDeletionPolicy	Decides whether to delete or snapshot the DB instance on environment termination.	Delete	Delete Snapshot

Name	Description	Default	Valid values
	Warning Deleting a DB instance results in permanent data loss.		
DBEngine	The name of the database engine to use for this instance.	mysql	mysql oracle-se1 sqlserver-ex sqlserver-web sqlserver-se postgres
DBEngineVersion	The version number of the database engine.	5.5	
DBInstanceClass	The database instance type.	db.t2.micro (db.m1.large for an environment not running in an Amazon VPC)	Go to DB Instance Class in the <i>Amazon Relational Database Service User Guide</i> .
DBPassword	The name of master user password for the database instance.	None	
DBSnapshotIdentifier	The identifier for the DB snapshot to restore from.	None	
DBUser	The name of master user for the DB Instance.	ebroot	
MultiAZDatabase	Specifies whether a database instance Multi-AZ deployment needs to be created. For more information about Multi-AZ deployments with Amazon Relational Database Service (RDS), go to Regions and Availability Zones in the <i>Amazon Relational Database Service User Guide</i> .	false	true false

Platform specific options

Some Elastic Beanstalk platforms define option namespaces that are specific to the platform. These namespaces and their options are listed below for each platform.

Note

Previously, in platform versions based on Amazon Linux AMI (preceding Amazon Linux 2), the following two features and their respective namespaces were considered to be platform-specific features, and were listed here per platform:

- **Proxy configuration for static files** – [aws:elasticbeanstalk:environment:proxy:staticfiles](#) (p. 577)
- **AWS X-Ray support** – [aws:elasticbeanstalk:xray](#) (p. 582)

In Amazon Linux 2 platform versions, Elastic Beanstalk implements these features in a consistent way across all supporting platforms. The related namespaces are now listed in the [the section called "General options" \(p. 555\)](#) page. We only kept mention of them on this page for platforms who had differently-named namespaces.

Platforms

- [Docker platform options \(p. 593\)](#)
- [Go platform options \(p. 593\)](#)
- [Java SE platform options \(p. 594\)](#)
- [Java with Tomcat platform options \(p. 594\)](#)
- [.NET platform options \(p. 595\)](#)
- [Node.js platform options \(p. 595\)](#)
- [PHP platform options \(p. 597\)](#)
- [Python platform options \(p. 597\)](#)
- [Ruby platform options \(p. 599\)](#)

Docker platform options

The following Docker-specific configuration options apply to the Single Container and Preconfigured Docker platforms.

Note

These configuration options do not apply to the Multicontainer Docker platform.

Namespace: `aws:elasticbeanstalk:environment:proxy`

Name	Description	Default	Valid values
ProxyServer	Specifies the web server to use as a proxy.	nginx	nginx none (on Amazon Linux AMI (preceding Amazon Linux 2) platform versions)

Go platform options

Amazon Linux AMI (pre-Amazon Linux 2) platform options

Namespace: `aws:elasticbeanstalk:container:golang:staticfiles`

You can use the following namespace to configure the proxy server to serve static files. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application. This reduces the number of requests that your application has to process.

Map a path served by the proxy server to a folder in your source code that contains static assets. Each option that you define in this namespace maps a different path.

Name	Value
Path where the proxy server will serve the files. Example: <code>/images</code> to serve files at <code>subdomain.eleasticbeanstalk.com/images</code> .	Name of the folder containing the files. Example: <code>staticimages</code> to serve files from a folder named <code>staticimages</code> at the top level of your source bundle.

Java SE platform options

Amazon Linux AMI (pre-Amazon Linux 2) platform options

Namespace: `aws:elasticbeanstalk:container:java:staticfiles`

You can use the following namespace to configure the proxy server to serve static files. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application. This reduces the number of requests that your application has to process.

Map a path served by the proxy server to a folder in your source code that contains static assets. Each option that you define in this namespace maps a different path.

Name	Value
Path where the proxy server will serve the files. Example: <code>/images</code> to serve files at <code>subdomain.elasticbeanstalk.com/images</code> .	Name of the folder containing the files. Example: <code>staticimages</code> to serve files from a folder named <code>staticimages</code> at the top level of your source bundle.

Java with Tomcat platform options

Namespace: `aws:elasticbeanstalk:application:environment`

Name	Description	Default	Valid values
<code>JDBC_CONNECTION_STRING</code>	The connection string to an external database.	n/a	n/a

See [Environment properties and other software settings \(p. 516\)](#) for more information.

Namespace: `aws:elasticbeanstalk:container:tomcat:jvmoptions`

Name	Description	Default	Valid values
JVM Options	Pass command-line options to the JVM at startup.	n/a	n/a
Xmx	Maximum JVM heap sizes.	256m	n/a
XX:MaxPermSize	Section of the JVM heap that is used to store class definitions and associated metadata. Note This option only applies to Java versions earlier than Java 8, and isn't supported on Elastic Beanstalk Tomcat platforms based on Amazon Linux 2.	64m	n/a
Xms	Initial JVM heap sizes.	256m	n/a
<code>optionName</code>	Specify arbitrary JVM options in addition to the those defined by the Tomcat platform.	n/a	n/a

Amazon Linux AMI (pre-Amazon Linux 2) platform options

Namespace: `aws:elasticbeanstalk:environment:proxy`

Name	Description	Default	Valid values
GzipCompression	Set to <code>false</code> to disable response compression.	<code>true</code>	<code>true</code> <code>false</code>
ProxyServer	<p>Set the proxy to use on your environment's instances. If you don't set this option, or if you set it to <code>apache</code>, Elastic Beanstalk uses Apache 2.4.</p> <p>Set to <code>apache/2.2</code> if your application isn't ready to migrate away from Apache 2.2 due to incompatible proxy configuration settings.</p> <p>Set to <code>nginx</code> to use nginx.</p> <p>For more information, see Configuring your Tomcat environment's proxy server (p. 108).</p>	<code>apache</code>	<code>apache</code> <code>apache/2.2</code> <code>nginx</code>

.NET platform options

Namespace: `aws:elasticbeanstalk:container:dotnet:apppool`

Name	Description	Default	Valid values
Target Runtime	Choose the version of .NET Framework for your application.	<code>4.0</code>	<code>2.0</code> <code>4.0</code>
Enable 32-bit Applications	Set to <code>True</code> to run 32-bit applications.	<code>False</code>	<code>True</code> <code>False</code>

Node.js platform options

Amazon Linux AMI (pre-Amazon Linux 2) platform options

Namespace: `aws:elasticbeanstalk:container:nodejs`

Name	Description	Default	Valid values
NodeCommand	Command used to start the Node.js application. If an empty string is specified, <code>app.js</code> is used, then <code>server.js</code> , then <code>npm start</code> in that order.	<code>""</code>	<code>n/a</code>
NodeVersion	Version of Node.js. For example, <code>4.4.6</code>	<code>varies</code>	<code>varies</code>
	Supported Node.js versions vary between Node.js platform versions. See Node.js in the		

Name	Description	Default	Valid values
	<p><i>AWS Elastic Beanstalk Platforms</i> document for a list of the currently supported versions.</p> <p>Note When support for the version of Node.js that you are using is removed from the platform, you must change or remove the version setting prior to doing a platform update (p. 415). This might occur when a security vulnerability is identified for one or more versions of Node.js. When this happens, attempting to update to a new version of the platform that doesn't support the configured NodeVersion (p. 595) fails. To avoid needing to create a new environment, change the <i>NodeVersion</i> configuration option to a Node.js version that is supported by both the old platform version and the new one, or remove the option setting (p. 547), and then perform the platform update.</p>		
GzipCompression	Specifies if gzip compression is enabled. If ProxyServer is set to none, then gzip compression is disabled.	false	true false
ProxyServer	Specifies which web server should be used to proxy connections to Node.js. If ProxyServer is set to none, then static file mappings doesn't take affect and gzip compression is disabled.	nginx	apache nginx none

Namespace: `aws:elasticbeanstalk:container:nodejs:staticfiles`

You can use the following namespace to configure the proxy server to serve static files. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application. This reduces the number of requests that your application has to process.

Map a path served by the proxy server to a folder in your source code that contains static assets. Each option that you define in this namespace maps a different path.

Note

Static file settings do not apply if `aws:elasticbeanstalk:container:nodejs::ProxyFiles` is set to none.

Name	Value
Path where the proxy server will serve the files. Example: <code>/images</code> to serve files at <code>subdomain.eleasticbeanstalk.com/images</code> .	Name of the folder containing the files. Example: <code>staticimages</code> to serve files from a folder named <code>staticimages</code> at the top level of your source bundle.

PHP platform options

Namespace: `aws:elasticbeanstalk:container:php:phpini`

Name	Description	Default	Valid values
<code>document_root</code>	Specify the child directory of your project that is treated as the public-facing web root.	/	A blank string is treated as /, or specify a string starting with /
<code>memory_limit</code>	Amount of memory allocated to the PHP environment.	256M	n/a
<code>zlib.output_compression</code>	Specifies whether or not PHP should use compression for output.	Off	On Off true false
<code>allow_url_fopen</code>	Specifies if PHP's file functions are allowed to retrieve data from remote locations, such as websites or FTP servers.	On	On Off true false
<code>display_errors</code>	Specifies if error messages should be part of the output.	Off	On Off
<code>max_execution_time</code>	Sets the maximum time, in seconds, a script is allowed to run before it is terminated by the environment.	60	0 to 9223372036854775807 (PHP_INT_MAX)
<code>composer_options</code>	Sets custom options to use when installing dependencies using Composer through <code>composer.phar install</code> . For more information including available options, go to http://getcomposer.org/doc/03-cli.md#install .	n/a	n/a

Note

For more information about the PHP platform, see [Using the Elastic Beanstalk PHP platform \(p. 230\)](#).

Python platform options

Namespace: `aws:elasticbeanstalk:application:environment`

Name	Description	Default	Valid values
<code>DJANGO_SETTINGS_MODULE</code>	Specifies which settings file to use.	n/a	n/a

See [Environment properties and other software settings \(p. 516\)](#) for more information.

Namespace: `aws:elasticbeanstalk:container:python`

Name	Description	Default	Valid values
WSGIPath	The file that contains the WSGI application. This file must have an <code>application</code> callable.	On Amazon Linux 2 Python platform versions: <code>application:app</code> On Amazon Linux AMI Python platform versions: <code>application.py</code>	n/a
NumProcesses	The number of daemon processes that should be started for the process group when running WSGI applications.	1	n/a
NumThreads	The number of threads to be created to handle requests in each daemon process within the process group when running WSGI applications.	15	n/a

Amazon Linux AMI (pre-Amazon Linux 2) platform options

Namespace: `aws:elasticbeanstalk:container:python:staticfiles`

You can use the following namespace to configure the proxy server to serve static files. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application. This reduces the number of requests that your application has to process.

Map a path served by the proxy server to a folder in your source code that contains static assets. Each option that you define in this namespace maps a different path.

By default, the proxy server in a Python environment serves any files in a folder named `static` at the `/static` path.

Namespace: `aws:elasticbeanstalk:container:python:staticfiles`

Name	Value
Path where the proxy server will serve the files. Example: <code>/images</code> to serve files at <code>subdomain.elasticbeanstalk.com/images</code> .	Name of the folder containing the files. Example: <code>staticimages</code> to serve files from a folder named <code>staticimages</code> at the top level of your source bundle.

Ruby platform options

Namespace: `aws:elasticbeanstalk:application:environment`

Name	Description	Default	Valid values
RAILS_SKIP_MIGRATIONS	Specifies whether to run <code> rake db:migrate </code> on behalf of the users' applications; or whether it should be skipped. This is only applicable to Rails 3 applications.	false	true false
RAILS_SKIP_ASSET_COMPILATION	Specifies whether the container should run <code> rake assets:precompile </code> on behalf of the users' applications; or whether it should be skipped. This is also only applicable to Rails 3 applications.	false	true false
BUNDLE_WITHOUT	A colon (:) separated list of groups to ignore when installing dependencies from a Gemfile.	test:development	n/a
RACK_ENV	Specifies what environment stage an application can be run in. Examples of common environments include development, production, test.	production	n/a

See [Environment properties and other software settings \(p. 516\)](#) for more information.

Custom options

Use the `aws:elasticbeanstalk:customoption` namespace to define options and values that can be read in `Resources` blocks in other configuration files. Use custom options to collect user specified settings in a single configuration file.

For example, you may have a complex configuration file that defines a resource that can be configured by the user launching the environment. If you use `Fn::GetOptionSetting` to retrieve the value of a custom option, you can put the definition of that option in a different configuration file, where it is more easily discovered and modified by the user.

Also, because they are configuration options, custom options can be set at the API level to override values set in a configuration file. See [Precedence \(p. 537\)](#) for more information.

Custom options are defined like any other option:

```
option_settings:
  aws:elasticbeanstalk:customoption:
    option name: option value
```

For example, the following configuration file creates an option named `ELBAlarmEmail` and sets the value to `someone@example.com`:

```
option_settings:
  aws:elasticbeanstalk:customoption:
```

```
ELBAlarmEmail: someone@example.com
```

Elsewhere, a configuration file defines an SNS topic that reads the option with `Fn::GetOptionSetting` to populate the value of the `Endpoint` attribute:

```
Resources:
  MySNSTopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint:
            Fn::GetOptionSetting:
              OptionName: ELBAlarmEmail
              DefaultValue: nobody@example.com
            Protocol: email
```

You can find more example snippets using `Fn::GetOptionSetting` at [Adding and customizing Elastic Beanstalk environment resources \(p. 622\)](#).

Advanced environment customization with configuration files (.ebextensions)

You can add AWS Elastic Beanstalk configuration files (.ebextensions) to your web application's source code to configure your environment and customize the AWS resources that it contains. Configuration files are YAML- or JSON-formatted documents with a `.config` file extension that you place in a folder named `.ebextensions` and deploy in your application source bundle.

Example .ebextensions/network-load-balancer.config

This example makes a simple configuration change. It modifies a configuration option to set the type of your environment's load balancer to Network Load Balancer.

```
option_settings:
  aws:elasticbeanstalk:environment:
    LoadBalancerType: network
```

We recommend using YAML for your configuration files, because it's more readable than JSON. YAML supports comments, multi-line commands, several alternatives for using quotes, and more. However, you can make any configuration change in Elastic Beanstalk configuration files identically using either YAML or JSON.

Tip

When you are developing or testing new configuration files, launch a clean environment running the default application and deploy to that. Poorly formatted configuration files will cause a new environment launch to fail unrecoverably.

The `option_settings` section of a configuration file defines values for [configuration options \(p. 536\)](#). Configuration options let you configure your Elastic Beanstalk environment, the AWS resources in it, and the software that runs your application. Configuration files are only one of several ways to set configuration options.

The [Resources section \(p. 622\)](#) lets you further customize the resources in your application's environment, and define additional AWS resources beyond the functionality provided by configuration

options. You can add and configure any resources supported by AWS CloudFormation, which Elastic Beanstalk uses to create environments.

The other sections of a configuration file (`packages`, `sources`, `files`, `users`, `groups`, `commands`, `container_commands`, and `services`) let you configure the EC2 instances that are launched in your environment. Whenever a server is launched in your environment, Elastic Beanstalk runs the operations defined in these sections to prepare the operating system and storage system for your application.

For examples of commonly used `.ebextensions`, see the [Elastic Beanstalk Configuration Files Repository](#).

Requirements

- **Location** – Place all of your configuration files in a single folder, named `.ebextensions`, in the root of your source bundle. Folders starting with a dot can be hidden by file browsers, so make sure that the folder is added when you create your source bundle. See [Create an application source bundle \(p. 342\)](#) for instructions.
- **Naming** – Configuration files must have the `.config` file extension.
- **Formatting** – Configuration files must conform to YAML or JSON specifications.

When using YAML, always use spaces to indent keys at different nesting levels. For more information about YAML, see [YAML Ain't Markup Language \(YAML™\) Version 1.1](#).

- **Uniqueness** – Use each key only once in each configuration file.

Warning

If you use a key (for example, `option_settings`) twice in the same configuration file, one of the sections will be dropped. Combine duplicate sections into a single section, or place them in separate configuration files.

The process for deploying varies slightly depending on the client that you use to manage your environments. See the following sections for details:

- [Elastic Beanstalk console \(p. 543\)](#)
- [EB CLI \(p. 545\)](#)
- [AWS CLI \(p. 546\)](#)

Topics

- [Option settings \(p. 601\)](#)
- [Customizing software on Linux servers \(p. 603\)](#)
- [Customizing software on Windows servers \(p. 615\)](#)
- [Adding and customizing Elastic Beanstalk environment resources \(p. 622\)](#)

Option settings

You can use the `option_settings` key to modify the Elastic Beanstalk configuration and define variables that can be retrieved from your application using environment variables. Some namespaces allow you to extend the number of parameters, and specify the parameter names. For a list of namespaces and configuration options, see [Configuration options \(p. 536\)](#).

Option settings can also be applied directly to an environment during environment creation or an environment update. Settings applied directly to the environment override the settings for the same options in configuration files. If you remove settings from an environment's configuration, settings in configuration files will take effect. See [Precedence \(p. 537\)](#) for details.

Syntax

The standard syntax for option settings is an array of objects, each having a `namespace`, `option_name` and `value` key.

```
option_settings:
  - namespace: namespace
    option_name: option name
    value: option value
  - namespace: namespace
    option_name: option name
    value: option value
```

The `namespace` key is optional. If you do not specify a namespace, the default used is `aws:elasticbeanstalk:application:environment`:

```
option_settings:
  - option_name: option name
    value: option value
  - option_name: option name
    value: option value
```

Elastic Beanstalk also supports a shorthand syntax for option settings that lets you specify options as key-value pairs underneath the namespace:

```
option_settings:
  namespace:
    option name: option value
    option name: option value
```

Examples

The following examples set a Tomcat platform-specific option in the `aws:elasticbeanstalk:container:tomcat:jvmoptions` namespace and an environment property named `MYPARAMETER`.

In standard YAML format:

Example `.ebextensions/options.config`

```
option_settings:
  - namespace: aws:elasticbeanstalk:container:tomcat:jvmoptions
    option_name: Xmx
    value: 256m
  - option_name: MYPARAMETER
    value: parametervalue
```

In shorthand format:

Example `.ebextensions/options.config`

```
option_settings:
  aws:elasticbeanstalk:container:tomcat:jvmoptions:
    Xmx: 256m
  aws:elasticbeanstalk:application:environment:
    MYPARAMETER: parametervalue
```

In JSON:

Example `.ebextensions/options.config`

```
{
  "option_settings": [
    {
      "namespace": "aws:elasticbeanstalk:container:tomcat:jvmoptions",
      "option_name": "Xmx",
      "value": "256m"
    },
    {
      "option_name": "MYPARAMETER",
      "value": "parametervalue"
    }
  ]
}
```

Customizing software on Linux servers

You may want to customize and configure the software that your application depends on. You can add commands to be executed during instance provisioning; define Linux users and groups; and download or directly create files on your environment instances. These files might be either dependencies required by the application—for example, additional packages from the yum repository—or they might be configuration files such as a replacement for a proxy configuration file to override specific settings that are defaulted by Elastic Beanstalk.

This section describes the type of information you can include in a configuration file to customize the software on your EC2 instances running Linux. For general information about customizing and configuring your Elastic Beanstalk environments, see [Configuring Elastic Beanstalk environments](#) (p. 446). For information about customizing software on your EC2 instances running Windows, see [Customizing software on Windows servers](#) (p. 615).

Notes

- On Amazon Linux 2 platforms, instead of providing files and commands in `.ebextensions` configuration files, we highly recommend that you use *Buildfile*, *Procfile*, and *platform hooks* whenever possible to configure and run custom code on your environment instances during instance provisioning. For details about these mechanisms, see [the section called “Extending Linux platforms”](#) (p. 31).
- YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

Configuration files support the following keys that affect the Linux server your application runs on.

Keys

- [Packages](#) (p. 604)
- [Groups](#) (p. 605)
- [Users](#) (p. 605)
- [Sources](#) (p. 606)
- [Files](#) (p. 607)
- [Commands](#) (p. 608)
- [Services](#) (p. 610)
- [Container commands](#) (p. 611)
- [Example: Using custom amazon CloudWatch metrics](#) (p. 612)

Keys are processed in the order that they are listed here.

Watch your environment's [events \(p. 728\)](#) while developing and testing configuration files. Elastic Beanstalk ignores a configuration file that contains validation errors, like an invalid key, and doesn't process any of the other keys in the same file. When this happens, Elastic Beanstalk adds a warning event to the event log.

Packages

You can use the `packages` key to download and install prepackaged applications and components.

Syntax

```
packages:  
  name of package manager:  
    package name: version  
    ...  
  name of package manager:  
    package name: version  
    ...  
  ...
```

You can specify multiple packages under each package manager's key.

Supported package formats

Elastic Beanstalk currently supports the following package managers: yum, rubygems, python, and rpm. Packages are processed in the following order: rpm, yum, and then rubygems and python. There is no ordering between rubygems and python. Within each package manager, package installation order isn't guaranteed. Use a package manager supported by your operating system.

Note

Elastic Beanstalk supports two underlying package managers for Python, pip and easy_install. However, in the syntax of the configuration file, you must specify the package manager name as `python`. When you use a configuration file to specify a Python package manager, Elastic Beanstalk uses Python 2.7. If your application relies on a different version of Python, you can specify the packages to install in a `requirements.txt` file. For more information, see [Specifying dependencies using a requirements file \(p. 294\)](#).

Specifying versions

Within each package manager, each package is specified as a package name and a list of versions. The version can be a string, a list of versions, or an empty string or list. An empty string or list indicates that you want the latest version. For rpm manager, the version is specified as a path to a file on disk or a URL. Relative paths are not supported.

If you specify a version of a package, Elastic Beanstalk attempts to install that version even if a newer version of the package is already installed on the instance. If a newer version is already installed, the deployment fails. Some package managers support multiple versions, but others may not. Please check the documentation for your package manager for more information. If you do not specify a version and a version of the package is already installed, Elastic Beanstalk does not install a new version—it assumes that you want to keep and use the existing version.

Example snippet

The following snippet specifies a version URL for rpm, requests the latest version from yum, and version 0.10.2 of chef from rubygems.

```
packages:
  yum:
    libmemcached: []
    ruby-devel: []
    gcc: []
  rpm:
    epel: http://download.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm
  rubygems:
    chef: '0.10.2'
```

Groups

You can use the `groups` key to create Linux/UNIX groups and to assign group IDs. To create a group, add a new key-value pair that maps a new group name to an optional group ID. The `groups` key can contain one or more group names. The following table lists the available keys.

Syntax

```
groups:
  name of group: {}
  name of group:
    gid: "group id"
```

Options

`gid`

A group ID number.

If a group ID is specified, and the group already exists by name, the group creation will fail. If another group has the specified group ID, the operating system may reject the group creation.

Example snippet

The following snippet specifies a group named `groupOne` without assigning a group ID and a group named `groupTwo` that specified a group ID value of 45.

```
groups:
  groupOne: {}
  groupTwo:
    gid: "45"
```

Users

You can use the `users` key to create Linux/UNIX users on the EC2 instance.

Syntax

```
users:
  name of user:
    groups:
      - name of group
    uid: "id of the user"
    homeDir: "user's home directory"
```

Options

uid

A user ID. The creation process fails if the user name exists with a different user ID. If the user ID is already assigned to an existing user, the operating system may reject the creation request.

groups

A list of group names. The user is added to each group in the list.

homeDir

The user's home directory.

Users are created as noninteractive system users with a shell of `/sbin/nologin`. This is by design and cannot be modified.

Example snippet

```
users:
  myuser:
    groups:
      - group1
      - group2
    uid: "50"
    homeDir: "/tmp"
```

Sources

You can use the `sources` key to download an archive file from a public URL and unpack it in a target directory on the EC2 instance.

Syntax

```
sources:
  target directory: location of archive file
```

Supported formats

Supported formats are tar, tar+gzip, tar+bz2, and zip. You can reference external locations such as Amazon Simple Storage Service (Amazon S3) (e.g., `https://mybucket.s3.amazonaws.com/myobject`) as long as the URL is publicly accessible.

Example snippet

The following example downloads a public .zip file from an Amazon S3 bucket and unpacks it into `/etc/myapp`:

```
sources:
  /etc/myapp: https://mybucket.s3.amazonaws.com/myobject
```

Note

Multiple extractions should not reuse the same target path. Extracting another source to the same target path will replace rather than append to the contents.

Files

You can use the `files` key to create files on the EC2 instance. The content can be either inline in the configuration file, or the content can be pulled from a URL. The files are written to disk in lexicographic order.

You can use the `files` key to download private files from Amazon S3 by providing an instance profile for authorization.

If the file path you specify already exists on the instance, the existing file is retained with the extension `.bak` appended to its name.

Syntax

```
files:
  "target file location on disk":
    mode: "six-digit octal value"
    owner: name of owning user for file
    group: name of owning group for file
    source: URL
    authentication: authentication name:

  "target file location on disk":
    mode: "six-digit octal value"
    owner: name of owning user for file
    group: name of owning group for file
    content: |
      # this is my
      # file content
    encoding: encoding format
    authentication: authentication name:
```

Options

content

String content to add to the file. Specify either `content` or `source`, but not both.

source

URL of a file to download. Specify either `content` or `source`, but not both.

encoding

The encoding format of the string specified with the `content` option.

Valid values: `plain` | `base64`

group

Linux group that owns the file.

owner

Linux user that owns the file.

mode

A six-digit octal value representing the mode for this file. Not supported for Windows systems. Use the first three digits for symlinks and the last three digits for setting permissions. To create a symlink, specify `120xxx`, where `xxx` defines the permissions of the target file. To specify permissions for a file, use the last three digits, such as `000644`.

authentication

The name of a [AWS CloudFormation authentication method](#) to use. You can add authentication methods to the autoscaling group metadata with the Resources key. See below for an example.

Example snippet

```
files:
  "/home/ec2-user/myfile" :
    mode: "000755"
    owner: root
    group: root
    source: http://foo.bar/myfile

  "/home/ec2-user/myfile2" :
    mode: "000755"
    owner: root
    group: root
    content: |
      this is my
      file content
```

Example using a symlink. This creates a link `/tmp/myfile2.txt` that points at the existing file `/tmp/myfile1.txt`.

```
files:
  "/tmp/myfile2.txt" :
    mode: "120400"
    content: "/tmp/myfile1.txt"
```

The following example uses the Resources key to add an authentication method named `S3Auth` and uses it to download a private file from an Amazon S3 bucket:

```
Resources:
  AWSEBAutoScalingGroup:
    Metadata:
      AWS::CloudFormation::Authentication:
        S3Auth:
          type: "s3"
          buckets: ["elasticbeanstalk-us-west-2-123456789012"]
          roleName:
            Fn::GetOptionSetting:
              Namespace: "aws:autoscaling:launchconfiguration"
              OptionName: "IamInstanceProfile"
              DefaultValue: "aws-elasticbeanstalk-ec2-role"

files:
  "/tmp/data.json" :
    mode: "000755"
    owner: root
    group: root
    authentication: "S3Auth"
    source: https://elasticbeanstalk-us-west-2-123456789012.s3-us-west-2.amazonaws.com/
    data.json
```

Commands

You can use the `commands` key to execute commands on the EC2 instance. The commands run before the application and web server are set up and the application version file is extracted.

The specified commands run as the root user, and are processed in alphabetical order by name. By default, commands run in the root directory. To run commands from another directory, use the `cwd` option.

To troubleshoot issues with your commands, you can find their output in [instance logs \(p. 732\)](#).

Syntax

```
commands:
  command name:
    command: command to run
    cwd: working directory
    env:
      variable name: variable value
    test: conditions for command
    ignoreErrors: true
```

Options

command

Either an array ([block sequence collection](#) in YAML syntax) or a string specifying the command to run. Some important notes:

- If you use a string, you don't need to enclose the entire string in quotes. If you do use quotes, escape literal occurrences of the same type of quote.
- If you use an array, you don't need to escape space characters or enclose command parameters in quotes. Each array element is a single command argument. Don't use an array to specify multiple commands.

The following examples are all equivalent:

```
commands:
  command1:
    command: git commit -m "This is a comment."
  command2:
    command: "git commit -m \"This is a comment.\""
  command3:
    command: 'git commit -m "This is a comment."'
  command4:
    command:
      - git
      - commit
      - -m
      - This is a comment.
```

To specify multiple commands, use a [literal block scalar](#), as shown in the following example.

```
commands:
  command block:
    command: |
      git commit -m "This is a comment."
      git push
```

env

(Optional) Sets environment variables for the command. This property overwrites, rather than appends, the existing environment.

`cwd`

(Optional) The working directory. If not specified, commands run from the root directory (/).

`test`

(Optional) A command that must return the value `true` (exit code 0) in order for Elastic Beanstalk to process the command, such as a shell script, contained in the `command` key.

`ignoreErrors`

(Optional) A boolean value that determines if other commands should run if the command contained in the `command` key fails (returns a nonzero value). Set this value to `true` if you want to continue running commands even if the command fails. Set it to `false` if you want to stop running commands if the command fails. The default value is `false`.

Example snippet

The following example snippet runs a Python script.

```
commands:
  python_install:
    command: myscript.py
    cwd: /home/ec2-user
    env:
      myvarname: myvarvalue
    test: "[ -x /usr/bin/python ]"
```

Services

You can use the `services` key to define which services should be started or stopped when the instance is launched. The `services` key also allows you to specify dependencies on sources, packages, and files so that if a restart is needed due to files being installed, Elastic Beanstalk takes care of the service restart.

Syntax

```
services:
  sysvinit:
    name of service:
      enabled: "true"
      ensureRunning: "true"
      files:
        - "file name"
      sources:
        - "directory"
      packages:
        name of package manager:
          "package name[: version]"
      commands:
        - "name of command"
```

Options

`ensureRunning`

Set to `true` to ensure that the service is running after Elastic Beanstalk finishes.

Set to `false` to ensure that the service is not running after Elastic Beanstalk finishes.

Omit this key to make no changes to the service state.

`enabled`

Set to `true` to ensure that the service is started automatically upon boot.

Set to `false` to ensure that the service is not started automatically upon boot.

Omit this key to make no changes to this property.

`files`

A list of files. If Elastic Beanstalk changes one directly via the `files` block, the service is restarted.

`sources`

A list of directories. If Elastic Beanstalk expands an archive into one of these directories, the service is restarted.

`packages`

A map of the package manager to a list of package names. If Elastic Beanstalk installs or updates one of these packages, the service is restarted.

`commands`

A list of command names. If Elastic Beanstalk runs the specified command, the service is restarted.

Example snippet

The following is an example snippet:

```
services:
  sysvinit:
    myservice:
      enabled: true
      ensureRunning: true
```

Container commands

You can use the `container_commands` key to execute commands that affect your application source code. Container commands run after the application and web server have been set up and the application version archive has been extracted, but before the application version is deployed. Non-container commands and other customization operations are performed prior to the application source code being extracted.

The specified commands run as the root user, and are processed in alphabetical order by name. Container commands are run from the staging directory, where your source code is extracted prior to being deployed to the application server. Any changes you make to your source code in the staging directory with a container command will be included when the source is deployed to its final location.

To troubleshoot issues with your container commands, you can find their output in [instance logs \(p. 732\)](#).

You can use `leader_only` to only run the command on a single instance, or configure a `test` to only run the command when a test command evaluates to `true`. Leader-only container commands are only executed during environment creation and deployments, while other commands and server customization operations are performed every time an instance is provisioned or updated. Leader-only container commands are not executed due to launch configuration changes, such as a change in the AMI Id or instance type.

Syntax

```
container_commands:  
  name_of_container_command:  
    command: "command to run"  
    leader_only: true  
  name_of_container_command:  
    command: "command to run"
```

Options

command

A string or array of strings to run.

env

(Optional) Set environment variables prior to running the command, overriding any existing value.

cwd

(Optional) The working directory. By default, this is the staging directory of the unzipped application.

leader_only

(Optional) Only run the command on a single instance chosen by Elastic Beanstalk. Leader-only container commands are run before other container commands. A command can be leader-only or have a `test`, but not both (`leader_only` takes precedence).

test

(Optional) Run a test command that must return the `true` in order to run the container command. A command can be leader-only or have a `test`, but not both (`leader_only` takes precedence).

ignoreErrors

(Optional) Do not fail deployments if the container command returns a value other than 0 (success). Set to `true` to enable.

Example snippet

The following is an example snippet.

```
container_commands:  
  collectstatic:  
    command: "django-admin.py collectstatic --noinput"  
  01syncdb:  
    command: "django-admin.py syncdb --noinput"  
    leader_only: true  
  02migrate:  
    command: "django-admin.py migrate"  
    leader_only: true  
  99customize:  
    command: "scripts/customize.sh"
```

Example: Using custom amazon CloudWatch metrics

Amazon CloudWatch is a web service that enables you to monitor, manage, and publish various metrics, as well as configure alarm actions based on data from metrics. You can define custom metrics for

your own use, and Elastic Beanstalk will push those metrics to Amazon CloudWatch. Once Amazon CloudWatch contains your custom metrics, you can view those in the Amazon CloudWatch console.

The Amazon CloudWatch Monitoring Scripts for Linux are available to demonstrate how to produce and consume Amazon CloudWatch custom metrics. The scripts comprise a fully functional example that reports memory, swap, and disk space utilization metrics for an Amazon Elastic Compute Cloud (Amazon EC2) Linux instance. For more information about the Amazon CloudWatch Monitoring Scripts, go to [Amazon CloudWatch Monitoring Scripts for Linux](#) in the *Amazon CloudWatch Developer Guide*.

Note

Elastic Beanstalk [Enhanced Health Reporting](#) (p. 691) has native support for publishing a wide range of instance and environment metrics to CloudWatch. See [Publishing Amazon CloudWatch custom metrics for an environment](#) (p. 714) for details.

Topics

- [.Ebextensions configuration file](#) (p. 613)
- [Permissions](#) (p. 614)
- [Viewing metrics in the CloudWatch console](#) (p. 614)

.Ebextensions configuration file

This example uses commands and option settings in an .ebextensions configuration file to download, install, and run monitoring scripts provided by Amazon CloudWatch.

To use this sample, save it to a file named `cloudwatch.config` in a directory named `.ebextensions` at the top level of your project directory, then deploy your application using the Elastic Beanstalk console (include the `.ebextensions` directory in your [source bundle](#) (p. 342)) or the [EB CLI](#) (p. 852).

For more information about configuration files, see [Advanced environment customization with configuration files \(.ebextensions\)](#) (p. 600).

.ebextensions/cloudwatch.config

```
packages:
  yum:
    perl-DateTime: []
    perl-Sys-Syslog: []
    perl-LWP-Protocol-https: []
    perl-Switch: []
    perl-URI: []
    perl-Bundle-LWP: []

sources:
  /opt/cloudwatch: https://aws-cloudwatch.s3.amazonaws.com/downloads/
  CloudWatchMonitoringScripts-1.2.1.zip

container_commands:
  01-setupcron:
    command: |
      echo '*/*5 * * * * root perl /opt/cloudwatch/aws-scripts-mon/mon-put-instance-data.pl
      `{"Fn::GetOptionSetting" : { "OptionName" : "CloudWatchMetrics", "DefaultValue" : "--mem-
      util --disk-space-util --disk-path=/" }}` >> /var/log/cwpump.log 2>&1' > /etc/cron.d/cwpump
  02-changeperm:
    command: chmod 644 /etc/cron.d/cwpump
  03-changeperm:
    command: chmod u+x /opt/cloudwatch/aws-scripts-mon/mon-put-instance-data.pl

option_settings:
  "aws:autoscaling:launchconfiguration" :
```

```
IamInstanceProfile : "aws-elasticbeanstalk-ec2-role"  
"aws:elasticbeanstalk:customoption" :  
  CloudWatchMetrics : "--mem-util --mem-used --mem-avail --disk-space-util --disk-space-  
used --disk-space-avail --disk-path=/ --auto-scaling"
```

After you verify the configuration file works, you can conserve disk usage by changing the command redirect from a log file (>> /var/log/cwpump.log 2>&1) to /dev/null (> /dev/null).

Permissions

In order to publish custom Amazon CloudWatch metrics, the instances in your environment need permission to use CloudWatch. You can grant permissions to your environment's instances by adding them to the environment's [instance profile \(p. 21\)](#). You can add permissions to the instance profile before or after deploying your application.

To grant permissions to publish CloudWatch metrics

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**.
3. Choose your environment's instance profile role. By default, when you create an environment with the Elastic Beanstalk console or [EB CLI \(p. 852\)](#), this is `aws-elasticbeanstalk-ec2-role`.
4. Choose the **Permissions** tab.
5. Under **Inline Policies**, in the **Permissions** section, choose **Create Role Policy**.
6. Choose **Custom Policy**, and then choose **Select**.
7. Complete the following fields, and then choose **Apply Policy**:

Policy Name

The name of the policy.

Policy Document

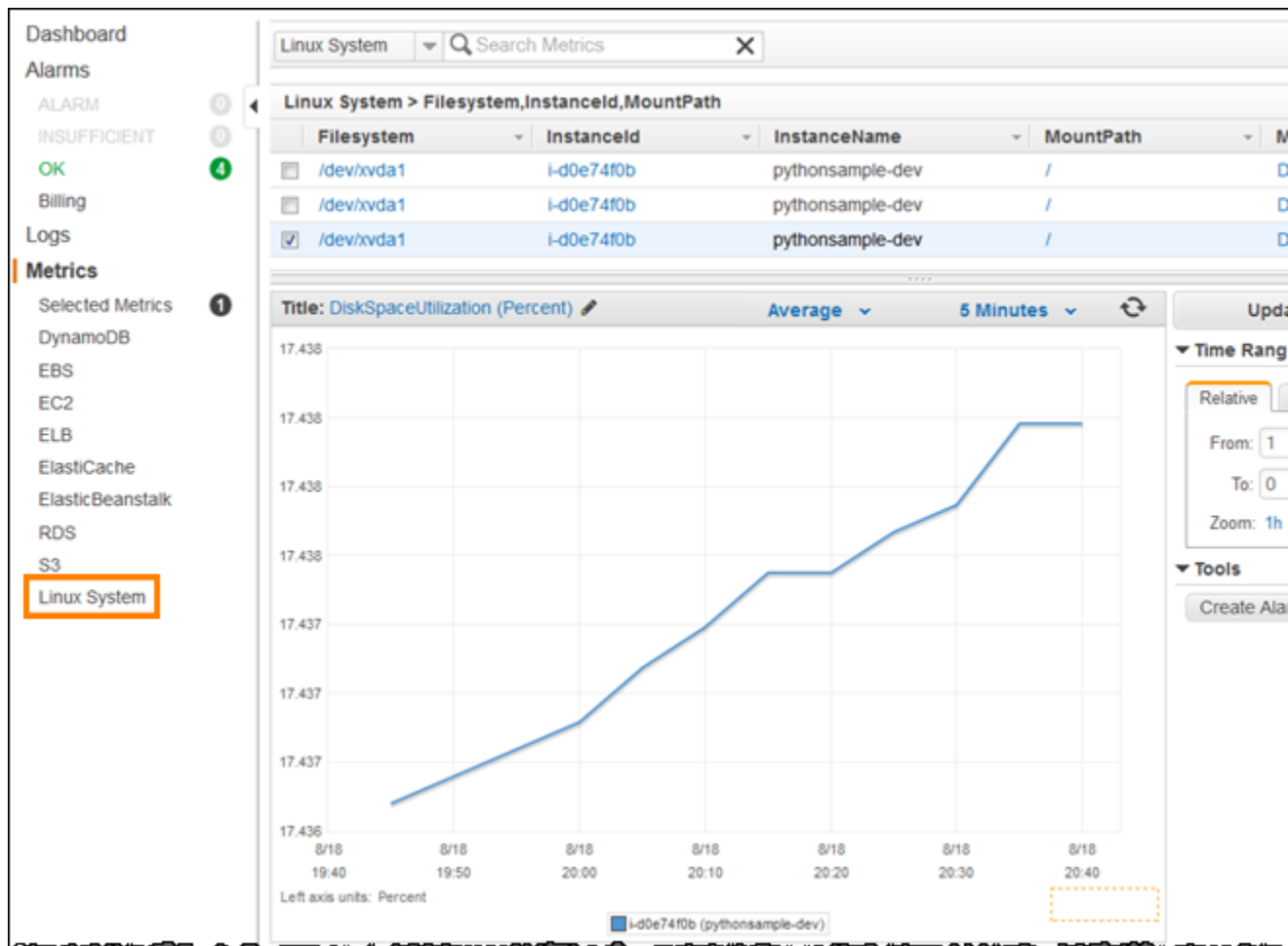
Copy and paste the following text into the policy document:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Action": [  
        "cloudwatch:PutMetricData",  
        "ec2:DescribeTags"  
      ],  
      "Effect": "Allow",  
      "Resource": [  
        "*" ]  
    }  
  ]  
}
```

For more information about managing policies, see [Working with Policies](#) in the *IAM User Guide*.

Viewing metrics in the CloudWatch console

After deploying the CloudWatch configuration file to your environment, check the [Amazon CloudWatch console](#) to view your metrics. Custom metrics will have the prefix **Linux System**.



Customizing software on Windows servers

You may want to customize and configure the software that your application depends on. These files could be either dependencies required by the application—for example, additional packages or services that need to be run. For general information on customizing and configuring your Elastic Beanstalk environments, see [Configuring Elastic Beanstalk environments \(p. 446\)](#).

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

Configuration files support the following keys that affect the Windows server on which your application runs.

Keys

- [Packages \(p. 616\)](#)
- [Sources \(p. 616\)](#)
- [Files \(p. 617\)](#)
- [Commands \(p. 618\)](#)
- [Services \(p. 619\)](#)

- [Container commands \(p. 620\)](#)

Keys are processed in the order that they are listed here.

Note

Older (non-versioned) .NET platform versions do not process configuration files in the correct order. Learn more at [Migrating across major versions of the Elastic Beanstalk Windows server platform \(p. 143\)](#).

Watch your environment's [events \(p. 728\)](#) while developing and testing configuration files. Elastic Beanstalk ignores a configuration file that contains validation errors, like an invalid key, and doesn't process any of the other keys in the same file. When this happens, Elastic Beanstalk adds a warning event to the event log.

Packages

Use the `packages` key to download and install prepackaged applications and components.

In Windows environments, Elastic Beanstalk supports downloading and installing MSI packages. (Linux environments support additional package managers. For details, see [Packages \(p. 604\)](#) on the *Customizing Software on Linux Servers* page.)

You can reference any external location, such as an Amazon Simple Storage Service (Amazon S3) object, as long as the URL is publicly accessible.

If you specify several `msi:` packages, their installation order isn't guaranteed.

Syntax

Specify a name of your choice as the package name, and a URL to an MSI file location as the value. You can specify multiple packages under the `msi:` key.

```
packages:
  msi:
    package name: package url
    ...
```

Examples

The following example specifies a URL to download **mysql** from `https://dev.mysql.com/`.

```
packages:
  msi:
    mysql: https://dev.mysql.com/get/Downloads/Connector-Net/mysql-connector-net-8.0.11.msi
```

The following example specifies an Amazon S3 object as the MSI file location.

```
packages:
  msi:
    mymsi: https://mybucket.s3.amazonaws.com/myobject.msi
```

Sources

Use the `sources` key to download an archive file from a public URL and unpack it in a target directory on the EC2 instance.

Syntax

```
sources:  
  target directory: location of archive file
```

Supported formats

In Windows environments, Elastic Beanstalk supports the .zip format. (Linux environments support additional formats. For details, see [Sources \(p. 606\)](#) on the *Customizing Software on Linux Servers* page.)

You can reference any external location, such as an Amazon Simple Storage Service (Amazon S3) object, as long as the URL is publicly accessible.

Example

The following example downloads a public .zip file from an Amazon S3 bucket and unpacks it into `c:/myproject/myapp`.

```
sources:  
  "c:/myproject/myapp": https://mybucket.s3.amazonaws.com/myobject.zip
```

Files

Use the `files` key to create files on the EC2 instance. The content can be either inline in the configuration file, or from a URL. The files are written to disk in lexicographic order. To download private files from Amazon S3, provide an instance profile for authorization.

Syntax

```
files:  
  "target file location on disk":  
    source: URL  
    authentication: authentication name:  
  
  "target file location on disk":  
    content: |  
      this is my content  
    encoding: encoding format
```

Options

`content`

(Optional) A string.

`source`

(Optional) The URL from which the file is loaded. This option cannot be specified with the `content` key.

`encoding`

(Optional) The encoding format. This option is only used for a provided `content` key value. The default value is `plain`.

Valid values: `plain` | `base64`

authentication

(Optional) The name of a [AWS CloudFormation authentication method](#) to use. You can add authentication methods to the autoscaling group metadata with the Resources key.

Examples

The following example shows the two ways to provide file content: from a URL, or inline in the configuration file.

```
files:
  "c:\\targetdirectory\\targetfile.txt":
    source: http://foo.bar/myfile

  "c:/targetdirectory/targetfile.txt":
    content: |
      # this is my file
      # with content
```

Note

If you use a backslash (\) in your file path, you must precede that with another backslash (the escape character) as shown in the previous example.

The following example uses the Resources key to add an authentication method named S3Auth and uses it to download a private file from an Amazon S3 bucket:

```
files:
  "c:\\targetdirectory\\targetfile.zip":
    source: https://elasticbeanstalk-us-east-2-123456789012.s3.amazonaws.com/prefix/
    myfile.zip
    authentication: S3Auth

Resources:
  AWSEBAutoScalingGroup:
    Metadata:
      AWS::CloudFormation::Authentication:
        S3Auth:
          type: "s3"
          buckets: ["elasticbeanstalk-us-east-2-123456789012"]
          roleName:
            "Fn::GetOptionSetting":
              Namespace: "aws:autoscaling:launchconfiguration"
              OptionName: "IamInstanceProfile"
              DefaultValue: "aws-elasticbeanstalk-ec2-role"
```

Commands

Use the `commands` key to execute commands on the EC2 instance. The commands are processed in alphabetical order by name, and they run before the application and web server are set up and the application version file is extracted.

The specified commands run as the Administrator user.

To troubleshoot issues with your commands, you can find their output in [instance logs \(p. 732\)](#).

Syntax

```
commands:
  command name:
```

```
command: command to run
```

Options

`command`

Either an array or a string specifying the command to run. If you use an array, you don't need to escape space characters or enclose command parameters in quotation marks.

`cwd`

(Optional) The working directory. By default, Elastic Beanstalk attempts to find the directory location of your project. If not found, it uses `c:\Windows\System32` as the default.

`env`

(Optional) Sets environment variables for the command. This property overwrites, rather than appends, the existing environment.

`ignoreErrors`

(Optional) A Boolean value that determines if other commands should run if the command contained in the `command` key fails (returns a nonzero value). Set this value to `true` if you want to continue running commands even if the command fails. Set it to `false` if you want to stop running commands if the command fails. The default value is `false`.

`test`

(Optional) A command that must return the value `true` (exit code 0) in order for Elastic Beanstalk to process the command contained in the `command` key.

`waitAfterCompletion`

(Optional) Seconds to wait after the command completes before running the next command. If the system requires a reboot after the command completes, the system reboots after the specified number of seconds elapses. If the system reboots as a result of a command, Elastic Beanstalk will recover to the point after the command in the configuration file. The default value is **60** seconds. You can also specify **forever**, but the system must reboot before you can run another command.

Example

The following example saves the output of the `set` command to the specified file. If there is a subsequent command, Elastic Beanstalk runs that command immediately after this command completes. If this command requires a reboot, Elastic Beanstalk reboots the instance immediately after the command completes.

```
commands:  
  test:  
    command: set > c:\\myapp\\set.txt  
    waitAfterCompletion: 0
```

Services

Use the `services` key to define which services should be started or stopped when the instance is launched. The `services` key also enables you to specify dependencies on sources, packages, and files so that if a restart is needed due to files being installed, Elastic Beanstalk takes care of the service restart.

Syntax

```
services:
```

```
windows:
  name of service:
    files:
      - "file name"
    sources:
      - "directory"
    packages:
      name of package manager:
        "package name[: version]"
    commands:
      - "name of command"
```

Options

ensureRunning

(Optional) Set to `true` to ensure that the service is running after Elastic Beanstalk finishes.

Set to `false` to ensure that the service is not running after Elastic Beanstalk finishes.

Omit this key to make no changes to the service state.

enabled

(Optional) Set to `true` to ensure that the service is started automatically upon boot.

Set to `false` to ensure that the service is not started automatically upon boot.

Omit this key to make no changes to this property.

files

A list of files. If Elastic Beanstalk changes one directly via the files block, the service is restarted.

sources

A list of directories. If Elastic Beanstalk expands an archive into one of these directories, the service is restarted.

packages

A map of the package manager to a list of package names. If Elastic Beanstalk installs or updates one of these packages, the service is restarted.

commands

A list of command names. If Elastic Beanstalk runs the specified command, the service is restarted.

Example

```
services:
  windows:
    myservice:
      enabled: true
      ensureRunning: true
```

Container commands

Use the `container_commands` key to execute commands that affect your application source code. Container commands run after the application and web server have been set up and the application version archive has been extracted, but before the application version is deployed. Non-container commands and other customization operations are performed prior to the application source code being extracted.

Container commands are run from the staging directory, where your source code is extracted prior to being deployed to the application server. Any changes you make to your source code in the staging directory with a container command will be included when the source is deployed to its final location.

To troubleshoot issues with your container commands, you can find their output in [instance logs](#) (p. 732).

Use the `leader_only` option to only run the command on a single instance, or configure a `test` to only run the command when a test command evaluates to `true`. Leader-only container commands are only executed during environment creation and deployments, while other commands and server customization operations are performed every time an instance is provisioned or updated. Leader-only container commands are not executed due to launch configuration changes, such as a change in the AMI Id or instance type.

Syntax

```
container_commands:  
  name_of_container_command:  
    command: command_to_run
```

Options

command

A string or array of strings to run.

env

(Optional) Set environment variables prior to running the command, overriding any existing value.

cwd

(Optional) The working directory. By default, this is the staging directory of the unzipped application.

leader_only

(Optional) Only run the command on a single instance chosen by Elastic Beanstalk. Leader-only container commands are run before other container commands. A command can be leader-only or have a `test`, but not both (`leader_only` takes precedence).

test

(Optional) Run a test command that must return the `true` in order to run the container command. A command can be leader-only or have a `test`, but not both (`leader_only` takes precedence).

ignoreErrors

(Optional) Do not fail deployments if the container command returns a value other than 0 (success). Set to `true` to enable.

waitAfterCompletion

(Optional) Seconds to wait after the command completes before running the next command. If the system requires a reboot after the command completes, the system reboots after the specified number of seconds elapses. If the system reboots as a result of a command, Elastic Beanstalk will recover to the point after the command in the configuration file. The default value is **60** seconds. You can also specify **forever**, but the system must reboot before you can run another command.

Example

The following example saves the output of the `set` command to the specified file. Elastic Beanstalk runs the command on one instance, and reboots the instance immediately after the command completes.

```
container_commands:
  foo:
    command: set > c:\\myapp\\set.txt
    leader_only: true
    waitAfterCompletion: 0
```

Adding and customizing Elastic Beanstalk environment resources

You may also want to customize your environment resources that are part of your Elastic Beanstalk environment. For example, you may want to add an Amazon SQS queue and an alarm on queue depth, or you might want to add an Amazon ElastiCache cluster. You can easily customize your environment at the same time that you deploy your application version by including a configuration file with your source bundle.

You can use the `Resources` key in a [configuration file \(p. 600\)](#) to create and customize AWS resources in your environment. Resources defined in configuration files are added to the AWS CloudFormation template used to launch your environment. All AWS CloudFormation [resources types](#) are supported.

For example, the following configuration file adds an Auto Scaling lifecycle hook to the default Auto Scaling group created by Elastic Beanstalk:

~/my-app/.ebextensions/as-hook.config

```
Resources:
  hookrole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument: {
        "Version": "2012-10-17",
        "Statement": [ {
          "Effect": "Allow",
          "Principal": {
            "Service": [ "autoscaling.amazonaws.com" ]
          },
          "Action": [ "sts:AssumeRole" ]
        } ]
      }
    Policies: [ {
      "PolicyName": "SNS",
      "PolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [{
          "Effect": "Allow",
          "Resource": "*",
          "Action": [
            "sqs:SendMessage",
            "sqs:GetQueueUrl",
            "sns:Publish"
          ]
        } ]
      }
    ]
  hooktopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint: "my-email@example.com"
          Protocol: email
  lifecyclehook:
```

```
Type: AWS::AutoScaling::LifecycleHook
Properties:
  AutoScalingGroupName: { "Ref" : "AWSEBAutoScalingGroup" }
  LifecycleTransition: autoscaling:EC2_INSTANCE_TERMINATING
  NotificationTargetARN: { "Ref" : "hooktopic" }
  RoleARN: { "Fn::GetAtt" : [ "hookrole", "Arn" ] }
```

This example defines three resources, `hookrole`, `hooktopic` and `lifecyclehook`. The first two resources are an IAM role, which grants Amazon EC2 Auto Scaling permission to publish messages to Amazon SNS, and an SNS topic, which relays messages from the Auto Scaling group to an email address. Elastic Beanstalk creates these resources with the specified properties and types.

The final resource, `lifecyclehook`, is the lifecycle hook itself:

```
lifecyclehook:
  Type: AWS::AutoScaling::LifecycleHook
  Properties:
    AutoScalingGroupName: { "Ref" : "AWSEBAutoScalingGroup" }
    LifecycleTransition: autoscaling:EC2_INSTANCE_TERMINATING
    NotificationTargetARN: { "Ref" : "hooktopic" }
    RoleARN: { "Fn::GetAtt" : [ "hookrole", "Arn" ] }
```

The lifecycle hook definition uses two [functions \(p. 626\)](#) to populate values for the hook's properties. `{ "Ref" : "AWSEBAutoScalingGroup" }` retrieves the name of the Auto Scaling group created by Elastic Beanstalk for the environment. `AWSEBAutoScalingGroup` is one of the standard [resource names \(p. 623\)](#) provided by Elastic Beanstalk.

For `AWS::IAM::Role`, `Ref` only returns the name of the role, not the ARN. To get the ARN for the `RoleARN` parameter, you use another intrinsic function, `Fn::GetAtt` instead, which can get any attribute from a resource. `RoleARN: { "Fn::GetAtt" : ["hookrole", "Arn"] }` gets the `Arn` attribute from the `hookrole` resource.

`{ "Ref" : "hooktopic" }` gets the ARN of the Amazon SNS topic created earlier in the configuration file. The value returned by `Ref` varies per resource type and can be found in the AWS CloudFormation User Guide [topic for the AWS::SNS::Topic resource type](#).

Modifying the resources that Elastic Beanstalk creates for your environment

The resources that Elastic Beanstalk creates for your environment have names. You can use these names to get information about the resources with a [function \(p. 626\)](#), or modify properties on the resources to customize their behavior.

Web server environments have the following resources.

Web server environments

- `AWSEBAutoScalingGroup` ([AWS::AutoScaling::AutoScalingGroup](#)) – The Auto Scaling group attached to your environment.
- One of the following two resources.
 - `AWSEBAutoScalingLaunchConfiguration` ([AWS::AutoScaling::LaunchConfiguration](#)) – The launch configuration attached to your environment's Auto Scaling group.
 - `AWSEBEC2LaunchTemplate` ([AWS::EC2::LaunchTemplate](#)) – The Amazon EC2 launch template used by your environment's Auto Scaling group.

Note

If your environment uses functionality that requires Amazon EC2 launch templates, and your user policy lacks the required permissions, creating or updating the environment might fail.

Use the [AWSElasticBeanstalkFullAccess managed user policy \(p. 775\)](#), or add the required permissions to your [custom policy \(p. 777\)](#).

- `AWSEBEnvironmentName` ([AWS::ElasticBeanstalk::Environment](#)) – Your environment.
- `AWSEBSecurityGroup` ([AWS::EC2::SecurityGroup](#)) – The security group attached to your Auto Scaling group.
- `AWSEBRDSDatabase` ([AWS::RDS::DBInstance](#)) – The Amazon RDS DB instance attached to your environment (if applicable).

In a load balanced environment, you can access additional resources related to the load balancer. Classic load balancers have a resource for the load balancer and one for the security group attached to it. Application and network load balancers have additional resources for the load balancer's default listener, listener rule, and target group.

Load balanced environments

- `AWSEBLoadBalancer` ([AWS::ElasticLoadBalancing::LoadBalancer](#)) – Your environment's classic load balancer.
- `AWSEBV2LoadBalancer` ([AWS::ElasticLoadBalancingV2::LoadBalancer](#)) – Your environment's application or network load balancer.
- `AWSEBLoadBalancerSecurityGroup` ([AWS::EC2::SecurityGroup](#)) – In a custom [Amazon Virtual Private Cloud](#) (Amazon VPC) only, the name of the security group that Elastic Beanstalk creates for the load balancer. In a default VPC or EC2 classic, Elastic Load Balancing assigns a default security group to the load balancer.
- `AWSEBV2LoadBalancerListener` ([AWS::ElasticLoadBalancingV2::Listener](#)) – A listener that allows the load balancer to check for connection requests and forward them to one or more target groups.
- `AWSEBV2LoadBalancerListenerRule` ([AWS::ElasticLoadBalancingV2::ListenerRule](#)) – Defines which requests an Elastic Load Balancing listener takes action on and the action that it takes.
- `AWSEBV2LoadBalancerTargetGroup` ([AWS::ElasticLoadBalancingV2::TargetGroup](#)) – An Elastic Load Balancing target group that routes requests to one or more registered targets, such as Amazon EC2 instances.

Worker environments have resources for the SQS queue that buffers incoming requests, and a Amazon DynamoDB table that the instances use for leader election.

Worker environments

- `AWSEBWorkerQueue` ([AWS::SQS::Queue](#)) – The Amazon SQS queue from which the daemon pulls requests that need to be processed.
- `AWSEBWorkerDeadLetterQueue` ([AWS::SQS::Queue](#)) – The Amazon SQS queue that stores messages that cannot be delivered or otherwise were not successfully processed by the daemon.
- `AWSEBWorkerCronLeaderRegistry` ([AWS::DynamoDB::Table](#)) – The Amazon DynamoDB table that is the internal registry used by the daemon for periodic tasks.

Other AWS CloudFormation template keys

We've already introduced configuration file keys from AWS CloudFormation such as `Resources`, `files`, and `packages`. Elastic Beanstalk adds the contents of configurations files to the AWS CloudFormation template that supports your environment, so you can use other AWS CloudFormation sections to perform advanced tasks in your configuration files.

Keys

- [Parameters \(p. 625\)](#)

- [Outputs \(p. 625\)](#)
- [Mappings \(p. 626\)](#)

Parameters

Parameters are an alternative to Elastic Beanstalk's own [custom options \(p. 599\)](#) that you can use to define values that you use in other places in your configuration files. Like custom options, you can use parameters to gather user configurable values in one place. Unlike custom options, you can not use Elastic Beanstalk's API to set parameter values, and the number of parameters you can define in a template is limited by AWS CloudFormation.

One reason you might want to use parameters is to make your configuration files double as AWS CloudFormation templates. If you use parameters instead of custom options, you can use the configuration file to create the same resource in AWS CloudFormation as its own stack. For example, you could have a configuration file that adds an Amazon EFS file system to your environment for testing, and then use the same file to create an independent file system that isn't tied to your environment's lifecycle for production use.

The following example shows the use of parameters to gather user-configurable values at the top of a configuration file.

Example [Loadbalancer-accesslogs-existingbucket.config](#) – Parameters

```
Parameters:
  bucket:
    Type: String
    Description: "Name of the Amazon S3 bucket in which to store load balancer logs"
    Default: "my-bucket"
  bucketprefix:
    Type: String
    Description: "Optional prefix. Can't start or end with a /, or contain the word
AWSLogs"
    Default: ""
```

Outputs

You can use an `Outputs` block to export information about created resources to AWS CloudFormation. You can then use the `Fn::ImportValue` function to pull the value into a AWS CloudFormation template outside of Elastic Beanstalk.

The following example creates an Amazon SNS topic and exports its ARN to AWS CloudFormation with the name `NotificationTopicArn`.

Example [sns-topic.config](#)

```
Resources:
  NotificationTopic:
    Type: AWS::SNS::Topic

Outputs:
  NotificationTopicArn:
    Description: Notification topic ARN
    Value: { "Ref" : "NotificationTopic" }
    Export:
      Name: NotificationTopicArn
```

In a configuration file for a different environment, or a AWS CloudFormation template outside of Elastic Beanstalk, you can use the `Fn::ImportValue` function to get the exported ARN. This example assigns the exported value to an environment property named `TOPIC_ARN`.

Example env.config

```
option_settings:
  aws:elasticbeanstalk:application:environment:
    TOPIC_ARN: '{ Fn::ImportValue : "NotificationTopicArn" }`'
```

Mappings

You can use a mapping to store key-value pairs organized by namespace. A mapping can help you organize values that you use throughout your configs, or change a parameter value depending on another value. For example, the following configuration sets the value of an account ID parameter based on the current region.

Example `Loadbalancer-accesslogs-newbucket.config` – Mappings

```
Mappings:
  Region2ELBAccountId:
    us-east-1:
      AccountId: "111122223333"
    us-west-2:
      AccountId: "444455556666"
    us-west-1:
      AccountId: "123456789012"
    eu-west-1:
      AccountId: "777788889999"
  ...
  Principal:
    AWS:
      ? "Fn::FindInMap"
      :
        - Region2ELBAccountId
        -
          Ref: "AWS::Region"
        - AccountId
```

Functions

You can use functions in your configuration files to populate values for resource properties with information from other resources or from Elastic Beanstalk configuration option settings. Elastic Beanstalk supports AWS CloudFormation functions (`Ref`, `Fn::GetAtt`, `Fn::Join`), and one Elastic Beanstalk-specific function, `Fn::GetOptionSetting`.

Functions

- [Ref \(p. 626\)](#)
- [Fn::GetAtt \(p. 627\)](#)
- [Fn::Join \(p. 627\)](#)
- [Fn::GetOptionSetting \(p. 627\)](#)

Ref

Use `Ref` to retrieve the default string representation of an AWS resource. The value returned by `Ref` depends on the resource type, and sometimes depends on other factors as well. For example, a security group (`AWS::EC2::SecurityGroup`) returns either the name or ID of the security group, depending on if the security group is in a default [Amazon Virtual Private Cloud](#) (Amazon VPC), EC2 classic, or a custom VPC.

```
{ "Ref" : "resource name" }
```

Note

For details on each resource type, including the return value(s) of `Ref`, see [AWS Resource Types Reference](#) in the *AWS CloudFormation User Guide*.

From the sample [Auto Scaling lifecycle hook \(p. 622\)](#):

```
Resources:
  lifecyclehook:
    Type: AWS::AutoScaling::LifecycleHook
    Properties:
      AutoScalingGroupName: { "Ref" : "AWSEBAutoScalingGroup" }
```

You can also use `Ref` to retrieve the value of a AWS CloudFormation parameter defined elsewhere in the same file or in a different configuration file.

Fn::GetAtt

Use `Fn::GetAtt` to retrieve the value of an attribute on an AWS resource.

```
{ "Fn::GetAtt" : [ "resource name", "attribute name" ] }
```

From the sample [Auto Scaling lifecycle hook \(p. 622\)](#):

```
Resources:
  lifecyclehook:
    Type: AWS::AutoScaling::LifecycleHook
    Properties:
      RoleARN: { "Fn::GetAtt" : [ "hookrole", "Arn" ] }
```

See [Fn::GetAtt](#) for more information.

Fn::Join

Use `Fn::Join` to combine strings with a delimiter. The strings can be hard-coded or use the output from `Fn::GetAtt` or `Ref`.

```
{ "Fn::Join" : [ "delimiter", [ "string1", "string2" ] ] }
```

See [Fn::Join](#) for more information.

Fn::GetOptionSetting

Use `Fn::GetOptionSetting` to retrieve the value of a [configuration option \(p. 536\)](#) setting applied to the environment.

```
"Fn::GetOptionSetting":
  Namespace: "namespace"
  OptionName: "option name"
  DefaultValue: "default value"
```

From the [storing private keys \(p. 682\)](#) example:

```
Resources:
  AWSEBAutoScalingGroup:
    Metadata:
      AWS::CloudFormation::Authentication:
        S3Auth:
          type: "s3"
```

```
buckets: ["elasticbeanstalk-us-west-2-123456789012"]
roleName:
  "Fn::GetOptionSetting":
    Namespace: "aws:autoscaling:launchconfiguration"
    OptionName: "IamInstanceProfile"
    DefaultValue: "aws-elasticbeanstalk-ec2-role"
```

Custom resource examples

The following is a list of example configuration files that you can use to customize your Elastic Beanstalk environments:

- [DynamoDB, CloudWatch, and SNS](#)
- [Elastic Load Balancing and CloudWatch](#)
- [ElastiCache](#)
- [RDS and CloudWatch](#)
- [SQS, SNS, and CloudWatch](#)

Subtopics of this page provide some extended examples for adding and configuring custom resources in an Elastic Beanstalk environment.

Examples

- [Example: ElastiCache \(p. 628\)](#)
- [Example: SQS, CloudWatch, and SNS \(p. 634\)](#)
- [Example: DynamoDB, CloudWatch, and SNS \(p. 636\)](#)

Example: ElastiCache

The following samples add an Amazon ElastiCache cluster to EC2-Classic and EC2-VPC (both default and custom [Amazon Virtual Private Cloud](#) (Amazon VPC)) platforms. For more information about these platforms and how you can determine which ones EC2 supports for your region and your AWS account, see <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-supported-platforms.html>. Then refer to the section in this topic that applies to your platform.

- [EC2-classic platforms \(p. 628\)](#)
- [EC2-VPC \(default\) \(p. 630\)](#)
- [EC2-VPC \(custom\) \(p. 632\)](#)

EC2-classic platforms

This sample adds an Amazon ElastiCache cluster to an environment with instances launched into the EC2-Classic platform. All of the properties that are listed in this example are the minimum required properties that must be set for each resource type. You can download the example at [ElastiCache example](#).

Note

This example creates AWS resources, which you might be charged for. For more information about AWS pricing, see <https://aws.amazon.com/pricing/>. Some services are part of the AWS Free Usage Tier. If you are a new customer, you can test drive these services for free. See <https://aws.amazon.com/free/> for more information.

To use this example, do the following:

1. Create an `.ebextensions` (p. 600) directory in the top-level directory of your source bundle.

2. Create two configuration files with the `.config` extension and place them in your `.ebextensions` directory. One configuration file defines the resources, and the other configuration file defines the options.
3. Deploy your application to Elastic Beanstalk.

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

Create a configuration file (e.g., `elasticache.config`) that defines the resources. In this example, we create the `ElastiCache` cluster by specifying the name of the `ElastiCache` cluster resource (`MyElastiCache`), declaring its type, and then configuring the properties for the cluster. The example references the name of the `ElastiCache` security group resource that gets created and defined in this configuration file. Next, we create an `ElastiCache` security group. We define the name for this resource, declare its type, and add a description for the security group. Finally, we set the ingress rules for the `ElastiCache` security group to allow access only from instances inside the `ElastiCache` security group (`MyCacheSecurityGroup`) and the `Elastic Beanstalk` security group (`AWSEBSecurityGroup`). The parameter name, `AWSEBSecurityGroup`, is a fixed resource name provided by Elastic Beanstalk. You must add `AWSEBSecurityGroup` to your `ElastiCache` security group ingress rules in order for your `Elastic Beanstalk` application to connect to the instances in your `ElastiCache` cluster.

```
#This sample requires you to create a separate configuration file that defines the custom
option settings for CacheCluster properties.

Resources:
  MyElastiCache:
    Type: AWS::ElastiCache::CacheCluster
    Properties:
      CacheNodeType:
        Fn::GetOptionSetting:
          OptionName : CacheNodeType
          DefaultValue: cache.m1.small
      NumCacheNodes:
        Fn::GetOptionSetting:
          OptionName : NumCacheNodes
          DefaultValue: 1
      Engine:
        Fn::GetOptionSetting:
          OptionName : Engine
          DefaultValue: memcached
      CacheSecurityGroupNames:
        - Ref: MyCacheSecurityGroup
  MyCacheSecurityGroup:
    Type: AWS::ElastiCache::SecurityGroup
    Properties:
      Description: "Lock cache down to webserver access only"
  MyCacheSecurityGroupIngress:
    Type: AWS::ElastiCache::SecurityGroupIngress
    Properties:
      CacheSecurityGroupName:
        Ref: MyCacheSecurityGroup
      EC2SecurityGroupName:
        Ref: AWSEBSecurityGroup
```

For more information about the resources used in this example configuration file, see the following references:

- [AWS::ElastiCache::CacheCluster](#)
- [AWS::ElastiCache::SecurityGroup](#)
- [AWS::ElastiCache::SecurityGroupIngress](#)

Create a separate configuration file called `options.config` and define the custom option settings.

```
option_settings:
  "aws:elasticbeanstalk:customoption":
    CacheNodeType : cache.m1.small
    NumCacheNodes : 1
    Engine : memcached
```

These lines tell Elastic Beanstalk to get the values for the **CacheNodeType**, **NumCacheNodes**, and **Engine** properties from the **CacheNodeType**, **NumCacheNodes**, and **Engine** values in a config file (`options.config` in our example) that contains an `option_settings` section with an **aws:elasticbeanstalk:customoption** section that contains a name-value pair that contains the actual value to use. In the example above, this means `cache.m1.small`, `1`, and `memcached` would be used for the values. For more information about `Fn::GetOptionSetting`, see [Functions \(p. 626\)](#).

EC2-VPC (default)

This sample adds an Amazon ElastiCache cluster to an environment with instances launched into the EC2-VPC platform. Specifically, the information in this section applies to a scenario where EC2 launches instances into the default VPC. All of the properties in this example are the minimum required properties that must be set for each resource type. For more information about default VPCs, see [Your Default VPC and Subnets](#).

Note

This example creates AWS resources, which you might be charged for. For more information about AWS pricing, see <https://aws.amazon.com/pricing/>. Some services are part of the AWS Free Usage Tier. If you are a new customer, you can test drive these services for free. See <https://aws.amazon.com/free/> for more information.

To use this example, do the following:

1. Create an `.ebextensions` (p. 600) directory in the top-level directory of your source bundle.
2. Create two configuration files with the `.config` extension and place them in your `.ebextensions` directory. One configuration file defines the resources, and the other configuration file defines the options.
3. Deploy your application to Elastic Beanstalk.

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

Now name the resources configuration file `elasticache.config`. To create the ElastiCache cluster, this example specifies the name of the ElastiCache cluster resource (`MyElasticache`), declares its type, and then configures the properties for the cluster. The example references the ID of the security group resource that we create and define in this configuration file.

Next, we create an EC2 security group. We define the name for this resource, declare its type, add a description, and set the ingress rules for the security group to allow access only from instances inside the Elastic Beanstalk security group (`AWSEBSecurityGroup`). (The parameter name, `AWSEBSecurityGroup`, is a fixed resource name provided by Elastic Beanstalk. You must add `AWSEBSecurityGroup` to your ElastiCache security group ingress rules in order for your Elastic Beanstalk application to connect to the instances in your ElastiCache cluster.)

The ingress rules for the EC2 security group also define the IP protocol and port numbers on which the cache nodes can accept connections. For Redis, the default port number is `6379`.

```
#This sample requires you to create a separate configuration file that defines the custom
option settings for CacheCluster properties.
```

```

Resources:
  MyCacheSecurityGroup:
    Type: "AWS::EC2::SecurityGroup"
    Properties:
      GroupDescription: "Lock cache down to webserver access only"
      SecurityGroupIngress :
        - IpProtocol : "tcp"
          FromPort :
            Fn::GetOptionSetting:
              OptionName : "CachePort"
              DefaultValue: "6379"
          ToPort :
            Fn::GetOptionSetting:
              OptionName : "CachePort"
              DefaultValue: "6379"
      SourceSecurityGroupName:
        Ref: "AWSEBSecurityGroup"
  MyElasticCache:
    Type: "AWS::ElasticCache::CacheCluster"
    Properties:
      CacheNodeType:
        Fn::GetOptionSetting:
          OptionName : "CacheNodeType"
          DefaultValue : "cache.t2.micro"
      NumCacheNodes:
        Fn::GetOptionSetting:
          OptionName : "NumCacheNodes"
          DefaultValue : "1"
      Engine:
        Fn::GetOptionSetting:
          OptionName : "Engine"
          DefaultValue : "redis"
      VpcSecurityGroupIds:
        -
          Fn::GetAtt:
            - MyCacheSecurityGroup
            - GroupId
Outputs:
  ElasticCache:
    Description : "ID of ElasticCache Cache Cluster with Redis Engine"
    Value :
      Ref : "MyElasticCache"

```

For more information about the resources used in this example configuration file, see the following references:

- [AWS::ElasticCache::CacheCluster](#)
- [AWS::EC2::SecurityGroup](#)

Next, name the options configuration file `options.config` and define the custom option settings.

```

option_settings:
  "aws:elasticbeanstalk:customoption":
    CacheNodeType : cache.t2.micro
    NumCacheNodes : 1
    Engine : redis
    CachePort : 6379

```

These lines tell Elastic Beanstalk to get the values for the `CacheNodeType`, `NumCacheNodes`, `Engine`, and `CachePort` properties from the `CacheNodeType`, `NumCacheNodes`, `Engine`,

and `CachePort` values in a config file (`options.config` in our example). That file includes an `aws:elasticbeanstalk:customoption` section (under `option_settings`) that contains name-value pairs with the actual values to use. In the preceding example, `cache.t2.micro`, `1`, `redis`, and `6379` would be used for the values. For more information about `Fn::GetOptionSetting`, see [Functions](#) (p. 626).

EC2-VPC (custom)

If you create a custom VPC on the EC2-VPC platform and specify it as the VPC into which EC2 launches instances, the process of adding an Amazon ElastiCache cluster to your environment differs from that of a default VPC. The main difference is that you must create a subnet group for the ElastiCache cluster. All of the properties in this example are the minimum required properties that must be set for each resource type.

Note

This example creates AWS resources, which you might be charged for. For more information about AWS pricing, see <https://aws.amazon.com/pricing/>. Some services are part of the AWS Free Usage Tier. If you are a new customer, you can test drive these services for free. See <https://aws.amazon.com/free/> for more information.

To use this example, do the following:

1. Create an `.ebextensions` (p. 600) directory in the top-level directory of your source bundle.
2. Create two configuration files with the `.config` extension and place them in your `.ebextensions` directory. One configuration file defines the resources, and the other configuration file defines the options.
3. Deploy your application to Elastic Beanstalk.

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

Now name the resources configuration file `elasticache.config`. To create the ElastiCache cluster, this example specifies the name of the ElastiCache cluster resource (`MyElasticache`), declares its type, and then configures the properties for the cluster. The properties in the example reference the name of the subnet group for the ElastiCache cluster as well as the ID of security group resource that we create and define in this configuration file.

Next, we create an EC2 security group. We define the name for this resource, declare its type, add a description, the VPC ID, and set the ingress rules for the security group to allow access only from instances inside the Elastic Beanstalk security group (`AWSEBSecurityGroup`). (The parameter name, `AWSEBSecurityGroup`, is a fixed resource name provided by Elastic Beanstalk. You must add `AWSEBSecurityGroup` to your ElastiCache security group ingress rules in order for your Elastic Beanstalk application to connect to the instances in your ElastiCache cluster.)

The ingress rules for the EC2 security group also define the IP protocol and port numbers on which the cache nodes can accept connections. For Redis, the default port number is `6379`. Finally, this example creates a subnet group for the ElastiCache cluster. We define the name for this resource, declare its type, and add a description and ID of the subnet in the subnet group.

Note

We recommend that you use private subnets for the ElastiCache cluster. For more information about a VPC with a private subnet, see https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Scenario2.html.

```
#This sample requires you to create a separate configuration file that defines the custom  
option settings for CacheCluster properties.
```

```
Resources:
```

```
MyElasticCache:
  Type: "AWS::ElasticCache::CacheCluster"
  Properties:
    CacheNodeType:
      Fn::GetOptionSetting:
        OptionName : "CacheNodeType"
        DefaultValue : "cache.t2.micro"
    NumCacheNodes:
      Fn::GetOptionSetting:
        OptionName : "NumCacheNodes"
        DefaultValue : "1"
    Engine:
      Fn::GetOptionSetting:
        OptionName : "Engine"
        DefaultValue : "redis"
    CacheSubnetGroupName:
      Ref: "MyCacheSubnets"
    VpcSecurityGroupIds:
      - Ref: "MyCacheSecurityGroup"
MyCacheSecurityGroup:
  Type: "AWS::EC2::SecurityGroup"
  Properties:
    GroupDescription: "Lock cache down to webserver access only"
    VpcId:
      Fn::GetOptionSetting:
        OptionName : "VpcId"
    SecurityGroupIngress :
      - IpProtocol : "tcp"
        FromPort :
          Fn::GetOptionSetting:
            OptionName : "CachePort"
            DefaultValue: "6379"
        ToPort :
          Fn::GetOptionSetting:
            OptionName : "CachePort"
            DefaultValue: "6379"
        SourceSecurityGroupId:
          Ref: "AWSEBSecurityGroup"
MyCacheSubnets:
  Type: "AWS::ElasticCache::SubnetGroup"
  Properties:
    Description: "Subnets for ElasticCache"
    SubnetIds:
      Fn::GetOptionSetting:
        OptionName : "CacheSubnets"
Outputs:
  ElasticCache:
    Description : "ID of ElasticCache Cache Cluster with Redis Engine"
    Value :
      Ref : "MyElasticCache"
```

For more information about the resources used in this example configuration file, see the following references:

- [AWS::ElasticCache::CacheCluster](#)
- [AWS::EC2::SecurityGroup](#)
- [AWS::ElasticCache::SubnetGroup](#)

Next, name the options configuration file `options.config` and define the custom option settings.

Note

In the following example, replace the example `CacheSubnets` and `VpcId` values with your own subnets and VPC.


```
option_settings:
  "aws:elasticbeanstalk:customoption":
    CacheNodeType : cache.t2.micro
    NumCacheNodes : 1
    Engine : redis
    CachePort : 6379
    CacheSubnets:
      - subnet-1a1a1a1a
      - subnet-2b2b2b2b
      - subnet-3c3c3c3c
    VpcId: vpc-4d4d4d4d
```

These lines tell Elastic Beanstalk to get the values for the `CacheNodeType`, `NumCacheNodes`, `Engine`, `CachePort`, `CacheSubnets`, and `VpcId` properties from the `CacheNodeType`, `NumCacheNodes`, `Engine`, `CachePort`, `CacheSubnets`, and `VpcId` values in a config file (`options.config` in our example). That file includes an `aws:elasticbeanstalk:customoption` section (under `option_settings`) that contains name-value pairs with sample values. In the example above, `cache.t2.micro`, `1`, `redis`, `6379`, `subnet-1a1a1a1a`, `subnet-2b2b2b2b`, `subnet-3c3c3c3c`, and `vpc-4d4d4d4d` would be used for the values. For more information about `Fn::GetOptionSetting`, see [Functions \(p. 626\)](#).

Example: SQS, CloudWatch, and SNS

This example adds an Amazon SQS queue and an alarm on queue depth to the environment. The properties that you see in this example are the minimum required properties that you must set for each of these resources. You can download the example at [SQS, SNS, and CloudWatch](#).

Note

This example creates AWS resources, which you might be charged for. For more information about AWS pricing, see <https://aws.amazon.com/pricing/>. Some services are part of the AWS Free Usage Tier. If you are a new customer, you can test drive these services for free. See <https://aws.amazon.com/free/> for more information.

To use this example, do the following:

1. Create an `.ebextensions` (p. 600) directory in the top-level directory of your source bundle.
2. Create two configuration files with the `.config` extension and place them in your `.ebextensions` directory. One configuration file defines the resources, and the other configuration file defines the options.
3. Deploy your application to Elastic Beanstalk.

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

Create a configuration file (e.g., `sqs.config`) that defines the resources. In this example, we create an SQS queue and define the `VisibilityTimeout` property in the `MySQSQueue` resource. Next, we create an SNS Topic and specify that email gets sent to `someone@example.com` when the alarm is fired. Finally, we create a CloudWatch alarm if the queue grows beyond 10 messages. In the `Dimensions` property, we specify the name of the dimension and the value representing the dimension measurement. We use `Fn::GetAtt` to return the value of `QueueName` from `MySQSQueue`.

```
#This sample requires you to create a separate configuration file to define the custom
options for the SNS topic and SQS queue.
Resources:
  MySQSQueue:
    Type: AWS::SQS::Queue
    Properties:
      VisibilityTimeout:
        Fn::GetOptionSetting:
```

```

        OptionName: VisibilityTimeout
        DefaultValue: 30
AlarmTopic:
  Type: AWS::SNS::Topic
  Properties:
    Subscription:
      - Endpoint:
          Fn::GetOptionSetting:
            OptionName: AlarmEmail
            DefaultValue: "nobody@amazon.com"
          Protocol: email
QueueDepthAlarm:
  Type: AWS::CloudWatch::Alarm
  Properties:
    AlarmDescription: "Alarm if queue depth grows beyond 10 messages"
    Namespace: "AWS/SQS"
    MetricName: ApproximateNumberOfMessagesVisible
    Dimensions:
      - Name: QueueName
        Value : { "Fn::GetAtt" : [ "MySQSQueue", "QueueName" ] }
    Statistic: Sum
    Period: 300
    EvaluationPeriods: 1
    Threshold: 10
    ComparisonOperator: GreaterThanThreshold
    AlarmActions:
      - Ref: AlarmTopic
    InsufficientDataActions:
      - Ref: AlarmTopic

Outputs :
  QueueURL:
    Description : "URL of newly created SQS Queue"
    Value : { Ref : "MySQSQueue" }
  QueueARN :
    Description : "ARN of newly created SQS Queue"
    Value : { "Fn::GetAtt" : [ "MySQSQueue", "Arn" ] }
  QueueName :
    Description : "Name newly created SQS Queue"
    Value : { "Fn::GetAtt" : [ "MySQSQueue", "QueueName" ] }

```

For more information about the resources used in this example configuration file, see the following references:

- [AWS::SQS::Queue](#)
- [AWS::SNS::Topic](#)
- [AWS::CloudWatch::Alarm](#)

Create a separate configuration file called `options.config` and define the custom option settings.

```

option_settings:
  "aws:elasticbeanstalk:customoption":
    VisibilityTimeout : 30
    AlarmEmail : "nobody@example.com"

```

These lines tell Elastic Beanstalk to get the values for the **VisibilityTimeout** and **Subscription Endpoint** properties from the **VisibilityTimeout** and **Subscription Endpoint** values in a config file (`options.config` in our example) that contains an `option_settings` section with an `aws:elasticbeanstalk:customoption` section that contains a name-value pair that contains the actual value to use. In the example above, this means 30 and "nobody@amazon.com" would be used for the values. For more information about `Fn::GetOptionSetting`, see [the section called "Functions"](#) (p. 626).

Example: DynamoDB, CloudWatch, and SNS

This configuration file sets up the DynamoDB table as a session handler for a PHP-based application using the AWS SDK for PHP 2. To use this example, you must have an IAM instance profile, which is added to the instances in your environment and used to access the DynamoDB table.

You can download the sample that we'll use in this step at [DynamoDB session Support example](#). The sample contains the following files:

- The sample application, `index.php`
- A configuration file, `dynamodb.config`, to create and configure a DynamoDB table and other AWS resources and install software on the EC2 instances that host the application in an Elastic Beanstalk environment
- A configuration file, `options.config`, that overrides the defaults in `dynamodb.config` with specific settings for this particular installation

`index.php`

```
<?php

// Include the SDK using the Composer autoloader
require '../vendor/autoload.php';

use Aws\DynamoDb\DynamoDbClient;

// Grab the session table name and region from the configuration file
list($tableName, $region) = file(__DIR__ . '/../sessiontable');
$tableName = rtrim($tableName);
$region = rtrim($region);

// Create a DynamoDB client and register the table as the session handler
$dynamodb = DynamoDbClient::factory(array('region' => $region));
$handler = $dynamodb->registerSessionHandler(array('table_name' => $tableName, 'hash_key'
=> 'username'));

// Grab the instance ID so we can display the EC2 instance that services the request
$instanceId = file_get_contents("http://169.254.169.254/latest/meta-data/instance-id");
?>

<h1>Elastic Beanstalk PHP Sessions Sample</h1>
<p>This sample application shows the integration of the Elastic Beanstalk PHP
container and the session support for DynamoDB from the AWS SDK for PHP 2.
Using DynamoDB session support, the application can be scaled out across
multiple web servers. For more details, see the
<a href="https://aws.amazon.com/php/">PHP Developer Center</a>.</p>

<form id="SimpleForm" name="SimpleForm" method="post" action="index.php">
<?php
echo 'Request serviced from instance ' . $instanceId . '<br/>';
echo '<br/>';

if (isset($_POST['continue'])) {
    session_start();
    $_SESSION['visits'] = $_SESSION['visits'] + 1;
    echo 'Welcome back ' . $_SESSION['username'] . '<br/>';
    echo 'This is visit number ' . $_SESSION['visits'] . '<br/>';
    session_write_close();
    echo '<br/>';
    echo '<input type="Submit" value="Refresh" name="continue" id="continue"/>';
    echo '<input type="Submit" value="Delete Session" name="killsession" id="killsession"/>';
} elseif (isset($_POST['killsession'])) {
    session_start();
```

```

echo 'Goodbye ' . $_SESSION['username'] . '<br/>';
session_destroy();
echo 'Username: <input type="text" name="username" id="username" size="30"/><br/>';
echo '<br/>';
echo '<input type="Submit" value="New Session" name="newsession" id="newsession"/>';
} elseif (isset($_POST['newsession'])) {
    session_start();
    $_SESSION['username'] = $_POST['username'];
    $_SESSION['visits'] = 1;
    echo 'Welcome to a new session ' . $_SESSION['username'] . '<br/>';
    session_write_close();
    echo '<br/>';
    echo '<input type="Submit" value="Refresh" name="continue" id="continue"/>';
    echo '<input type="Submit" value="Delete Session" name="killsession" id="killsession"/>';
} else {
    echo 'To get started, enter a username.<br/>';
    echo '<br/>';
    echo 'Username: <input type="text" name="username" id="username" size="30"/><br/>';
    echo '<input type="Submit" value="New Session" name="newsession" id="newsession"/>';
}
?>
</form>

```

.ebextensions/dynamodb.config

```

Resources:
  SessionTable:
    Type: AWS::DynamoDB::Table
    Properties:
      KeySchema:
        HashKeyElement:
          AttributeName:
            Fn::GetOptionSetting:
              OptionName : SessionHashKeyName
              DefaultValue: "username"
          AttributeType:
            Fn::GetOptionSetting:
              OptionName : SessionHashKeyType
              DefaultValue: "S"
      ProvisionedThroughput:
        ReadCapacityUnits:
          Fn::GetOptionSetting:
            OptionName : SessionReadCapacityUnits
            DefaultValue: 1
        WriteCapacityUnits:
          Fn::GetOptionSetting:
            OptionName : SessionWriteCapacityUnits
            DefaultValue: 1

  SessionWriteCapacityUnitsLimit:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmDescription: { "Fn::Join" : [ "", [ { "Ref" : "AWSEBEnvironmentName" }, " write
capacity limit on the session table." ] ] }
      Namespace: "AWS/DynamoDB"
      MetricName: ConsumedWriteCapacityUnits
      Dimensions:
        - Name: TableName
          Value: { "Ref" : "SessionTable" }
      Statistic: Sum
      Period: 300
      EvaluationPeriods: 12
      Threshold:
        Fn::GetOptionSetting:
          OptionName : SessionWriteCapacityUnitsAlarmThreshold

```

```
        DefaultValue: 240
    ComparisonOperator: GreaterThanThreshold
    AlarmActions:
      - Ref: SessionAlarmTopic
    InsufficientDataActions:
      - Ref: SessionAlarmTopic

    SessionReadCapacityUnitsLimit:
      Type: AWS::CloudWatch::Alarm
      Properties:
        AlarmDescription: { "Fn::Join" : [ "", [ { "Ref" : "AWSEBEnvironmentName" }, " read
capacity limit on the session table." ] ] }
        Namespace: "AWS/DynamoDB"
        MetricName: ConsumedReadCapacityUnits
        Dimensions:
          - Name: TableName
            Value: { "Ref" : "SessionTable" }
        Statistic: Sum
        Period: 300
        EvaluationPeriods: 12
        Threshold:
          Fn::GetOptionSetting:
            OptionName : SessionReadCapacityUnitsAlarmThreshold
            DefaultValue: 240
        ComparisonOperator: GreaterThanThreshold
        AlarmActions:
          - Ref: SessionAlarmTopic
        InsufficientDataActions:
          - Ref: SessionAlarmTopic

    SessionThrottledRequestsAlarm:
      Type: AWS::CloudWatch::Alarm
      Properties:
        AlarmDescription: { "Fn::Join" : [ "", [ { "Ref" : "AWSEBEnvironmentName" }, " :
requests are being throttled." ] ] }
        Namespace: AWS/DynamoDB
        MetricName: ThrottledRequests
        Dimensions:
          - Name: TableName
            Value: { "Ref" : "SessionTable" }
        Statistic: Sum
        Period: 300
        EvaluationPeriods: 1
        Threshold:
          Fn::GetOptionSetting:
            OptionName: SessionThrottledRequestsThreshold
            DefaultValue: 1
        ComparisonOperator: GreaterThanThreshold
        AlarmActions:
          - Ref: SessionAlarmTopic
        InsufficientDataActions:
          - Ref: SessionAlarmTopic

    SessionAlarmTopic:
      Type: AWS::SNS::Topic
      Properties:
        Subscription:
          - Endpoint:
              Fn::GetOptionSetting:
                OptionName: SessionAlarmEmail
                DefaultValue: "nobody@amazon.com"
            Protocol: email

files:
  "/var/app/sessiontable":
    mode: "000444"
```

```
content: |
  `{"Ref" : "SessionTable"}`
  `{"Ref" : "AWS::Region"}`

"/var/app/composer.json":
  mode: "000744"
  content:
    {
      "require": {
        "aws/aws-sdk-php": "*"
      }
    }

container_commands:
  "1-install-composer":
    command: "cd /var/app; curl -s http://getcomposer.org/installer | php"
  "2-install-dependencies":
    command: "cd /var/app; php composer.phar install"
  "3-cleanup-composer":
    command: "rm -Rf /var/app/composer.*"
```

In the sample configuration file, we first create the DynamoDB table and configure the primary key structure for the table and the capacity units to allocate sufficient resources to provide the requested throughput. Next, we create CloudWatch alarms for `WriteCapacity` and `ReadCapacity`. We create an SNS topic that sends email to "nobody@amazon.com" if the alarm thresholds are breached.

After we create and configure our AWS resources for our environment, we need to customize the EC2 instances. We use the `files` key to pass the details of the DynamoDB table to the EC2 instances in our environment as well as add a "require" in the `composer.json` file for the AWS SDK for PHP 2. Finally, we run container commands to install composer, the required dependencies, and then remove the installer.

.ebextensions/options.config

```
option_settings:
  "aws:elasticbeanstalk:customoption":
    SessionHashKeyName           : username
    SessionHashKeyType           : S
    SessionReadCapacityUnits     : 1
    SessionReadCapacityUnitsAlarmThreshold : 240
    SessionWriteCapacityUnits    : 1
    SessionWriteCapacityUnitsAlarmThreshold : 240
    SessionThrottledRequestsThreshold : 1
    SessionAlarmEmail            : me@example.com
```

Replace the `SessionAlarmEmail` value with the email where you want alarm notifications sent. The `options.config` file contains the values used for some of the variables defined in `dynamodb.config`. For example, `dynamodb.config` contains the following lines:

```
Subscription:
- Endpoint:
  Fn::GetOptionSetting:
    OptionName: SessionAlarmEmail
    DefaultValue: "nobody@amazon.com"
```

These lines that tell Elastic Beanstalk to get the value for the **Endpoint** property from the **SessionAlarmEmail** value in a config file (`options.config` in our sample application) that contains an `option_settings` section with an **aws:elasticbeanstalk:customoption** section that contains a name-value pair that contains the actual value to use. In the example above, this means **SessionAlarmEmail** would be assigned the value `nobody@amazon.com`.

For more information about the CloudFormation resources used in this example, see the following references:

- [AWS::DynamoDB::Table](#)
- [AWS::CloudWatch::Alarm](#)
- [AWS::SNS::Topic](#)

Using Elastic Beanstalk saved configurations

You can save your environment's configuration as an object in Amazon Simple Storage Service (Amazon S3) that can be applied to other environments during environment creation, or applied to a running environment. *Saved configurations* are YAML formatted templates that define an environment's [platform version](#) (p. 29), [tier](#) (p. 13), [configuration option](#) (p. 536) settings, and tags.

You can apply tags to a saved configuration when you create it, and edit tags of existing saved configurations. For details, see [Tagging saved configurations](#) (p. 644).

Note

The tags applied to a saved configuration aren't related to the tags specified in a saved configuration using the `Tags :` key. The latter are applied to an environment when you apply the saved configuration to the environment.

Create a saved configuration from the current state of your environment in the Elastic Beanstalk management console.

To save an environment's configuration

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Save configuration**.
4. Use the on-screen form to name the saved configuration. Optionally, provide a brief description, and add tag keys and values.
5. Choose **Save**.
6. Choose **Save**.

Elastic Beanstalk > Environments > GettingStartedApp-env

Save Configuration

Save this environment's current configuration.

Environment:
GettingStartedApp-env

Configuration name:

Description:

Tags

Apply up to 50 tags. You can use tags to group and filter your resources. A tag is a key-value pair. The key must be unique within the resource and is case-sensitive. [Learn more](#)

Key	Value	
<input type="text" value="mytag1"/>	<input type="text" value="value1"/>	<input type="button" value="Remove tag"/>

49 remaining

The saved configuration includes any settings that you have applied to the environment with the console or any other client that uses the Elastic Beanstalk API. You can then apply the saved configuration to your environment at a later date to restore it to its previous state, or apply it to a new environment during [environment creation](#) (p. 365).

You can download a configuration using the EB CLI [the section called “eb config” \(p. 891\)](#) command, as shown in the following example. *NAME* is the name of your saved configuration.

```
eb config get NAME
```

To apply a saved configuration during environment creation (Elastic Beanstalk console)

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.

Note

If you have many applications, use the search bar to filter the application list.

3. In the navigation pane, find your application's name and choose **Saved configurations**.
4. Select the saved configuration you want to apply, and then choose **Launch environment**.
5. Proceed through the wizard to create your environment.

Saved configurations don't include settings applied with [configuration files \(p. 600\)](#) in your application's source code. If the same setting is applied in both a configuration file and saved configuration, the setting in the saved configuration takes precedence. Likewise, options specified in the Elastic Beanstalk console override options in saved configurations. For more information, see [Precedence \(p. 537\)](#).

Saved configurations are stored in the Elastic Beanstalk S3 bucket in a folder named after your application. For example, configurations for an application named `my-app` in the `us-west-2` region for account number `123456789012` can be found at `s3://elasticbeanstalk-us-west-2-123456789012/resources/templates/my-app/`.

View the contents of a saved configuration by opening it in a text editor. The following example configuration shows the configuration of a web server environment launched with the Elastic Beanstalk management console.

```
EnvironmentConfigurationMetadata:
  Description: Saved configuration from a multicontainer Docker environment created with
the Elastic Beanstalk Management Console
  DateCreated: '1520633151000'
  DateModified: '1520633151000'
Platform:
  PlatformArn: arn:aws:elasticbeanstalk:us-east-2::platform/Java 8 running on 64bit Amazon
Linux/2.5.0
OptionSettings:
  aws:elasticbeanstalk:command:
    BatchSize: '30'
    BatchSizeType: Percentage
  aws:elasticbeanstalk:sns:topics:
    Notification Endpoint: me@example.com
  aws:elb:policies:
    ConnectionDrainingEnabled: true
    ConnectionDrainingTimeout: '20'
  aws:elb:loadbalancer:
    CrossZone: true
  aws:elasticbeanstalk:environment:
    ServiceRole: aws-elasticbeanstalk-service-role
  aws:elasticbeanstalk:application:
    Application Healthcheck URL: /
  aws:elasticbeanstalk:healthreporting:system:
    SystemType: enhanced
  aws:autoscaling:launchconfiguration:
    IamInstanceProfile: aws-elasticbeanstalk-ec2-role
```

```
InstanceType: t2.micro
EC2KeyName: workstation-uswest2
aws:autoscaling:updatepolicy:rollingupdate:
  RollingUpdateType: Health
  RollingUpdateEnabled: true
EnvironmentTier:
  Type: Standard
  Name: WebServer
AWSConfigurationTemplateVersion: 1.1.0.0
Tags:
  Cost Center: WebApp Dev
```

You can modify the contents of a saved configuration and save it in the same location in Amazon S3. Any properly formatted saved configuration stored in the right location can be applied to an environment by using the Elastic Beanstalk management console.

The following keys are supported.

- **AWSConfigurationTemplateVersion** (required) – The configuration template version (1.1.0.0).

```
AWSConfigurationTemplateVersion: 1.1.0.0
```

- **Platform** – The Amazon Resource Name (ARN) of the environment's platform version. You can specify the platform by ARN or solution stack name.

```
Platform:
  PlatformArn: arn:aws:elasticbeanstalk:us-east-2::platform/Java 8 running on 64bit
  Amazon Linux/2.5.0
```

- **SolutionStack** – The full name of the [solution stack](#) (p. 29) used to create the environment.

```
SolutionStack: 64bit Amazon Linux 2017.03 v2.5.0 running Java 8
```

- **OptionSettings** – [Configuration option](#) (p. 536) settings to apply to the environment. For example, the following entry sets the instance type to t2.micro.

```
OptionSettings:
  aws:autoscaling:launchconfiguration:
    InstanceType: t2.micro
```

- **Tags** – Up to 47 tags to apply to resources created within the environment.

```
Tags:
  Cost Center: WebApp Dev
```

- **EnvironmentTier** – The type of environment to create. For a web server environment, you can exclude this section (web server is the default). For a worker environment, use the following.

```
EnvironmentTier:
  Name: Worker
  Type: SQS/HTTP
```

See the following topics for alternate methods of creating and applying saved configurations:

- [Setting configuration options before environment creation](#) (p. 539)
- [Setting configuration options during environment creation](#) (p. 543)
- [Setting configuration options after environment creation](#) (p. 547)

Tagging saved configurations

You can apply tags to your AWS Elastic Beanstalk saved configurations. Tags are key-value pairs associated with AWS resources. For information about Elastic Beanstalk resource tagging, use cases, tag key and value constraints, and supported resource types, see [Tagging Elastic Beanstalk application resources \(p. 349\)](#).

You can specify tags when you create a saved configuration. In an existing saved configuration, you can add or remove tags, and update the values of existing tags. You can add up to 50 tags to each saved configuration.

Adding tags during saved configuration creation

When you use the Elastic Beanstalk console to [save a configuration \(p. 640\)](#), you can specify tag keys and values on the **Save Configuration** page.

If you use the EB CLI to save a configuration, use the `--tags` option with [eb config \(p. 891\)](#) to add tags.

```
~/workspace/my-app$ eb config --tags mytag1=value1,mytag2=value2
```

With the AWS CLI or other API-based clients, add tags by using the `--tags` parameter on the [create-configuration-template](#) command.

```
$ aws elasticbeanstalk create-configuration-template \  
  --tags Key=mytag1,Value=value1 Key=mytag2,Value=value2 \  
  --application-name my-app --template-name my-template --solution-stack-name solution-stack
```

Managing tags of an existing saved configuration

You can add, update, and delete tags in an existing Elastic Beanstalk saved configuration.

To manage a saved configuration's tags using the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.

Note

If you have many applications, use the search bar to filter the application list.

3. In the navigation pane, find your application's name and choose **Saved configurations**.
4. Select the saved configuration you want to manage.
5. Choose **Actions**, and then choose **Manage tags**.
6. Use the on-screen form to add, update, or delete tags.
7. Choose **Apply**.

If you use the EB CLI to update your saved configuration, use [eb tags \(p. 930\)](#) to add, update, delete, or list tags.

For example, the following command lists the tags in a saved configuration.

```
~/workspace/my-app$ eb tags --list --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:configurationtemplate/my-app/my-template"
```

The following command updates the tag `mytag1` and deletes the tag `mytag2`.

```
~/workspace/my-app$ eb tags --update mytag1=newvalue --delete mytag2 \
--resource "arn:aws:elasticbeanstalk:us-east-2:my-account-
id:configurationtemplate/my-app/my-template"
```

For a complete list of options and more examples, see [eb tags](#) (p. 930).

With the AWS CLI or other API-based clients, use the [list-tags-for-resource](#) command to list the tags of a saved configuration.

```
$ aws elasticbeanstalk list-tags-for-resource --resource-arn "arn:aws:elasticbeanstalk:us-
east-2:my-account-id:configurationtemplate/my-app/my-template"
```

Use the [update-tags-for-resource](#) command to add, update, or delete tags in a saved configuration.

```
$ aws elasticbeanstalk update-tags-for-resource \
--tags-to-add Key=mytag1,Value=newvalue --tags-to-remove mytag2 \
--resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-
id:configurationtemplate/my-app/my-template"
```

Specify both tags to add and tags to update in the `--tags-to-add` parameter of [update-tags-for-resource](#). A nonexistent tag is added, and an existing tag's value is updated.

Note

To use some of the EB CLI and AWS CLI commands with an Elastic Beanstalk saved configuration, you need the saved configuration's ARN. To construct the ARN, first use the following command to retrieve the saved configuration's name.

```
$ aws elasticbeanstalk describe-applications --application-names my-app
```

Look for the `ConfigurationTemplates` key in the command's output. This element shows the saved configuration's name. Use this name where `my-template` is specified in the commands mentioned on this page.

Environment manifest (env.yaml)

You can include a YAML formatted environment manifest in the root of your application source bundle to configure the environment name, solution stack and [environment links](#) (p. 444) to use when creating your environment.

This file format includes support for environment groups. To use groups, specify the environment name in the manifest with a `+` symbol at the end. When you create or update the environment, specify the group name with `--group-name` (AWS CLI) or `--env-group-suffix` (EB CLI). For more information on groups, see [Creating and updating groups of Elastic Beanstalk environments](#) (p. 395).

The following example manifest defines a web server environment with a link to a worker environment component that it is dependent upon. The manifest uses groups to allow creating multiple environments with the same source bundle:

```
~/myapp/frontend/env.yaml
```

```
AWSConfigurationTemplateVersion: 1.1.0.0
SolutionStack: 64bit Amazon Linux 2015.09 v2.0.6 running Multi-container Docker 1.7.1
(Generic)
OptionSettings:
  aws:elasticbeanstalk:command:
    BatchSize: '30'
    BatchSizeType: Percentage
```

```
aws:elasticbeanstalk:sns:topics:
  Notification Endpoint: me@example.com
aws:elb:policies:
  ConnectionDrainingEnabled: true
  ConnectionDrainingTimeout: '20'
aws:elb:loadbalancer:
  CrossZone: true
aws:elasticbeanstalk:environment:
  ServiceRole: aws-elasticbeanstalk-service-role
aws:elasticbeanstalk:application:
  Application Healthcheck URL: /
aws:elasticbeanstalk:healthreporting:system:
  SystemType: enhanced
aws:autoscaling:launchconfiguration:
  IamInstanceProfile: aws-elasticbeanstalk-ec2-role
  InstanceType: t2.micro
  EC2KeyName: workstation-uswest2
aws:autoscaling:updatepolicy:rollingupdate:
  RollingUpdateType: Health
  RollingUpdateEnabled: true
Tags:
  Cost Center: WebApp Dev
CName: front-A08G28LG+
EnvironmentName: front+
EnvironmentLinks:
  "WORKERQUEUE" : "worker+"
```

The following keys are supported.

- **AWSConfigurationTemplateVersion** (required) – The configuration template version (1.1.0.0).

```
AWSConfigurationTemplateVersion: 1.1.0.0
```

- **Platform** – The Amazon Resource Name (ARN) of the environment's platform version. You can specify the platform by ARN or solution stack name.

```
Platform:
  PlatformArn: arn:aws:elasticbeanstalk:us-east-2::platform/Java 8 running on 64bit
  Amazon Linux/2.5.0
```

- **SolutionStack** – The full name of the [solution stack](#) (p. 29) used to create the environment.

```
SolutionStack: 64bit Amazon Linux 2017.03 v2.5.0 running Java 8
```

- **OptionSettings** – [Configuration option](#) (p. 536) settings to apply to the environment. For example, the following entry sets the instance type to t2.micro.

```
OptionSettings:
  aws:autoscaling:launchconfiguration:
    InstanceType: t2.micro
```

- **Tags** – Up to 47 tags to apply to resources created within the environment.

```
Tags:
  Cost Center: WebApp Dev
```

- **EnvironmentTier** – The type of environment to create. For a web server environment, you can exclude this section (web server is the default). For a worker environment, use the following.

```
EnvironmentTier:
  Name: Worker
```

```
Type: SQS/HTTP
```

- **CName** – The CNAME for the environment. Include a + character at the end of the name to enable groups.

```
CName: front-A08G28LG+
```

- **EnvironmentName** – The name of the environment to create. Include a + character at the end of the name to enable groups.

```
EnvironmentName: front+
```

With groups enabled, you must specify a group name when you create the environments. Elastic Beanstalk appends the group name to the environment name with a hyphen. For example, with the environment name `front+` and the group name `dev`, Elastic Beanstalk will create the environment with the name `front-dev`.

- **EnvironmentLinks** – A map of variable names and environment names of dependencies. The following example makes the `worker+` environment a dependency and tells Elastic Beanstalk to save the link information to a variable named `WORKERQUEUE`.

```
EnvironmentLinks:  
  "WORKERQUEUE" : "worker+"
```

The value of the link variable varies depending on the type of the linked environment. For a web server environment, the link is the environment's CNAME. For a worker environment, the link is the name of the environment's Amazon Simple Queue Service (Amazon SQS) queue.

The **CName**, **EnvironmentName** and **EnvironmentLinks** keys can be used to create [environment groups](#) (p. 395) and [links to other environments](#) (p. 444). These features are currently supported when using the EB CLI, AWS CLI or an SDK.

Using a custom Amazon machine image (AMI)

When you create an AWS Elastic Beanstalk environment, you can specify an Amazon Machine Image (AMI) to use instead of the standard Elastic Beanstalk AMI included in your platform version. A custom AMI can improve provisioning times when instances are launched in your environment if you need to install a lot of software that isn't included in the standard AMIs.

Using [configuration files](#) (p. 600) is great for configuring and customizing your environment quickly and consistently. Applying configurations, however, can start to take a long time during environment creation and updates. If you do a lot of server configuration in configuration files, you can reduce this time by making a custom AMI that already has the software and configuration that you need.

A custom AMI also allows you to make changes to low-level components, such as the Linux kernel, that are difficult to implement or take a long time to apply in configuration files. To create a custom AMI, launch an Elastic Beanstalk platform AMI in Amazon EC2, customize the software and configuration to your needs, and then stop the instance and save an AMI from it.

Creating a custom AMI

To identify the base Elastic Beanstalk AMI

1. In a command window, run a command like the following. Specify the AWS Region where you want to use your custom AMI, and replace the platform ARN and version number with the Elastic Beanstalk platform that your application is based on.

```
$ aws elasticbeanstalk describe-platform-version --region us-east-2 \
  --platform-arn "arn:aws:elasticbeanstalk:us-east-2::platform/Tomcat 8.5 with Java
  8 running on 64bit Amazon Linux/3.1.6" \
  --query PlatformDescription.CustomAmiList
[
  {
    "VirtualizationType": "pv",
    "ImageId": ""
  },
  {
    "VirtualizationType": "hvm",
    "ImageId": "ami-020ae06fdda6a0f66"
  }
]
```

2. Take note of the ImageId value that looks like `ami-020ae06fdda6a0f66` in the result.

The value is the stock Elastic Beanstalk AMI for the platform version, EC2 instance architecture, and AWS Region that are relevant for your application. If you need to create AMIs for multiple platforms, architectures or AWS Regions, repeat this process to identify the correct base AMI for each combination.

Notes

- Don't create an AMI from an instance that has been launched in an Elastic Beanstalk environment. Elastic Beanstalk makes changes to instances during provisioning that can cause issues in the saved AMI. Saving an image from an instance in an Elastic Beanstalk environment will also make the version of your application that was deployed to the instance a fixed part of the image.
- We recommend that you always use the latest platform version. When you update to a new platform version, we also recommend that you rebase your custom AMI to the new platform version's AMI. This minimizes deployment failures due to incompatible package or library versions.

It is also possible to create a custom AMI from a community AMI that wasn't published by Elastic Beanstalk. You can use the latest [Amazon Linux](#) AMI as a starting point. When you launch an environment with a Linux AMI that isn't managed by Elastic Beanstalk, Elastic Beanstalk attempts to install platform software (language, framework, proxy server, etc.) and additional components to support features such as [Enhanced Health Reporting](#) (p. 691).

Note

AMIs that aren't managed by Elastic Beanstalk aren't supported for Windows Server-based Elastic Beanstalk platforms.

Although Elastic Beanstalk can use an AMI that isn't managed by Elastic Beanstalk, the increase in provisioning time that results from Elastic Beanstalk installing missing components can reduce or eliminate the benefits of creating a custom AMI in the first place. Other Linux distributions might work with some troubleshooting but are not officially supported. If your application requires a specific Linux distribution, one alternative is to create a Docker image and run it on the Elastic Beanstalk [single container Docker platform](#) (p. 51) or [multicontainer Docker platform](#) (p. 58).

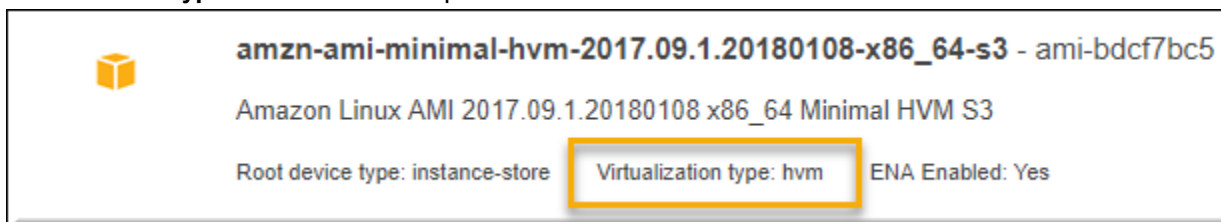
To create a custom AMI

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Choose **Launch Instance**.
3. Choose **Community AMIs**.
4. If you identified a base Elastic Beanstalk or Amazon Linux AMI that you want to customize to create a custom AMI, enter its AMI ID in the search box, and then press **Enter**.

You can also search the list for another community AMI that suits your needs.

Note

We recommend that you choose an AMI that uses HVM virtualization. These AMIs show **Virtualization type: hvm** in their description.



For details about instance virtualization types, see [Linux AMI Virtualization Types](#) in the *Amazon EC2 User Guide for Linux Instances*.

5. Choose **Select** to select the AMI.
6. Select an instance type, and then choose **Next: Configure Instance Details**.
7. **(Linux platforms)** Expand the **Advanced Details** section and paste the following text in the **User Data** field.

```
#cloud-config
repo_releasever: repository version number
repo_upgrade: none
```

The *repository version number* is the year and month version in the AMI name. For example, AMIs based on the March 2015 release of Amazon Linux have a repository version number 2015.03. For an Elastic Beanstalk image, this matches the date shown in the solution stack name for your [platform version](#) (p. 29).

Note

These settings configure the lock-on-launch feature. This causes the AMI to use a fixed, specific repository version when it launches, and disables the automatic installation of security updates. Both are required to use a custom AMI with Elastic Beanstalk.

8. Proceed through the wizard to [launch the EC2 instance](#). When prompted, select a key pair that you have access to so that you can connect to the instance for the next steps.
9. [Connect to the instance](#) with SSH or RDP.
10. Perform any customizations you want.
11. **(Windows platforms)** Run the EC2Config service Sysprep. For information about EC2Config, see [Configuring a Windows Instance Using the EC2Config Service](#). Ensure that Sysprep is configured to generate a random password that can be retrieved from the AWS Management Console.
12. In the Amazon EC2 console, stop the EC2 instance. Then on the **Instance Actions** menu, choose **Create Image (EBS AMI)**.
13. To avoid incurring additional AWS charges, [terminate the EC2 instance](#).

To use your custom AMI in an Elastic Beanstalk environment

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. For **AMI ID**, enter your custom AMI ID.
6. Choose **Apply**.

When you create a new environment with the custom AMI, you should use the same platform version that you used as a base to create the AMI. If you later apply a [platform update \(p. 415\)](#) to an environment using a custom AMI, Elastic Beanstalk attempts to apply the library and configuration updates during the bootstrapping process.

Cleaning up a custom AMI

When you are done with a custom AMI and don't need it to launch Elastic Beanstalk environments anymore, consider cleaning it up to minimize storage cost. Cleaning up a custom AMI involves deregistering it from Amazon EC2 and deleting other associated resources. For details, see [Deregistering Your Linux AMI](#) or [Deregistering Your Windows AMI](#).

Serving static files

To improve performance, you can configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

Elastic Beanstalk supports configuring the proxy to serve static files on most platform branches based on Amazon Linux 2. The one exception is Docker.

Note

On the Python and Ruby platforms, Elastic Beanstalk configures some static file folders by default. For details, see the static file configuration sections for [Python \(p. 292\)](#) and [Ruby \(p. 317\)](#). You can configure additional folders as explained on this page.

Configure static files using the console

To configure the proxy server to serve static files

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. In the **Static files** section, enter a path for serving static files and the directory of the static files to serve into the empty row at the bottom of the list.

Note

If you aren't seeing the **Static files** section, you have to add at least one mapping by using a [configuration file \(p. 600\)](#). For details, see [the section called "Configure static files using configuration options" \(p. 651\)](#) on this page.

Start the path with a slash (/). Specify a directory name in the root of your application's source code; don't start it with a slash.

When you add a mapping, an extra row appears in case you want to add another one. To remove a mapping, click the **Remove** icon.

Path (Example: /assets)	Directory (Example: /static/assets)
<input type="text" value="/html"/>	<input type="text" value="statichtml"/> X
<input type="text" value="/images"/>	<input type="text" value="staticimages"/> X
<input type="text"/>	<input type="text"/>

6. Choose **Apply**.

Configure static files using configuration options

You can use a [configuration file \(p. 600\)](#) to configure static file paths and directory locations using configuration options. You can add a configuration file to your application's source bundle and deploy it during environment creation or a later deployment.

If your environment uses a platform branch based on Amazon Linux 2, use the [aws:elasticbeanstalk:environment:proxy:staticfiles \(p. 577\)](#) namespace.

The following example configuration file tells the proxy server to serve files in the `statichtml` folder at the path `/html`, and files in the `staticimages` folder at the path `/images`.

Example `.ebextensions/static-files.config`

```
option_settings:  
  aws:elasticbeanstalk:environment:proxy:staticfiles:  
    /html: statichtml  
    /images: staticimages
```

If your Elastic Beanstalk environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the following additional information:

Amazon Linux AMI platform-specific namespaces

On Amazon Linux AMI platform branches, static file configuration namespaces vary by platform. For details, see one of the following pages:

- [Go configuration namespace \(p. 88\)](#)
- [Java SE configuration namespace \(p. 114\)](#)
- [Tomcat configuration namespaces \(p. 104\)](#)
- [Node.js configuration namespace \(p. 197\)](#)
- [Python configuration namespaces \(p. 292\)](#)

Configuring HTTPS for your Elastic Beanstalk environment

If you've purchased and configured a [custom domain name \(p. 534\)](#) for your Elastic Beanstalk environment, you can use HTTPS to allow users to connect to your web site securely. If you don't own a domain name, you can still use HTTPS with a self-signed certificate for development and testing purposes. HTTPS is a must for any application that transmits user data or login information.

The simplest way to use HTTPS with an Elastic Beanstalk environment is to [assign a server certificate to your environment's load balancer \(p. 656\)](#). When you configure your load balancer to terminate HTTPS, the connection between the client and the load balancer is secure. Backend connections between the load balancer and EC2 instances use HTTP, so no additional configuration of the instances is required.

Note

With [AWS Certificate Manager \(ACM\)](#), you can create a trusted certificate for your domain names for free. ACM certificates can only be used with AWS load balancers and Amazon CloudFront distributions, and ACM is [available only in certain AWS Regions](#).

To use an ACM certificate with Elastic Beanstalk, see [Configuring your Elastic Beanstalk environment's load balancer to terminate HTTPS \(p. 656\)](#).

If you run your application in a single instance environment, or need to secure the connection all the way to the EC2 instances behind the load balancer, you can [configure the proxy server that runs on the instance to terminate HTTPS \(p. 658\)](#). Configuring your instances to terminate HTTPS connections requires the use of [configuration files \(p. 600\)](#) to modify the software running on the instances, and to modify security groups to allow secure connections.

For end-to-end HTTPS in a load balanced environment, you can [combine instance and load balancer termination \(p. 679\)](#) to encrypt both connections. By default, if you configure the load balancer to forward traffic using HTTPS, it will trust any certificate presented to it by the backend instances. For maximum security, you can attach policies to the load balancer that prevent it from connecting to instances that don't present a public certificate that it trusts.

Note

You can also configure the load balancer to [relay HTTPS traffic without decrypting it \(p. 682\)](#). The down side to this method is that the load balancer cannot see the requests and thus cannot optimize routing or report response metrics.

If ACM is not available in your region, you can purchase a trusted certificate from a third party. A third-party certificate can be used to decrypt HTTPS traffic at your load balancer, on the backend instances, or both.

For development and testing, you can [create and sign a certificate \(p. 653\)](#) yourself with open source tools. Self-signed certificates are free and easy to create, but cannot be used for front-end decryption on public sites. If you attempt to use a self-signed certificate for an HTTPS connection to a client, the user's

browser displays an error message indicating that your web site is unsafe. You can, however, use a self-signed certificate to secure backend connections without issue.

ACM is the preferred tool to provision, manage, and deploy your server certificates programmatically or using the AWS CLI. If ACM is not [available in your AWS Region](#), you can [upload a third-party or self-signed certificate and private key \(p. 655\)](#) to AWS Identity and Access Management (IAM) by using the AWS CLI. Certificates stored in IAM can be used with load balancers and CloudFront distributions.

Note

The [Does it have Snakes?](#) sample application on GitHub includes configuration files and instructions for each method of configuring HTTPS with a Tomcat web application. See the [readme file](#) and [HTTPS instructions](#) for details.

Topics

- [Create and sign an X509 certificate \(p. 653\)](#)
- [Upload a certificate to IAM \(p. 655\)](#)
- [Configuring your Elastic Beanstalk environment's load balancer to terminate HTTPS \(p. 656\)](#)
- [Configuring your application to terminate HTTPS connections at the instance \(p. 658\)](#)
- [Configuring end-to-end encryption in a load balanced Elastic Beanstalk environment \(p. 679\)](#)
- [Configuring your environment's load balancer for TCP Passthrough \(p. 682\)](#)
- [Storing private keys securely in Amazon S3 \(p. 682\)](#)
- [Configuring HTTP to HTTPS redirection \(p. 683\)](#)

Create and sign an X509 certificate

You can create an X509 certificate for your application with OpenSSL. OpenSSL is a standard, open source library that supports a wide range of cryptographic functions, including the creation and signing of x509 certificates. For more information about OpenSSL, visit www.openssl.org.

Note

You only need to create a certificate locally if you want to [use HTTPS in a single instance environment \(p. 658\)](#) or [re-encrypt on the backend \(p. 679\)](#) with a self-signed certificate. If you own a domain name, you can create a certificate in AWS and use it with a load balanced environment for free by using AWS Certificate Manager (ACM). See [Request a Certificate](#) in the *AWS Certificate Manager User Guide* for instructions.

Run `openssl version` at the command line to see if you already have OpenSSL installed. If you don't, you can build and install the source code using the instructions at the [public GitHub repository](#), or use your favorite package manager. OpenSSL is also installed on Elastic Beanstalk's Linux images, so a quick alternative is to connect to an EC2 instance in a running environment by using the [EB CLI \(p. 852\)](#)'s `eb ssh` command:

```
~/eb$ eb ssh  
[ec2-user@ip-255-55-55-255 ~]$ openssl version  
OpenSSL 1.0.1k-fips 8 Jan 2015
```

You need to create an RSA private key to create your certificate signing request (CSR). To create your private key, use the `openssl genrsa` command:

```
[ec2-user@ip-255-55-55-255 ~]$ openssl genrsa 2048 > privatekey.pem  
Generating RSA private key, 2048 bit long modulus  
.....  
+++  
.....+++  
e is 65537 (0x10001)
```

privatekey.pem

The name of the file where you want to save the private key. Normally, the **openssl genrsa** command prints the private key contents to the screen, but this command pipes the output to a file. Choose any file name, and store the file in a secure place so that you can retrieve it later. If you lose your private key, you won't be able to use your certificate.

A CSR is a file you send to a certificate authority (CA) to apply for a digital server certificate. To create a CSR, use the **openssl req** command:

```
$ openssl req -new -key privatekey.pem -out csr.pem
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
```

Enter the information requested and press **Enter**. The following table describes and shows examples for each field.

Name	Description	Example
Country Name	The two-letter ISO abbreviation for your country.	US = United States
State or Province	The name of the state or province where your organization is located. You cannot abbreviate this name.	Washington
Locality Name	The name of the city where your organization is located.	Seattle
Organization Name	The full legal name of your organization. Do not abbreviate your organization name.	Example Corporation
Organizational Unit	Optional, for additional organization information.	Marketing
Common Name	The fully qualified domain name for your web site. This must match the domain name that users see when they visit your site, otherwise certificate errors will be shown.	www.example.com
Email address	The site administrator's email address.	someone@example.com

You can submit the signing request to a third party for signing, or sign it yourself for development and testing. Self-signed certificates can also be used for backend HTTPS between a load balancer and EC2 instances.

To sign the certificate, use the **openssl x509** command. The following example uses the private key from the previous step (*privatekey.pem*) and the signing request (*csr.pem*) to create a public certificate named *public.crt* that is valid for 365 days.

```
$ openssl x509 -req -days 365 -in csr.pem -signkey privatekey.pem -out public.crt
Signature ok
subject=/C=us/ST=Washington/L=Seattle/O=example corporation/OU=marketing/
CN=www.example.com/emailAddress=someone@example.com
```

```
Getting Private key
```

Keep the private key and public certificate for later use. You can discard the signing request. Always [store the private key in a secure location \(p. 682\)](#) and avoid adding it to your source code.

To use the certificate with the Windows Server platform, you must convert it to a PFX format. Use the following command to create a PFX certificate from the private key and public certificate files:

```
$ openssl pkcs12 -export -out example.com.pfx -inkey privatekey.pem -in public.crt  
Enter Export Password: password  
Verifying - Enter Export Password: password
```

Now that you have a certificate, you can [upload it to IAM \(p. 655\)](#) for use with a load balancer, or [configure the instances in your environment to terminate HTTPS \(p. 658\)](#).

Upload a certificate to IAM

To use your certificate with your Elastic Beanstalk environment's load balancer, upload the certificate and private key to AWS Identity and Access Management (IAM). You can use a certificate stored in IAM with Elastic Load Balancing load balancers and Amazon CloudFront distributions.

Note

AWS Certificate Manager (ACM) is the preferred tool to provision, manage, and deploy your server certificates. For more information about requesting an ACM certificate, see [Request a Certificate](#) in the *AWS Certificate Manager User Guide*. For more information about importing third-party certificates into ACM, see [Importing Certificates](#) in the *AWS Certificate Manager User Guide*. Use IAM to upload a certificate only if ACM is not [available in your AWS Region](#).

You can use the [AWS Command Line Interface \(p. 850\)](#) (AWS CLI) to upload your certificate. The following command uploads a self-signed certificate named `https-cert.crt` with a private key named `private-key.pem`:

```
$ aws iam upload-server-certificate --server-certificate-name elastic-beanstalk-x509 --  
certificate-body file://https-cert.crt --private-key file://private-key.pem  
{  
  "ServerCertificateMetadata": {  
    "ServerCertificateId": "AS5YBEIONO2Q7CAIHKNGC",  
    "ServerCertificateName": "elastic-beanstalk-x509",  
    "Expiration": "2017-01-31T23:06:22Z",  
    "Path": "/",  
    "Arn": "arn:aws:iam::123456789012:server-certificate/elastic-beanstalk-x509",  
    "UploadDate": "2016-02-01T23:10:34.167Z"  
  }  
}
```

The `file://` prefix tells the AWS CLI to load the contents of a file in the current directory. `elastic-beanstalk-x509` specifies the name to call the certificate in IAM.

If you purchased a certificate from a certificate authority and received a certificate chain file, upload that as well by including the `--certificate-chain` option:

```
$ aws iam upload-server-certificate --server-certificate-name elastic-beanstalk-x509 --  
certificate-chain file://certificate-chain.pem --certificate-body file://https-cert.crt --  
private-key file://private-key.pem
```

Make note of the Amazon Resource Name (ARN) for your certificate. You'll use it when you update your load balancer configuration settings to use HTTPS.

Note

A certificate uploaded to IAM stays stored even after it's no longer used in any environment's load balancer. It contains sensitive data. When you no longer need the certificate for any environment, be sure to delete it. For details about deleting a certificate from IAM, see https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_server-certs.html#delete-server-certificate.

For more information about server certificates in IAM, see [Working with Server Certificates](#) in the *IAM User Guide*.

Configuring your Elastic Beanstalk environment's load balancer to terminate HTTPS

To update your AWS Elastic Beanstalk environment to use HTTPS, you need to configure an HTTPS listener for the load balancer in your environment. Two types of load balancer support an HTTPS listener: Classic Load Balancer and Application Load Balancer.

You can use the Elastic Beanstalk console or a configuration file to configure a secure listener and assign the certificate.

Note

Single-instance environments don't have a load balancer and don't support HTTPS termination at the load balancer.

Configuring a secure listener using the Elastic Beanstalk console

To assign a certificate to your environment's load balancer

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Load balancer** configuration category, choose **Edit**.

Note

If the **Load balancer** configuration category doesn't have an **Edit** button, your environment doesn't have a [load balancer](#) (p. 436).

5. On the **Modify load balancer** page, the procedure varies depending on the type of load balancer associated with your environment.
 - **Classic Load Balancer**
 - a. Choose **Add listener**.
 - b. In the **Classic Load Balancer listener** dialog box, configure the following settings:
 - For **Listener port**, type the incoming traffic port, typically 443.
 - For **Listener protocol**, choose **HTTPS**.
 - For **Instance port**, type 80.
 - For **Instance protocol**, choose **HTTP**.
 - For **SSL certificate**, choose your certificate.
 - c. Choose **Add**.
 - **Application Load Balancer**

- a. Choose **Add listener**.
- b. In the **Application Load Balancer listener** dialog box, configure the following settings:
 - For **Port**, type the incoming traffic port, typically 443.
 - For **Protocol**, choose **HTTPS**.
 - For **SSL certificate**, choose your certificate.
- c. Choose **Add**.

Note

For Classic Load Balancer and Application Load Balancer, if the drop-down menu doesn't show any certificates, you should create or upload a certificate for your [custom domain name \(p. 534\)](#) in [AWS Certificate Manager \(ACM\)](#) (preferred). Alternatively, upload a certificate to IAM with the AWS CLI.

- **Network Load Balancer**
 - a. Choose **Add listener**.
 - b. In the **Network Load Balancer listener** dialog box, for **Port**, type the incoming traffic port, typically 443.
 - c. Choose **Add**.
6. Choose **Apply**.

Configuring a secure listener using a configuration file

You can configure a secure listener on your load balancer with one of the following [configuration files \(p. 600\)](#).

Example `.ebextensions/securelistener-clb.config`

Use this example when your environment has a Classic Load Balancer. The example uses options in the `aws:elb:listener` namespace to configure an HTTPS listener on port 443 with the specified certificate, and to forward the decrypted traffic to the instances in your environment on port 80.

```
option_settings:
  aws:elb:listener:443:
    SSLCertificateId: arn:aws:acm:us-east-2:1234567890123:certificate/
    #####
    ListenerProtocol: HTTPS
    InstancePort: 80
```

Replace the highlighted text with the ARN of your certificate. The certificate can be one that you created or uploaded in AWS Certificate Manager (ACM) (preferred), or one that you uploaded to IAM with the AWS CLI.

For more information about Classic Load Balancer configuration options, see [Classic Load Balancer configuration namespaces \(p. 479\)](#).

Example `.ebextensions/securelistener-alb.config`

Use this example when your environment has an Application Load Balancer. The example uses options in the `aws:elbv2:listener` namespace to configure an HTTPS listener on port 443 with the specified certificate. The listener routes traffic to the default process.

```
option_settings:
  aws:elbv2:listener:443:
    ListenerEnabled: 'true'
```



```
Protocol: HTTPS
SSLCertificateArns: arn:aws:acm:us-east-2:1234567890123:certificate/
#####
```

Example `.ebextensions/securelistener-nlb.config`

Use this example when your environment has a Network Load Balancer. The example uses options in the `aws:elbv2:listener` namespace to configure a listener on port 443. The listener routes traffic to the default process.

```
option_settings:
  aws:elbv2:listener:443:
    ListenerEnabled: 'true'
```

Configuring a security group

If you configure your load balancer to forward traffic to an instance port other than port 80, you must add a rule to your security group that allows inbound traffic over the instance port from your load balancer. If you create your environment in a custom VPC, Elastic Beanstalk adds this rule for you.

You add this rule by adding a `Resources` key to a [configuration file \(p. 600\)](#) in the `.ebextensions` directory for your application.

The following example configuration file adds an ingress rule to the `AWSEBSecurityGroup` security group. This allows traffic on port 1000 from the load balancer's security group.

Example `.ebextensions/sg-ingressfromlb.config`

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 1000
      FromPort: 1000
      SourceSecurityGroupName: {"Fn::GetAtt" : ["AWSEBLoadBalancer" ,
"SourceSecurityGroup.GroupName"]}
```

Configuring your application to terminate HTTPS connections at the instance

You can use [configuration files \(p. 600\)](#) to configure the proxy server that passes traffic to your application to terminate HTTPS connections. This is useful if you want to use HTTPS with a single instance environment, or if you configure your load balancer to pass traffic through without decrypting it.

To enable HTTPS, you must allow incoming traffic on port 443 to the EC2 instance that your Elastic Beanstalk application is running on. You do this by using the `Resources` key in the configuration file to add a rule for port 443 to the ingress rules for the `AWSEBSecurityGroup` security group.

The following snippet adds an ingress rule to the `AWSEBSecurityGroup` security group that opens port 443 to all traffic for a single instance environment:

`.ebextensions/https-instance-securitygroup.config`

```
Resources:
  sslSecurityGroupIngress:
```

```
Type: AWS::EC2::SecurityGroupIngress
Properties:
  GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
  IpProtocol: tcp
  ToPort: 443
  FromPort: 443
  CidrIp: 0.0.0.0/0
```

In a load balanced environment in a default [Amazon Virtual Private Cloud \(Amazon VPC\)](#), you can modify this policy to only accept traffic from the load balancer. See [Configuring end-to-end encryption in a load balanced Elastic Beanstalk environment \(p. 679\)](#) for an example.

Platforms

- [Terminating HTTPS on EC2 instances running Docker \(p. 659\)](#)
- [Terminating HTTPS on EC2 instances running Go \(p. 661\)](#)
- [Terminating HTTPS on EC2 instances running Java SE \(p. 663\)](#)
- [Terminating HTTPS on EC2 instances running Node.js \(p. 665\)](#)
- [Terminating HTTPS on EC2 instances running PHP \(p. 667\)](#)
- [Terminating HTTPS on EC2 instances running Python \(p. 669\)](#)
- [Terminating HTTPS on EC2 instances running Ruby \(p. 671\)](#)
- [Terminating HTTPS on EC2 instances running Tomcat \(p. 675\)](#)
- [Terminating HTTPS on Amazon EC2 instances running .NET \(p. 677\)](#)

Terminating HTTPS on EC2 instances running Docker

For Docker containers, you use a [configuration file \(p. 600\)](#) to enable HTTPS.

Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `files` key creates the following files on the instance:

```
/etc/nginx/conf.d/https.conf
```

Configures the nginx server. This file is loaded when the nginx service starts.

```
/etc/pki/tls/certs/server.crt
```

Creates the certificate file on the instance. Replace *certificate file contents* with the contents of your certificate.

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

If you have intermediate certificates, include them in `server.crt` after your site certificate:

```
-----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

```
/etc/pki/tls/certs/server.key
```

Creates the private key file on the instance. Replace *private key contents* with the contents of the private key used to create the certificate request or self-signed certificate.

Example .ebextensions/https-instance.config

```
files:
  /etc/nginx/conf.d/https.conf:
    mode: "000644"
    owner: root
    group: root
    content: |
      # HTTPS Server

      server {
        listen 443;
        server_name localhost;

        ssl on;
        ssl_certificate /etc/pki/tls/certs/server.crt;
        ssl_certificate_key /etc/pki/tls/certs/server.key;

        ssl_session_timeout 5m;

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        ssl_prefer_server_ciphers on;

        location / {
          proxy_pass http://docker;
          proxy_http_version 1.1;

          proxy_set_header Connection "";
          proxy_set_header Host $host;
          proxy_set_header X-Real-IP $remote_addr;
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
          proxy_set_header X-Forwarded-Proto https;
        }
      }

  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3

and modify the configuration to download it during deployment. For instructions, see [Storing private keys securely in Amazon S3 \(p. 682\)](#).

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an [AWS CloudFormation function \(p. 626\)](#) and adds a rule to it.

Example `.ebextensions/https-instance-single.config`

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load balanced environment, you configure the load balancer to either [pass secure traffic through untouched \(p. 682\)](#), or [decrypt and re-encrypt \(p. 679\)](#) for end-to-end encryption.

Terminating HTTPS on EC2 instances running Go

For Go container types, you enable HTTPS with a [configuration file \(p. 600\)](#) and an nginx configuration file that configures the nginx server to use HTTPS.

Add the following snippet to your configuration file, replacing the certificate and private key placeholders as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `Resources` key enables port 443 on the security group used by your environment's instance.
- The `files` key creates the following files on the instance:

```
/etc/pki/tls/certs/server.crt
```

Creates the certificate file on the instance. Replace *certificate file contents* with the contents of your certificate.

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

If you have intermediate certificates, include them in `server.crt` after your site certificate:

```
-----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

```
/etc/pki/tls/certs/server.key
```

Creates the private key file on the instance. Replace *private key contents* with the contents of the private key used to create the certificate request or self-signed certificate.

- The `container_commands` key restarts the nginx server after everything is configured so that the server loads the nginx configuration file.

Example `.ebextensions/https-instance.config`

```
files:
  /etc/pki/tls/certs/server.crt:
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----

container_commands:
  01restart_nginx:
    command: "service nginx restart"
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see [Storing private keys securely in Amazon S3 \(p. 682\)](#).

Place the following in a file with the `.conf` extension in the `.ebextensions/nginx/conf.d/` directory of your source bundle (e.g., `.ebextensions/nginx/conf.d/https.conf`). Replace `app_port` with the port number that your application listens on. This example configures the nginx server to listen on port 443 using SSL. For more information about these configuration files on the Go platform, see [Configuring the reverse proxy \(p. 90\)](#).

Example `.ebextensions/nginx/conf.d/https.conf`

```
# HTTPS server

server {
    listen      443;
    server_name localhost;

    ssl        on;
    ssl_certificate /etc/pki/tls/certs/server.crt;
    ssl_certificate_key /etc/pki/tls/certs/server.key;

    ssl_session_timeout 5m;

    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;

    location / {
        proxy_pass http://localhost:app_port;
        proxy_set_header    Connection "";
        proxy_http_version 1.1;
        proxy_set_header    Host      $host;
        proxy_set_header    X-Real-IP $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header    X-Forwarded-Proto https;
    }
}
```

```
}

```

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation [function](#) (p. 626) and adds a rule to it.

Example `.ebextensions/https-instance-single.config`

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load balanced environment, you configure the load balancer to either [pass secure traffic through untouched](#) (p. 682), or [decrypt and re-encrypt](#) (p. 679) for end-to-end encryption.

Terminating HTTPS on EC2 instances running Java SE

For Java SE container types, you enable HTTPS with an `.ebextensions` [configuration file](#) (p. 600), and an nginx configuration file that configures the nginx server to use HTTPS.

Add the following snippet to your configuration file, replacing the certificate and private key placeholders as instructed, and save it in the `.ebextensions` directory. The configuration file performs the following tasks:

- The `files` key creates the following files on the instance:

```
/etc/pki/tls/certs/server.crt
```

Creates the certificate file on the instance. Replace *certificate file contents* with the contents of your certificate.

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

If you have intermediate certificates, include them in `server.crt` after your site certificate:

```
-----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

```
/etc/pki/tls/certs/server.key
```

Creates the private key file on the instance. Replace *private key contents* with the contents of the private key used to create the certificate request or self-signed certificate.

- The `container_commands` key restarts the nginx server after everything is configured so that the server loads the nginx configuration file.

Example .ebextensions/https-instance.config

```
files:
  /etc/pki/tls/certs/server.crt:
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----

container_commands:
  01restart_nginx:
    command: "service nginx restart"
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see [Storing private keys securely in Amazon S3 \(p. 682\)](#).

Place the following in a file with the .conf extension in the .ebextensions/nginx/conf.d/ directory of your source bundle (e.g., .ebextensions/nginx/conf.d/https.conf). Replace *app_port* with the port number that your application listens on. This example configures the nginx server to listen on port 443 using SSL. For more information about these configuration files on the Java SE platform, see [Configuring the reverse proxy \(p. 116\)](#).

Example .ebextensions/nginx/conf.d/https.conf

```
# HTTPS server

server {
    listen      443;
    server_name localhost;

    ssl         on;
    ssl_certificate      /etc/pki/tls/certs/server.crt;
    ssl_certificate_key  /etc/pki/tls/certs/server.key;

    ssl_session_timeout 5m;

    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;

    location / {
        proxy_pass http://localhost:app_port;
        proxy_set_header    Connection "";
        proxy_http_version  1.1;
        proxy_set_header    Host          $host;
        proxy_set_header    X-Real-IP    $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header    X-Forwarded-Proto https;
    }
}
```

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation [function \(p. 626\)](#) and adds a rule to it.

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load balanced environment, you configure the load balancer to either [pass secure traffic through untouched \(p. 682\)](#), or [decrypt and re-encrypt \(p. 679\)](#) for end-to-end encryption.

Terminating HTTPS on EC2 instances running Node.js

The following example configuration file [extends the default nginx configuration \(p. 201\)](#) to listen on port 443 and terminate SSL/TLS connections with a public certificate and private key.

If you configured your environment for [enhanced health reporting \(p. 691\)](#), you need to configure nginx to generate access logs. To do that, uncomment the block of lines under the comment that reads `# For enhanced health...` by removing the leading `#` characters.

Example .ebextensions/https-instance.config

```
files:
  /etc/nginx/conf.d/https.conf:
    mode: "000644"
    owner: root
    group: root
    content: |
      # HTTPS server

      server {
        listen      443;
        server_name localhost;

        ssl          on;
        ssl_certificate      /etc/pki/tls/certs/server.crt;
        ssl_certificate_key  /etc/pki/tls/certs/server.key;

        ssl_session_timeout 5m;

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        ssl_prefer_server_ciphers on;

        # For enhanced health reporting support, uncomment this block:

        #if ($time_iso8601 ~ "^(\\d{4})-(\\d{2})-(\\d{2})T(\\d{2})") {
        #   set $year $1;
        #   set $month $2;
        #   set $day $3;
        #   set $hour $4;
        #}
        #access_log /var/log/nginx/healthd/application.log.$year-$month-$day-$hour
healthd;
        #access_log /var/log/nginx/access.log main;

        location / {
          proxy_pass http://nodejs;
          proxy_set_header    Connection "";
          proxy_http_version 1.1;
        }
      }
```



```

        proxy_set_header    Host            $host;
        proxy_set_header    X-Real-IP       $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header    X-Forwarded-Proto https;
    }
}
/etc/pki/tls/certs/server.crt:
mode: "000400"
owner: root
group: root
content: |
    -----BEGIN CERTIFICATE-----
    certificate file contents
    -----END CERTIFICATE-----

/etc/pki/tls/certs/server.key:
mode: "000400"
owner: root
group: root
content: |
    -----BEGIN RSA PRIVATE KEY-----
    private key contents # See note below.
    -----END RSA PRIVATE KEY-----

```

The files key creates the following files on the instance:

/etc/nginx/conf.d/https.conf

Configures the nginx server. This file is loaded when the nginx service starts.

/etc/pki/tls/certs/server.crt

Creates the certificate file on the instance. Replace *certificate file contents* with the contents of your certificate.

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

If you have intermediate certificates, include them in `server.crt` after your site certificate:

```

    -----BEGIN CERTIFICATE-----
    certificate file contents
    -----END CERTIFICATE-----
    -----BEGIN CERTIFICATE-----
    first intermediate certificate
    -----END CERTIFICATE-----
    -----BEGIN CERTIFICATE-----
    second intermediate certificate
    -----END CERTIFICATE-----

```

/etc/pki/tls/certs/server.key

Creates the private key file on the instance. Replace *private key contents* with the contents of the private key used to create the certificate request or self-signed certificate.

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see [Storing private keys securely in Amazon S3 \(p. 682\)](#).

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation [function \(p. 626\)](#) and adds a rule to it.

Example `.ebextensions/https-instance-single.config`

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load balanced environment, you configure the load balancer to either [pass secure traffic through untouched \(p. 682\)](#), or [decrypt and re-encrypt \(p. 679\)](#) for end-to-end encryption.

Terminating HTTPS on EC2 instances running PHP

For PHP container types, you use a [configuration file \(p. 600\)](#) to enable the Apache HTTP Server to use HTTPS.

Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory.

The configuration file performs the following tasks:

- The `packages` key uses `yum` to install `mod24_ssl`.
- The `files` key creates the following files on the instance:

```
/etc/httpd/conf.d/ssl.conf
```

Configures the Apache server. This file loads when the Apache service starts.

```
/etc/pki/tls/certs/server.crt
```

Creates the certificate file on the instance. Replace *certificate file contents* with the contents of your certificate.

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

If you have intermediate certificates, include them in `server.crt` after your site certificate:

```
-----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

```
/etc/pki/tls/certs/server.key
```

Creates the private key file on the instance. Replace *private key contents* with the contents of the private key used to create the certificate request or self-signed certificate.

Example .ebextensions/https-instance.config

```
packages:
  yum:
    mod24_ssl : []

files:
  /etc/httpd/conf.d/ssl.conf:
    mode: "000644"
    owner: root
    group: root
    content: |
      LoadModule ssl_module modules/mod_ssl.so
      Listen 443
      <VirtualHost *:443>
        <Proxy *>
          Order deny,allow
          Allow from all
        </Proxy>

        SSLEngine                on
        SSLCertificateFile        "/etc/pki/tls/certs/server.crt"
        SSLCertificateKeyFile     "/etc/pki/tls/certs/server.key"
        SSLCipherSuite            ECDH+AESGCM:EDH+AESGCM:AES256+ECDH:AES256+EDH
        SSLProtocol               All -SSLv2 -SSLv3
        SSLHonorCipherOrder       On
        SSLSessionTickets         Off

        Header always set Strict-Transport-Security "max-age=63072000; includeSubdomains;
preload"
        Header always set X-Frame-Options DENY
        Header always set X-Content-Type-Options nosniff

        ProxyPass / http://localhost:80/ retry=0
        ProxyPassReverse / http://localhost:80/
        ProxyPreserveHost on
        RequestHeader set X-Forwarded-Proto "https" early

      </VirtualHost>

  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see [Storing private keys securely in Amazon S3 \(p. 682\)](#).

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation function (p. 626) and adds a rule to it.

Example `.ebextensions/https-instance-single.config`

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load balanced environment, you configure the load balancer to either [pass secure traffic through untouched \(p. 682\)](#), or [decrypt and re-encrypt \(p. 679\)](#) for end-to-end encryption.

Terminating HTTPS on EC2 instances running Python

For Python container types using Apache HTTP Server with the Web Server Gateway Interface (WSGI), you use a [configuration file \(p. 600\)](#) to enable the Apache HTTP Server to use HTTPS.

Add the following snippet to your [configuration file \(p. 600\)](#), replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `packages` key uses `yum` to install `mod24_ssl`.
- The `files` key creates the following files on the instance:

```
/etc/httpd/conf.d/ssl.conf
```

Configures the Apache server. If your application is not named `application.py`, replace the highlighted text in the value for `WSGIScriptAlias` with the local path to your application. For example, a `django` application's may be at `django/wsgi.py`. The location should match the value of the `WSGIPath` option that you set for your environment.

Depending on your application requirements, you may also need to add other directories to the `python-path` parameter.

```
/etc/pki/tls/certs/server.crt
```

Creates the certificate file on the instance. Replace *certificate file contents* with the contents of your certificate.

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

If you have intermediate certificates, include them in `server.crt` after your site certificate:

```
-----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
```

```
-----END CERTIFICATE-----
```

```
/etc/pki/tls/certs/server.key
```

Creates the private key file on the instance. Replace *private key contents* with the contents of the private key used to create the certificate request or self-signed certificate.

- The `container_commands` key stops the `httpd` service after everything has been configured so that the service uses the new `https.conf` file and certificate.

Note

The example works only in environments using the [Python \(p. 290\)](#) platform.

Example `.ebextensions/https-instance.config`

```
packages:
  yum:
    mod24_ssl : []

files:
  /etc/httpd/conf.d/ssl.conf:
    mode: "000644"
    owner: root
    group: root
    content: |
      LoadModule wsgi_module modules/mod_wsgi.so
      WSGIPythonHome /opt/python/run/baselinenv
      WSGISocketPrefix run/wsgi
      WSGIRestrictEmbedded On
      Listen 443
      <VirtualHost *:443>
        SSLEngine on
        SSLCertificateFile "/etc/pki/tls/certs/server.crt"
        SSLCertificateKeyFile "/etc/pki/tls/certs/server.key"

        Alias /static/ /opt/python/current/app/static/
        <Directory /opt/python/current/app/static>
          Order allow,deny
          Allow from all
        </Directory>

        WSGIScriptAlias / /opt/python/current/app/application.py

        <Directory /opt/python/current/app>
          Require all granted
        </Directory>

        WSGIDaemonProcess wsgi-ssl processes=1 threads=15 display-name=%{GROUP} \
          python-path=/opt/python/current/app \
          python-home=/opt/python/run/venv \
          home=/opt/python/current/app \
          user=wsgi \
          group=wsgi
        WSGIProcessGroup wsgi-ssl

      </VirtualHost>

  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
```

```
    certificate file contents
    -----END CERTIFICATE-----

/etc/pki/tls/certs/server.key:
mode: "000400"
owner: root
group: root
content: |
    -----BEGIN RSA PRIVATE KEY-----
    private key contents # See note below.
    -----END RSA PRIVATE KEY-----

container_commands:
01killhttpd:
  command: "killall httpd"
02waitforhttpddeath:
  command: "sleep 3"
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see [Storing private keys securely in Amazon S3 \(p. 682\)](#).

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an [AWS CloudFormation function \(p. 626\)](#) and adds a rule to it.

Example `.ebextensions/https-instance-single.config`

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load balanced environment, you configure the load balancer to either [pass secure traffic through untouched \(p. 682\)](#), or [decrypt and re-encrypt \(p. 679\)](#) for end-to-end encryption.

Terminating HTTPS on EC2 instances running Ruby

For Ruby container types, the way you enable HTTPS depends on the type of application server used.

Topics

- [Configure HTTPS for Ruby with Puma \(p. 671\)](#)
- [Configure HTTPS for Ruby with Passenger \(p. 673\)](#)

Configure HTTPS for Ruby with Puma

For Ruby container types that use Puma as the application server, you use a [configuration file \(p. 600\)](#) to enable HTTPS.

Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `files` key creates the following files on the instance:

```
/etc/nginx/conf.d/https.conf
```

Configures the nginx server. This file is loaded when the nginx service starts.

```
/etc/pki/tls/certs/server.crt
```

Creates the certificate file on the instance. Replace *certificate file contents* with the contents of your certificate.

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

If you have intermediate certificates, include them in `server.crt` after your site certificate:

```
-----BEGIN CERTIFICATE-----  
certificate file contents  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
first intermediate certificate  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
second intermediate certificate  
-----END CERTIFICATE-----
```

```
/etc/pki/tls/certs/server.key
```

Creates the private key file on the instance. Replace *private key contents* with the contents of the private key used to create the certificate request or self-signed certificate.

- The `container_commands` key restarts the nginx server after everything is configured so that the server uses the new `https.conf` file.

Example `.ebextensions/https-instance.config`

```
files:  
  /etc/nginx/conf.d/https.conf:  
    content: |  
      # HTTPS server  
  
      server {  
        listen      443;  
        server_name localhost;  
  
        ssl          on;  
        ssl_certificate      /etc/pki/tls/certs/server.crt;  
        ssl_certificate_key  /etc/pki/tls/certs/server.key;  
  
        ssl_session_timeout 5m;  
  
        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;  
        ssl_prefer_server_ciphers on;  
  
        location / {  
          proxy_pass http://my_app;  
          proxy_set_header    Host $host;  
          proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;  
          proxy_set_header    X-Forwarded-Proto https;  
        }  
      }
```

```

location /assets {
    alias /var/app/current/public/assets;
    gzip_static on;
    gzip on;
    expires max;
    add_header Cache-Control public;
}

location /public {
    alias /var/app/current/public;
    gzip_static on;
    gzip on;
    expires max;
    add_header Cache-Control public;
}
}

/etc/pki/tls/certs/server.crt:
content: |
-----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----

/etc/pki/tls/certs/server.key:
content: |
-----BEGIN RSA PRIVATE KEY-----
private key contents # See note below.
-----END RSA PRIVATE KEY-----

container_commands:
01restart_nginx:
  command: "service nginx restart"

```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see [Storing private keys securely in Amazon S3 \(p. 682\)](#).

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an [AWS CloudFormation function \(p. 626\)](#) and adds a rule to it.

Example `.ebextensions/https-instance-single.config`

```

Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0

```

For a load balanced environment, you configure the load balancer to either [pass secure traffic through untouched \(p. 682\)](#), or [decrypt and re-encrypt \(p. 679\)](#) for end-to-end encryption.

Configure HTTPS for Ruby with Passenger

For Ruby container types that use Passenger as the application server, you use both a configuration file and a JSON file to enable HTTPS.

To configure HTTPS for Ruby with Passenger

1. Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `files` key creates the following files on the instance:

```
/etc/pki/tls/certs/server.crt
```

Creates the certificate file on the instance. Replace *certificate file contents* with the contents of your certificate.

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

If you have intermediate certificates, include them in `server.crt` after your site certificate:

```
-----BEGIN CERTIFICATE-----  
certificate file contents  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
first intermediate certificate  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
second intermediate certificate  
-----END CERTIFICATE-----
```

```
/etc/pki/tls/certs/server.key
```

Creates the private key file on the instance. Replace *private key contents* with the contents of the private key used to create the certificate request or self-signed certificate.

Example .Ebextensions snippet for configuring HTTPS for Ruby with Passenger

```
files:  
  /etc/pki/tls/certs/server.crt:  
    content: |  
      -----BEGIN CERTIFICATE-----  
      certificate file contents  
      -----END CERTIFICATE-----  
  
  /etc/pki/tls/certs/server.key:  
    content: |  
      -----BEGIN RSA PRIVATE KEY-----  
      private key contents # See note below.  
      -----END RSA PRIVATE KEY-----
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see [Storing private keys securely in Amazon S3 \(p. 682\)](#).

2. Create a text file and add the following JSON to the file. Save it in your source bundle's root directory with the name `passenger-standalone.json`. This JSON file configures Passenger to use HTTPS.

Important

This JSON file must not contain a byte order mark (BOM). If it does, the Passenger JSON library will not read the file correctly and the Passenger service will not start.

Example `passenger-standalone.json`

```
{
  "ssl" : true,
  "ssl_port" : 443,
  "ssl_certificate" : "/etc/pki/tls/certs/server.crt",
  "ssl_certificate_key" : "/etc/pki/tls/certs/server.key"
}
```

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation [function \(p. 626\)](#) and adds a rule to it.

Example `.ebextensions/https-instance-single.config`

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load balanced environment, you configure the load balancer to either [pass secure traffic through untouched \(p. 682\)](#), or [decrypt and re-encrypt \(p. 679\)](#) for end-to-end encryption.

Terminating HTTPS on EC2 instances running Tomcat

For Tomcat container types, you use a [configuration file \(p. 600\)](#) to enable the Apache HTTP Server to use HTTPS when acting as the reverse proxy for Tomcat.

Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `files` key creates the following files on the instance:

```
/etc/pki/tls/certs/server.crt
```

Creates the certificate file on the instance. Replace *certificate file contents* with the contents of your certificate.

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

```
/etc/pki/tls/certs/server.key
```

Creates the private key file on the instance. Replace *private key contents* with the contents of the private key used to create the certificate request or self-signed certificate.

```
/opt/elasticbeanstalk/hooks/appdeploy/post/99_start_httpd.sh
```

Creates a post-deployment hook script to restart the httpd service.

Example `.ebextensions/https-instance.config`

```
files:
  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----

  /opt/elasticbeanstalk/hooks/appdeploy/post/99_start_httpd.sh:
    mode: "000755"
    owner: root
    group: root
    content: |
      #!/usr/bin/env bash
      sudo service httpd restart
```

You must also configure your environment's proxy server to listen on port 443. The following Apache 2.4 configuration adds a listener on port 443. To learn more, see [Configuring your Tomcat environment's proxy server \(p. 108\)](#).

Example `.ebextensions/httpd/conf.d/ssl.conf`

```
Listen 443
<VirtualHost *:443>
  ServerName server-name
  SSLEngine on
  SSLCertificateFile "/etc/pki/tls/certs/server.crt"
  SSLCertificateKeyFile "/etc/pki/tls/certs/server.key"

  <Proxy *>
    Require all granted
  </Proxy>
  ProxyPass / http://localhost:8080/ retry=0
  ProxyPassReverse / http://localhost:8080/
  ProxyPreserveHost on

  ErrorLog /var/log/httpd/elasticbeanstalk-ssl-error_log
</VirtualHost>
```

Your certificate vendor may include intermediate certificates that you can install for better compatibility with mobile clients. Configure Apache with an intermediate certificate authority (CA) bundle by

adding the following to your SSL configuration file (see [Extending and overriding the default Apache configuration \(p. 110\)](#) for the location):

- In the `ssl.conf` file contents, specify the chain file:

```
SSLCertificateKeyFile "/etc/pki/tls/certs/server.key"  
SSLCertificateChainFile "/etc/pki/tls/certs/gd_bundle.crt"  
SSLCipherSuite          EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
```

- Add a new entry to the `files` key with the contents of the intermediate certificates:

```
files:  
  /etc/pki/tls/certs/gd_bundle.crt:  
    mode: "000400"  
    owner: root  
    group: root  
    content: |  
      -----BEGIN CERTIFICATE-----  
      First intermediate certificate  
      -----END CERTIFICATE-----  
      -----BEGIN CERTIFICATE-----  
      Second intermediate certificate  
      -----END CERTIFICATE-----
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see [Storing private keys securely in Amazon S3 \(p. 682\)](#).

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an [AWS CloudFormation function \(p. 626\)](#) and adds a rule to it.

Example `.ebextensions/https-instance-single.config`

```
Resources:  
  sslSecurityGroupIngress:  
    Type: AWS::EC2::SecurityGroupIngress  
    Properties:  
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}  
      IpProtocol: tcp  
      ToPort: 443  
      FromPort: 443  
      CidrIp: 0.0.0.0/0
```

For a load balanced environment, you configure the load balancer to either [pass secure traffic through untouched \(p. 682\)](#), or [decrypt and re-encrypt \(p. 679\)](#) for end-to-end encryption.

Terminating HTTPS on Amazon EC2 instances running .NET

The following [configuration file \(p. 600\)](#) creates and runs a Windows PowerShell script that performs the following tasks:

- Checks for an existing HTTPS certificate binding to port 443
- Gets the [PFX certificate \(p. 653\)](#) and password from an Amazon S3 bucket

Note

Add an AmazonS3ReadOnlyAccess policy to the aws-elasticbeanstalk-ec2-role to access the SSL certificate and password files in the Amazon S3 bucket.

- Installs the certificate
- Binds the certificate to port 443

Note

To remove the HTTP endpoint (port 80), include the Remove-WebBinding command under the **Remove the HTTP binding** section of the example.

Example .ebextensions/https-instance-dotnet.config

```
files:
  "C:\\certs\\install-cert.ps1":
    content: |
      import-module webadministration
      ## Settings - replace the following values with your own
      $bucket = "my-bucket"      ## S3 bucket name
      $certkey = "example.com.pfx" ## S3 object key for your PFX certificate
      $pwdkey = "password.txt"   ## S3 object key for a text file containing the
      certificate's password
      ##

      # Set variables
      $certfile = "C:\cert.pfx"
      $pwdfile = "C:\certs\pwdcontent"
      Read-S3Object -BucketName $bucket -Key $pwdkey -File $pwdfile
      $pwd = Get-Content $pwdfile -Raw

      # Clean up existing binding
      if ( Get-WebBinding "Default Web Site" -Port 443 ) {
        Echo "Removing WebBinding"
        Remove-WebBinding -Name "Default Web Site" -BindingInformation *:443:
      }
      if ( Get-Item -path IIS:\SslBindings\0.0.0.0!443 ) {
        Echo "Deregistering WebBinding from IIS"
        Remove-Item -path IIS:\SslBindings\0.0.0.0!443
      }

      # Download certificate from S3
      Read-S3Object -BucketName $bucket -Key $certkey -File $certfile

      # Install certificate
      Echo "Installing cert..."
      $securepwd = ConvertTo-SecureString -String $pwd -Force -AsPlainText
      $cert = Import-PfxCertificate -FilePath $certfile cert:\localMachine\my -Password
      $securepwd

      # Create site binding
      Echo "Creating and registering WebBinding"
      New-WebBinding -Name "Default Web Site" -IP "*" -Port 443 -Protocol https
      New-Item -path IIS:\SslBindings\0.0.0.0!443 -value $cert -Force

      ## Remove the HTTP binding
      ## (optional) Uncomment the following line to unbind port 80
      # Remove-WebBinding -Name "Default Web Site" -BindingInformation *:80:
      ##

      # Update firewall
      netsh advfirewall firewall add rule name="Open port 443" protocol=TCP localport=443
      action=allow dir=OUT
```

```
commands:
  00_install_ssl:
    command: powershell -NoProfile -ExecutionPolicy Bypass -file C:\\certs\\install-
cert.ps1
```

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an [AWS CloudFormation function \(p. 626\)](#) and adds a rule to it.

Example `.ebextensions/https-instance-single.config`

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load balanced environment, you configure the load balancer to either [pass secure traffic through untouched \(p. 682\)](#), or [decrypt and re-encrypt \(p. 679\)](#) for end-to-end encryption.

Configuring end-to-end encryption in a load balanced Elastic Beanstalk environment

Terminating secure connections at the load balancer and using HTTP on the backend might be sufficient for your application. Network traffic between AWS resources can't be listened to by instances that are not part of the connection, even if they are running under the same account.

However, if you are developing an application that needs to comply with strict external regulations, you might be required to secure all network connections. You can use the Elastic Beanstalk console or [configuration files \(p. 600\)](#) to make your Elastic Beanstalk environment's load balancer connect to backend instances securely to meet these requirements. The following procedure focuses on configuration files.

First, [add a secure listener to your load balancer \(p. 656\)](#), if you haven't already.

You must also configure the instances in your environment to listen on the secure port and terminate HTTPS connections. The configuration varies per platform. See [Configuring your application to terminate HTTPS connections at the instance \(p. 658\)](#) for instructions. You can use a [self-signed certificate \(p. 653\)](#) for the EC2 instances without issue.

Next, configure the listener to forward traffic using HTTPS on the secure port used by your application. Use one of the following configuration files, based on the type of load balancer that your environment uses.

`.ebextensions/https-reencrypt-clb.config`

Use this configuration file with a Classic Load Balancer. In addition to configuring the load balancer, the configuration file also changes the default health check to use port 443 and HTTPS, to ensure that the load balancer can connect securely.

```
option_settings:
  aws:elb:listener:443:
    InstancePort: 443
    InstanceProtocol: HTTPS
  aws:elasticbeanstalk:application:
```

```
Application Healthcheck URL: HTTPS:443/
```

.ebextensions/https-reencrypt-alb.config

Use this configuration file with an Application Load Balancer.

```
option_settings:
  aws:elbv2:listener:443:
    DefaultProcess: https
    ListenerEnabled: 'true'
    Protocol: HTTPS
  aws:elasticbeanstalk:environment:process:https:
    Port: '443'
    Protocol: HTTPS
```

.ebextensions/https-reencrypt-nlb.config

Use this configuration file with a Network Load Balancer.

```
option_settings:
  aws:elbv2:listener:443:
    DefaultProcess: https
    ListenerEnabled: 'true'
  aws:elasticbeanstalk:environment:process:https:
    Port: '443'
```

The `DefaultProcess` option is named this way because of Application Load Balancers, which can have nondefault listeners on the same port for traffic to specific paths (see [Application Load Balancer \(p. 479\)](#) for details). For a Network Load Balancer the option specifies the only target process for this listener.

In this example, we named the process `https` because it listens to secure (HTTPS) traffic. The listener sends traffic to the process on the designated port using the TCP protocol, because a Network Load Balancer works only with TCP. This is okay, because network traffic for HTTP and HTTPS is implemented on top of TCP.

Note

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See [Recommended values \(p. 537\)](#) for details.

In the next task, you need to modify the load balancer's security group to allow traffic. Depending on the [Amazon Virtual Private Cloud \(Amazon VPC\)](#) in which you launch your environment—the default VPC or a custom VPC—the load balancer's security group will vary. In a default VPC, Elastic Load Balancing provides a default security group that all load balancers can use. In an Amazon VPC that you create, Elastic Beanstalk creates a security group for the load balancer to use.

To support both scenarios, you can create a security group and tell Elastic Beanstalk to use it. The following configuration file creates a security group and attaches it to the load balancer.

.ebextensions/https-lbsecuritygroup.config

```
option_settings:
  # Use the custom security group for the load balancer
  aws:elb:loadbalancer:
    SecurityGroups: '`{ "Ref" : "loadbalancersg" }`'
    ManagedSecurityGroup: '`{ "Ref" : "loadbalancersg" }`'

Resources:
  loadbalancersg:
    Type: AWS::EC2::SecurityGroup
    Properties:
```

```
GroupDescription: load balancer security group
VpcId: vpc-#####
SecurityGroupIngress:
  - IpProtocol: tcp
    FromPort: 443
    ToPort: 443
    CidrIp: 0.0.0.0/0
  - IpProtocol: tcp
    FromPort: 80
    ToPort: 80
    CidrIp: 0.0.0.0/0
SecurityGroupEgress:
  - IpProtocol: tcp
    FromPort: 80
    ToPort: 80
    CidrIp: 0.0.0.0/0
```

Replace the highlighted text with your default or custom VPC ID. The previous example includes ingress and egress over port 80 to allow HTTP connections. You can remove those properties if you want to allow only secure connections.

Finally, add ingress and egress rules that allow communication over port 443 between the load balancer's security group and the instances' security group.

.ebextensions/https-backendsecurity.config

```
Resources:
  # Add 443-inbound to instance security group (AWSEBSecurityGroup)
  httpsFromLoadBalancerSG:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      SourceSecurityGroupId: {"Fn::GetAtt" : ["loadbalancersg", "GroupId"]}
  # Add 443-outbound to load balancer security group (loadbalancersg)
  httpsToBackendInstances:
    Type: AWS::EC2::SecurityGroupEgress
    Properties:
      GroupId: {"Fn::GetAtt" : ["loadbalancersg", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      DestinationSecurityGroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
```

Doing this separately from security group creation enables you to restrict the source and destination security groups without creating a circular dependency.

After you have completed all the previous tasks, the load balancer connects to your backend instances securely using HTTPS. The load balancer doesn't care if your instance's certificate is self-signed or issued by a trusted certificate authority, and will accept any certificate presented to it.

You can change this behavior by adding policies to the load balancer that tell it to trust only a specific certificate. The following configuration file creates two policies. One policy specifies a public certificate, and the other tells the load balancer to only trust that certificate for connections to instance port 443.

.ebextensions/https-backendauth.config

```
option_settings:
  # Backend Encryption Policy
  aws:elb:policies:backendencryption:
```



```
PublicKeyPolicyNames: backendkey
InstancePorts: 443
# Public Key Policy
aws:elb:policies:backendkey:
  PublicKey: |
    -----BEGIN CERTIFICATE-----
    #####
    #####
    #####
    #####
    #####
    -----END CERTIFICATE-----
```

Replace the highlighted text with the contents of your EC2 instance's public certificate.

Configuring your environment's load balancer for TCP Passthrough

If you don't want the load balancer in your AWS Elastic Beanstalk environment to decrypt HTTPS traffic, you can configure the secure listener to relay requests to backend instances as-is.

First [configure your environment's EC2 instances to terminate HTTPS \(p. 658\)](#). Test the configuration on a single instance environment to make sure everything works before adding a load balancer to the mix.

Add a [configuration file \(p. 600\)](#) to your project to configure a listener on port 443 that passes TCP packets as-is to port 443 on backend instances:

.ebextensions/https-lb-passthrough.config

```
option_settings:
  aws:elb:listener:443:
    ListenerProtocol: TCP
    InstancePort: 443
    InstanceProtocol: TCP
```

In a default [Amazon Virtual Private Cloud \(Amazon VPC\)](#), you also need to add a rule to the instances' security group to allow inbound traffic on 443 from the load balancer:

.ebextensions/https-instance-securitygroup.config

```
Resources:
  443inboundfromloadbalancer:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      SourceSecurityGroupName: { "Fn::GetAtt": ["AWSEBLoadBalancer",
"SourceSecurityGroup.GroupName"] }
```

In a custom VPC, Elastic Beanstalk updates the security group configuration for you.

Storing private keys securely in Amazon S3

The private key that you use to sign your public certificate is private and should not be committed to source code. You can avoid storing private keys in configuration files by uploading them to Amazon S3, and configuring Elastic Beanstalk to download the file from Amazon S3 during application deployment.

The following example shows the [Resources \(p. 622\)](#) and [files \(p. 607\)](#) sections of a [configuration file \(p. 600\)](#) downloads a private key file from an Amazon S3 bucket.

Example `.ebextensions/privatekey.config`

```
Resources:
  AWSEBAutoScalingGroup:
    Metadata:
      AWS::CloudFormation::Authentication:
        S3Auth:
          type: "s3"
          buckets: ["elasticbeanstalk-us-west-2-123456789012"]
          roleName:
            "Fn::GetOptionSetting":
              Namespace: "aws:autoscaling:launchconfiguration"
              OptionName: "IamInstanceProfile"
              DefaultValue: "aws-elasticbeanstalk-ec2-role"
files:
  # Private key
  "/etc/pki/tls/certs/server.key":
    mode: "000400"
    owner: root
    group: root
    authentication: "S3Auth"
    source: https://elasticbeanstalk-us-west-2-123456789012.s3.us-west-2.amazonaws.com/
server.key
```

Replace the bucket name and URL in the example with your own. The first entry in this file adds an authentication method named `S3Auth` to the environment's Auto Scaling group's metadata. If you have configured a custom [instance profile \(p. 21\)](#) for your environment, that will be used, otherwise the default value of `aws-elasticbeanstalk-ec2-role` is applied. The default instance profile has permission to read from the Elastic Beanstalk storage bucket. If you use a different bucket, [add permissions to the instance profile \(p. 763\)](#).

The second entry uses the `S3Auth` authentication method to download the private key from the specified URL and save it to `/etc/pki/tls/certs/server.key`. The proxy server can then read the private key from this location to [terminate HTTPS connections at the instance \(p. 658\)](#).

The instance profile assigned to your environment's EC2 instances must have permission to read the key object from the specified bucket. [Verify that the instance profile has permission \(p. 763\)](#) to read the object in IAM, and that the permissions on the bucket and object do not prohibit the instance profile.

To view a bucket's permissions

1. Open the [Amazon S3 Management Console](#).
2. Choose a bucket.
3. Choose **Properties** and then choose **Permissions**.
4. Verify that your account is a grantee on the bucket with read permission.
5. If a bucket policy is attached, choose **Bucket policy** to view the permissions assigned to the bucket.

Configuring HTTP to HTTPS redirection

In [Configuring HTTPS for your Elastic Beanstalk environment \(p. 652\)](#) and its subtopics, we discuss configuring your Elastic Beanstalk environment to use HTTPS to ensure traffic encryption into your application. This topic describes how to elegantly handle HTTP traffic to your application if end users still initiate it. You do this by configuring *HTTP to HTTPS redirection*, sometimes referred to as *forcing HTTPS*.

To configure redirection, you first configure your environment to handle HTTPS traffic. Then you redirect HTTP traffic to HTTPS. These two steps are discussed in the following subsections.

Configure your environment to handle HTTPS traffic

Depending on your environment's load balancing configuration, do one of the following:

- **Load balanced environment** – [Configure your load balancer to terminate HTTPS \(p. 656\)](#).
- **Single instance environment** – [Configure your application to terminate HTTPS connections at the instance \(p. 658\)](#). This configuration depends on your environment's platform.

Redirect HTTP traffic to HTTPS

You can configure either the web servers on your environment's instances or the environment's Application Load Balancer to redirect HTTP traffic to HTTPS. Do one of the following:

- **Configure instance web servers** – This method works on any web server environment. Configure web servers on your Amazon Elastic Compute Cloud (Amazon EC2) instances to respond to HTTP traffic with an HTTP redirection response status. This configuration depends on your environment's platform. Find the folder for your platform in the [https-redirect](#) collection on GitHub, and use the example configuration file in that folder.

If your environment uses [Elastic Load Balancing health checks \(p. 689\)](#), the load balancer expects a healthy instance to respond to the HTTP health check messages with HTTP 200 (OK) responses. Therefore, your web server shouldn't redirect these messages to HTTPS. The example configuration files in [https-redirect](#) handle this requirement correctly.

- **Configure load balancer** – This method works if you have a load-balanced environment that uses an [Application Load Balancer \(p. 479\)](#). Application Load Balancer can send redirection responses as HTTP traffic comes in. In this case, you don't need to configure redirection on your environment's instances. We have two example configuration files on GitHub that show how to configure Application Load Balancer for redirection. The [alb-http-to-https-redirection-full.config](#) configuration file creates an HTTPS listener on port 443, and modifies the default port 80 listener to redirect incoming HTTP traffic to HTTPS. The [alb-http-to-https-redirection.config](#) configuration file expects the 443 listener to be defined (you can use standard Elastic Beanstalk configuration namespaces, or the Elastic Beanstalk console). Then it takes care of modifying the port 80 listener for redirection.

Monitoring an environment

When you are running a production website, it is important to know that your application is available and responding to requests. To assist with monitoring your application's responsiveness, Elastic Beanstalk provides features that monitor statistics about your application and create alerts that trigger when thresholds are exceeded.

Topics

- [Monitoring environment health in the AWS management console \(p. 685\)](#)
- [Basic health reporting \(p. 688\)](#)
- [Enhanced health reporting and monitoring \(p. 691\)](#)
- [Manage alarms \(p. 725\)](#)
- [Viewing an Elastic Beanstalk environment's event stream \(p. 728\)](#)
- [Listing and connecting to server instances \(p. 730\)](#)
- [Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment \(p. 732\)](#)

Monitoring environment health in the AWS management console

You can access operational information about your application from the Elastic Beanstalk console. The console displays your environment's status and application health at a glance. In the console's **Environments** page and in each application's page, the environments on the list are color-coded to indicate status.

To monitor an environment in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Monitoring**.

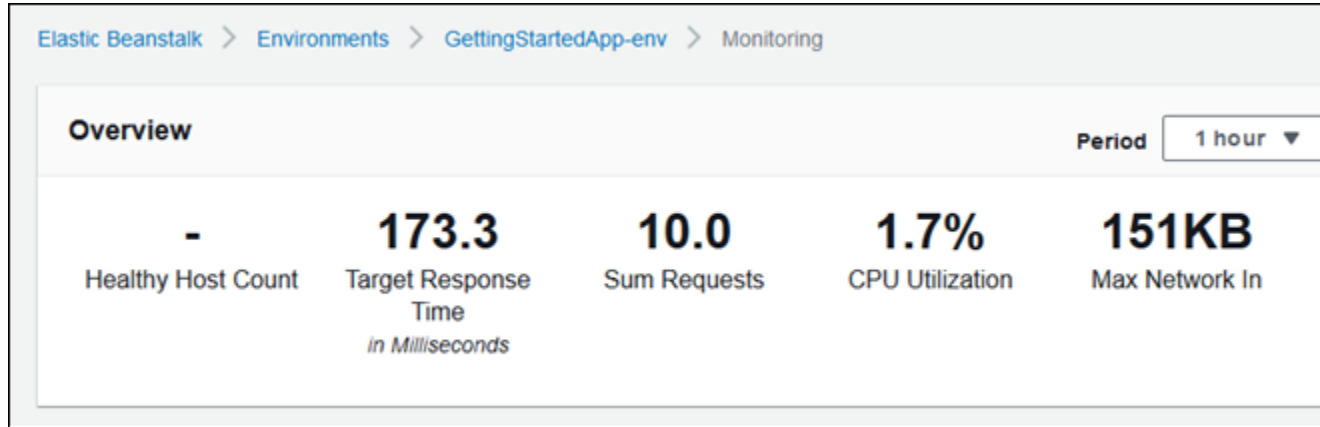
The Monitoring page shows you overall statistics about your environment, such as CPU utilization and average latency. In addition to the overall statistics, you can view monitoring graphs that show resource usage over time. You can click any of the graphs to view more detailed information.

Note

By default, only basic CloudWatch metrics are enabled, which return data in five-minute periods. You can enable more granular one-minute CloudWatch metrics by editing your environment's configuration settings.

Overview

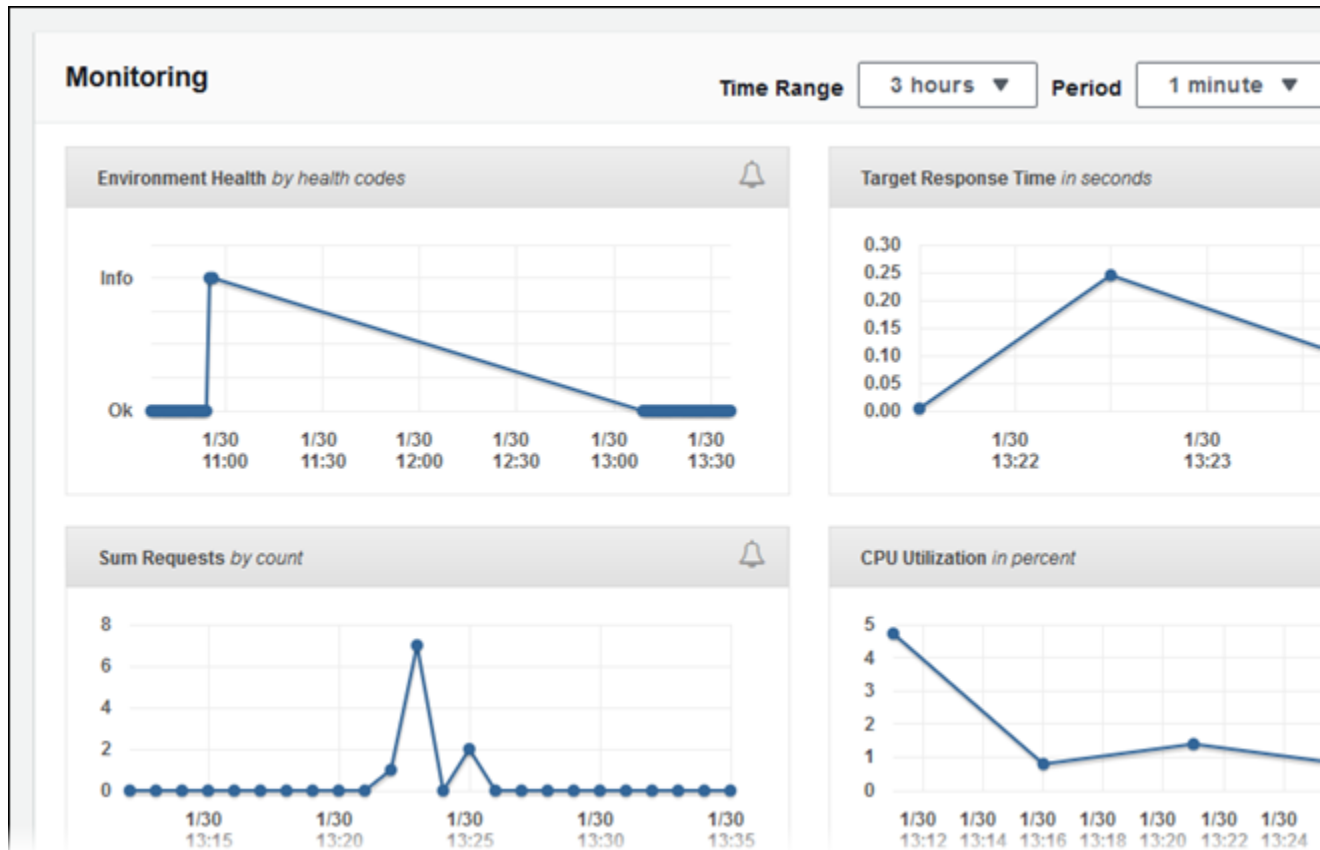
An overview of the environment's health is shown near the top of the screen.



The overview section shows a customizable summary of the activity in your environment over period of time. Choose the **Period** drop-down and select a length of time to view information for a period between one minute and one day.

Monitoring graphs

Below the overview are graphs that show data about overall environment health over a period of time. Choose the **Period** drop-down and select a length of time to set the time between each two plot points to a period between one minute and one day. Choose the **Time Range** drop-down and select a length of time to set the graph time axis to a period between three hours and two weeks.



Customizing the monitoring console

Choose **Edit** next to either monitoring section to customize the information shown.

The screenshot displays the AWS Elastic Beanstalk monitoring console. The top section is titled "Overview" and includes a "Period" dropdown set to "1 hour" and a "Cancel" button. Below this, there are six summary cards: "Healthy Host Count" (value: -), "Target Response Time in Milliseconds" (value: 173.3), "Sum Requests" (value: 10.0), "CPU Utilization" (value: 0.8%), "Max Network In" (value: 54KB), and "Max Network Out" (value: 91KB). Each card has a small "x" icon in the top right corner for removal. Below the overview is the "Add Overview" section, which contains a form with the following fields: "Resource" (dropdown menu showing "AWSEBV2LoadBalancer"), "CloudWatch metric" (text input), "Statistic" (dropdown menu), "Description" (text input), and "Dimensions" (text input). A "Refresh" button is located to the right of the "CloudWatch metric" field. At the bottom of the form, a message states: "Added 'Avg ActiveConnectionCount.' Click Save to preserve your config".

To remove any of the existing items, choose the **x** in the top right corner.

To add an overview or graph

1. Choose **Edit** in the **Overview** or **Monitoring** section.
2. Select a **Resource**. The supported resources are your environment's Auto Scaling group, Elastic Load Balancing load balancer, and the environment itself.
3. Select a **CloudWatch metric** for the resource. See [Publishing Amazon CloudWatch custom metrics for an environment \(p. 714\)](#) for a full list of supported metrics.
4. Select a **Statistic**. The default statistic is the average value of the selected cloudwatch metric during the time range (overview) or between plot points (graph).
5. Enter a **Description**. The description is the label for the item shown in the monitoring console.
6. Choose **Add**.
7. Repeat the previous steps to add more items or choose **Save** to finish modifying the section.

For more information about metrics and dimensions for each resource, see [Amazon CloudWatch Metrics, Namespaces, and Dimensions Reference](#) in the *Amazon CloudWatch User Guide*.

[Elastic Load Balancing](#) and [Amazon EC2](#) metrics are enabled for all environments.

With [enhanced health](#) (p. 691), the EnvironmentHealth metric is enabled and a graph is added to the monitoring console automatically. Additional metrics become available for use in the monitoring console when you enabled them in the environment configuration. Enhanced health also adds the [Health page](#) (p. 702) to the management console.

Note

When you enable additional CloudWatch metrics for your environment, it takes a few minutes for them to start being reported and appear in the list of metrics that you use to add graphs and overview stats.

See [Publishing Amazon CloudWatch custom metrics for an environment](#) (p. 714) for a list of available enhanced health metrics.

Basic health reporting

AWS Elastic Beanstalk uses information from multiple sources to determine if your environment is available and processing requests from the Internet. An environment's health is represented by one of four colors, and is displayed on the [environment overview](#) (p. 353) page of the Elastic Beanstalk console. It's also available from the [DescribeEnvironments](#) API and by calling `eb status` with the [EB CLI](#) (p. 852).

Prior to version 2 Linux platform versions, the only health reporting system was basic health. The basic health reporting system provides information about the health of instances in an Elastic Beanstalk environment based on health checks performed by Elastic Load Balancing for load balanced environments or Amazon Elastic Compute Cloud for single instance environments.

In addition to checking the health of your EC2 instances, Elastic Beanstalk also monitors the other resources in your environment and reports missing or incorrectly configured resources that can cause your environment to become unavailable to users.

Metrics gathered by the resources in your environment is published to Amazon CloudWatch in five minute intervals. This includes operating system metrics from EC2, request metrics from Elastic Load Balancing. You can view graphs based on these CloudWatch metrics on the [Monitoring page](#) (p. 685) of the environment console. For basic health, these metrics are not used to determine an environment's health.

Topics

- [Health colors](#) (p. 688)
- [Elastic Load Balancing health checks](#) (p. 689)
- [Single instance and worker tier environment health checks](#) (p. 689)
- [Additional checks](#) (p. 690)
- [Amazon CloudWatch metrics](#) (p. 690)

Health colors

Elastic Beanstalk reports the health of a web server environment depending on how the application running in it responds to the health check. Elastic Beanstalk uses one of four colors to describe status, as shown in the following table:

Color	Description
Grey	Your environment is being updated.
Green	Your environment has passed the most recent health check. At least one instance in your environment is available and taking requests.
Yellow	Your environment has failed one or more health checks. Some requests to your environment are failing.
Red	Your environment has failed three or more health checks, or an environment resource has become unavailable. Requests are consistently failing.

These descriptions only apply to environments using basic health reporting. See [Health colors and statuses \(p. 706\)](#) for details related to enhanced health.

Elastic Load Balancing health checks

In a load balanced environment, Elastic Load Balancing sends a request to each instance in an environment every 10 seconds to confirm that instances are healthy. By default, the load balancer is configured to open a TCP connection on port 80. If the instance acknowledges the connection, it is considered healthy.

You can choose to override this setting by specifying an existing resource in your application. If you specify a path, such as `/health`, the health check URL is set to `HTTP:80/health`. The health check URL should be set to a path that is always served by your application. If it is set to a static page that is served or cached by the web server in front of your application, health checks will not reveal issues with the application server or web container. For instructions on modifying your health check URL, see [Health check \(p. 477\)](#).

If a health check URL is configured, Elastic Load Balancing expects a GET request that it sends to return a response of `200 OK`. The application fails the health check if it fails to respond within 5 seconds or if it responds with any other HTTP status code. After 5 consecutive health check failures, Elastic Load Balancing takes the instance out of service.

For more information about Elastic Load Balancing health checks, see [Health Check](#) in the *Elastic Load Balancing User Guide*.

Note

Configuring a health check URL does not change the health check behavior of an environment's Auto Scaling group. An unhealthy instance is removed from the load balancer, but is not automatically replaced by Amazon EC2 Auto Scaling unless you configure Amazon EC2 Auto Scaling to use the Elastic Load Balancing health check as a basis for replacing instances. To configure Amazon EC2 Auto Scaling to replace instances that fail an Elastic Load Balancing health check, see [Auto Scaling health check setting \(p. 469\)](#).

Single instance and worker tier environment health checks

In a single instance or worker tier environment, Elastic Beanstalk determines the instance's health by monitoring its Amazon EC2 instance status. Elastic Load Balancing health settings, including HTTP health check URLs, cannot be used in these environment types.

For more information on Amazon EC2 instance status checks, see [Monitoring Instances with Status Checks](#) in the *Amazon EC2 User Guide for Linux Instances*.

Additional checks

In addition to Elastic Load Balancing health checks, Elastic Beanstalk monitors resources in your environment and changes health status to red if they fail to deploy, are not configured correctly, or become unavailable. These checks confirm that:

- The environment's Auto Scaling group is available and has a minimum of at least one instance.
- The environment's security group is available and is configured to allow incoming traffic on port 80.
- The environment CNAME exists and is pointing to the right load balancer.
- In a worker environment, the Amazon Simple Queue Service (Amazon SQS) queue is being polled at least once every three minutes.

Amazon CloudWatch metrics

With basic health reporting, the Elastic Beanstalk service does not publish any metrics to Amazon CloudWatch. The CloudWatch metrics used to produce graphs on the [Monitoring page \(p. 685\)](#) of the environment console are published by the resources in your environment.

For example, EC2 publishes the following metrics for the instances in your environment's Auto Scaling group:

`CPUUtilization`

Percentage of compute units currently in use.

`DiskReadBytes`, `DiskReadOps`, `DiskWriteBytes`, `DiskWriteOps`

Number of bytes read and written, and number of read and write operations.

`NetworkIn`, `NetworkOut`

Number of bytes sent and received.

Elastic Load Balancing publishes the following metrics for your environment's load balancer:

`BackendConnectionErrors`

Number of connection failures between the load balancer and environment instances.

`HTTPCode_Backend_2XX`, `HTTPCode_Backend_4XX`

Number of successful (2XX) and client error (4XX) response codes generated by instances in your environment.

`Latency`

Number of seconds between when the load balancer relays a request to an instance and when the response is received.

`RequestCount`

Number of completed requests.

These lists are not comprehensive. For a full list of metrics that can be reported for these resources, see the following topics in the Amazon CloudWatch Developer Guide:

Metrics

Namespace	Topic
AWS::ElasticLoadBalancing::LoadBalancer	Elastic Load Balancing Metrics and Resources
AWS::AutoScaling::AutoScalingGroup	Amazon Elastic Compute Cloud Metrics and Resources
AWS::SQS::Queue	Amazon SQS Metrics and Resources
AWS::RDS::DBInstance	Amazon RDS Dimensions and Metrics

Worker environment health metric

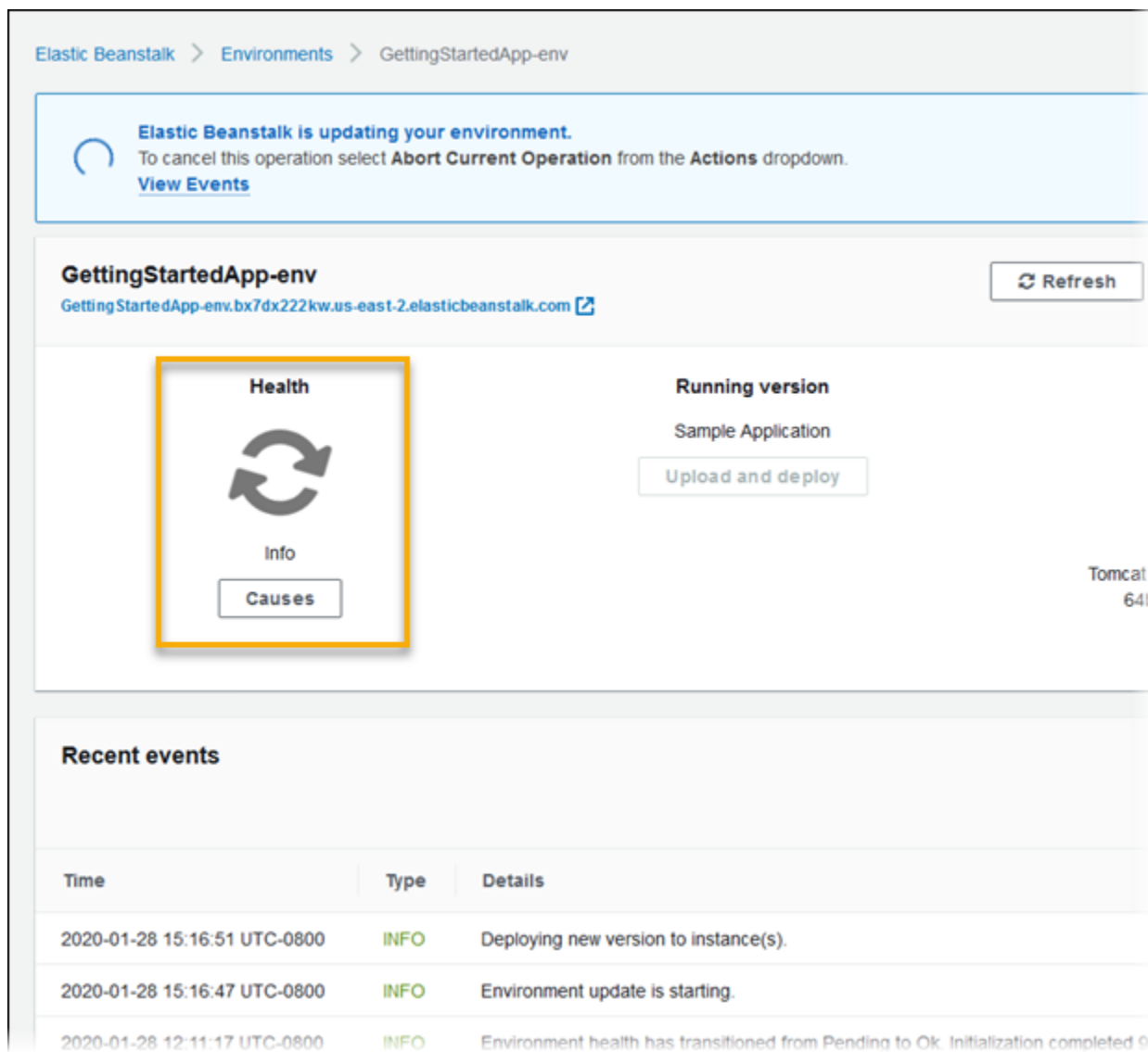
For worker environments only, the SQS daemon publishes a custom metric for environment health to CloudWatch, where a value of 1 is Green. You can review the CloudWatch health metric data in your account using the `ElasticBeanstalk/SQSD` namespace. The metric dimension is `EnvironmentName`, and the metric name is `Health`. All instances publish their metrics to the same namespace.

To enable the daemon to publish metrics, the environment's instance profile must have permission to call `cloudwatch:PutMetricData`. This permission is included in the default instance profile. For more information, see [Managing Elastic Beanstalk instance profiles \(p. 760\)](#).

Enhanced health reporting and monitoring

Enhanced health reporting is a feature that you can enable on your environment to allow AWS Elastic Beanstalk to gather additional information about resources in your environment. Elastic Beanstalk analyzes the information gathered to provide a better picture of overall environment health and aid in the identification of issues that can cause your application to become unavailable.

In addition to changes in how health color works, enhanced health adds a *status* descriptor that provides an indicator of the severity of issues observed when an environment is yellow or red. When more information is available about the current status, you can choose the **Causes** button to view detailed health information on the [health page \(p. 701\)](#).



To provide detailed health information about the Amazon EC2 instances running in your environment, Elastic Beanstalk includes a [health agent \(p. 693\)](#) in the Amazon Machine Image (AMI) for each platform version that supports enhanced health. The health agent monitors web server logs and system metrics and relays them to the Elastic Beanstalk service. Elastic Beanstalk analyzes these metrics and data from Elastic Load Balancing and Amazon EC2 Auto Scaling to provide an overall picture of an environment's health.

In addition to collecting and presenting information about your environment's resources, Elastic Beanstalk monitors the resources in your environment for several error conditions and provides notifications to help you avoid failures and resolve configuration issues. [Factors that influence your environment's health \(p. 694\)](#) include the results of each request served by your application, metrics from your instances' operating system, and the status of the most recent deployment.

You can view health status in real time by using the [environment overview \(p. 701\)](#) page of the Elastic Beanstalk console or the [eb health \(p. 875\)](#) command in the [Elastic Beanstalk command line interface \(p. 852\)](#) (EB CLI). To record and track environment and instance health over time, you can configure your environment to publish the information gathered by Elastic Beanstalk for enhanced

health reporting to Amazon CloudWatch as custom metrics. CloudWatch [charges](#) for custom metrics apply to all metrics other than `EnvironmentHealth`, which is free of charge.

Enhanced health reporting requires a version 2 or newer [platform version \(p. 29\)](#). To monitor resources and publish metrics, your environment must have both an [instance profile and service \(p. 691\)](#) role. The Multicontainer Docker platform doesn't include a web server by default, but can be used with enhanced health reporting if you configure your web server to [provide logs in the proper format \(p. 721\)](#).

Windows platform notes

- This feature isn't available on [Windows Server platform versions](#) earlier than version 2 (v2).
- When you enable enhanced health reporting on a Windows Server environment, don't change [IIS logging configuration](#). For enhanced health monitoring to work correctly, IIS logging must be configured with the **W3C** format and the **ETW event only** or **Both log file and ETW event** log event destinations.

In addition, don't disable or stop the [Elastic Beanstalk health agent \(p. 693\)](#) Windows service on any of your environment's instances. To collect and report enhanced health information on an instance, this service should be enabled and running.

Enhanced health requires the environment to have an instance profile. The instance profile should have roles that provide permissions for your environment instances to collect and report enhanced health information. The first time you create an environment with a v2 platform version in the Elastic Beanstalk console, Elastic Beanstalk prompts you to create the required roles and enables enhanced health reporting by default. Continue reading for details on how enhanced health reporting works, or see [Enabling Elastic Beanstalk enhanced health reporting \(p. 698\)](#) to get started using it right away.

Amazon Linux 2 platforms require instance profiles, so they can support enhanced health unconditionally. When you create an environment using an Amazon Linux 2 platform, Elastic Beanstalk always enables enhanced health. This is true regardless of how you create the environment—using the Elastic Beanstalk console, the EB CLI, the AWS CLI, or the API.

Topics

- [The Elastic Beanstalk health agent \(p. 693\)](#)
- [Factors in determining instance and environment health \(p. 694\)](#)
- [Health check rule customization \(p. 696\)](#)
- [Enhanced health roles \(p. 696\)](#)
- [Enhanced health events \(p. 696\)](#)
- [Enhanced health reporting behavior during updates, deployments, and scaling \(p. 697\)](#)
- [Enabling Elastic Beanstalk enhanced health reporting \(p. 698\)](#)
- [Enhanced health monitoring with the environment management console \(p. 701\)](#)
- [Health colors and statuses \(p. 706\)](#)
- [Instance metrics \(p. 708\)](#)
- [Configuring enhanced health rules for an environment \(p. 710\)](#)
- [Publishing Amazon CloudWatch custom metrics for an environment \(p. 714\)](#)
- [Using enhanced health reporting with the Elastic Beanstalk API \(p. 719\)](#)
- [Enhanced health log format \(p. 721\)](#)
- [Notifications and troubleshooting \(p. 723\)](#)

The Elastic Beanstalk health agent

The Elastic Beanstalk health agent is a daemon process (or service, on Windows environments) that runs on each Amazon EC2 instance in your environment, monitoring operating system and application-level

health metrics and reporting issues to Elastic Beanstalk. The health agent is included in all platform versions starting with version 2.0 of each platform.

The health agent reports similar metrics to those [published to CloudWatch \(p. 690\)](#) by Amazon EC2 Auto Scaling and Elastic Load Balancing as part of [basic health reporting \(p. 688\)](#), including CPU load, HTTP codes, and latency. The health agent, however, reports directly to Elastic Beanstalk, with greater granularity and frequency than basic health reporting.

For basic health, these metrics are published every five minutes and can be monitored with graphs in the environment management console. With enhanced health, the Elastic Beanstalk health agent reports metrics to Elastic Beanstalk every 10 seconds. Elastic Beanstalk uses the metrics provided by the health agent to determine the health status of each instance in the environment and, combined with other [factors \(p. 694\)](#), to determine the overall health of the environment.

The overall health of the environment can be viewed in real time in the environment overview page of the Elastic Beanstalk console, and is published to CloudWatch by Elastic Beanstalk every 60 seconds. You can view detailed metrics reported by the health agent in real time with the [eb health \(p. 875\)](#) command in the [EB CLI \(p. 852\)](#).

For an additional charge, you can choose to publish individual instance and environment-level metrics to CloudWatch every 60 seconds. Metrics published to CloudWatch can then be used to create [monitoring graphs \(p. 687\)](#) in the [environment management console \(p. 353\)](#).

Enhanced health reporting only incurs a charge if you choose to publish enhanced health metrics to CloudWatch. When you use enhanced health, you still get the basic health metrics published for free, even if you don't choose to publish enhanced health metrics.

See [Instance metrics \(p. 708\)](#) for details on the metrics reported by the health agent. For details on publishing enhanced health metrics to CloudWatch, see [Publishing Amazon CloudWatch custom metrics for an environment \(p. 714\)](#).

Factors in determining instance and environment health

In addition to the basic health reporting system checks, including [Elastic Load Balancing health checks \(p. 689\)](#) and [resource monitoring \(p. 690\)](#), Elastic Beanstalk enhanced health reporting gathers additional data about the state of the instances in your environment. This includes operating system metrics, server logs, and the state of ongoing environment operations such as deployments and updates. The Elastic Beanstalk health reporting service combines information from all available sources and analyzes it to determine the overall health of the environment.

Operations and commands

When you perform an operation on your environment, such as deploying a new version of an application, Elastic Beanstalk makes several changes that affect the environment's health status.

For example, when you deploy a new version of an application to an environment that is running multiple instances, you might see messages similar to the following as you monitor the environment's health [with the EB CLI \(p. 875\)](#).

```
id          status      cause
Overall    Info        Command is executing on 3 out of 5 instances
i-bb65c145 Pending     91 % of CPU is in use. 24 % in I/O wait
            Performing application deployment (running for 31 seconds)
i-ba65c144 Pending     Performing initialization (running for 12 seconds)
i-f6a2d525 Ok          Application deployment completed 23 seconds ago and took 26
seconds
```

```
i-e8a2d53b    Pending    94 % of CPU is in use. 52 % in I/O wait
              Performing application deployment (running for 33 seconds)
i-e81cca40    Ok
```

In this example, the overall status of the environment is `Ok` and the cause of this status is that the *Command is executing on 3 out of 5 instances*. Three of the instances in the environment have the status *Pending*, indicating that an operation is in progress.

When an operation completes, Elastic Beanstalk reports additional information about the operation. For the example, Elastic Beanstalk displays the following information about an instance that has already been updated with the new version of the application:

```
i-f6a2d525    Ok          Application deployment completed 23 seconds ago and took 26
seconds
```

Instance health information also includes details about the most recent deployment to each instance in your environment. Each instance reports a deployment ID and status. The deployment ID is an integer that increases by one each time you deploy a new version of your application or change settings for on-instance configuration options, such as environment variables. You can use the deployment information to identify instances that are running the wrong version of your application after a failed [rolling deployment \(p. 400\)](#).

In the cause column, Elastic Beanstalk includes informational messages about successful operations and other healthy states across multiple health checks, but they don't persist indefinitely. Causes for unhealthy environment statuses persist until the environment returns to a healthy status.

Command timeout

Elastic Beanstalk applies a command timeout from the time an operation begins to allow an instance to transition into a healthy state. This command timeout is set in your environment's update and deployment configuration (in the [aws:elasticbeanstalk:command \(p. 571\)](#) namespace) and defaults to 10 minutes.

During rolling updates, Elastic Beanstalk applies a separate timeout to each batch in the operation. This timeout is set as part of the environment's rolling update configuration (in the [aws:autoscaling:updatepolicy:rollingupdate \(p. 564\)](#) namespace). If all instances in the batch are healthy within the command timeout, the operation continues to the next batch. If not, the operation fails.

Note

If your application does not pass health checks with an **OK** status but is stable at a different level, you can set the `HealthCheckSuccessThreshold` option in the [aws:elasticbeanstalk:command namespace \(p. 571\)](#) to change the level at which Elastic Beanstalk considers an instance to be healthy.

For a web server environment to be considered healthy, each instance in the environment or batch must pass 12 consecutive health checks over the course of two minutes. For a worker tier environment, each instance must pass 18 health checks. Before the command times out, Elastic Beanstalk doesn't lower an environment's health status when health checks fail. If the instances in the environment become healthy within the command timeout, the operation succeeds.

HTTP requests

When no operation is in progress on an environment, the primary source of information about instance and environment health is the web server logs for each instance. To determine the health of an instance and the overall health of the environment, Elastic Beanstalk considers the number of requests, the result of each request, and the speed at which each request was resolved.

On Linux-based platforms, Elastic Beanstalk reads and parses web server logs to get information about HTTP requests. On the Windows Server platform, Elastic Beanstalk receives this information [directly from the IIS web server \(p. 710\)](#).

Your environment might not have an active web server. For example, the Multicontainer Docker platform doesn't include a web server. Other platforms include a web server, and your application might disable it. In these cases, your environment requires additional configuration to provide the [Elastic Beanstalk health agent \(p. 693\)](#) with logs in the format that it needs to relay health information to the Elastic Beanstalk service. See [Enhanced health log format \(p. 721\)](#) for details.

Operating system metrics

Elastic Beanstalk monitors operating system metrics reported by the health agent to identify instances that are consistently low on system resources.

See [Instance metrics \(p. 708\)](#) for details on the metrics reported by the health agent.

Health check rule customization

Elastic Beanstalk enhanced health reporting relies on a set of rules to determine the health of your environment. Some of these rules might not be appropriate for your particular application. A common case is an application that returns frequent HTTP 4xx errors by design. Elastic Beanstalk, using one of its default rules, concludes that something is going wrong, and changes your environment health status from OK to Warning, Degraded, or Severe, depending on the error rate. To handle this case correctly, Elastic Beanstalk allows you to configure this rule and ignore application HTTP 4xx errors. For details, see [Configuring enhanced health rules for an environment \(p. 710\)](#).

Enhanced health roles

Enhanced health reporting requires two roles—a service role for Elastic Beanstalk and an instance profile for the environment. The service role allows Elastic Beanstalk to interact with other AWS services on your behalf to gather information about the resources in your environment. The instance profile allows the instances in your environment to write logs to Amazon S3.

When you create an Elastic Beanstalk environment in the Elastic Beanstalk console, the console prompts you to create an instance profile and service role with appropriate permissions. The EB CLI also assists you in creating these roles when you call `eb create` to create an environment.

If you use the API, an SDK, or the AWS CLI to create environments, you must create these roles in advance, and specify them during environment creation to use enhanced health. For instructions on creating appropriate roles for your environments, see [Service roles, instance profiles, and user policies \(p. 20\)](#).

Enhanced health events

The enhanced health system generates events when an environment transitions between states. The following example shows events output by an environment transitioning between **Info**, **OK**, and **Severe** states.

Recent events		
Time	Type	Details
2020-01-28 16:06:04 UTC-0800	INFO	Environment health has transitioned from Severe to Ok.
2020-01-28 16:05:04 UTC-0800	INFO	Added instance [i-03280193ba1ba4171] to your environment.
2020-01-28 16:05:04 UTC-0800	WARN	Removed instance [i-0a4a27bbf9994ba5] from your environment due to a EC2 health check failure
2020-01-28 16:03:04 UTC-0800	WARN	Environment health has transitioned from Ok to Severe. ELB processes are not healthy on all instances. ELB health is failing or not available for all instances.
2020-01-28 15:19:06 UTC-0800	INFO	Environment health has transitioned from Info to Ok. Application update completed 75 seconds ago

When transitioning to a worse state, the enhanced health event includes a message indicating the transition cause.

Not all changes in status at an instance level cause Elastic Beanstalk to emit an event. To prevent false alarms, Elastic Beanstalk generates a health-related event only if an issue persists across multiple checks.

Real-time environment-level health information, including status, color, and cause, is available in the [environment overview \(p. 355\)](#) page of the Elastic Beanstalk console and the [EB CLI \(p. 852\)](#). By attaching the EB CLI to your environment and running the [eb health \(p. 875\)](#) command, you can also view real-time statuses from each of the instances in your environment.

Enhanced health reporting behavior during updates, deployments, and scaling

Enabling enhanced health reporting can affect how your environment behaves during configuration updates and deployments. Elastic Beanstalk won't complete a batch of updates until all of the instances pass health checks consistently. Also, because enhanced health reporting applies a higher standard for health and monitors more factors, instances that pass basic health reporting's [ELB health check \(p. 689\)](#) won't necessarily pass with enhanced health reporting. See the topics on [rolling configuration updates \(p. 409\)](#) and [rolling deployments \(p. 400\)](#) for details on how health checks affect the update process.

Enhanced health reporting can also highlight the need to set a proper [health check URL \(p. 477\)](#) for Elastic Load Balancing. When your environment scales up to meet demand, new instances will start taking requests as soon as they pass enough ELB health checks. If a health check URL is not configured, this can be as little as 20 seconds after a new instance is able to accept a TCP connection.

If your application hasn't finished starting up by the time the load balancer declares it healthy enough to receive traffic, you will see a flood of failed requests, and your environment will start to fail health checks. A health check URL that hits a path served by your application can prevent this issue. ELB health checks won't pass until a GET request to the health check URL returns a 200 status code.

Enabling Elastic Beanstalk enhanced health reporting

New environments created with the latest [platform versions \(p. 29\)](#) include the AWS Elastic Beanstalk [health agent \(p. 693\)](#), which supports enhanced health reporting. If you create your environment in the Elastic Beanstalk console or with the EB CLI, enhanced health is enabled by default. You can also set your health reporting preference in your application's source code using [configuration files \(p. 600\)](#).

Enhanced health reporting requires an [instance profile \(p. 21\)](#) and [service role \(p. 20\)](#) with the standard set of permissions. When you create an environment in the Elastic Beanstalk console, Elastic Beanstalk creates the required roles automatically. See [Getting started using Elastic Beanstalk \(p. 3\)](#) for instructions on creating your first environment.

Topics

- [Enabling enhanced health reporting using the Elastic Beanstalk console \(p. 698\)](#)
- [Enabling enhanced health reporting using the EB CLI \(p. 700\)](#)
- [Enabling enhanced health reporting using a configuration file \(p. 701\)](#)

Enabling enhanced health reporting using the Elastic Beanstalk console

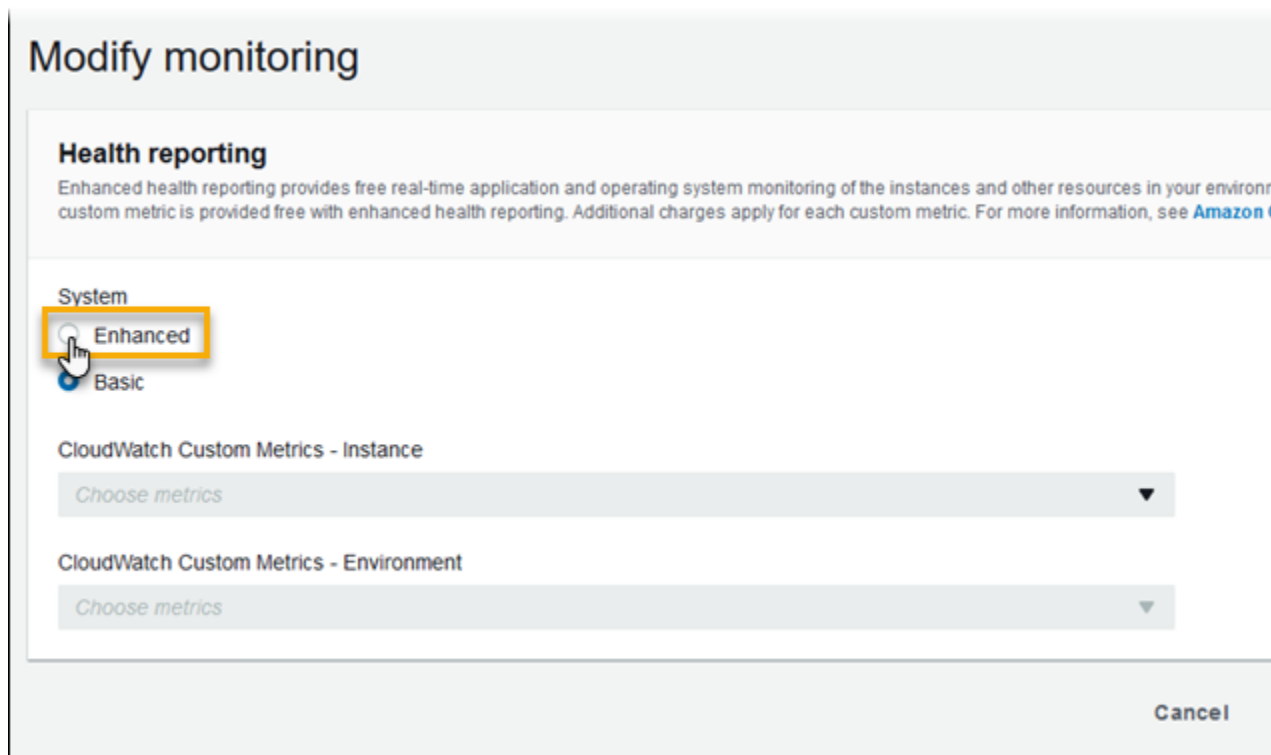
To enable enhanced health reporting in a running environment using the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Monitoring** configuration category, choose **Edit**.
5. Under **Health reporting**, for **System**, choose **Enhanced**.



Note

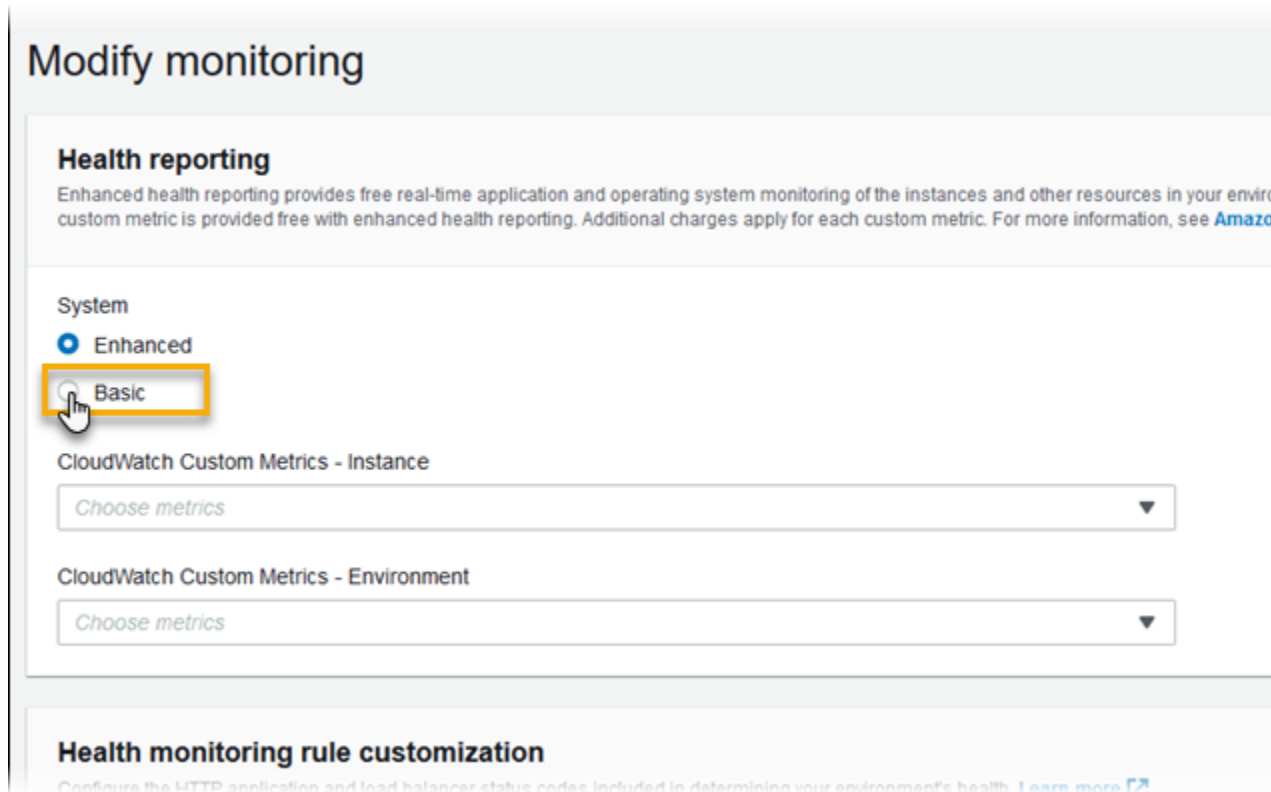
The options for enhanced health reporting don't appear if you are using an [unsupported platform or version](#) (p. 691).

6. Choose **Apply**.

The Elastic Beanstalk console defaults to enhanced health reporting when you create a new environment with a version 2 (v2) platform version. You can disable enhanced health reporting by changing the health reporting option during environment creation.

To disable enhanced health reporting when creating an environment using the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. [Create an application](#) (p. 334) or select an existing one.
3. [Create an environment](#) (p. 363). On the **Create a new environment** page, before choosing **Create environment**, choose **Configure more options**.
4. In the **Monitoring** configuration category, choose **Edit**.
5. Under **Health reporting**, for **System**, choose **Basic**.



6. Choose **Save**.

Enabling enhanced health reporting using the EB CLI

When you create a new environment with the **eb create** command, the EB CLI enables enhanced health reporting by default and applies the default instance profile and service role.

You can specify a different service role by name by using the `--service-role` option.

If you have an environment running with basic health reporting on a v2 platform version and you want to switch to enhanced health, follow these steps.

To enable enhanced health on a running environment using the EB CLI (p. 852)

1. Use the **eb config** command to open the configuration file in the default text editor.

```
~/project$ eb config
```

2. Locate the `aws:elasticbeanstalk:environment` namespace in the settings section. Ensure that the value of `ServiceRole` is not null and that it matches the name of your [service role \(p. 20\)](#).

```
aws:elasticbeanstalk:environment:  
  EnvironmentType: LoadBalanced  
  ServiceRole: aws-elasticbeanstalk-service-role
```

3. Under the `aws:elasticbeanstalk:healthreporting:system:` namespace, change the value of `SystemType` to **enhanced**.

```
aws:elasticbeanstalk:healthreporting:system:
```

```
SystemType: enhanced
```

4. Save the configuration file and close the text editor.
5. The EB CLI starts an environment update to apply your configuration changes. Wait for the operation to complete or press **Ctrl+C** to exit safely.

```
~/project$ eb config  
Printing Status:  
INFO: Environment update is starting.  
INFO: Health reporting type changed to ENHANCED.  
INFO: Updating environment no-role-test's configuration settings.
```

Enabling enhanced health reporting using a configuration file

You can enable enhanced health reporting by including a [configuration file \(p. 600\)](#) in your source bundle. The following example shows a configuration file that enables enhanced health reporting and assigns the default service and instance profile to the environment:

Example `.ebextensions/enhanced-health.config`

```
option_settings:  
  aws:elasticbeanstalk:healthreporting:system:  
    SystemType: enhanced  
  aws:autoscaling:launchconfiguration:  
    IamInstanceProfile: aws-elasticbeanstalk-ec2-role  
  aws:elasticbeanstalk:environment:  
    ServiceRole: aws-elasticbeanstalk-service-role
```

If you created your own instance profile or service role, replace the highlighted text with the names of those roles.

Enhanced health monitoring with the environment management console

When you enable enhanced health reporting in AWS Elastic Beanstalk, you can monitor environment health in the [environment management console \(p. 353\)](#).

Topics

- [Environment overview \(p. 701\)](#)
- [Environment health page \(p. 702\)](#)
- [Monitoring page \(p. 706\)](#)

Environment overview

The [environment overview \(p. 355\)](#) displays the [health status \(p. 706\)](#) of the environment and lists events that provide information about recent changes in health status.

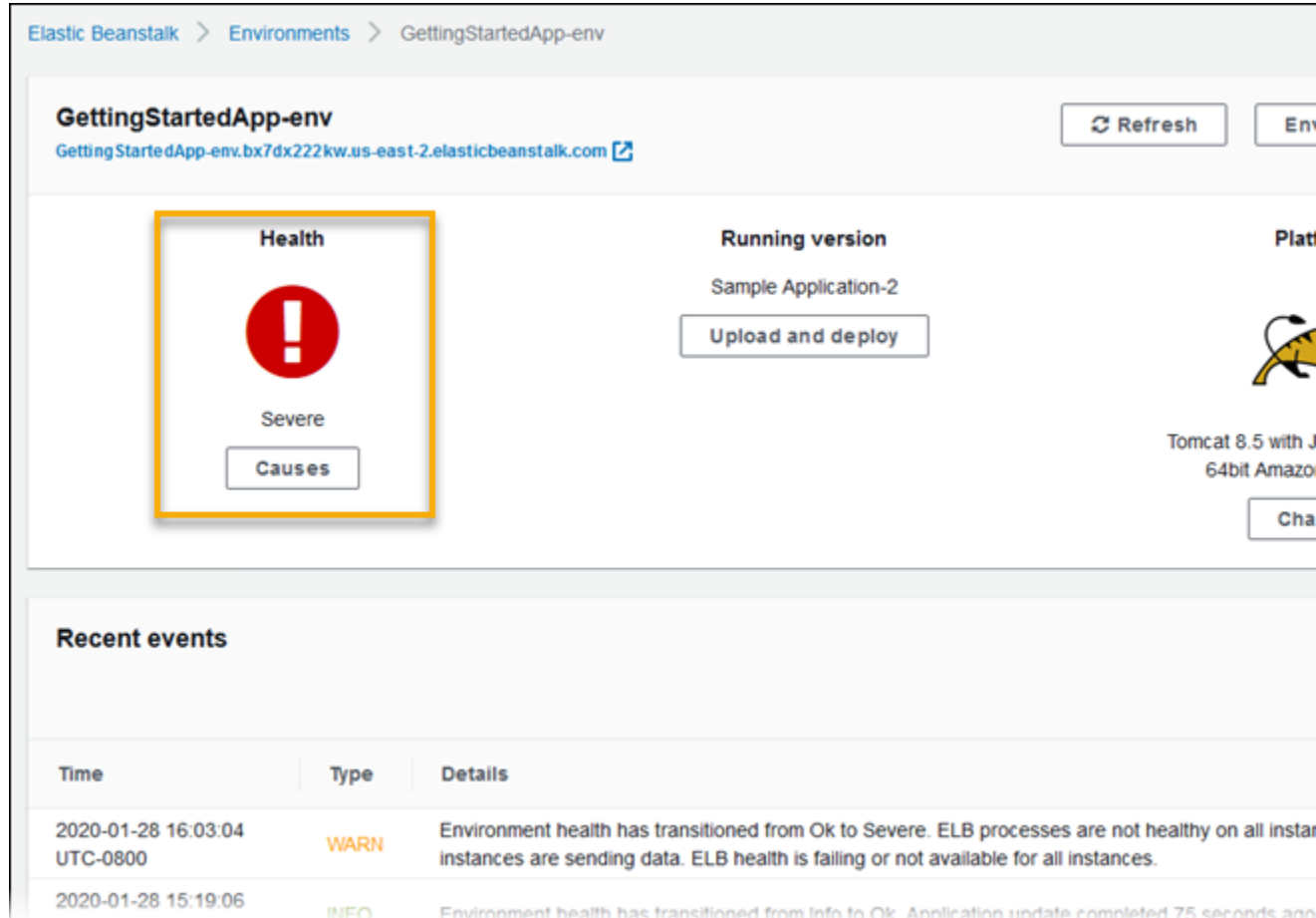
To view the environment overview

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

For detailed information about the current environment's health, open the **Health** page by choosing **Causes**. Alternatively, in the navigation pane, choose **Health**.



Environment health page

The **Health** page displays health status, metrics, and causes for the environment and for each Amazon EC2 instance in the environment.

Note

Elastic Beanstalk displays the **Health** page only if you have [enabled enhanced health monitoring \(p. 698\)](#) for the environment.

The following image shows the **Health** page for a Linux environment.

Elastic Beanstalk > Environments > GettingStartedApp-env > Health

Enhanced health overview
Instances: 2 Total, 2 Ok
[Learn more](#) about enhanced health.

Instance ID	Status	Running	Deployment ID	Requests/sec	2xx Responses	3xx Responses	4xx Responses	5xx Responses	P99 Latency	P99 Latency
Overall	Ok	N/A	N/A	0.4	100%	0.0%	0.0%	0.0%	0.002	0.00
i-00227807c4c4a1334	Ok	2 hours	3	0.2	2	0	0	0	0.002	0.00
i-03280193ba1ba4171	Ok	19 days	3	0.2	2	0	0	0	0.001	0.00

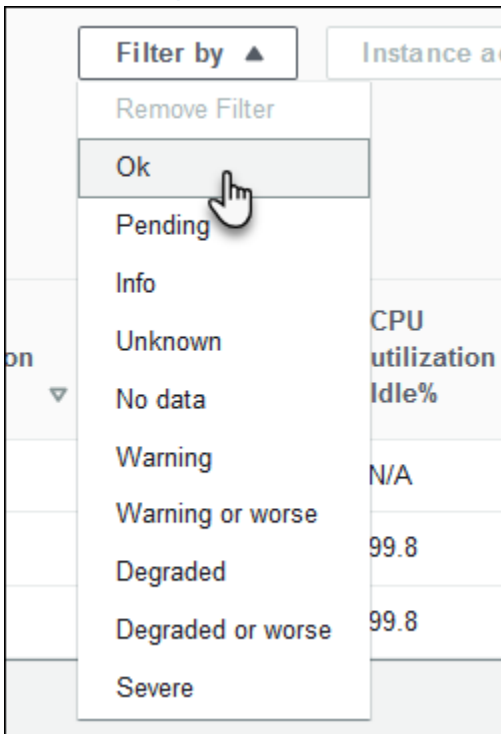
The following image shows the **Health** page for a Windows environment. Notice that CPU metrics are different from those on a Linux environment.

Elastic Beanstalk > Environments > GettingStartedApp-Windows > Health

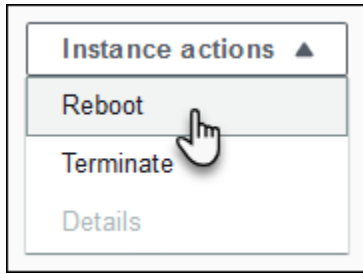
Enhanced health overview
Instances: 1 Total, 1 Ok
[Learn more](#) about enhanced health.

Instance ID	Status	Running	Deployment ID	Requests/sec	2xx Responses	3xx Responses	4xx Responses	5xx Responses	P99 Latency
Overall	Ok	N/A	N/A	0.2	100%	0.0%	0.0%	0.0%	0.015
i-04b33b4c983d318af	Ok	20 days	1	0.2	2	0	0	0	0.015

At the top of the page you can see the total number of environment instances, as well as the number of instances per status. To display only instances that have a particular status, choose **Filter By**, and then select a **status** (p. 706).




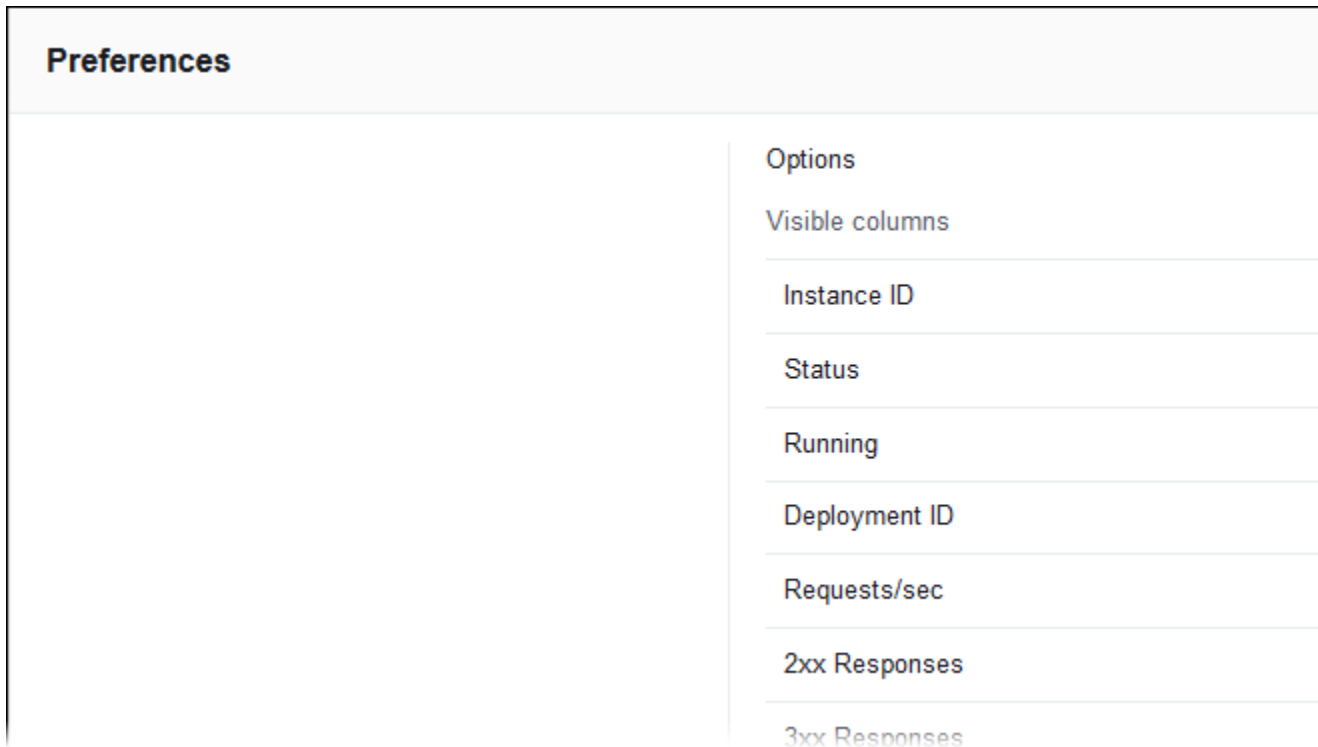
To reboot or terminate an unhealthy instance, choose **Instance Actions**, and then choose **Reboot** or **Terminate**.



Elastic Beanstalk updates the **Health** page every 10 seconds. It reports information about environment and instance health.

For each Amazon EC2 instance in the environment, the page displays the instance's ID and [status \(p. 706\)](#), the amount of time since the instance was launched, the ID of the most recent deployment executed on the instance, the responses and latency of requests that the instance served, and load and CPU utilization information. The **Overall** row displays average response and latency information for the entire environment.

The page displays many details in a very wide table. To hide some of the columns, choose  (**Preferences**). Select or clear column names, and then choose **Confirm**.



Choose the **Instance ID** of any instance to view more information about the instance, including its Availability Zone and instance type.

	Instance ID ▾	Status ▲	Running ▾	Deployment ID ▾	Reque
●	Overall	Ok	N/A	N/A	0.2
○	i-00227807c4c4a1334	Ok	1 day	3	0.1
○	i-03280193ba1ba4171	Ok	20 days	3	0.1



i-00227807c4c4a1334 details ×

Instance ID: i-00227807c4c4a1334
Instance type: t2.micro
Availability zone: us-east-2b

Choose the **Deployment ID** of any instance to view information about the last deployment (p. 397) to the instance.

	Instance ID ▾	Status ▲	Running ▾	Deployment ID ▾	Reque
●	Overall	Ok	N/A	N/A	0.2
○	i-00227807c4c4a1334	Ok	1 day	3	0.1
○	i-03280193ba1ba4171	Ok	20 days	3	0.1



Deployment details ×

Deployment ID 3
Version: Sample Application-3
Deployed 1 day ago

Deployment information includes the following:

- **Deployment ID**—The unique identifier for the [deployment \(p. 397\)](#). Deployment IDs starts at 1 and increase by one each time you deploy a new application version or change configuration settings that affect the software or operating system running on the instances in your environment.
- **Version**—The version label of the application source code used in the deployment.
- **Status**—The status of the deployment, which can be `In Progress`, `Deployed`, or `Failed`.
- **Time**— For in-progress deployments, the time that the deployment started. For completed deployments, the time that the deployment ended.

If you [enable X-Ray integration \(p. 521\)](#) on your environment and instrument your application with the AWS X-Ray SDK, the **Health** page adds links to the AWS X-Ray console in the overview row.

Requests/sec ▾	2xx Responses ▾	3xx Responses ▾	4xx Responses ▾	5xx Responses ▾	P99 Latency ▾	P90 Latency ▾	P75 Latency ▾	P50 Latency ▾
100% 🔗	0.0%	0.0%	0.0%	0.0%	0.002 🔗	0.002 🔗	0.002 🔗	0.002 🔗
1	0	0	0	0	0.002	0.002	0.002	0.002
1	0	0	0	0	0.001	0.001	0.001	0.001

Choose a link to view traces related to the highlighted statistic in the AWS X-Ray console.

Monitoring page

The **Monitoring** page displays summary statistics and graphs for the custom Amazon CloudWatch metrics generated by the enhanced health reporting system. See [Monitoring environment health in the AWS management console \(p. 685\)](#) for instructions on adding graphs and statistics to this page.

Health colors and statuses

Enhanced health reporting represents instance and overall environment health by using four colors, similar to [basic health reporting \(p. 688\)](#). Enhanced health reporting also provides seven health statuses, which are single-word descriptors that provide a better indication of the state of your environment.

Instance status and environment status

Every time Elastic Beanstalk runs a health check on your environment, enhanced health reporting checks the health of each instance in your environment by analyzing all of [the data \(p. 694\)](#) available. If any lower-level check fails, Elastic Beanstalk downgrades the health of the instance.

Elastic Beanstalk displays the health information for the overall environment (color, status, and cause) in the [environment management console \(p. 353\)](#). This information is also available in the EB CLI. Health status and cause messages for individual instances are updated every 10 seconds and are available from the [EB CLI \(p. 852\)](#) when you view health status with [eb health \(p. 875\)](#).

Elastic Beanstalk uses changes in instance health to evaluate environment health, but does not immediately change environment health status. When an instance fails health checks at least three times in any one-minute period, Elastic Beanstalk may downgrade the health of the environment. Depending on the number of instances in the environment and the issue identified, one unhealthy instance can cause Elastic Beanstalk to display an informational message or to change the environment's health status from green (**OK**) to yellow (**Warning**) or red (**Degraded** or **Severe**).

OK (green)

This status is displayed when:

- An instance is passing health checks and the health agent is not reporting any problems.
- Most instances in the environment are passing health checks and the health agent is not reporting major issues.
- An instance is passing health checks and is completing requests normally.

Example: Your environment was recently deployed and is taking requests normally. Five percent of requests are returning 400 series errors. Deployment completed normally on each instance.

Message (instance): Application deployment completed 23 seconds ago and took 26 seconds.

Warning (yellow)

This status is displayed when:

- The health agent is reporting a moderate number of request failures or other issues for an instance or environment.
- An operation is in progress on an instance and is taking a very long time.

Example: One instance in the environment has a status of **Severe**.

Message (environment): Impaired services on 1 out of 5 instances.

Degraded (red)

This status is displayed when the health agent is reporting a high number of request failures or other issues for an instance or environment.

Example: environment is in the process of scaling up to 5 instances.

Message (environment): 4 active instances is below Auto Scaling group minimum size 5.

Severe (red)

This status is displayed when the health agent is reporting a very high number of request failures or other issues for an instance or environment.

Example: Elastic Beanstalk is unable to contact the load balancer to get instance health.

Message (environment): ELB health is failing or not available for all instances. None of the instances are sending data. Unable to assume role "arn:aws:iam::123456789012:role/aws-elasticbeanstalk-service-role". Verify that the role exists and is configured correctly.

Message (Instances): Instance ELB health has not been available for 37 minutes. No data. Last seen 37 minutes ago.

Info (green)

This status is displayed when:

- An operation is in progress on an instance.
- An operation is in progress on several instances in an environment.

Example: A new application version is being deployed to running instances.

Message (environment): Command is executing on 3 out of 5 instances.

Message (instance): Performing application deployment (running for 3 seconds).

Pending (grey)

This status is displayed when an operation is in progress on an instance within the [command timeout](#) (p. 695).

Example: You have recently created the environment and instances are being bootstrapped.

Message: Performing initialization (running for 12 seconds).

Unknown (grey)

This status is displayed when Elastic Beanstalk and the health agent are reporting an insufficient amount of data on an instance.

Example: No data is being received.

Suspended (grey)

This status is displayed when Elastic Beanstalk stopped monitoring the environment's health. The environment might not work correctly. Some severe health conditions, if they last a long time, cause Elastic Beanstalk to transition the environment to the **Suspended** status.

Example: Elastic Beanstalk can't access the environment's [service role](#) (p. 764).

Example: The [Auto Scaling group](#) (p. 457) that Elastic Beanstalk created for the environment has been deleted.

Message: Environment health has transitioned from **OK** to **Severe**. There are no instances. Auto Scaling group desired capacity is set to 1.

Instance metrics

Instance metrics provide information about the health of instances in your environment. The [Elastic Beanstalk health agent](#) (p. 693) runs on each instance. It gathers and relays metrics about instances to Elastic Beanstalk, which analyzes the metrics to determine the health of the instances in your environments.

The on-instance Elastic Beanstalk health agent gathers metrics about instances from web servers and the operating system. To get web server information on Linux-based platforms, Elastic Beanstalk reads and parses web server logs. On the Windows Server platform, Elastic Beanstalk receives this information directly from the IIS web server. Web servers provide information about incoming HTTP requests: how many requests came in, how many resulted in errors, and how long they took to resolve. The operating system provides snapshot information about the state of the instances' resources: the CPU load and distribution of time spent on each process type.

The health agent gathers web server and operating system metrics and relays them to Elastic Beanstalk every 10 seconds. Elastic Beanstalk analyzes the data and uses the results to update the health status for each instance and the environment.

Topics

- [Web server metrics](#) (p. 709)

- [Operating system metrics \(p. 709\)](#)
- [Web server metrics capture in IIS on Windows server \(p. 710\)](#)

Web server metrics

On Linux-based platforms, the Elastic Beanstalk health agent reads web server metrics from logs generated by the web container or server that processes requests on each instance in your environment. Elastic Beanstalk platforms are configured to generate two logs: one in human-readable format and one in machine-readable format. The health agent relays machine-readable logs to Elastic Beanstalk every 10 seconds.

For more information on the log format used by Elastic Beanstalk, see [Enhanced health log format \(p. 721\)](#).

On the Windows Server platform, Elastic Beanstalk adds a module to the IIS web server's request pipeline and captures metrics about HTTP request times and response codes. The module sends these metrics to the on-instance health agent using a high-performance interprocess communication (IPC) channel. For implementation details, see [Web server metrics capture in IIS on Windows server \(p. 710\)](#).

Reported Web Server Metrics

RequestCount

Number of requests handled by the web server per second over the last 10 seconds. Shown as an average r/sec (requests per second) in the EB CLI and [Environment health page \(p. 702\)](#).

Status2xx, Status3xx, Status4xx, Status5xx

Number of requests that resulted in each type of status code over the last 10 seconds. For example, successful requests return a 200 OK, redirects are a 301, and a 404 is returned if the URL entered doesn't match any resources in the application.

The EB CLI and [Environment health page \(p. 702\)](#) show these metrics both as a raw number of requests for instances, and as a percentage of overall requests for environments.

p99.9, p99, p95, p90, p85, p75, p50, p10

Average latency for the slowest x percent of requests over the last 10 seconds, where x is the difference between the number and 100. For example, p99 1.403 indicates the slowest 1% of requests over the last 10 seconds had an average latency of 1.403 seconds.

Operating system metrics

The Elastic Beanstalk health agent reports the following operating system metrics. Elastic Beanstalk uses these metrics to identify instances that are under sustained heavy load. The metrics differ by operating system.

Reported operating system metrics—Linux

Running

The amount of time that has passed since the instance was launched.

Load 1, Load 5

Load average in the last one-minute and five-minute periods. Shown as a decimal value indicating the average number of processes running during that time. If the number shown is higher than the number of vCPUs (threads) available, then the remainder is the average number of processes that were waiting.

For example, if your instance type has four vCPUs, and the load is 4.5, there was an average of .5 processes in wait during that time period, equivalent to one process waiting 50 percent of the time.

```
User %, Nice %, System %, Idle %, I/O Wait %
```

Percentage of time that the CPU has spent in each state over the last 10 seconds.

Reported operating system metrics—Windows

Running

The amount of time that has passed since the instance was launched.

```
% User Time, % Privileged Time, % Idle Time
```

Percentage of time that the CPU has spent in each state over the last 10 seconds.

Web server metrics capture in IIS on Windows server

On the Windows Server platform, Elastic Beanstalk adds a module to the IIS web server's request pipeline and captures metrics about HTTP request times and response codes. The module sends these metrics to the on-instance health agent using a high-performance interprocess communication (IPC) channel. The health agent aggregates these metrics, combines them with operating system metrics, and sends them to the Elastic Beanstalk service.

Implementation details

To capture metrics from IIS, Elastic Beanstalk implements a managed `IHttpModule`, and subscribes to the `BeginRequest` and `EndRequest` events. This enables the module to report HTTP request latency and response codes for all web requests handled by IIS. To add the module to the IIS request pipeline, Elastic Beanstalk registers the module in the `<modules>` section of the IIS configuration file, `%windir%\System32\inetsrv\config\applicationHost.config`.

The Elastic Beanstalk module in IIS sends the captured web request metrics to the on-instance health agent, which is a Windows service named `HealthD`. To send this data, the module uses `NetNamedPipeBinding`, which provides a secure and reliable binding that is optimized for on-machine communication.

Configuring enhanced health rules for an environment

AWS Elastic Beanstalk enhanced health reporting relies on a set of rules to determine the health of your environment. Some of these rules might not be appropriate for your particular application. The following are some common examples:

- You use client-side test tools. In this case, frequent HTTP client (4xx) errors are expected.
- You use [AWS WAF](#) in conjunction with your environment's Application Load Balancer to block unwanted incoming traffic. In this case, Application Load Balancer returns HTTP 403 for each rejected incoming message.

By default, Elastic Beanstalk includes all application HTTP 4xx errors when determining the environment's health. It changes your environment health status from **OK** to **Warning**, **Degraded**, or **Severe**, depending on the error rate. To correctly handle cases such as the examples we mentioned, Elastic Beanstalk enables you to configure some enhanced health rules. You can choose to ignore application HTTP 4xx errors on the environment's instances, or to ignore HTTP 4xx errors returned by the environment's load balancer. This topic describes how to make these configuration changes.

Note

Currently, these are the only available enhanced health rule customizations. You can't configure enhanced health to ignore other HTTP errors in addition to 4xx.

Configuring enhanced health rules using the Elastic Beanstalk console

You can use the Elastic Beanstalk console to configure enhanced health rules in your environment.

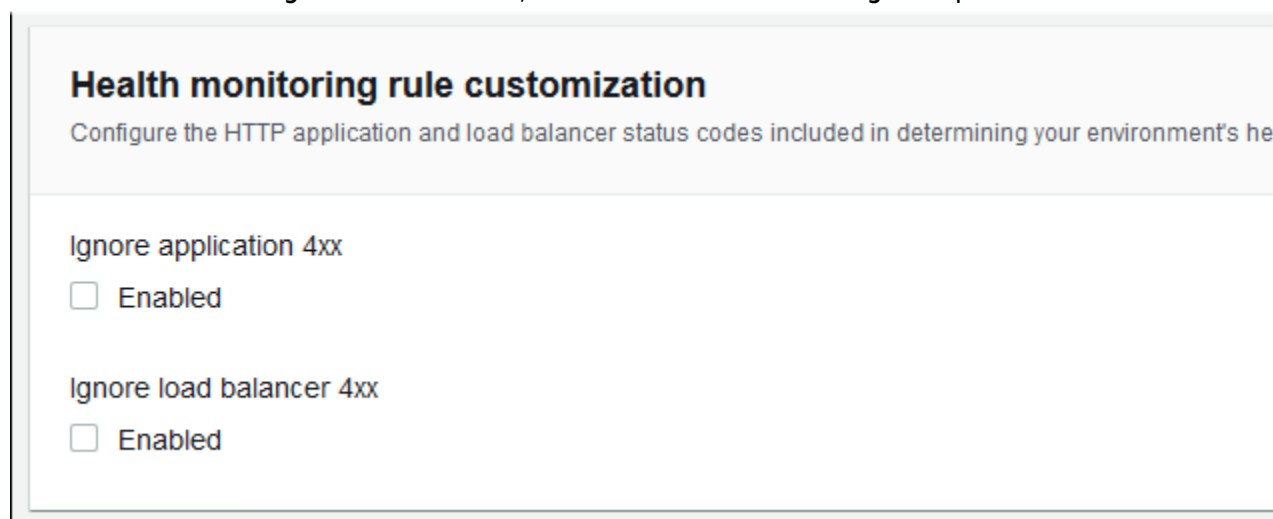
To configure HTTP 4xx status code checking using the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Monitoring** configuration category, choose **Edit**.
5. Under **Health monitoring rule customization**, enable or disable the desired **Ignore** options.



6. Choose **Apply**.

Configuring enhanced health rules using the EB CLI

You can use the EB CLI to configure enhanced health rules by saving your environment's configuration locally, adding an entry that configures enhanced health rules, and then uploading the configuration to Elastic Beanstalk. You can apply the saved configuration to an environment during or after creation.

To configure HTTP 4xx status code checking using the EB CLI and saved configurations

1. Initialize your project folder with **eb init** (p. 860).
2. Create an environment by running the **eb create** (p. 864) command.
3. Save a configuration template locally by running the **eb config save** command. The following example uses the `--cfg` option to specify the name of the configuration.

```
$ eb config save --cfg 01-base-state
```

```
Configuration saved at: ~/project/.elasticbeanstalk/saved_configs/01-base-state.cfg.yml
```

4. Open the saved configuration file in a text editor.
5. Under `OptionSettings > aws:elasticbeanstalk:healthreporting:system:`, add a `ConfigDocument` key to list each enhanced health rule to configure. The following `ConfigDocument` disables the checking of application HTTP 4xx status codes, while keeping the checking of load balancer HTTP 4xx code enabled.

```
OptionSettings:
  ...
  aws:elasticbeanstalk:healthreporting:system:
    ConfigDocument:
      Rules:
        Environment:
          Application:
            ApplicationRequests4xx:
              Enabled: false
        ELB:
          ELBRequests4xx:
            Enabled: true
      Version: 1
    SystemType: enhanced
  ...
```

Note

You can combine `Rules` and `CloudWatchMetrics` in the same `ConfigDocument` option setting. `CloudWatchMetrics` are described in [Publishing Amazon CloudWatch custom metrics for an environment \(p. 714\)](#).

If you previously enabled `CloudWatchMetrics`, the configuration file that you retrieve using the `eb config save` command already has a `ConfigDocument` key with a `CloudWatchMetrics` section. *Do not delete it*—add a `Rules` section into the same `ConfigDocument` option value.

6. Save the configuration file and close the text editor. In this example, the updated configuration file is saved with a name (`02-cloudwatch-enabled.cfg.yml`) that's different from the downloaded configuration file. This creates a separate saved configuration when the file is uploaded. You can use the same name as the downloaded file to overwrite the existing configuration without creating a new one.
7. Use the `eb config put` command to upload the updated configuration file to Elastic Beanstalk.

```
$ eb config put 02-cloudwatch-enabled
```

When using the `eb config get` and `put` commands with saved configurations, don't include the file name extension.

8. Apply the saved configuration to your running environment.

```
$ eb config --cfg 02-cloudwatch-enabled
```

The `--cfg` option specifies a named configuration file that is applied to the environment. You can save the configuration file locally or in Elastic Beanstalk. If a configuration file with the specified name exists in both locations, the EB CLI uses the local file.

Configuring enhanced health rules using a config document

The configuration (config) document for enhanced health rules is a JSON document that lists the rules to configure.

The following example shows a config document that disables the checking of application HTTP 4xx status codes and enables the checking of load balancer HTTP 4xx status codes.

```
{
  "Rules": {
    "Environment": {
      "Application": {
        "ApplicationRequests4xx": {
          "Enabled": false
        }
      },
      "ELB": {
        "ELBRequests4xx": {
          "Enabled": true
        }
      }
    }
  },
  "Version": 1
}
```

For the AWS CLI, you pass the document as a value for the `value` key in an option settings argument, which itself is a JSON object. In this case, you must escape quotation marks in the embedded document. The following command checks if the configuration settings are valid.

```
$ aws elasticbeanstalk validate-configuration-settings --application-name my-app --
environment-name my-env --option-settings '[
  {
    "Namespace": "aws:elasticbeanstalk:healthreporting:system",
    "OptionName": "ConfigDocument",
    "Value": "{\\"Rules\\": { \\"Environment\\": { \\"Application\\":
{ \\"ApplicationRequests4xx\\": { \\"Enabled\\": false } }, \\"ELB\\": { \\"ELBRequests4xx\\":
{ \\"Enabled\\": true } } } }, \\"Version\\": 1 }"
  }
]'
```

For an `.ebextensions` configuration file in YAML, you can provide the JSON document as is.

```
option_settings:
- namespace: aws:elasticbeanstalk:healthreporting:system
  option_name: ConfigDocument
  value: {
    "Rules": {
      "Environment": {
        "Application": {
          "ApplicationRequests4xx": {
            "Enabled": false
          }
        },
        "ELB": {
          "ELBRequests4xx": {
            "Enabled": true
          }
        }
      }
    }
  },
  "Version": 1
}
```


Publishing Amazon CloudWatch custom metrics for an environment

You can publish the data gathered by AWS Elastic Beanstalk enhanced health reporting to Amazon CloudWatch as custom metrics. Publishing metrics to CloudWatch lets you monitor changes in application performance over time and identify potential issues by tracking how resource usage and request latency scale with load.

By publishing metrics to CloudWatch, you also make them available for use with [monitoring graphs](#) (p. 686) and [alarms](#) (p. 725). One free metric, *EnvironmentHealth*, is enabled automatically when you use enhanced health reporting. Custom metrics other than *EnvironmentHealth* incur standard [CloudWatch charges](#).

To publish CloudWatch custom metrics for an environment, you must first enable enhanced health reporting on the environment. See [Enabling Elastic Beanstalk enhanced health reporting](#) (p. 698) for instructions.

Topics

- [Enhanced health reporting metrics](#) (p. 714)
- [Configuring CloudWatch metrics using the Elastic Beanstalk console](#) (p. 715)
- [Configuring CloudWatch custom metrics using the EB CLI](#) (p. 716)
- [Providing custom metric config documents](#) (p. 717)

Enhanced health reporting metrics

When you enable enhanced health reporting in your environment, the enhanced health reporting system automatically publishes one [CloudWatch custom metric](#), *EnvironmentHealth*. To publish additional metrics to CloudWatch, configure your environment with those metrics by using the [Elastic Beanstalk console](#) (p. 715), [EB CLI](#) (p. 716), or [.ebextensions](#) (p. 536).

You can publish the following enhanced health metrics from your environment to CloudWatch.

Available metrics—all platforms

`EnvironmentHealth`

Environment only. This is the only CloudWatch metric that the enhanced health reporting system publishes, unless you configure additional metrics. Environment health is represented by one of seven [statuses](#) (p. 706). In the CloudWatch console, these statuses map to the following values:

- 0 – OK
- 1 – Info
- 5 – Unknown
- 10 – No data
- 15 – Warning
- 20 – Degraded
- 25 – Severe

`InstancesSevere`, `InstancesDegraded`, `InstancesWarning`, `InstancesInfo`, `InstancesOk`, `InstancesPending`, `InstancesUnknown`, `InstancesNoData`

Environment only. These metrics indicate the number of instances in the environment with each health status. `InstancesNoData` indicates the number of instances for which no data is being received.

`ApplicationRequestsTotal`, `ApplicationRequests5xx`, `ApplicationRequests4xx`,
`ApplicationRequests3xx`, `ApplicationRequests2xx`

Instance and environment. Indicates the total number of requests completed by the instance or environment, and the number of requests that completed with each status code category.

`ApplicationLatencyP10`, `ApplicationLatencyP50`, `ApplicationLatencyP75`,
`ApplicationLatencyP85`, `ApplicationLatencyP90`, `ApplicationLatencyP95`,
`ApplicationLatencyP99`, `ApplicationLatencyP99.9`

Instance and environment. Indicates the average amount of time, in seconds, it takes to complete the fastest x percent of requests.

`InstanceHealth`

Instance only. Indicates the current health status of the instance. Instance health is represented by one of seven [statuses \(p. 706\)](#). In the CloudWatch console, these statuses map to the following values:

- 0 – OK
- 1 – Info
- 5 – Unknown
- 10 – No data
- 15 – Warning
- 20 – Degraded
- 25 – Severe

Available metrics—Linux

`CPU irq`, `CPUIdle`, `CPUUser`, `CPUSystem`, `CPUSoftirq`, `CPUiowait`, `CPUNice`

Instance only. Indicates the percentage of time that the CPU has spent in each state over the last minute.

`LoadAverage1min`

Instance only. The average CPU load of the instance over the last minute.

`RootFilesystemUtil`

Instance only. Indicates the percentage of disk space that's in use.

Available metrics—Windows

`CPUIdle`, `CPUUser`, `CPUPrivileged`

Instance only. Indicates the percentage of time that the CPU has spent in each state over the last minute.

Configuring CloudWatch metrics using the Elastic Beanstalk console

You can use the Elastic Beanstalk console to configure your environment to publish enhanced health reporting metrics to CloudWatch and make them available for use with monitoring graphs and alarms.

To configure CloudWatch custom metrics in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.

- In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

- In the navigation pane, choose **Configuration**.
- In the **Monitoring** configuration category, choose **Edit**.
- Under **Health reporting**, select the instance and environment metrics to publish to CloudWatch. To select multiple metrics, press the **Ctrl** key while choosing.
- Choose **Apply**.

Enabling CloudWatch custom metrics adds them to the list of metrics available on the [Monitoring page \(p. 685\)](#).

Configuring CloudWatch custom metrics using the EB CLI

You can use the EB CLI to configure custom metrics by saving your environment's configuration locally, adding an entry that defines the metrics to publish, and then uploading the configuration to Elastic Beanstalk. You can apply the saved configuration to an environment during or after creation.

To configure CloudWatch custom metrics with the EB CLI and saved configurations

- Initialize your project folder with [eb init \(p. 860\)](#).
- Create an environment by running the [eb create \(p. 864\)](#) command.
- Save a configuration template locally by running the **eb config save** command. The following example uses the `--cfg` option to specify the name of the configuration.

```
$ eb config save --cfg 01-base-state
Configuration saved at: ~/project/.elasticbeanstalk/saved_configs/01-base-state.cfg.yml
```

- Open the saved configuration file in a text editor.
- Under `OptionSettings > aws:elasticbeanstalk:healthreporting:system:`, add a `ConfigDocument` key to enable each of the CloudWatch metrics you want. For example, the following `ConfigDocument` publishes `ApplicationRequests5xx` and `ApplicationRequests4xx` metrics at the environment level, and `ApplicationRequestsTotal` metrics at the instance level.

```
OptionSettings:
  ...
  aws:elasticbeanstalk:healthreporting:system:
    ConfigDocument:
      CloudWatchMetrics:
        Environment:
          ApplicationRequests5xx: 60
          ApplicationRequests4xx: 60
        Instance:
          ApplicationRequestsTotal: 60
      Version: 1
    SystemType: enhanced
  ...
```

In the example, 60 indicates the number of seconds between measurements. Currently, this is the only supported value.

Note

You can combine `CloudWatchMetrics` and `Rules` in the same `ConfigDocument` option setting. Rules are described in [Configuring enhanced health rules for an environment \(p. 710\)](#).

If you previously used Rules to configure enhanced health rules, then the configuration file that you retrieve using the **eb config save** command already has a `ConfigDocument` key with a `Rules` section. *Do not delete it*—add a `CloudWatchMetrics` section into the same `ConfigDocument` option value.

6. Save the configuration file and close the text editor. In this example, the updated configuration file is saved with a name (`02-cloudwatch-enabled.cfg.yml`) that is different from the downloaded configuration file. This creates a separate saved configuration when the file is uploaded. You can use the same name as the downloaded file to overwrite the existing configuration without creating a new one.
7. Use the **eb config put** command to upload the updated configuration file to Elastic Beanstalk.

```
$ eb config put 02-cloudwatch-enabled
```

When using the **eb config get** and **put** commands with saved configurations, don't include the file extension.

8. Apply the saved configuration to your running environment.

```
$ eb config --cfg 02-cloudwatch-enabled
```

The `--cfg` option specifies a named configuration file that is applied to the environment. You can save the configuration file locally or in Elastic Beanstalk. If a configuration file with the specified name exists in both locations, the EB CLI uses the local file.

Providing custom metric config documents

The configuration (config) document for Amazon CloudWatch custom metrics is a JSON document that lists the metrics to publish at the environment and instance levels. The following example shows a config document that enables all custom metrics available on Linux.

```
{
  "CloudWatchMetrics": {
    "Environment": {
      "ApplicationLatencyP99.9": 60,
      "InstancesSevere": 60,
      "ApplicationLatencyP90": 60,
      "ApplicationLatencyP99": 60,
      "ApplicationLatencyP95": 60,
      "InstancesUnknown": 60,
      "ApplicationLatencyP85": 60,
      "InstancesInfo": 60,
      "ApplicationRequests2xx": 60,
      "InstancesDegraded": 60,
      "InstancesWarning": 60,
      "ApplicationLatencyP50": 60,
      "ApplicationRequestsTotal": 60,
      "InstancesNoData": 60,
      "InstancesPending": 60,
      "ApplicationLatencyP10": 60,
      "ApplicationRequests5xx": 60,
      "ApplicationLatencyP75": 60,
      "InstancesOk": 60,
    }
  }
}
```

```

    "ApplicationRequests3xx": 60,
    "ApplicationRequests4xx": 60
  },
  "Instance": {
    "ApplicationLatencyP99.9": 60,
    "ApplicationLatencyP90": 60,
    "ApplicationLatencyP99": 60,
    "ApplicationLatencyP95": 60,
    "ApplicationLatencyP85": 60,
    "CPUUser": 60,
    "ApplicationRequests2xx": 60,
    "CPUIdle": 60,
    "ApplicationLatencyP50": 60,
    "ApplicationRequestsTotal": 60,
    "RootFilesystemUtil": 60,
    "LoadAverage1min": 60,
    "CPUirq": 60,
    "CPUNice": 60,
    "CPUiowait": 60,
    "ApplicationLatencyP10": 60,
    "LoadAverage5min": 60,
    "ApplicationRequests5xx": 60,
    "ApplicationLatencyP75": 60,
    "CPUSystem": 60,
    "ApplicationRequests3xx": 60,
    "ApplicationRequests4xx": 60,
    "InstanceHealth": 60,
    "CPUSoftirq": 60
  }
},
"Version": 1
}

```

For the AWS CLI, you pass the document as a value for the `Value` key in an option settings argument, which itself is a JSON object. In this case, you must escape quotation marks in the embedded document.

```

$ aws elasticbeanstalk validate-configuration-settings --application-name my-app --
environment-name my-env --option-settings '[
  {
    "Namespace": "aws:elasticbeanstalk:healthreporting:system",
    "OptionName": "ConfigDocument",
    "Value": "{\"CloudWatchMetrics\": {\"Environment\": {\"ApplicationLatencyP99.9\":
60,\"InstancesSevere\": 60,\"ApplicationLatencyP90\": 60,\"ApplicationLatencyP99\":
60,\"ApplicationLatencyP95\": 60,\"InstancesUnknown\": 60,\"ApplicationLatencyP85\":
60,\"InstancesInfo\": 60,\"ApplicationRequests2xx\": 60,\"InstancesDegraded\": 60,
\"InstancesWarning\": 60,\"ApplicationLatencyP50\": 60,\"ApplicationRequestsTotal
\": 60,\"InstancesNoData\": 60,\"InstancesPending\": 60,\"ApplicationLatencyP10\":
60,\"ApplicationRequests5xx\": 60,\"ApplicationLatencyP75\": 60,\"InstancesOk\":
60,\"ApplicationRequests3xx\": 60,\"ApplicationRequests4xx\": 60},\"Instance\":
{\"ApplicationLatencyP99.9\": 60,\"ApplicationLatencyP90\": 60,\"ApplicationLatencyP99\":
60,\"ApplicationLatencyP95\": 60,\"ApplicationLatencyP85\": 60,\"CPUUser\": 60,
\"ApplicationRequests2xx\": 60,\"CPUIdle\": 60,\"ApplicationLatencyP50\": 60,
\"ApplicationRequestsTotal\": 60,\"RootFilesystemUtil\": 60,\"LoadAverage1min\":
60,\"CPUirq\": 60,\"CPUNice\": 60,\"CPUiowait\": 60,\"ApplicationLatencyP10\": 60,
\"LoadAverage5min\": 60,\"ApplicationRequests5xx\": 60,\"ApplicationLatencyP75\":
60,\"CPUSystem\": 60,\"ApplicationRequests3xx\": 60,\"ApplicationRequests4xx\": 60,
\"InstanceHealth\": 60,\"CPUSoftirq\": 60}},\"Version\": 1}"
  }
]'

```

For an `.ebextensions` configuration file in YAML, you can provide the JSON document as is.

```
option_settings:
```

```

- namespace: aws:elasticbeanstalk:healthreporting:system
  option_name: ConfigDocument
  value: {
"CloudWatchMetrics": {
  "Environment": {
    "ApplicationLatencyP99.9": 60,
    "InstancesSevere": 60,
    "ApplicationLatencyP90": 60,
    "ApplicationLatencyP99": 60,
    "ApplicationLatencyP95": 60,
    "InstancesUnknown": 60,
    "ApplicationLatencyP85": 60,
    "InstancesInfo": 60,
    "ApplicationRequests2xx": 60,
    "InstancesDegraded": 60,
    "InstancesWarning": 60,
    "ApplicationLatencyP50": 60,
    "ApplicationRequestsTotal": 60,
    "InstancesNoData": 60,
    "InstancesPending": 60,
    "ApplicationLatencyP10": 60,
    "ApplicationRequests5xx": 60,
    "ApplicationLatencyP75": 60,
    "InstancesOk": 60,
    "ApplicationRequests3xx": 60,
    "ApplicationRequests4xx": 60
  },
  "Instance": {
    "ApplicationLatencyP99.9": 60,
    "ApplicationLatencyP90": 60,
    "ApplicationLatencyP99": 60,
    "ApplicationLatencyP95": 60,
    "ApplicationLatencyP85": 60,
    "CPUUser": 60,
    "ApplicationRequests2xx": 60,
    "CPUIidle": 60,
    "ApplicationLatencyP50": 60,
    "ApplicationRequestsTotal": 60,
    "RootFilesystemUtil": 60,
    "LoadAverage1min": 60,
    "CPUIrq": 60,
    "CPUNice": 60,
    "CPUIowait": 60,
    "ApplicationLatencyP10": 60,
    "LoadAverage5min": 60,
    "ApplicationRequests5xx": 60,
    "ApplicationLatencyP75": 60,
    "CPUSystem": 60,
    "ApplicationRequests3xx": 60,
    "ApplicationRequests4xx": 60,
    "InstanceHealth": 60,
    "CPUSoftirq": 60
  }
},
"Version": 1
}

```

Using enhanced health reporting with the Elastic Beanstalk API

Because AWS Elastic Beanstalk enhanced health reporting has role and solution stack requirements, you must update scripts and code that you used prior to the release of enhanced health reporting before

you can use it. To maintain backward compatibility, enhanced health reporting is not enabled by default when you create an environment using the Elastic Beanstalk API.

You configure enhanced health reporting by setting the service role, the instance profile, and Amazon CloudWatch configuration options for your environment. You can do this in three ways: by setting the configuration options in the `.ebextensions` folder, with saved configurations, or by configuring them directly in the `create-environment` call's `option-settings` parameter.

To use the API, SDKs, or AWS command line interface (CLI) to create an environment that supports enhanced health, you must:

- Create a service role and instance profile with the appropriate [permissions \(p. 20\)](#)
- Create a new environment with a new [platform version \(p. 29\)](#)
- Set the health system type, instance profile, and service role [configuration options \(p. 536\)](#)

Use the following configuration options in the `aws:elasticbeanstalk:healthreporting:system`, `aws:autoscaling:launchconfiguration`, and `aws:elasticbeanstalk:environment` namespaces to configure your environment for enhanced health reporting.

Enhanced health configuration options

SystemType

Namespace: `aws:elasticbeanstalk:healthreporting:system`

To enable enhanced health reporting, set to **enhanced**.

IamInstanceProfile

Namespace: `aws:autoscaling:launchconfiguration`

Set to the name of an instance profile configured for use with Elastic Beanstalk.

ServiceRole

Namespace: `aws:elasticbeanstalk:environment`

Set to the name of a service role configured for use with Elastic Beanstalk.

ConfigDocument (optional)

Namespace: `aws:elasticbeanstalk:healthreporting:system`

A JSON document that defines the and instance and environment metrics to publish to CloudWatch. For example:

```
{
  "CloudWatchMetrics":
  {
    "Environment":
    {
      "ApplicationLatencyP99.9": 60,
      "InstancesSevere": 60
    }
    "Instance":
    {
      "ApplicationLatencyP85": 60,
      "CPUUser": 60
    }
  }
  "Version": 1
}
```

```
}

```

Note

Config documents may require special formatting, such as escaping quotes, depending on how you provide them to Elastic Beanstalk. See [Providing custom metric config documents \(p. 717\)](#) for examples.

Enhanced health log format

AWS Elastic Beanstalk platforms use a custom web server log format to efficiently relay information about HTTP requests to the enhanced health reporting system. The system analyzes the logs, identifies issues, and sets the instance and environment health accordingly. If you disable the web server proxy on your environment and serve requests directly from the web container, you can still make full use of enhanced health reporting by configuring your server to output logs in the location and format that the [Elastic Beanstalk health agent \(p. 693\)](#) uses.

Note

The information on this page is relevant only to Linux-based platforms. On the Windows Server platform, Elastic Beanstalk receives information about HTTP requests directly from the IIS web server. For details, see [Web server metrics capture in IIS on Windows server \(p. 710\)](#).

Web server log configuration

Elastic Beanstalk platforms are configured to output two logs with information about HTTP requests. The first is in verbose format and provides detailed information about the request, including the requester's user agent information and a human-readable timestamp.

`/var/log/nginx/access.log`

The following example is from an nginx proxy running on a Ruby web server environment, but the format is similar for Apache.

```
172.31.24.3 - - [23/Jul/2015:00:21:20 +0000] "GET / HTTP/1.1" 200 11 "-" "curl/7.22.0
(x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3"
"177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:21 +0000] "GET / HTTP/1.1" 200 11 "-" "curl/7.22.0
(x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3"
"177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:22 +0000] "GET / HTTP/1.1" 200 11 "-" "curl/7.22.0
(x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3"
"177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:22 +0000] "GET / HTTP/1.1" 200 11 "-" "curl/7.22.0
(x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3"
"177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:22 +0000] "GET / HTTP/1.1" 200 11 "-" "curl/7.22.0
(x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3"
"177.72.242.17"
```

The second log is in terse format. It includes information relevant only to enhanced health reporting. This log is output to a subfolder named `healthd` and rotates hourly. Old logs are deleted immediately after rotating out.

`/var/log/nginx/healthd/application.log.2015-07-23-00`

The following example shows a log in the machine-readable format.

```
1437609879.311"/"200"0.083"0.083"177.72.242.17
1437609879.874"/"200"0.347"0.347"177.72.242.17
```



```
1437609880.006"/bad/path"404"0.001"0.001"177.72.242.17
1437609880.058"/"200"0.530"0.530"177.72.242.17
1437609880.928"/bad/path"404"0.001"0.001"177.72.242.17
```

The enhanced health log format includes the following information:

- The time of the request, in Unix time
- The path of the request
- The HTTP status code for the result
- The request time
- The upstream time
- The X-Forwarded-For HTTP header

For nginx proxies, times are printed in floating-point seconds, with three decimal places. For Apache, whole microseconds are used.

Note

If you see a warning similar to the following in a log file, where `DATE-TIME` is a date and time, and you are using a custom proxy, such as in a multi-container Docker environment, you must use an `.ebextension` to configure your environment so that `healthd` can read your log files:

```
W, [DATE-TIME #1922] WARN -- : log file "/var/log/nginx/healthd/
application.log.DATE-TIME" does not exist
```

You can start with the `.ebextension` in the [Multicontainer Docker sample](#).

`/etc/nginx/conf.d/webapp_healthd.conf`

The following example shows the log configuration for nginx with the `healthd` log format highlighted.

```
upstream my_app {
    server unix:///var/run/puma/my_app.sock;
}

log_format healthd '$msec$uri'
    '$status$request_time$upstream_response_time'
    '$http_x_forwarded_for';

server {
    listen 80;
    server_name _ localhost; # need to listen to localhost for worker tier

    if ($time_iso8601 ~ "^(\\d{4})-(\\d{2})-(\\d{2})T(\\d{2})") {
        set $year $1;
        set $month $2;
        set $day $3;
        set $hour $4;
    }

    access_log /var/log/nginx/access.log main;
    access_log /var/log/nginx/healthd/application.log.$year-$month-$day-$hour healthd;

    location / {
        proxy_pass http://my_app; # match the name of upstream directive which is defined above
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    location /assets {
        alias /var/app/current/public/assets;
    }
}
```

```
gzip_static on;
gzip on;
expires max;
add_header Cache-Control public;
}

location /public {
    alias /var/app/current/public;
    gzip_static on;
    gzip on;
    expires max;
    add_header Cache-Control public;
}
}
```

/etc/httpd/conf.d/healthd.conf

The following example shows the log configuration for Apache.

```
LogFormat "%{s}t\"%U\"%s\"%D\"%D\"%{X-Forwarded-For}i" healthd
CustomLog "|/usr/sbin/rotatelogs /var/log/httpd/healthd/application.log.%Y-%m-%d-%H 3600"
healthd
```

Generating logs for enhanced health reporting

To provide logs to the health agent, you must do the following:

- Output logs in the correct format, as shown in the previous section
- Output logs to `/var/log/nginx/healthd/`
- Name logs using the following format: `application.log.$year-$month-$day-$hour`
- Rotate logs once per hour
- Do not truncate logs

Notifications and troubleshooting

This page lists example cause messages for common issues and links to more information. Cause messages appear in the [environment overview \(p. 685\)](#) page of the Elastic Beanstalk console and are recorded in [events \(p. 728\)](#) when health issues persist across several checks.

Deployments

Elastic Beanstalk monitors your environment for consistency following deployments. If a rolling deployment fails, the version of your application running on the instances in your environment may vary. This can occur if a deployment succeeds on one or more batches but fails prior to all batches completing.

Incorrect application version found on 2 out of 5 instances. Expected version "v1" (deployment 1).

Incorrect application version on environment instances. Expected version "v1" (deployment 1).

The expected application version is not running on some or all instances in an environment.

Incorrect application version "v2" (deployment 2). Expected version "v1" (deployment 1).

The application deployed to an instance differs from the expected version. If a deployment fails, the expected version is reset to the version from the most recent successful deployment. In the above example, the first deployment (version "v1") succeeded, but the second deployment (version "v2") failed. Any instances running "v2" are considered unhealthy.

To solve this issue, start another deployment. You can [redploy a previous version \(p. 397\)](#) that you know works, or configure your environment to [ignore health checks \(p. 401\)](#) during deployment and redeploy the new version to force the deployment to complete.

You can also identify and terminate the instances that are running the wrong application version. Elastic Beanstalk will launch instances with the correct version to replace any instances that you terminate. Use the [EB CLI health command \(p. 875\)](#) to identify instances that are running the wrong application version.

Application server

15% of requests are erroring with HTTP 4xx

20% of the requests to the ELB are erroring with HTTP 4xx.

A high percentage of HTTP requests to an instance or environment are failing with 4xx errors.

A 400 series status code indicates that the user made a bad request, such as requesting a page that doesn't exist (404 File Not Found) or that the user doesn't have access to (403 Forbidden). A low number of 404s is not unusual but a large number could mean that there are internal or external links to unavailable pages. These issues can be resolved by fixing bad internal links and adding redirects for bad external links.

5% of the requests are failing with HTTP 5xx

3% of the requests to the ELB are failing with HTTP 5xx.

A high percentage of HTTP requests to an instance or environment are failing with 500 series status codes.

A 500 series status code indicates that the application server encountered an internal error. These issues indicate that there is an error in your application code and should be identified and fixed quickly.

95% of CPU is in use

On an instance, the health agent is reporting an extremely high percentage of CPU usage and sets the instance health to **Warning** or **Degraded**.

Scale your environment to take load off of instances.

Worker instance

20 messages waiting in the queue (25 seconds ago)

Requests are being added to your worker environment's queue faster than they can be processed. Scale your environment to increase capacity.

5 messages in Dead Letter Queue (15 seconds ago)

Worker requests are failing repeatedly and being added to the [the section called "Dead-letter queues" \(p. 440\)](#). Check the requests in the dead-letter queue to see why they are failing.

Other resources

4 active instances is below Auto Scaling group minimum size 5

The number of instances running in your environment is fewer than the minimum configured for the Auto Scaling group.

Auto Scaling group (groupname) notifications have been deleted or modified

The notifications configured for your Auto Scaling group have been modified outside of Elastic Beanstalk.

Manage alarms

You can create alarms for metrics that you are monitoring by using the Elastic Beanstalk console. Alarms help you monitor changes to your AWS Elastic Beanstalk environment so that you can easily identify and mitigate problems before they occur. For example, you can set an alarm that notifies you when CPU utilization in an environment exceeds a certain threshold, ensuring that you are notified before a potential problem occurs. For more information, see [Using Elastic Beanstalk with Amazon CloudWatch \(p. 742\)](#).

Note

Elastic Beanstalk uses CloudWatch for monitoring and alarms, meaning CloudWatch costs are applied to your AWS account for any alarms that you use.

For more information about monitoring specific metrics, see [Basic health reporting \(p. 688\)](#).

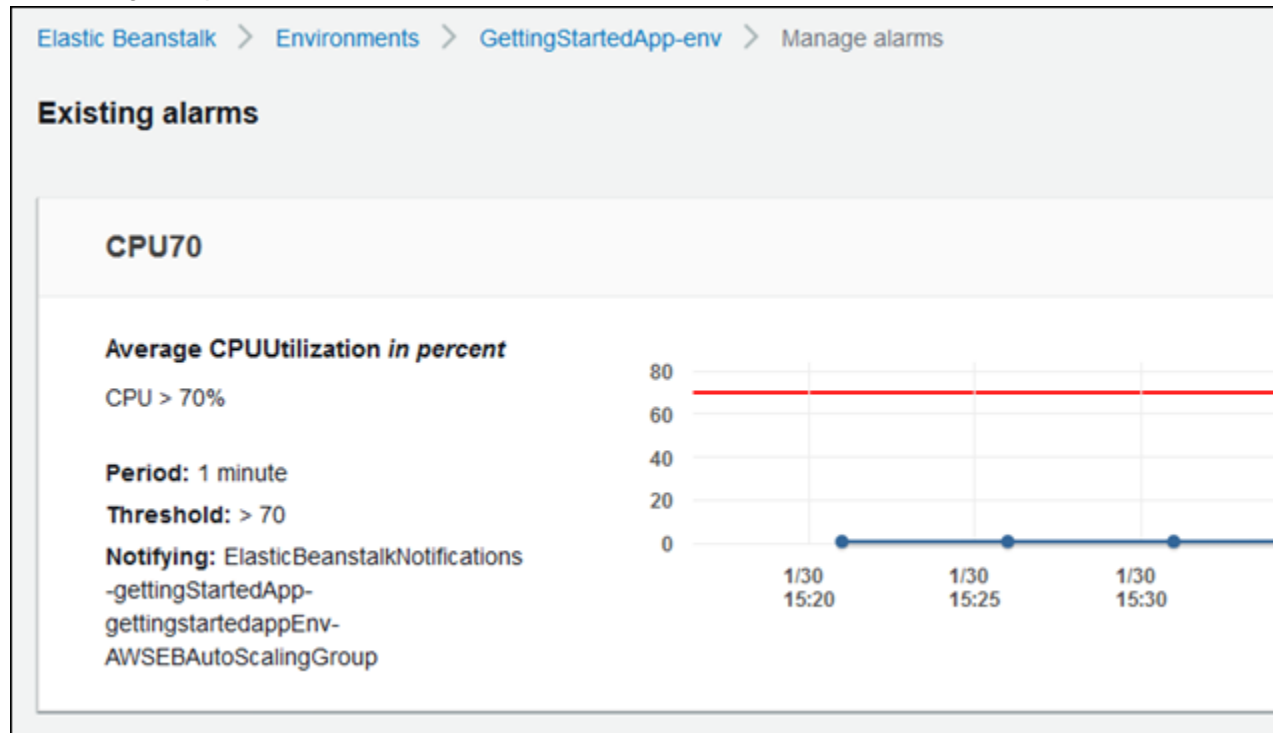
To check the state of your alarms


1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.



Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Alarms**.



The page displays a list of existing alarms. If any alarms are in the alarm state, they are flagged with  (warning).


4. To filter alarms, choose the drop-down menu, and then select a filter.
5. To edit or delete an alarm, choose  (edit) or  (delete), respectively.

To create an alarm

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Monitoring**.
4. Locate the metric for which you want to create an alarm, and then choose  (alarm). The **Add alarm** page is displayed.

Elastic Beanstalk > Environments > GettingStartedApp-env > Add alarm

Add Alarm

Average CPUUtilization *in percent*

Name:

Name should be less than 238 characters in length and can only contain numbers and letters

Description:

Optional.

Period:
1 minute ▼

Threshold: Average CPUUtilization
 ▼

Change state after:
 ▼

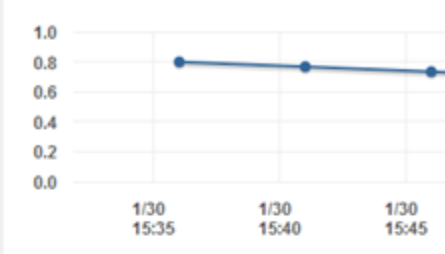
Notify:
A new SNS topic... ▼ Refresh ↻

Topic name:
ElasticBeanstalkNotifications-gettingStartedApp-gettingsta

E-mail address:

Notify when state changes to:
 OK
 Alarm
 Insufficient data

Cancel **Add**



Time	Average CPU Utilization (in percent)
1/30 15:35	0.8
1/30 15:40	0.75
1/30 15:45	0.7

Other Alarms For This Metric

CPU70

5. Enter details about the alarm:

- **Name:** A name for this alarm.
- **Description** (optional): A short description of what this alarm is.
- **Period:** The time interval between readings.
- **Threshold:** Describes the behavior and value that the metric must exceed in order to trigger an alarm.

- **Change state after:** The amount a time after a threshold has been exceed that triggers a change in state of the alarm.
 - **Notify:** The Amazon SNS topic that is notified when an alarm changes state.
 - **Notify when state changes to:**
 - **OK:** The metric is within the defined threshold.
 - **Alarm:** The metric exceeded the defined threshold.
 - **Insufficient data:** The alarm has just started, the metric is not available, or not enough data is available for the metric to determine the alarm state.
6. Choose **Add**. The environment status changes to gray while the environment updates. You can view the alarm that you created by choosing **Alarms** in the navigation pane.

Viewing an Elastic Beanstalk environment's event stream

You can use the AWS Management Console to access events and notifications associated with your application.

To view events


1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Events**.

Elastic Beanstalk > Environments > GettingStartedApp-env > Events

 Click the link to be routed to the previous Beanstalk Console
Switch to the previous console

Events

Severity < 1 2 3 4 5 6 7

Time	Type	Details
2020-03-09 17:14:06 UTC-0700	INFO	createConfigurationTemplate completed suc
2020-03-09 17:14:06 UTC-0700	INFO	createConfigurationTemplate is starting.
2020-03-03 04:16:55 UTC-0800	INFO	Environment health has transitioned from Int update completed 85 seconds ago and took
2020-03-03 04:16:07 UTC-0800	INFO	Environment update completed successfully
2020-03-03 04:16:07 UTC-0800	INFO	Successfully deployed new configuration to

The Events page shows a list of all events that have been recorded for the environment. You can page through the list choosing < (previous), > (next), or page numbers. You can filter the type of events shown by using the **Severity** drop-down list.

The [EB CLI \(p. 852\)](#) and [AWS CLI](#) both provide commands for retrieving events. If you are managing your environment using the EB CLI, use [eb events \(p. 904\)](#) to print a list of events. This command also has a `--follow` option that continues to show new events until you press **Ctrl+C** to stop output.

To pull events using the AWS CLI, use the `describe-events` command and specify the environment by name or ID:

```
$ aws elasticbeanstalk describe-events --environment-id e-gbjzqcra3
{
  "Events": [
    {
      "ApplicationName": "elastic-beanstalk-example",
      "EnvironmentName": "elasticBeanstalkExa-env",
      "Severity": "INFO",
      "RequestId": "a4c7bfd6-2043-11e5-91e2-9114455c358a",
      "Message": "Environment update completed successfully.",
      "EventDate": "2015-07-01T22:52:12.639Z"
    }
  ]
}
```



```
... },
```

For more information on the command line tools, see [Tools \(p. 852\)](#).

Listing and connecting to server instances

You can view a list of Amazon EC2 instances running your AWS Elastic Beanstalk application environment through the Elastic Beanstalk console. You can connect to the instances using any SSH client. You can connect to the instances running Windows using Remote Desktop.

Some notes about specific development environments:

- For more information about listing and connecting to server instances using the AWS Toolkit for Eclipse, see [Listing and connecting to server instances \(p. 136\)](#).
- For more information about listing and connecting to server instances using the AWS Toolkit for Visual Studio, see [Listing and connecting to server instances \(p. 188\)](#).

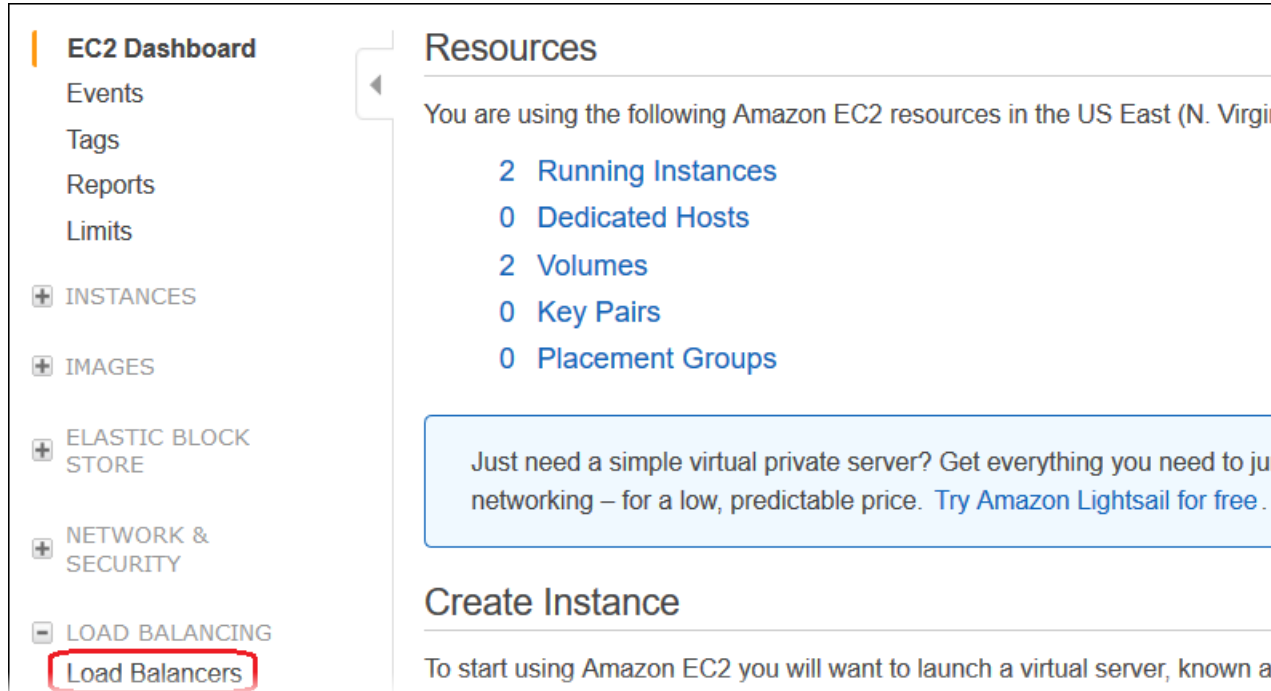
Important

Before you can access your Elastic Beanstalk–provisioned Amazon EC2 instances, you must create an Amazon EC2 key pair and configure your Elastic Beanstalk–provisioned Amazon EC2 instances to use the Amazon EC2 key pair. You can set up your Amazon EC2 key pairs using the [AWS Management Console](#). For instructions on creating a key pair for Amazon EC2, see the *Amazon EC2 Getting Started Guide*. For more information on how to configure your Amazon EC2 instances to use an Amazon EC2 key pair, see [EC2 key pair \(p. 512\)](#).

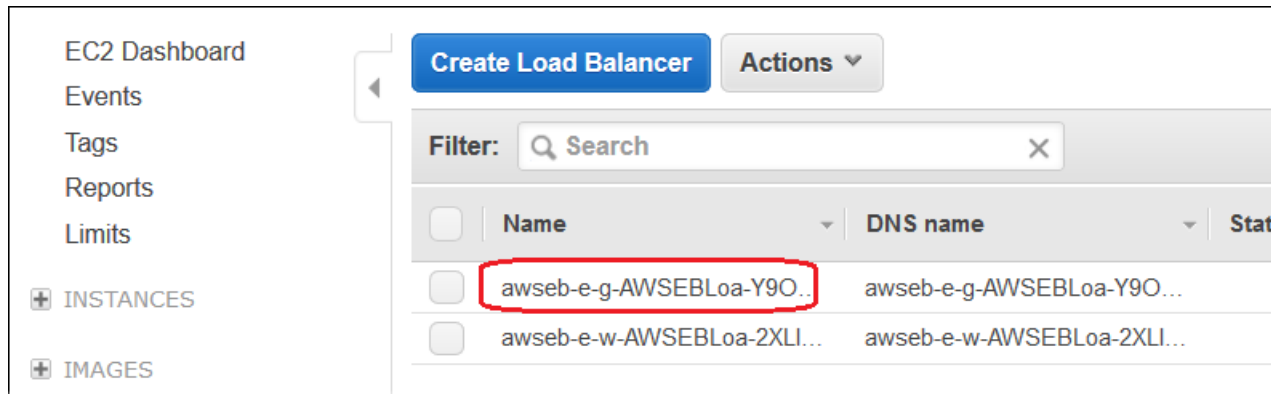
By default, Elastic Beanstalk does not enable remote connections to EC2 instances in a Windows container except for those in legacy Windows containers. (Elastic Beanstalk configures EC2 instances in legacy Windows containers to use port 3389 for RDP connections.) You can enable remote connections to your EC2 instances running Windows by adding a rule to a security group that authorizes inbound traffic to the instances. We strongly recommend that you remove the rule when you end your remote connection. You can add the rule again the next time you need to log in remotely. For more information, see [Adding a Rule for Inbound RDP Traffic to a Windows Instance](#) and [Connect to Your Windows Instance](#) in the *Amazon Elastic Compute Cloud User Guide for Microsoft Windows*.

To view and connect to Amazon EC2 instances for an environment

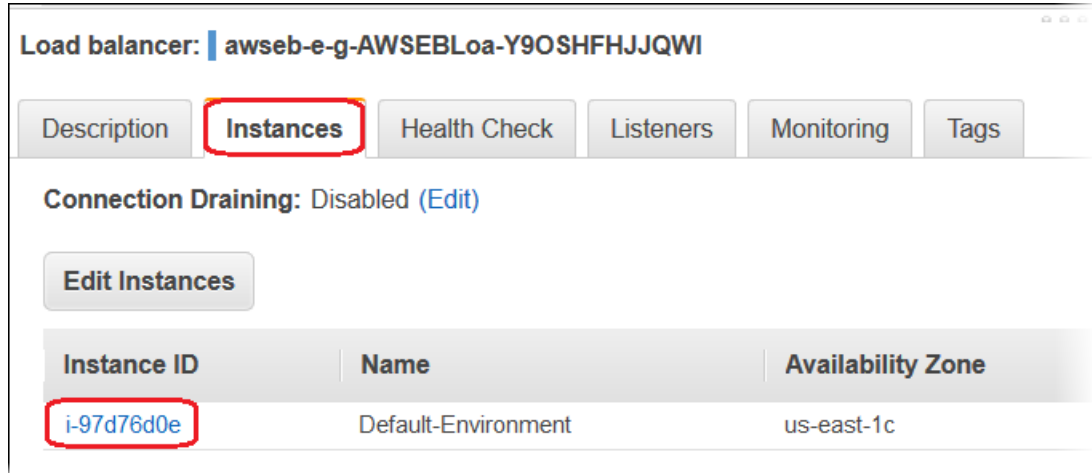
1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane of the console, choose **Load Balancers**.



3. Load balancers created by Elastic Beanstalk have **awseb** in the name. Find the load balancer for your environment and click it.

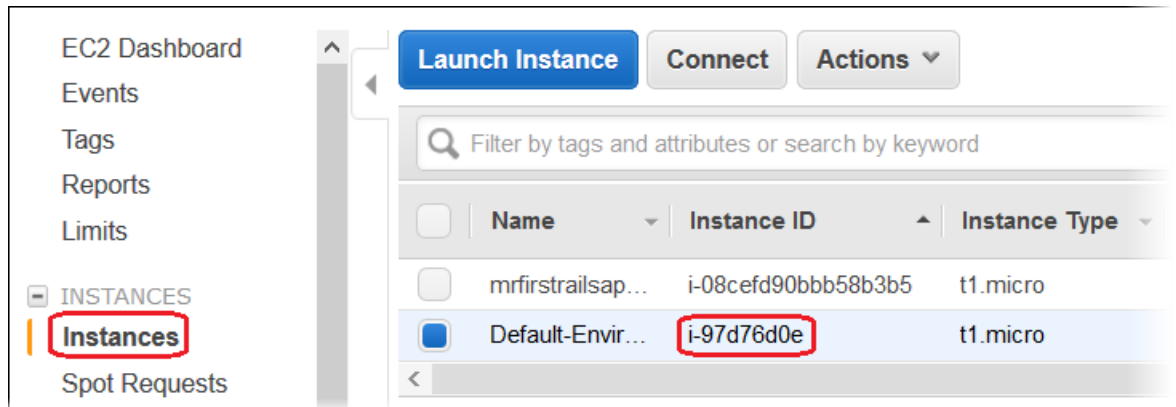


4. Choose the **Instances** tab in the bottom pane of the console.



A list of the instances that the load balancer for your Elastic Beanstalk environment uses is displayed. Make a note of an instance ID that you want to connect to.

5. In the navigation pane of the Amazon EC2 console, choose **Instances**, and find your instance ID in the list.



6. Right-click the instance ID for the Amazon EC2 instance running in your environment's load balancer, and then select **Connect** from the context menu.
7. Make a note of the instance's public DNS address on the **Description** tab.
8. Connect to an instance running Linux by using the SSH client of your choice, and then type `ssh -i .ec2/mykeypair.pem ec2-user@<public-DNS-of-the-instance>`.

For more information on connecting to an Amazon EC2 Linux instance, see [Getting Started with Amazon EC2 Linux Instances](#) in the *Amazon EC2 User Guide for Linux Instances*.

If your Elastic Beanstalk environment uses the [.NET on Windows Server platform \(p. 141\)](#), see [Getting Started with Amazon EC2 Windows Instances](#) in the *Amazon EC2 User Guide for Windows Instances*.

Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment

The Amazon EC2 instances in your Elastic Beanstalk environment generate logs that you can view to troubleshoot issues with your application or configuration files. Logs created by the web server,

application server, Elastic Beanstalk platform scripts, and AWS CloudFormation are stored locally on individual instances. You can easily retrieve them by using the [environment management console \(p. 353\)](#) or the EB CLI. You can also configure your environment to stream logs to Amazon CloudWatch Logs in real time.

Tail logs are the last 100 lines of the most commonly used log files—Elastic Beanstalk operational logs and logs from the web server or application server. When you request tail logs in the environment management console or with **eb logs**, an instance in your environment concatenates the most recent log entries into a single text file and uploads it to Amazon S3.

Bundle logs are full logs for a wider range of log files, including logs from yum and cron and several logs from AWS CloudFormation. When you request bundle logs, an instance in your environment packages the full log files into a ZIP archive and uploads it to Amazon S3.

Note

Elastic Beanstalk Windows Server platforms do not support bundle logs.

To upload rotated logs to Amazon S3, the instances in your environment must have an [instance profile \(p. 21\)](#) with permission to write to your Elastic Beanstalk Amazon S3 bucket. These permissions are included in the default instance profile that Elastic Beanstalk prompts you to create when you launch an environment in the Elastic Beanstalk console for the first time.

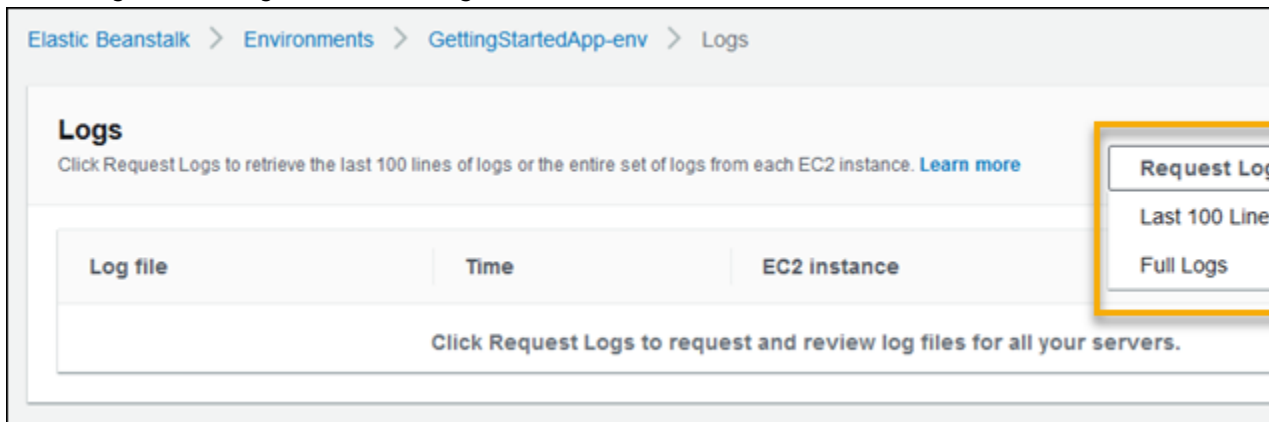
To retrieve instance logs

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Logs**.
4. Choose **Request Logs**, and then choose the type of logs to retrieve. To get tail logs, choose **Last 100 Lines**. To get bundle logs, choose **Full Logs**.



5. When Elastic Beanstalk finishes retrieving your logs, choose **Download**.

Elastic Beanstalk stores tail and bundle logs in an Amazon S3 bucket, and generates a presigned Amazon S3 URL that you can use to access your logs. Elastic Beanstalk deletes the files from Amazon S3 after a duration of 15 minutes.

Warning

Anyone in possession of the presigned Amazon S3 URL can access the files before they are deleted. Make the URL available only to trusted parties.

Note

Your user policy must have the `s3:DeleteObject` permission. Elastic Beanstalk uses your user permissions to delete the logs from Amazon S3.

To persist logs, you can configure your environment to publish logs to Amazon S3 automatically after they are rotated. To enable log rotation to Amazon S3, follow the procedure in [Configuring instance log viewing](#) (p. 524). Instances in your environment will attempt to upload logs that have been rotated once per hour.

If your application generates logs in a location that isn't part of the default configuration for your environment's platform, you can extend the default configuration by using configuration files ([.ebextensions](#) (p. 600)). You can add your application's log files to tail logs, bundle logs, or log rotation.

For real-time log streaming and long-term storage, configure your environment to [stream logs to Amazon CloudWatch Logs](#) (p. 737).

Sections

- [Log location on Amazon EC2 instances](#) (p. 734)
- [Log location in Amazon S3](#) (p. 735)
- [Log rotation settings on Linux](#) (p. 735)
- [Extending the default log task configuration](#) (p. 735)
- [Streaming log files to Amazon CloudWatch Logs](#) (p. 737)

Log location on Amazon EC2 instances

Logs are stored in standard locations on the Amazon EC2 instances in your environment. Elastic Beanstalk generates the following logs.

Linux

- `/var/log/eb-activity.log`
- `/var/log/eb-commandprocessor.log`

Windows Server

- `C:\Program Files\Amazon\ElasticBeanstalk\logs\`
- `C:\cfn\logs\cfn-init.log`

These logs contain messages about deployment activities, including messages related to configuration files ([.ebextensions](#) (p. 600)).

Each application and web server stores logs in its own folder:

- **Apache** – `/var/log/httpd/`
- **IIS** – `C:\inetpub\wwwroot\`
- **Node.js** – `/var/log/nodejs/`
- **nginx** – `/var/log/nginx/`
- **Passenger** – `/var/app/support/logs/`
- **Puma** – `/var/log/puma/`
- **Python** – `/opt/python/log/`

- **Tomcat** – `/var/log/tomcat8/`

Log location in Amazon S3

When you request tail or bundle logs from your environment, or when instances upload rotated logs, they're stored in your Elastic Beanstalk bucket in Amazon S3. Elastic Beanstalk creates a bucket named `elasticbeanstalk-region-account-id` for each AWS Region in which you create environments. Within this bucket, logs are stored under the path `resources/environments/logs/logtype/environment-id/instance-id`.

For example, logs from instance `i-0a1fd158`, in Elastic Beanstalk environment `e-mpcwnwheky` in AWS Region `us-west-2` in account `123456789012`, are stored in the following locations:

- **Tail Logs** –

```
s3://elasticbeanstalk-us-west-2-123456789012/resources/environments/logs/tail/e-mpcwnwheky/i-0a1fd158
```

- **Bundle Logs** –

```
s3://elasticbeanstalk-us-west-2-123456789012/resources/environments/logs/bundle/e-mpcwnwheky/i-0a1fd158
```

- **Rotated Logs** –

```
s3://elasticbeanstalk-us-west-2-123456789012/resources/environments/logs/publish/e-mpcwnwheky/i-0a1fd158
```

Note

You can find your environment ID in the environment management console.

Elastic Beanstalk deletes tail and bundle logs from Amazon S3 automatically 15 minutes after they are created. Rotated logs persist until you delete them or move them to S3 Glacier.

Log rotation settings on Linux

On Linux platforms, Elastic Beanstalk uses `logrotate` to rotate logs periodically. If configured, after a log is rotated locally, the log rotation task picks it up and uploads it to Amazon S3. Logs that are rotated locally don't appear in tail or bundle logs by default.

You can find Elastic Beanstalk configuration files for `logrotate` in `/etc/logrotate.elasticbeanstalk.hourly/`. These rotation settings are specific to the platform, and might change in future versions of the platform. For more information about the available settings and example configurations, run `man logrotate`.

The configuration files are invoked by cron jobs in `/etc/cron.hourly/`. For more information about cron, run `man cron`.

Extending the default log task configuration

Elastic Beanstalk uses files in subfolders of `/opt/elasticbeanstalk/tasks` (Linux) or `C:\Program Files\Amazon\ElasticBeanstalk\config` (Windows Server) on the Amazon EC2 instance to configure tasks for tail logs, bundle logs, and log rotation.

On Linux:

- **Tail Logs** –

```
/opt/elasticbeanstalk/tasks/taillogs.d/
```

- **Bundle Logs** –

```
/opt/elasticbeanstalk/tasks/bundlelogs.d/
```

- **Rotated Logs** –

```
/opt/elasticbeanstalk/tasks/publishlogs.d/
```

On Windows Server:

- **Tail Logs** –

```
c:\Program Files\Amazon\ElasticBeanstalk\config\taillogs.d\
```

- **Rotated Logs** –

```
c:\Program Files\Amazon\ElasticBeanstalk\config\publogs.d\
```

For example, the `eb-activity.conf` file on Linux adds two log files to the tail logs task.

`/opt/elasticbeanstalk/tasks/taillogs.d/eb-activity.conf`

```
/var/log/eb-commandprocessor.log  
/var/log/eb-activity.log
```

You can use environment configuration files ([.ebextensions](#) (p. 600)) to add your own `.conf` files to these folders. A `.conf` file lists log files specific to your application, which Elastic Beanstalk adds to the log file tasks.

Use the [files](#) (p. 607) section to add configuration files to the tasks that you want to modify. For example, the following configuration text adds a log configuration file to each instance in your environment. This log configuration file, `cloud-init.conf`, adds `/var/log/cloud-init.log` to tail logs.

```
files:  
  "/opt/elasticbeanstalk/tasks/taillogs.d/cloud-init.conf" :  
    mode: "000755"  
    owner: root  
    group: root  
    content: |  
      /var/log/cloud-init.log
```

Add this text to a file with the `.config` file name extension to your source bundle under a folder named `.ebextensions`.

```
~/workspace/my-app  
|-- .ebextensions  
|  |-- tail-logs.config  
|-- index.php  
`-- styles.css
```

On Linux platforms, you can also use wildcard characters in log task configurations. This configuration file adds all files with the `.log` file name extension from the `log` folder in the application root to bundle logs.

```
files:
```

```
"/opt/elasticbeanstalk/tasks/bundlelogs.d/applogs.conf" :  
mode: "000755"  
owner: root  
group: root  
content: |  
    /var/app/current/log/*.log
```

Note

Log task configurations don't support wildcard characters on Windows platforms.

For more information about using configuration files, see [Advanced environment customization with configuration files \(.ebextensions\)](#) (p. 600).

Much like extending tail logs and bundle logs, you can extend log rotation using a configuration file. Whenever Elastic Beanstalk rotates its own logs and uploads them to Amazon S3, it also rotates and uploads your additional logs. Log rotation extension behaves differently depending on the platform's operating system. The following sections describe the two cases.

Extending log rotation on Linux

As explained in [Log rotation settings on Linux](#) (p. 735), Elastic Beanstalk uses `logrotate` to rotate logs on Linux platforms. When you configure your application's log files for log rotation, the application doesn't need to create copies of log files. Elastic Beanstalk configures `logrotate` to create a copy of your application's log files for each rotation. Therefore, the application must keep log files unlocked when it isn't actively writing to them.

Extending log rotation on Windows server

On Windows Server, when you configure your application's log files for log rotation, the application must rotate the log files periodically. Elastic Beanstalk looks for files with names starting with the pattern you configured, and picks them up for uploading to Amazon S3. In addition, periods in the file name are ignored, and Elastic Beanstalk considers the name up to the period to be the base log file name.

Elastic Beanstalk uploads all versions of a base log file except for the newest one, because it considers that one to be the active application log file, which can potentially be locked. Your application can, therefore, keep the active log file locked between rotations.

For example, your application writes to a log file named `my_log.log`, and you specify this name in your `.conf` file. The application periodically rotates the file. During the Elastic Beanstalk rotation cycle, it finds the following files in the log file's folder: `my_log.log`, `my_log.0800.log`, `my_log.0830.log`. Elastic Beanstalk considers all of them to be versions of the base name `my_log`. The file `my_log.log` has the latest modification time, so Elastic Beanstalk uploads only the other two files, `my_log.0800.log` and `my_log.0830.log`.

Streaming log files to Amazon CloudWatch Logs

You can configure your environment to stream logs to Amazon CloudWatch Logs in the Elastic Beanstalk console or by using [configuration options](#) (p. 536). With CloudWatch Logs, each instance in your environment streams logs to log groups that you can configure to be retained for weeks or years, even after your environment is terminated.

The set of logs streamed varies per environment, but always includes `eb-activity.log` and access logs from the nginx or Apache proxy server that runs in front of your application.

You can configure log streaming in the Elastic Beanstalk console either [during environment creation](#) (p. 371) or [for an existing environment](#) (p. 524). In the following example, logs are saved for up to seven days, even when the environment is terminated.

Instance log streaming to CloudWatch Logs
Configure the instances in your environment to stream logs to CloudWatch Logs. You can set the retention to up to ten years and configure Elastic Beanstalk logs when you terminate your environment.

Log groups
[/aws/elasticbeanstalk/GettingStartedApp-env](#)

Log streaming
(Standard CloudWatch charges apply.)
 Enabled

Retention
7 days

Lifecycle
Keep logs after terminating environment

The following [configuration file \(p. 600\)](#) enables log streaming with 180 days retention, even if the environment is terminated.

Example .ebextensions/log-streaming.config

```
option_settings:  
  aws:elasticbeanstalk:cloudwatch:logs:  
    StreamLogs: true  
    DeleteOnTerminate: false  
    RetentionInDays: 180
```

Using Elastic Beanstalk with other AWS services

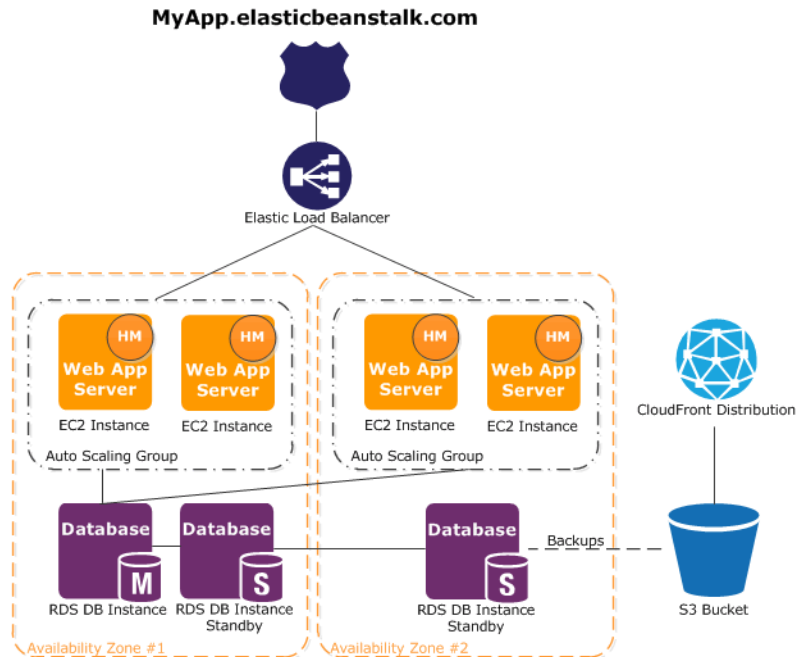
To implement your application's environments, Elastic Beanstalk manages resources of other AWS services or uses their functionality. In addition, Elastic Beanstalk integrates with AWS services that it doesn't use directly as part of your environments. The topics in this section describe many ways you can use these additional services with your Elastic Beanstalk application.

Topics

- [Architectural overview \(p. 739\)](#)
- [Using Elastic Beanstalk with Amazon CloudFront \(p. 740\)](#)
- [Logging Elastic Beanstalk API calls with AWS CloudTrail \(p. 741\)](#)
- [Using Elastic Beanstalk with Amazon CloudWatch \(p. 742\)](#)
- [Using Elastic Beanstalk with Amazon CloudWatch Logs \(p. 743\)](#)
- [Finding and tracking Elastic Beanstalk resources with AWS Config \(p. 752\)](#)
- [Using Elastic Beanstalk with Amazon DynamoDB \(p. 756\)](#)
- [Using Elastic Beanstalk with Amazon ElastiCache \(p. 757\)](#)
- [Using Elastic Beanstalk with Amazon Elastic File System \(p. 758\)](#)
- [Using Elastic Beanstalk with AWS Identity and Access Management \(p. 759\)](#)
- [Using Elastic Beanstalk with Amazon RDS \(p. 820\)](#)
- [Using Elastic Beanstalk with Amazon S3 \(p. 831\)](#)
- [Using Elastic Beanstalk with Amazon VPC \(p. 834\)](#)

Architectural overview

The following diagram illustrates an example architecture of Elastic Beanstalk across multiple Availability Zones working with other AWS products such as Amazon CloudFront, Amazon Simple Storage Service (Amazon S3), and Amazon Relational Database Service (Amazon RDS).



To plan for fault-tolerance, it is advisable to have N+1 Amazon EC2 instances and spread your instances across multiple Availability Zones. In the unlikely case that one Availability Zone goes down, you will still have your other Amazon EC2 instances running in another Availability Zone. You can adjust Amazon EC2 Auto Scaling to allow for a minimum number of instances as well as multiple Availability Zones. For instructions on how to do this, see [Auto Scaling group for your Elastic Beanstalk environment \(p. 457\)](#). For more information about building fault-tolerant applications, go to [Building Fault-Tolerant Applications on AWS](#).

The following sections discuss in more detail integration with Amazon CloudFront, Amazon CloudWatch, Amazon DynamoDB Amazon ElastiCache, Amazon RDS, Amazon Route 53, Amazon Simple Storage Service, Amazon VPC , and IAM.

Using Elastic Beanstalk with Amazon CloudFront

Amazon CloudFront is a web service that speeds up distribution of your static and dynamic web content, for example, .html, .css, .php, image, and media files, to end users. CloudFront delivers your content through a worldwide network of edge locations. When an end user requests content that you're serving with CloudFront, the user is routed to the edge location that provides the lowest latency, so content is delivered with the best possible performance. If the content is already in that edge location, CloudFront delivers it immediately. If the content is not currently in that edge location, CloudFront retrieves it from an Amazon S3 bucket or an HTTP server (for example, a web server) that you have identified as the source for the definitive version of your content.

After you have created and deployed your Elastic Beanstalk application you can sign up for CloudFront and start using CloudFront to distribute your content. Learn more about CloudFront from the [Amazon CloudFront Developer Guide](#).

Logging Elastic Beanstalk API calls with AWS CloudTrail

Elastic Beanstalk is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Elastic Beanstalk. CloudTrail captures all API calls for Elastic Beanstalk as events, including calls from the Elastic Beanstalk console, from the EB CLI, and from your code to the Elastic Beanstalk APIs. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Elastic Beanstalk. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Elastic Beanstalk, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

Elastic Beanstalk information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Elastic Beanstalk, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your AWS account, including events for Elastic Beanstalk, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all regions. The trail logs events from all regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- [Overview for Creating a Trail](#)
- [CloudTrail Supported Services and Integrations](#)
- [Configuring Amazon SNS Notifications for CloudTrail](#)
- [Receiving CloudTrail Log Files from Multiple Regions](#) and [Receiving CloudTrail Log Files from Multiple Accounts](#)

All Elastic Beanstalk actions are logged by CloudTrail and are documented in the [AWS Elastic Beanstalk API Reference](#). For example, calls to the `DescribeApplications`, `UpdateEnvironment`, and `ListTagsForResource` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](#).

Understanding Elastic Beanstalk log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from

any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `UpdateEnvironment` action called by an IAM user named `intern`, for the `sample-env` environment in the `sample-app` application.

```
{
  "Records": [{
    "eventVersion": "1.05",
    "userIdentity": {
      "type": "IAMUser",
      "principalId": "AIXDAYQEXAMPLEUMLYNGI",
      "arn": "arn:aws:iam:123456789012:user/intern",
      "accountId": "123456789012",
      "accessKeyId": "ASXIAGXEXAMPLEQULKXNV",
      "userName": "intern",
      "sessionContext": {
        "attributes": {
          "mfaAuthenticated": "false",
          "creationDate": "2016-04-22T00:23:24Z"
        }
      }
    },
    "invokedBy": "signin.amazonaws.com"
  },
  "eventTime": "2016-04-22T00:24:14Z",
  "eventSource": "elasticbeanstalk.amazonaws.com",
  "eventName": "UpdateEnvironment",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "255.255.255.54",
  "userAgent": "signin.amazonaws.com",
  "requestParameters": {
    "applicationName": "sample-app",
    "environmentName": "sample-env",
    "optionSettings": []
  },
  "responseElements": null,
  "requestID": "84ae9ecf-0280-17ce-8612-705c7b132321",
  "eventID": "e48b6a08-c6be-4a22-99e1-c53139chfb18",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}]
}
```

Using Elastic Beanstalk with Amazon CloudWatch

Amazon CloudWatch enables you to monitor, manage, and publish various metrics, as well as configure alarm actions based on data from metrics. Amazon CloudWatch monitoring enables you to collect, analyze, and view system and application metrics so that you can make operational and business decisions more quickly and with greater confidence.

You can use Amazon CloudWatch to collect metrics about your Amazon Web Services (AWS) resources—such as the performance of your Amazon EC2 instances. You can also publish your own metrics directly to Amazon CloudWatch. Amazon CloudWatch alarms help you implement decisions more easily by enabling you to send notifications or automatically make changes to the resources you are monitoring, based on rules that you define. For example, you can create alarms that initiate Amazon EC2 Auto Scaling and Amazon Simple Notification Service (Amazon SNS) actions on your behalf.

Elastic Beanstalk automatically uses Amazon CloudWatch to help you monitor your application and environment status. You can navigate to the Amazon CloudWatch console to see your dashboard and get

an overview of all of your resources as well as your alarms. You can also choose to view more metrics or add custom metrics.

For more information about Amazon CloudWatch, go to the [Amazon CloudWatch Developer Guide](#). For an example of how to use Amazon CloudWatch with Elastic Beanstalk, see the section called “[Example: Using custom amazon CloudWatch metrics](#)” (p. 612).

Using Elastic Beanstalk with Amazon CloudWatch Logs

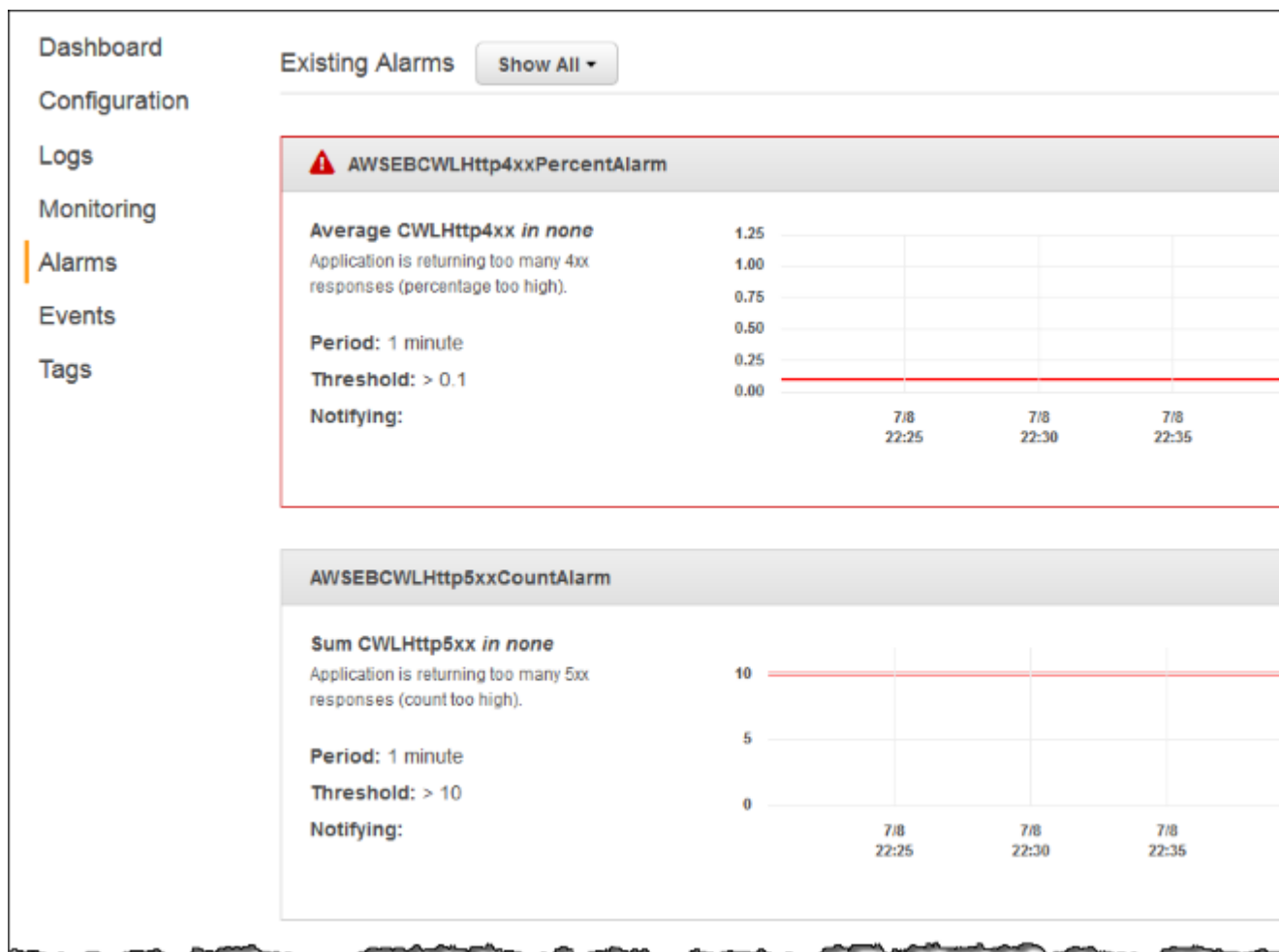
With CloudWatch Logs, you can monitor and archive your Elastic Beanstalk application, system, and custom log files from Amazon EC2 instances of your environments. You can also configure alarms that make it easier for you to react to specific log stream events that your metric filters extract. The CloudWatch Logs agent installed on each Amazon EC2 instance in your environment publishes metric data points to the CloudWatch service for each log group you configure. Each log group applies its own filter patterns to determine what log stream events to send to CloudWatch as data points. Log streams that belong to the same log group share the same retention, monitoring, and access control settings. You can configure Elastic Beanstalk to automatically stream logs to the CloudWatch service, as described in [Streaming instance logs to CloudWatch Logs](#) (p. 748). For more information about CloudWatch Logs, including terminology and concepts, see the [Amazon CloudWatch Logs User Guide](#).

In addition to instance logs, if you enable [enhanced health](#) (p. 691) for your environment, you can configure the environment to stream health information to CloudWatch Logs. See [Streaming Elastic Beanstalk environment health information to Amazon CloudWatch Logs](#) (p. 750).

The following figure shows the **Monitoring** page and graphs for an environment that is configured with CloudWatch Logs integration. The example metrics in this environment are named **CWLHttp4xx** and **CWLHttp5xx**. One of the graphs shows that the **CWLHttp4xx** metric has triggered an alarm based on conditions specified in the configuration files.



The following figure shows the **Alarms** page and graphs for the example alarms named **AWSEBCWLHttp4xxPercentAlarm** and **AWSEBCWLHttp5xxCountAlarm** that correspond to the **CWLHttp4xx** and **CWLHttp5xx** metrics, respectively.



Topics

- [Prerequisites to instance log streaming to CloudWatch Logs \(p. 745\)](#)
- [How Elastic Beanstalk sets up CloudWatch Logs \(p. 746\)](#)
- [Streaming instance logs to CloudWatch Logs \(p. 748\)](#)
- [Troubleshooting CloudWatch Logs integration \(p. 750\)](#)
- [Streaming Elastic Beanstalk environment health information to Amazon CloudWatch Logs \(p. 750\)](#)

Prerequisites to instance log streaming to CloudWatch Logs

To enable streaming of logs from your environment's Amazon EC2 instances to CloudWatch Logs, you must meet the following conditions.

- *Platform* – Because this feature is only available in platform versions released on or after [this release](#), if you are using an earlier platform version, update your environment to a current one.
- If you don't have the *AWSElasticBeanstalkWebTier* or *AWSElasticBeanstalkWorkerTier* Elastic Beanstalk managed policy in your [Elastic Beanstalk instance profile \(p. 21\)](#), you must add the following to your profile to enable this feature.


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogStream"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

How Elastic Beanstalk sets up CloudWatch Logs

Elastic Beanstalk installs a CloudWatch log agent with the default configuration settings on each instance it creates. Learn more in the [CloudWatch Logs Agent Reference](#).

When you enable instance log streaming to CloudWatch Logs, Elastic Beanstalk sends log files from your environment's instances to CloudWatch Logs. Different platforms stream different logs. The following table lists the logs, by platform.

Platform	Logs
Docker	<ul style="list-style-type: none">• /var/log/eb-activity.log• /var/log/nginx/error.log• /var/log/docker-events.log• /var/log/docker• /var/log/nginx/access.log• /var/log/eb-docker/containers/eb-current-app/stdouterr.log
Multi-Docker(generic)	<ul style="list-style-type: none">• /var/log/eb-activity.log• /var/log/ecs/ecs-init.log• /var/log/eb-ecs-mgr.log• /var/log/ecs/ecs-agent.log• /var/log/docker-events.log
Glass fish (Preconfigured Docker)	<ul style="list-style-type: none">• /var/log/eb-activity.log• /var/log/nginx/error.log• /var/log/docker-events.log• /var/log/docker• /var/log/nginx/access.log
Go (Preconfigured Docker)	<ul style="list-style-type: none">• /var/log/eb-activity.log• /var/log/nginx/error.log• /var/log/docker-events.log• /var/log/docker• /var/log/nginx/access.log

Platform	Logs
Python (Preconfigured Docker)	<ul style="list-style-type: none"> • /var/log/eb-activity.log • /var/log/nginx/error.log • /var/log/docker-events.log • /var/log/docker • /var/log/nginx/access.log
Go	<ul style="list-style-type: none"> • /var/log/eb-activity.log • /var/log/nginx/error.log • /var/log/nginx/access.log
Java	<ul style="list-style-type: none"> • /var/log/eb-activity.log • /var/log/nginx/access.log • /var/log/nginx/error.log • /var/log/web-1.error.log • /var/log/web-1.log
Tomcat	<ul style="list-style-type: none"> • /var/log/eb-activity.log • /var/log/httpd/error_log • /var/log/httpd/access_log • /var/log/nginx/error_log • /var/log/nginx/access_log
.NET on Windows Server	<ul style="list-style-type: none"> • C:\inetpub\logs\LogFiles\W3SVC1\u_ex*.log • C:\Program Files\Amazon\ElasticBeanstalk\logs\AWSDeployment.log • C:\Program Files\Amazon\ElasticBeanstalk\logs\Hooks.log
Node.js	<ul style="list-style-type: none"> • /var/log/eb-activity.log • /var/log/nodejs/nodejs.log • /var/log/nginx/error.log • /var/log/nginx/access.log • /var/log/httpd/error.log • /var/log/httpd/access.log
PHP	<ul style="list-style-type: none"> • /var/log/eb-activity.log • /var/log/httpd/error_log • /var/log/httpd/access_log
Python	<ul style="list-style-type: none"> • /var/log/eb-activity.log • /var/log/httpd/error_log • /var/log/httpd/access_log • /opt/python/log/supervisord.log
Ruby (Puma)	<ul style="list-style-type: none"> • /var/log/eb-activity.log • /var/log/nginx/error.log • /var/log/puma/puma.log • /var/log/nginx/access.log

Platform	Logs
Ruby (Passenger)	<ul style="list-style-type: none"> • /var/log/eb-activity.log • /var/app/support/logs/passenger.log • /var/app/support/logs/access.log • /var/app/support/logs/error.log

Elastic Beanstalk configures log groups in CloudWatch Logs for the various log files that it streams. To retrieve specific log files from CloudWatch Logs, you have to know the name of the corresponding log group. The log group naming scheme depends on the platform's operating system.

For Linux platforms, prefix the on-instance log file location with `/aws/elasticbeanstalk/environment_name` to get the log group name. For example, to retrieve the file `/var/log/nginx/error.log`, specify the log group `/aws/elasticbeanstalk/environment_name/var/log/nginx/error.log`.

For Windows platforms, see the following table for the log group corresponding to each log file.

On-instance log file	Log group
C:\Program Files\Amazon\ElasticBeanstalk\logs\AWSDeployment.log	/aws/elasticbeanstalk/<environment-name>/EBDeploy-Log
C:\Program Files\Amazon\ElasticBeanstalk\logs\Hooks.log	/aws/elasticbeanstalk/<environment-name>/EBHooks-Log
C:\inetpub\logs\LogFiles (the entire directory)	/aws/elasticbeanstalk/<environment-name>/IIS-Log

Streaming instance logs to CloudWatch Logs

You can enable instance log streaming to CloudWatch Logs using the Elastic Beanstalk console, the EB CLI, or configuration options.

Before you enable it, set up IAM permissions to use with the CloudWatch Logs agent. You can attach the following custom policy to the [instance profile \(p. 21\)](#) that you assign to your environment.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:GetLogEvents",
        "logs:PutLogEvents",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:PutRetentionPolicy"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

```
}  
  ]  
}
```

Instance log streaming using the Elastic Beanstalk console

To stream instance logs to CloudWatch Logs

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. Under **Instance log streaming to CloudWatch Logs**:
 - Enable **Log streaming**.
 - Set **Retention** to the number of days to save the logs.
 - Select the **Lifecycle** setting that determines whether the logs are saved after the environment is terminated.
6. Choose **Apply**.

After you enable log streaming, you can return to the **Software** configuration category or page and find the **Log Groups** link. Click this link to see your logs in the CloudWatch console.

Instance log streaming using the EB CLI

To enable instance log streaming to CloudWatch Logs using the EB CLI, use the [eb logs \(p. 913\)](#) command.

```
$ eb logs --cloudwatch-logs enable
```

You can also use **eb logs** to retrieve logs from CloudWatch Logs. You can retrieve all the environment's instance logs, or use the command's many options to specify subsets of logs to retrieve. For example, the following command retrieves the complete set of instance logs for your environment, and saves them to a directory under `.elasticbeanstalk/logs`.

```
$ eb logs --all
```

In particular, the `--log-group` option enables you to retrieve instance logs of a specific log group, corresponding to a specific on-instance log file. To do that, you need to know the name of the log group that corresponds to the log file you want to retrieve. You can find this information in [How Elastic Beanstalk sets up CloudWatch Logs \(p. 746\)](#).

Instance log streaming using configuration files

When you create or update an environment, you can use a configuration file to set up and configure instance log streaming to CloudWatch Logs. The following example configuration file enables default instance log streaming. Elastic Beanstalk streams the default set of log files for your

environment's platform. To use the example, copy the text into a file with the `.config` extension in the `.ebextensions` directory at the top level of your application source bundle.

```
option_settings:
- namespace: aws:elasticbeanstalk:cloudwatch:logs
  option_name: StreamLogs
  value: true
```

Custom log file streaming

The Elastic Beanstalk integration with CloudWatch Logs doesn't directly support the streaming of custom log files that your application generates. To stream custom logs, use a configuration file to directly install the CloudWatch Logs agent and to configure the files to be pushed. For an example configuration file, see [logs-streamtocloudwatch-linux.config](#).

Note

The example doesn't work on the Windows platform.

For more information about configuring CloudWatch Logs, see the [CloudWatch Logs Agent Reference](#) in the *Amazon CloudWatch Logs User Guide*.

Troubleshooting CloudWatch Logs integration

If you can't find some of the environment's instance logs you expect in CloudWatch Logs, you can investigate the following common issues:

- Your IAM role lacks the required IAM permissions.
- You launched your environment in an AWS Region that doesn't support CloudWatch Logs.
- One of your custom log files doesn't exist in the path you specified.

Streaming Elastic Beanstalk environment health information to Amazon CloudWatch Logs

If you enable [enhanced health \(p. 691\)](#) reporting for your environment, you can configure the environment to stream health information to CloudWatch Logs. This streaming is independent from Amazon EC2 instance log streaming. This topic describes environment health information streaming. For information about instance log streaming, see [Using Elastic Beanstalk with Amazon CloudWatch Logs \(p. 743\)](#).

When you configure environment health streaming, Elastic Beanstalk creates a CloudWatch Logs log group for environment health. The log group's name is `/aws/elasticbeanstalk/environment-name/environment-health`.log. Within this log group, Elastic Beanstalk creates log streams named `YYYY-MM-DD#<hash-suffix>` (there might be more than one log stream per date).

When the environment's health status changes, Elastic Beanstalk adds a record to the health log stream. The record represents the health status transition—the new status and a description of the cause of change. For example, an environment's status might change to Severe because the load balancer is failing. For a description of enhanced health statuses, see [Health colors and statuses \(p. 706\)](#).

Prerequisites to environment health streaming to CloudWatch Logs

To enable environment health streaming to CloudWatch Logs, you must meet the following conditions:

- *Platform* – You must be using a platform version that supports enhanced health reporting.
- *Permissions* – You must grant certain logging-related permissions to Elastic Beanstalk so that it can act on your behalf to stream health information for your environment. If your environment isn't using a service role that Elastic Beanstalk created for it, `aws-elasticbeanstalk-service-role`, or your account's service-linked role, `AWSServiceRoleForElasticBeanstalk`, be sure to add the following permissions to your custom service role.

```
{
  "Effect": "Allow",
  "Action": [
    "logs:DescribeLogStreams",
    "logs:CreateLogStream",
    "logs:PutLogEvents"
  ],
  "Resource": "arn:aws:logs:*:*:log-group:/aws/elasticbeanstalk/*:log-stream:*"
}
```

Streaming environment health logs to CloudWatch Logs

You can enable environment health streaming to CloudWatch Logs using the Elastic Beanstalk console, the EB CLI, or configuration options.

Environment health log streaming using the Elastic Beanstalk console

To stream environment health logs to CloudWatch Logs

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Monitoring** configuration category, choose **Edit**.
5. Under **Health reporting**, make sure that the reporting **System** is set to **Enhanced**.
6. Under **Health event streaming to CloudWatch Logs**
 - Enable **Log streaming**.
 - Set **Retention** to the number of days to save the logs.
 - Select the **Lifecycle** setting that determines whether the logs are saved after the environment is terminated.
7. Choose **Apply**.

After you enable log streaming, you can return to the **Monitoring** configuration category or page and find the **Log Group** link. Click this link to see your environment health logs in the CloudWatch console.

Environment health log streaming using the EB CLI

To enable environment health log streaming to CloudWatch Logs using the EB CLI, use the [eb logs \(p. 913\)](#) command.

```
$ eb logs --cloudwatch-logs enable --cloudwatch-log-source environment-health
```

You can also use **eb logs** to retrieve logs from CloudWatch Logs. For example, the following command retrieves all the health logs for your environment, and saves them to a directory under `.elasticbeanstalk/logs`.

```
$ eb logs --all --cloudwatch-log-source environment-health
```

Environment health log streaming using configuration files

When you create or update an environment, you can use a configuration file to set up and configure environment health log streaming to CloudWatch Logs. To use the example below, copy the text into a file with the `.config` extension in the `.ebextensions` directory at the top level of your application source bundle. The example configures Elastic Beanstalk to enable environment health log streaming, keep the logs after terminating the environment, and save them for 30 days.

Example Health streaming configuration file

```
#####  
## Sets up Elastic Beanstalk to stream environment health information  
## to Amazon CloudWatch Logs.  
## Works only for environments that have enhanced health reporting enabled.  
#####  
  
option_settings:  
  aws:elasticbeanstalk:cloudwatch:logs:health:  
    HealthStreamingEnabled: true  
    ### Settings below this line are optional.  
    # DeleteOnTerminate: Delete the log group when the environment is  
    # terminated. Default is false. If false, the health data is kept  
    # RetentionInDays days.  
    DeleteOnTerminate: false  
    # RetentionInDays: The number of days to keep the archived health data  
    # before it expires, if DeleteOnTerminate isn't set. Default is 7 days.  
    RetentionInDays: 30
```

For option defaults and valid values, see [aws:elasticbeanstalk:cloudwatch:logs:health \(p. 570\)](#).

Finding and tracking Elastic Beanstalk resources with AWS Config

[AWS Config](#) provides a detailed view of the configuration of AWS resources in your AWS account. You can see how resources are related, get a history of configuration changes, and see how relationships and configurations change over time. You can use AWS Config to define rules that evaluate resource configurations for data compliance.

Several Elastic Beanstalk resource types are integrated with AWS Config:

- Applications
- Application Versions
- Environments

The following section shows how to configure AWS Config to record resources of these types.

For more information about AWS Config, see the [AWS Config Developer Guide](#). For pricing information, see the [AWS Config pricing information page](#).

Setting up AWS Config

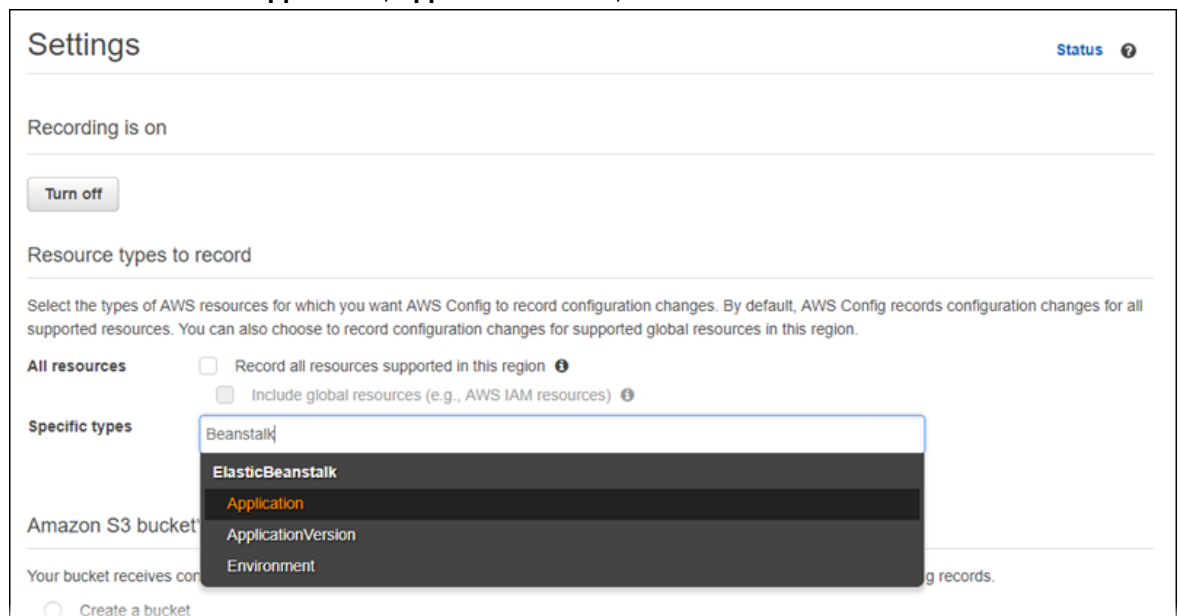
To initially set up AWS Config, see the following topics in the [AWS Config Developer Guide](#).

- [Setting up AWS Config with the Console](#)
- [Setting up AWS Config with the AWS CLI](#)

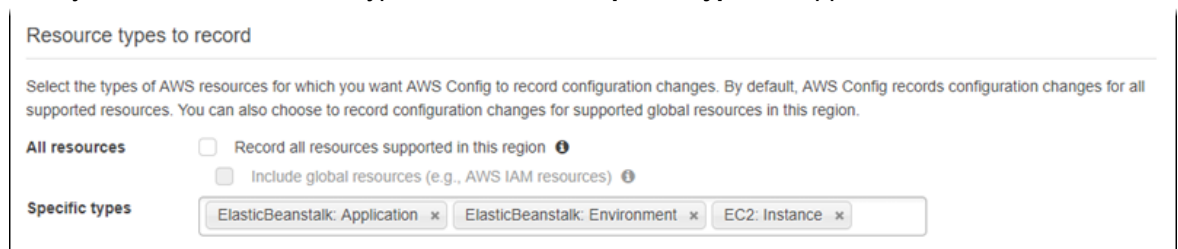
Configuring AWS Config to record Elastic Beanstalk resources

By default, AWS Config records configuration changes for all supported types of *regional resources* that it discovers in the region in which your environment is running. You can customize AWS Config to record changes only for specific resource types, or changes to *global resources*. For example, you can configure AWS Config to record changes for Elastic Beanstalk resources and a subset of other AWS resources that Elastic Beanstalk starts for you.

The following figure shows the AWS Config **Settings** page, with Elastic Beanstalk resource types that you can choose to record: **Application**, **ApplicationVersion**, and **Environment**.



After you select a few resource types, this is how the **Specific types** list appears.



To learn about *regional* vs. *global* resources, and for the full customization procedure, see [Selecting which Resources AWS Config Records](#).

Viewing Elastic Beanstalk configuration details in the AWS Config console

You can use the AWS Config console to look for Elastic Beanstalk resources, and get current and historical details about their configurations. The following example shows how to find information about an Elastic Beanstalk environment.

To find an Elastic Beanstalk environment in the AWS Config console

1. Open the [AWS Config console](#).
2. Choose **Resources**.
3. On the **Resource inventory** page, choose **Resources**.
4. Open the **Resource type** menu, scroll to **ElasticBeanstalk**, and then choose one or more of the Elastic Beanstalk resource types. See 1 in the following figure.

Note

To view configuration details for other resources that Elastic Beanstalk created for your application, choose additional resource types. For example, you can choose **Instance** under **EC2**.

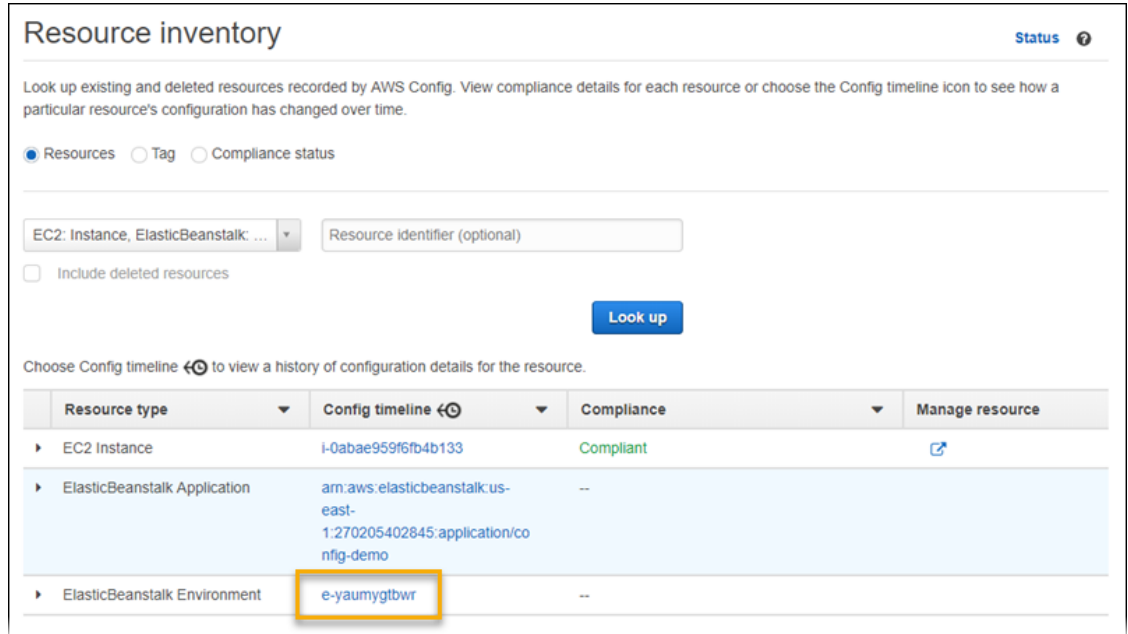
5. Choose **Look up**. See 2 in the following figure.

The screenshot shows the AWS Config console's 'Resource inventory' page. At the top, there's a 'Status' icon. Below it, a description states: 'Look up existing and deleted resources recorded by AWS Config. View compliance details for each resource or choose the Config timeline icon to see how a particular resource's configuration has changed over time.' There are three radio buttons: 'Resources' (selected), 'Tag', and 'Compliance status'. A search bar labeled 'Resource identifier (optional)' is present. Below the search bar is a 'Look up' button. To the left, a dropdown menu for 'ElasticBeanstalk: Application, Ela...' is open, showing a list of resource types with checkboxes. 'ElasticBeanstalk' is selected, and its sub-items 'Application', 'ApplicationVersion', and 'Environment' are also checked. The main table displays a list of resources with columns for 'Config timeline', 'Compliance', and 'Manage resource'. The first resource has ID 'i-0abae959f6fb4b133' and is 'Compliant'. Other resources include 'am:aws:elasticbeanstalk:us-east-1:270205402845:application/config-demo' and 'e-yaumygtbwr'.

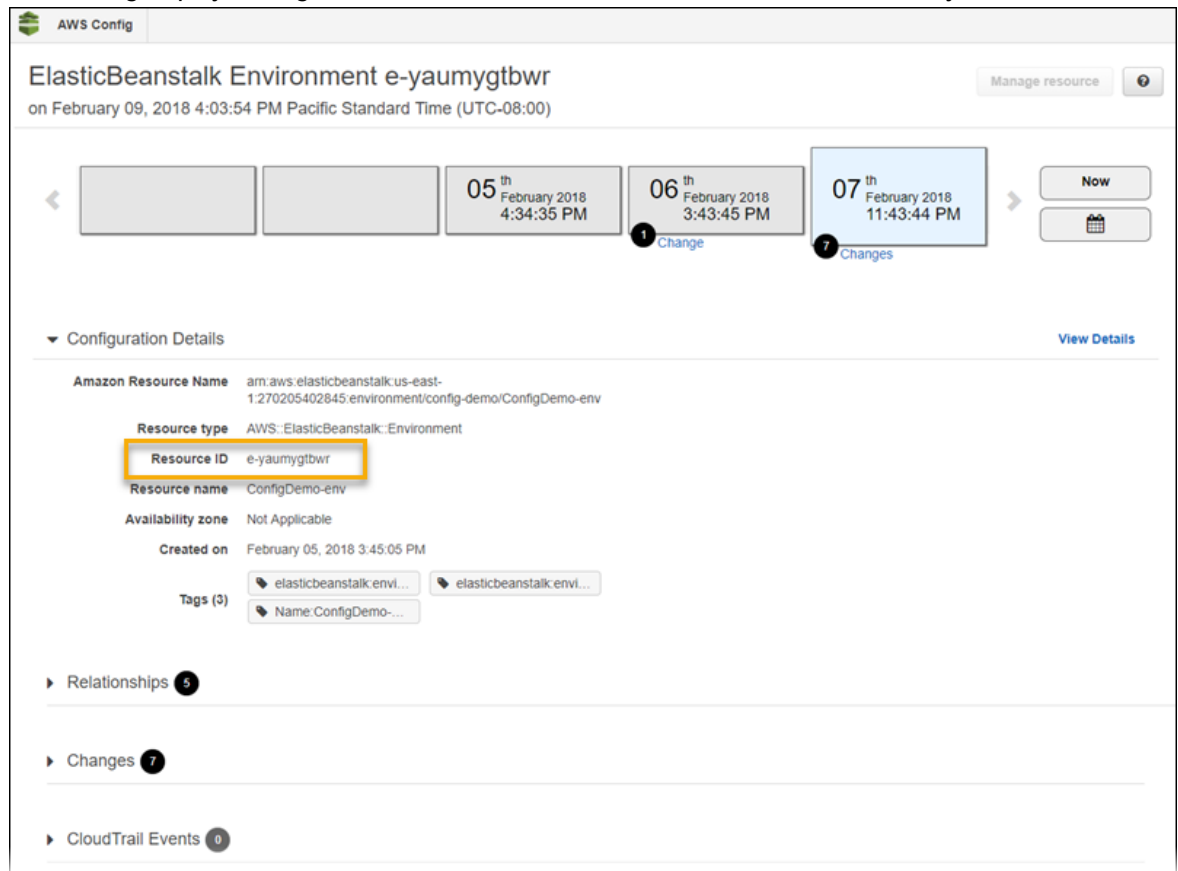
6. Choose a resource ID in the list of resources that AWS Config displays.

AWS Elastic Beanstalk Developer Guide

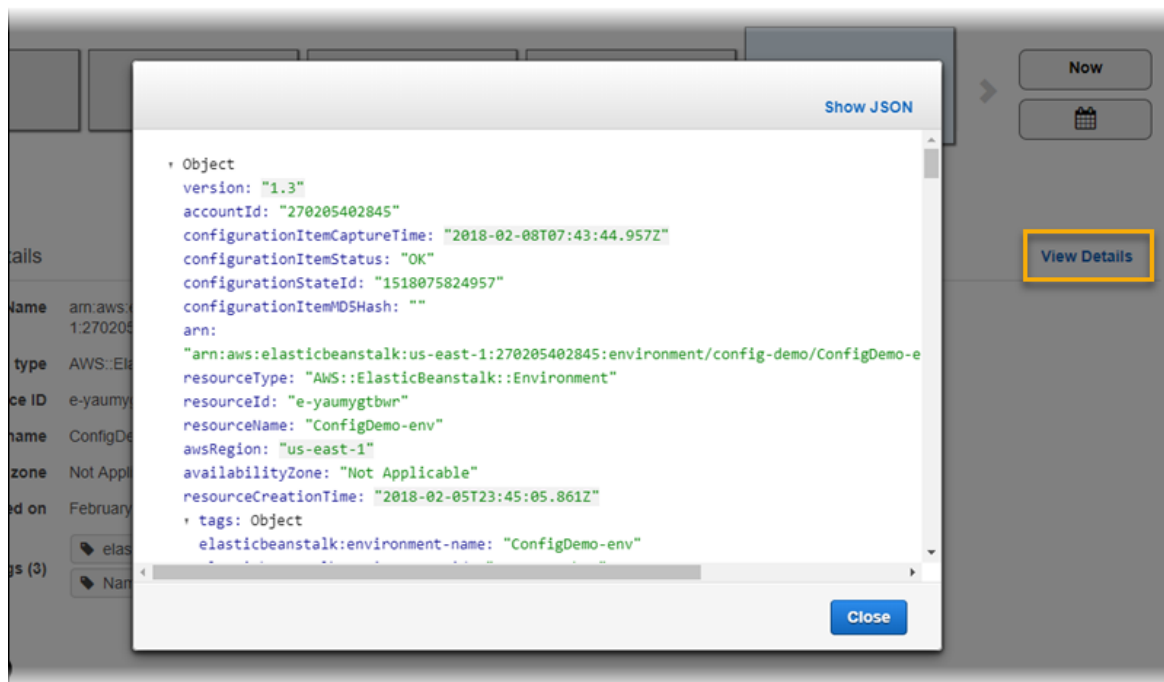
Viewing Elastic Beanstalk configuration details in the AWS Config console



AWS Config displays configuration details and other information about the resource you selected.



To see the full details of the recorded configuration, choose **View Details**.



To learn more ways to find a resource and view information on this page, see [Viewing AWS Resource Configurations and History](#) in the *AWS Config Developer Guide*.

Evaluating Elastic Beanstalk resources using AWS Config rules

You can create AWS Config rules, which represent the ideal configuration settings for your Elastic Beanstalk resources. You can use predefined *AWS Managed Config Rules*, or define custom rules. AWS Config continuously tracks changes to the configuration of your resources to determine whether those changes violate any of the conditions in your rules. The AWS Config console shows the compliance status of your rules and resources.

If a resource violates a rule and is flagged as *noncompliant*, AWS Config can alert you using an [Amazon Simple Notification Service \(Amazon SNS\)](#) topic. To programmatically consume the data in these AWS Config alerts, use an [Amazon Simple Queue Service \(Amazon SQS\)](#) queue as the notification endpoint for the Amazon SNS topic. For example, you might want to write code that starts a workflow when someone modifies your environment's Auto Scaling group configuration.

To learn more about setting up and using rules, see [Evaluating Resources with AWS Config Rules](#) in the *AWS Config Developer Guide*.

Using Elastic Beanstalk with Amazon DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. If you are a developer, you can use DynamoDB to create a database table that can store and retrieve any amount of data, and serve any level of request traffic. DynamoDB automatically spreads the data and traffic for the table over a sufficient number of servers to handle the request capacity specified by the customer and the amount of data stored, while maintaining consistent and fast performance. All data items are stored on solid state drives (SSDs) and

are automatically replicated across multiple Availability Zones in an AWS Region to provide built-in high availability and data durability.

If you use [periodic tasks \(p. 440\)](#) in a worker environment, Elastic Beanstalk creates a DynamoDB table and uses it to perform leader election and store information about the task. Each instance in the environment attempts to write to the table every few seconds to become leader and perform the task when scheduled.

You can use [configuration files \(p. 600\)](#) to create a DynamoDB table for your application. See [eb-node-express-sample](#) on GitHub for a sample Node.js application that creates a table with a configuration file and connects to it with the AWS SDK for JavaScript in Node.js. For an example walkthrough using DynamoDB with PHP, see [Example: DynamoDB, CloudWatch, and SNS \(p. 636\)](#). For an example that uses the AWS SDK for Java, see [Manage Tomcat Session State with DynamoDB](#) in the AWS SDK for Java documentation.

When you create a DynamoDB table using configuration files, the table isn't tied to your environment's lifecycle, and isn't deleted when you terminate your environment. To ensure that personal information isn't unnecessarily retained, delete any records that you don't need anymore, or delete the table.

For more information about DynamoDB, see the [DynamoDB Developer Guide](#).

Using Elastic Beanstalk with Amazon ElastiCache

Amazon ElastiCache is a web service that enables setting up, managing, and scaling distributed in-memory cache environments in the cloud. It provides a high-performance, scalable, and cost-effective in-memory cache, while removing the complexity associated with deploying and managing a distributed cache environment. ElastiCache is protocol-compliant with Redis and Memcached, so the code, applications, and most popular tools that you use today with your existing Redis and Memcached environments will work seamlessly with the service. For more information about ElastiCache, go to the [Amazon ElastiCache](#) product page.

To use Elastic Beanstalk with Amazon ElastiCache

1. Create an ElastiCache cluster.
 - For instructions on how to create an ElastiCache cluster with Redis, go to [Getting Started with Amazon ElastiCache for Redis](#) in the *ElastiCache for Redis User Guide*.
 - For instructions on how to create an ElastiCache cluster with Memcached, go to [Getting Started with Amazon ElastiCache for Memcached](#) in the *ElastiCache for Memcached User Guide*.
2. Configure your ElastiCache Security Group to allow access from the Amazon EC2 security group used by your Elastic Beanstalk application. For instructions on how to find the name of your EC2 security group using the AWS Management Console, see [Security groups \(p. 455\)](#) on the *EC2 Instances* document page.
 - For more information on Redis, go to [Authorize Access](#) in the *ElastiCache for Redis User Guide*.
 - For more information on Memcached, go to [Authorize Access](#) in the *ElastiCache for Memcached User Guide*.

You can use configuration files to customize your Elastic Beanstalk environment to use ElastiCache. For configuration file examples that integrate ElastiCache with Elastic Beanstalk, see [Example: ElastiCache \(p. 628\)](#).

Using Elastic Beanstalk with Amazon Elastic File System

With Amazon Elastic File System (Amazon EFS), you can create network file systems that can be mounted by instances across multiple Availability Zones. An Amazon EFS file system is an AWS resource that uses security groups to control access over the network in your default or custom VPC.

In an Elastic Beanstalk environment, you can use Amazon EFS to create a shared directory that stores files uploaded or modified by users of your application. Your application can treat a mounted Amazon EFS volume like local storage, so you don't have to change your application code to scale up to multiple instances.

For more information about Amazon EFS, see the [Amazon Elastic File System User Guide](#).

Sections

- [Configuration files \(p. 758\)](#)
- [Encrypted file systems \(p. 759\)](#)
- [Sample applications \(p. 759\)](#)
- [Cleaning up file systems \(p. 759\)](#)

Configuration files

Elastic Beanstalk provides [configuration files \(p. 600\)](#) that you can use to create and mount Amazon EFS file systems. You can create an Amazon EFS volume as part of your environment, or mount an Amazon EFS volume that you created independently of Elastic Beanstalk.

- **`storage-efs-createfilesystem.config`** – Uses the `Resources` key to create a new file system and mount points in Amazon EFS. All instances in your environment can connect to the same file system for shared, scalable storage. Use `storage-efs-mountfilesystem.config` to mount the file system on each instance.

Internal resources

Any resources that you create with configuration files are tied to the lifecycle of your environment and will be lost if you terminate your environment or remove the configuration file.

- **`storage-efs-mountfilesystem.config`** – Mount an Amazon EFS file system to a local path on the instances in your environment. You can create the volume as part of the environment with `storage-efs-createfilesystem.config`, or external to your environment by using the Amazon EFS console, AWS CLI, or AWS SDK.

To use the configuration files, start by creating your Amazon EFS file system with `storage-efs-createfilesystem.config`. Follow the instructions in the configuration file and add it to the [.ebextensions \(p. 600\)](#) directory in your source code to create the file system in your VPC.

Deploy your updated source code to your Elastic Beanstalk environment to confirm that the file system is created successfully. Then, add the `storage-efs-mountfilesystem.config` to mount the file system to the instances in your environment. Doing this in two separate deployments ensures that if the mount operation fails, the file system is left intact. If you do both in the same deployment, an issue with either step will cause the file system to terminate when the deployment fails.

Encrypted file systems

Amazon EFS supports encrypted file systems. The `storage-efs-createfilesystem.config` configuration file discussed in this topic defines two custom options that you can use to create an Amazon EFS encrypted file system. For details, follow the instructions in the configuration file.

Sample applications

Elastic Beanstalk also provides sample applications that use Amazon EFS for shared storage. The two projects are configuration files that you can use with a standard WordPress or Drupal installer to run a blog or other content management system in a load balanced environment. When a user uploads a photo or other media, it is stored on an Amazon EFS file system, avoiding the need to use a plugin to store uploaded files in Amazon S3.

- **Load Balanced WordPress** – Configuration files for installing WordPress securely and running it in a load balanced Elastic Beanstalk environment.
- **Load Balanced Drupal** – Configuration files and instructions for installing Drupal securely and running it in a load balanced Elastic Beanstalk environment.

Cleaning up file systems

If you created an Amazon EFS file system using a configuration file as part of your Elastic Beanstalk environment, Elastic Beanstalk removes the file system when you terminate the environment. To minimize storage costs of a running application, routinely delete files that your application doesn't need, or ensure that the application code maintains file lifecycle correctly.

In addition, if you created an Amazon EFS file system outside of an Elastic Beanstalk environment and mounted it to the environment's instances, be aware that Elastic Beanstalk doesn't remove the file system when you terminate the environment. To ensure that personal information isn't unnecessarily retained, delete files that your application stored if you don't need them anymore, or remove the file system.

Using Elastic Beanstalk with AWS Identity and Access Management

AWS Identity and Access Management (IAM) helps you securely control access to your AWS resources. This section includes reference materials for working with IAM policies, instance profiles, and service roles.

For an overview of permissions, see [Service roles, instance profiles, and user policies \(p. 20\)](#). For most environments, the service role and instance profile that the Elastic Beanstalk console prompts you to create when you launch your first environment have all of the permissions that you need. Likewise, the [managed policies \(p. 775\)](#) provided by Elastic Beanstalk for full access and read-only access contain all of the user permissions required for daily use.

The [IAM User Guide](#) provides in-depth coverage of AWS permissions.

Topics

- [Managing Elastic Beanstalk instance profiles \(p. 760\)](#)
- [Managing Elastic Beanstalk service roles \(p. 764\)](#)
- [Using service-linked roles for Elastic Beanstalk \(p. 769\)](#)

- [Managing Elastic Beanstalk user policies \(p. 775\)](#)
- [Amazon resource name format for Elastic Beanstalk \(p. 781\)](#)
- [Resources and conditions for Elastic Beanstalk actions \(p. 782\)](#)
- [Using tags to control access to Elastic Beanstalk resources \(p. 807\)](#)
- [Example policies based on managed policies \(p. 810\)](#)
- [Example policies based on resource permissions \(p. 813\)](#)

Managing Elastic Beanstalk instance profiles

An instance profile is a container for an AWS Identity and Access Management (IAM) role that you can use to pass role information to an Amazon EC2 instance when the instance starts. When you launch an environment using the Elastic Beanstalk console or the EB CLI, Elastic Beanstalk creates a default instance profile, called `aws-elasticbeanstalk-ec2-role`, and assigns managed policies with default permissions to it.

Elastic Beanstalk provides three managed policies: one for the web server tier, one for the worker tier, and one with additional permissions required for multicontainer Docker environments. The console assigns all of these policies to the role attached to the default instance profile. The policies follow.

Managed instance profile policies

- **AWSElasticBeanstalkWebTier** – Grants permissions for the application to upload logs to Amazon S3 and debugging information to AWS X-Ray.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "BucketAccess",
      "Action": [
        "s3:Get*",
        "s3:List*",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::elasticbeanstalk-*",
        "arn:aws:s3:::elasticbeanstalk-*/*"
      ]
    },
    {
      "Sid": "XRayAccess",
      "Action": [
        "xray:PutTraceSegments",
        "xray:PutTelemetryRecords"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchLogsAccess",
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogStream",
        "logs:DescribeLogStreams",
        "logs:DescribeLogGroups"
      ],
      "Effect": "Allow",
      "Resource": [
```

```

        "arn:aws:logs:*:*:log-group:/aws/elasticbeanstalk*"
    ]
}

```

- **AWSElasticBeanstalkWorkerTier** – Grants permissions for log uploads, debugging, metric publication, and worker instance tasks, including queue management, leader election, and periodic tasks.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "MetricsAccess",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "XRayAccess",
      "Action": [
        "xray:PutTraceSegments",
        "xray:PutTelemetryRecords"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "QueueAccess",
      "Action": [
        "sqs:ChangeMessageVisibility",
        "sqs:DeleteMessage",
        "sqs:ReceiveMessage",
        "sqs:SendMessage"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "BucketAccess",
      "Action": [
        "s3:Get*",
        "s3:List*",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::elasticbeanstalk-*",
        "arn:aws:s3:::elasticbeanstalk-*/*"
      ]
    },
    {
      "Sid": "DynamoPeriodicTasks",
      "Action": [
        "dynamodb:BatchGetItem",
        "dynamodb:BatchWriteItem",
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem"
      ],
    }
  ],
}

```



```
    "Effect": "Allow",
    "Resource": [
      "arn:aws:dynamodb:*:*:table/*-stack-AWSEBWorkerCronLeaderRegistry*"
    ]
  },
  {
    "Sid": "CloudWatchLogsAccess",
    "Action": [
      "logs:PutLogEvents",
      "logs:CreateLogStream"
    ],
    "Effect": "Allow",
    "Resource": [
      "arn:aws:logs:*:*:log-group:/aws/elasticbeanstalk*"
    ]
  }
]
}
```

- **AWSElasticBeanstalkMulticontainerDocker** – Grants permissions for the Amazon Elastic Container Service to coordinate cluster tasks.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ECSAccess",
      "Effect": "Allow",
      "Action": [
        "ecs:Poll",
        "ecs:StartTask",
        "ecs:StopTask",
        "ecs:DiscoverPollEndpoint",
        "ecs:StartTelemetrySession",
        "ecs:RegisterContainerInstance",
        "ecs:DeregisterContainerInstance",
        "ecs:DescribeContainerInstances",
        "ecs:Submit*"
      ],
      "Resource": "*"
    }
  ]
}
```

To allow the EC2 instances in your environment to assume the `aws-elasticbeanstalk-ec2-role` role, the instance profile specifies Amazon EC2 as a trusted entity in the trust relationship policy, as follows.

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

To customize permissions, you can add policies to the role attached to the default instance profile or create your own instance profile with a restricted set of permissions.

Sections

- [Verifying the permissions assigned to the default instance profile \(p. 763\)](#)
- [Updating an out-of-date default instance profile \(p. 763\)](#)
- [Adding permissions to the default instance profile \(p. 763\)](#)
- [Creating an instance profile \(p. 764\)](#)
- [Instance profiles with Amazon Linux 2 platforms \(p. 764\)](#)

Verifying the permissions assigned to the default instance profile

The permissions assigned to your default instance profile can vary depending on when it was created, the last time you launched an environment, and which client you used. You can verify the permissions on the default instance profile in the IAM console.

To verify the default instance profile's permissions

1. Open the [Roles page](#) in the IAM console.
2. Choose **aws-elasticbeanstalk-ec2-role**.
3. On the **Permissions** tab, review the list of policies attached to the role.
4. To see the permissions that a policy grants, choose the policy.

Updating an out-of-date default instance profile

If the default instance profile lacks the required permissions, you can update it by [creating a new environment \(p. 363\)](#) in the Elastic Beanstalk environment management console.

Alternatively, you can add the managed policies to the role attached to the default instance profile manually.

To add managed policies to the role attached to the default instance profile

1. Open the [Roles page](#) in the IAM console.
2. Choose **aws-elasticbeanstalk-ec2-role**.
3. On the **Permissions** tab, choose **Attach policies**.
4. Type **AWSElasticBeanstalk** to filter the policies.
5. Select the following policies, and then choose **Attach policy**:
 - **AWSElasticBeanstalkWebTier**
 - **AWSElasticBeanstalkWorkerTier**
 - **AWSElasticBeanstalkMulticontainerDocker**

Adding permissions to the default instance profile

If your application accesses AWS APIs or resources to which permissions aren't granted in the default instance profile, add policies that grant permissions in the IAM console.

To add policies to the role attached to the default instance profile

1. Open the [Roles page](#) in the IAM console.

2. Choose **aws-elasticbeanstalk-ec2-role**.
3. On the **Permissions** tab, choose **Attach policies**.
4. Select the managed policy for the additional services that your application uses. For example, `AmazonS3FullAccess` or `AmazonDynamoDBFullAccess`.
5. Choose **Attach policy**.

Creating an instance profile

An instance profile is a wrapper around a standard IAM role that allows an EC2 instance to assume the role. You can create additional instance profiles to customize permissions for different applications or to create an instance profile that doesn't grant permissions for worker tier or multicontainer Docker environments, if you don't use those features.

To create an instance profile

1. Open the [Roles page](#) in the IAM console.
2. Choose **Create role**.
3. Under **AWS service**, choose **EC2**.
4. Choose **Next: Permissions**.
5. Attach the appropriate managed policies provided by Elastic Beanstalk and any additional policies that provide permissions that your application needs.
6. Choose **Next: Tags**.
7. (Optional) Add tags to the role.
8. Choose **Next: Review**.
9. Enter a name for the role.
10. Choose **Create role**.

Instance profiles with Amazon Linux 2 platforms

Amazon Linux 2 platforms require an instance profile for proper operation. For example, all Amazon Linux 2 platform versions enable enhanced health by default during environment creation. Instances need the right permissions to collect and report enhanced health information.

Managing Elastic Beanstalk service roles

To manage and monitor your environment, AWS Elastic Beanstalk performs actions on the environment's resources on your behalf. Elastic Beanstalk needs certain permissions to perform these actions, and it assumes AWS Identity and Access Management (IAM) service roles to get these permissions.

Elastic Beanstalk needs to use temporary security credentials whenever it assumes a service role. To get these credentials, Elastic Beanstalk sends a request to AWS Security Token Service (AWS STS) on a global endpoint. For more information, see [Temporary Security Credentials](#) in the *IAM User Guide*.

When you launch an environment in the Elastic Beanstalk console, the console creates a default service role, named `aws-elasticbeanstalk-service-role`, and attaches managed policies with default permissions to it.

Elastic Beanstalk provides a managed policy for [enhanced health monitoring \(p. 691\)](#), and one with additional permissions required for [managed platform updates \(p. 420\)](#). The console assigns both of these policies to the default service role. The managed service role policies follow.

Managed service role policies

- **AWSElasticBeanstalkEnhancedHealth** – Grants permissions for Elastic Beanstalk to monitor instance and environment health.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:DescribeInstanceHealth",
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeTargetHealth",
        "ec2:DescribeInstances",
        "ec2:DescribeInstanceStatus",
        "ec2:GetConsoleOutput",
        "ec2:AssociateAddress",
        "ec2:DescribeAddresses",
        "ec2:DescribeSecurityGroups",
        "sqs:GetQueueAttributes",
        "sqs:GetQueueUrl",
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeAutoScalingInstances",
        "autoscaling:DescribeScalingActivities",
        "autoscaling:DescribeNotificationConfigurations",
        "sns:Publish"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

- **AWSElasticBeanstalkService** – Grants permissions for Elastic Beanstalk to update environments on your behalf to perform managed platform updates.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCloudformationOperationsOnElasticBeanstalkStacks",
      "Effect": "Allow",
      "Action": [
        "cloudformation:*"
      ],
      "Resource": [
        "arn:aws:cloudformation:*:*:stack/awseb-*",
        "arn:aws:cloudformation:*:*:stack/eb-*"
      ]
    },
    {
      "Sid": "AllowS3OperationsOnElasticBeanstalkBuckets",
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": [
        "arn:aws:s3:::elasticbeanstalk-*",
        "arn:aws:s3:::elasticbeanstalk-*/*"
      ]
    }
  ]
}
```

```

"Sid": "AllowOperations",
"Effect": "Allow",
"Action": [
    "autoscaling:AttachInstances",
    "autoscaling:CreateAutoScalingGroup",
    "autoscaling:CreateLaunchConfiguration",
    "autoscaling>DeleteLaunchConfiguration",
    "autoscaling>DeleteAutoScalingGroup",
    "autoscaling>DeletePolicy",
    "autoscaling>DeleteScheduledAction",
    "autoscaling:DescribeAccountLimits",
    "autoscaling:DescribeAutoScalingGroups",
    "autoscaling:DescribeAutoScalingInstances",
    "autoscaling:DescribeLaunchConfigurations",
    "autoscaling:DescribeLoadBalancers",
    "autoscaling:DescribeNotificationConfigurations",
    "autoscaling:DescribeScalingActivities",
    "autoscaling:DescribeScheduledActions",
    "autoscaling:DetachInstances",
    "autoscaling:PutNotificationConfiguration",
    "autoscaling:PutScalingPolicy",
    "autoscaling:PutScheduledUpdateGroupAction",
    "autoscaling:ResumeProcesses",
    "autoscaling:SetDesiredCapacity",
    "autoscaling:SuspendProcesses",
    "autoscaling:TerminateInstanceInAutoScalingGroup",
    "autoscaling:UpdateAutoScalingGroup",
    "cloudwatch:PutMetricAlarm",
    "ec2:AuthorizeSecurityGroupEgress",
    "ec2:AuthorizeSecurityGroupIngress",
    "ec2:CreateLaunchTemplate",
    "ec2:CreateLaunchTemplateVersion",
    "ec2:CreateSecurityGroup",
    "ec2>DeleteLaunchTemplate",
    "ec2>DeleteLaunchTemplateVersions",
    "ec2>DeleteSecurityGroup",
    "ec2:DescribeAccountAttributes",
    "ec2:DescribeImages",
    "ec2:DescribeInstances",
    "ec2:DescribeKeyPairs",
    "ec2:DescribeLaunchTemplates",
    "ec2:DescribeLaunchTemplateVersions",
    "ec2:DescribeSecurityGroups",
    "ec2:DescribeSubnets",
    "ec2:DescribeVpcs",
    "ec2:RevokeSecurityGroupEgress",
    "ec2:RevokeSecurityGroupIngress",
    "ec2:RunInstances",
    "ec2:TerminateInstances",
    "ecs:CreateCluster",
    "ecs>DeleteCluster",
    "ecs:DescribeClusters",
    "ecs:RegisterTaskDefinition",
    "elasticbeanstalk:*",
    "elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",
    "elasticloadbalancing:ConfigureHealthCheck",
    "elasticloadbalancing:CreateLoadBalancer",
    "elasticloadbalancing>DeleteLoadBalancer",
    "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
    "elasticloadbalancing:DescribeInstanceHealth",
    "elasticloadbalancing:DescribeLoadBalancers",
    "elasticloadbalancing:DescribeTargetHealth",
    "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
    "iam:ListRoles",
    "iam:PassRole",
    "logs:CreateLogGroup",

```

```

        "logs:PutRetentionPolicy",
        "rds:DescribeDBInstances",
        "rds:DescribeOrderableDBInstanceOptions",
        "s3:CopyObject",
        "s3:GetObject",
        "s3:GetObjectAcl",
        "s3:GetObjectMetadata",
        "s3:ListBucket",
        "s3:listBuckets",
        "sns:CreateTopic",
        "sns:GetTopicAttributes",
        "sns:ListSubscriptionsByTopic",
        "sns:Subscribe",
        "sqs:GetQueueAttributes",
        "sqs:GetQueueUrl"
    ],
    "Resource": [
        "*"
    ]
}
]
}

```

To allow Elastic Beanstalk to assume the `aws-elasticbeanstalk-service-role` role, the service role specifies Elastic Beanstalk as a trusted entity in the trust relationship policy.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "elasticbeanstalk.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "elasticbeanstalk"
        }
      }
    }
  ]
}

```

When you launch an environment using the [the section called “eb create” \(p. 894\)](#) command of the Elastic Beanstalk Command Line Interface (EB CLI) and don't specify a service role through the `--service-role` option, Elastic Beanstalk creates the default service role `aws-elasticbeanstalk-service-role`. If the default service role already exists, Elastic Beanstalk uses it for the new environment.

If you use the `CreateEnvironment` action of the Elastic Beanstalk API to create an environment, specify a service role with the `ServiceRole` configuration option in the `aws:elasticbeanstalk:environment` namespace. See [Using enhanced health reporting with the Elastic Beanstalk API \(p. 719\)](#) for details on using enhanced health monitoring with the Elastic Beanstalk API.

When you create an environment by using the Elastic Beanstalk API, and don't specify a service role, Elastic Beanstalk creates a monitoring service-linked role for your account, if it doesn't already exist, and uses it for the new environment. A service-linked role is a unique type of service role that is predefined by Elastic Beanstalk to include all the permissions that the service requires to call other AWS services

on your behalf. The service-linked role is associated with your account. Elastic Beanstalk creates it once, then reuses it when creating additional environments. You can also use IAM to create your account's monitoring service-linked role in advance. When your account has a monitoring service-linked role, you can use it to create an environment by using the Elastic Beanstalk API, the Elastic Beanstalk console, or the EB CLI. For details about using service-linked roles with Elastic Beanstalk environments, see [Using service-linked roles for Elastic Beanstalk \(p. 769\)](#).

Note

When Elastic Beanstalk tries to create the monitoring service-linked role for your account when you create an environment, you must have the `iam:CreateServiceLinkedRole` permission. If you don't have this permission, environment creation fails, and you see a message explaining the issue.

As an alternative, another user with permission to create service-linked roles can use IAM to create the service linked-role in advance. You can then create your environment even without having the `iam:CreateServiceLinkedRole` permission.

Verifying the default service role's permissions

The permissions granted by your default service role can vary depending on when it was created, the last time you launched an environment, and which client you used. You can verify the permissions granted by the default service role in the IAM console.

To verify the default service role's permissions

1. Open the [Roles page](#) in the IAM console.
2. Choose **aws-elasticbeanstalk-service-role**.
3. On the **Permissions** tab, review the list of policies attached to the role.
4. To view the permissions that a policy grants, choose the policy.

Updating an out-of-date default service role

If the default service role lacks the required permissions, you can update it by [creating a new environment \(p. 363\)](#) in the Elastic Beanstalk environment management console.

Alternatively, you can add the managed policies to the default service role manually.

To add managed policies to the default service role

1. Open the [Roles page](#) in the IAM console.
2. Choose **aws-elasticbeanstalk-service-role**.
3. On the **Permissions** tab, choose **Attach policies**.
4. Type **AWSElasticBeanstalk** to filter the policies.
5. Select the following policies, and then choose **Attach policy**:
 - **AWSElasticBeanstalkEnhancedHealth**
 - **AWSElasticBeanstalkService**

Adding permissions to the default service role

If your application includes configuration files that refer to AWS resources for which permissions aren't included in the default service role, Elastic Beanstalk might need additional permissions to resolve these references when it processes the configuration files during a managed update. If permissions are missing,

the update fails and Elastic Beanstalk returns a message indicating which permission it needs. Add permissions for additional services to the default service role in the IAM console.

To add additional policies to the default service role

1. Open the [Roles page](#) in the IAM console.
2. Choose `aws-elasticbeanstalk-service-role`.
3. On the **Permissions** tab, choose **Attach policies**.
4. Select the managed policy for the additional services that your application uses. For example, `AmazonAPIGatewayAdministrator` or `AmazonElasticFileSystemFullAccess`.
5. Choose **Attach policy**.

Creating a service role

If you can't use the default service role, create a service role.

To create a service role

1. Open the [Roles page](#) in the IAM console.
2. Choose **Create role**.
3. Under **AWS service**, choose **AWS Elastic Beanstalk**, and then select your use case.
4. Choose **Next: Permissions**.
5. Attach the `AWSElasticBeanstalkService` and `AWSElasticBeanstalkEnhancedHealth` managed policies and any additional policies that provide permissions that your application needs.
6. Choose **Next: Tags**.
7. (Optional) Add tags to the role.
8. Choose **Next: Review**.
9. Enter a name for the role.
10. Choose **Create role**.

You can apply your custom service role when you create an environment in the [environment creation wizard \(p. 365\)](#) or with the `--service-role` option on the `eb create (p. 894)` command.

Using service-linked roles for Elastic Beanstalk

AWS Elastic Beanstalk uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to Elastic Beanstalk. Service-linked roles are predefined by Elastic Beanstalk and include all the permissions that the service requires to call other AWS services on your behalf.

Elastic Beanstalk defines two types of service-linked roles:

- *Monitoring service-linked role* – Allows Elastic Beanstalk to monitor the health of running environments and publish health event notifications.
- *Maintenance service-linked role* – Allows Elastic Beanstalk to perform regular maintenance activities for your running environments.

Topics

- [The Monitoring Service-Linked Role \(p. 770\)](#)
- [The Maintenance Service-Linked Role \(p. 772\)](#)

The Monitoring Service-Linked Role

AWS Elastic Beanstalk uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to Elastic Beanstalk. Service-linked roles are predefined by Elastic Beanstalk and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Elastic Beanstalk easier because you don't have to manually add the necessary permissions. Elastic Beanstalk defines the permissions of its service-linked roles, and unless defined otherwise, only Elastic Beanstalk can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Elastic Beanstalk resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see [AWS Services That Work with IAM](#) and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Elastic Beanstalk

Elastic Beanstalk uses the service-linked role named **AWSServiceRoleForElasticBeanstalk** – Allows Elastic Beanstalk to monitor the health of running environments and publish health event notifications.

The **AWSServiceRoleForElasticBeanstalk** service-linked role trusts the following services to assume the role:

- `elasticbeanstalk.amazonaws.com`

The permissions policy of the **AWSServiceRoleForElasticBeanstalk** service-linked role contains all of the permissions that Elastic Beanstalk needs to complete actions on your behalf:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCloudformationReadOperationsOnElasticBeanstalkStacks",
      "Effect": "Allow",
      "Action": [
        "cloudformation:DescribeStackResource",
        "cloudformation:DescribeStackResources",
        "cloudformation:DescribeStacks"
      ],
      "Resource": [
        "arn:aws:cloudformation:*:*:stack/awseb-*",
        "arn:aws:cloudformation:*:*:stack/eb-*"
      ]
    },
    {
      "Sid": "AllowOperations",
      "Effect": "Allow",
      "Action": [
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeAutoScalingInstances",
        "autoscaling:DescribeNotificationConfigurations",
        "autoscaling:DescribeScalingActivities",
        "autoscaling:PutNotificationConfiguration",
        "ec2:DescribeInstanceStatus",
        "ec2:AssociateAddress"
      ]
    }
  ]
}
```

```
        "ec2:DescribeAddresses",
        "ec2:DescribeInstances",
        "ec2:DescribeSecurityGroups",
        "elasticloadbalancing:DescribeInstanceHealth",
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeTargetHealth",
        "elasticloadbalancing:DescribeTargetGroups",
        "sqs:GetQueueAttributes",
        "sqs:GetQueueUrl",
        "sns:Publish"
    ],
    "Resource": [
        "*"
    ]
}
]
```

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see [Service-Linked Role Permissions](#) in the *IAM User Guide*.

Alternatively, you can use an AWS managed policy to [provide full access \(p. 775\)](#) to Elastic Beanstalk.

Creating a service-linked role for Elastic Beanstalk

You don't need to manually create a service-linked role. When you create an Elastic Beanstalk environment using the Elastic Beanstalk API and don't specify a service role, Elastic Beanstalk creates the service-linked role for you.

Important

If you were using the Elastic Beanstalk service before September 27, 2017, when it began supporting the `AWSServiceRoleForElasticBeanstalk` service-linked role, and your account needed it, then Elastic Beanstalk created the `AWSServiceRoleForElasticBeanstalk` role in your account. To learn more, see [A New Role Appeared in My IAM Account](#).

When Elastic Beanstalk tries to create the `AWSServiceRoleForElasticBeanstalk` service-linked role for your account when you create an environment, you must have the `iam:CreateServiceLinkedRole` permission. If you don't have this permission, environment creation fails, and you see a message explaining the issue.

As an alternative, another user with permission to create service-linked roles can use IAM to pre-create the service linked-role in advance. You can then create your environment even without having the `iam:CreateServiceLinkedRole` permission.

You (or another user) can use the IAM console to create a service-linked role with the **Elastic Beanstalk** use case. In the IAM CLI or the IAM API, create a service-linked role with the `elasticbeanstalk.amazonaws.com` service name. For more information, see [Creating a Service-Linked Role](#) in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create an Elastic Beanstalk environment using the Elastic Beanstalk API and don't specify a service role, Elastic Beanstalk creates the service-linked role for you again.

Editing a service-linked role for Elastic Beanstalk

Elastic Beanstalk does not allow you to edit the `AWSServiceRoleForElasticBeanstalk` service-linked role. After you create a service-linked role, you cannot change the name of the role because various

entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a Service-Linked Role](#) in the *IAM User Guide*.

Deleting a service-linked role for Elastic Beanstalk

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first terminate any Elastic Beanstalk environments that uses the role.

Note

If the Elastic Beanstalk service is using the role when you try to terminate the environments, then the termination might fail. If that happens, wait for a few minutes and try the operation again.

To terminate an Elastic Beanstalk environment that uses the `AWSServiceRoleForElasticBeanstalk` (console)

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

See [eb terminate \(p. 933\)](#) for details about terminating an Elastic Beanstalk environment using the EB CLI.

See [TerminateEnvironment](#) for details about terminating an Elastic Beanstalk environment using the API.

Manually delete the service-linked role

Use the IAM console, the IAM CLI, or the IAM API to delete the `AWSServiceRoleForElasticBeanstalk` service-linked role. For more information, see [Deleting a Service-Linked Role](#) in the *IAM User Guide*.

Supported regions for Elastic Beanstalk service-linked roles

Elastic Beanstalk supports using service-linked roles in all of the regions where the service is available. For more information, see [AWS Elastic Beanstalk Endpoints and Quotas](#).

The Maintenance Service-Linked Role

AWS Elastic Beanstalk uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to Elastic Beanstalk. Service-linked roles are predefined by Elastic Beanstalk and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Elastic Beanstalk easier because you don't have to manually add the necessary permissions. Elastic Beanstalk defines the permissions of its service-linked roles, and unless defined otherwise, only Elastic Beanstalk can assume its roles. The defined permissions include

the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Elastic Beanstalk resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see [AWS Services That Work with IAM](#) and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Elastic Beanstalk

Elastic Beanstalk uses the service-linked role named **AWSServiceRoleForElasticBeanstalkMaintenance** – Allows Elastic Beanstalk to perform regular maintenance activities for your running environments.

The AWSServiceRoleForElasticBeanstalkMaintenance service-linked role trusts the following services to assume the role:

- `maintenance.elasticbeanstalk.amazonaws.com`

The permissions policy of the AWSServiceRoleForElasticBeanstalkMaintenance service-linked role contains all of the permissions that Elastic Beanstalk needs to complete actions on your behalf:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCloudformationChangeSetOperationsOnElasticBeanstalkStacks",
      "Effect": "Allow",
      "Action": [
        "cloudformation:CreateChangeSet",
        "cloudformation:DescribeChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation>DeleteChangeSet",
        "cloudformation:ListChangeSets",
        "cloudformation:DescribeStacks"
      ],
      "Resource": [
        "arn:aws:cloudformation:*:*:stack/awseb-*",
        "arn:aws:cloudformation:*:*:stack/eb-*"
      ]
    }
  ]
}
```

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see [Service-Linked Role Permissions](#) in the *IAM User Guide*.

Alternatively, you can use an AWS managed policy to [provide full access \(p. 775\)](#) to Elastic Beanstalk.

Creating a service-linked role for Elastic Beanstalk

You don't need to manually create a service-linked role. When you create an Elastic Beanstalk environment using the Elastic Beanstalk API and don't specify an instance profile, Elastic Beanstalk creates the service-linked role for you.

Important

This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role. If you were using the Elastic Beanstalk service before April 18, 2019, when it began supporting the

AWSServiceRoleForElasticBeanstalkMaintenance service-linked role, and your account needed it, then Elastic Beanstalk created the AWSServiceRoleForElasticBeanstalkMaintenance role in your account. To learn more, see [A New Role Appeared in My IAM Account](#).

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create an Elastic Beanstalk environment using the Elastic Beanstalk API and don't specify an instance profile, Elastic Beanstalk creates the service-linked role for you again.

Editing a service-linked role for Elastic Beanstalk

Elastic Beanstalk does not allow you to edit the AWSServiceRoleForElasticBeanstalkMaintenance service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a Service-Linked Role](#) in the *IAM User Guide*.

Deleting a service-linked role for Elastic Beanstalk

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first terminate any Elastic Beanstalk environments that uses the role.

Note

If the Elastic Beanstalk service is using the role when you try to terminate the environments, then the termination might fail. If that happens, wait for a few minutes and try the operation again.

To terminate an Elastic Beanstalk environment that uses the AWSServiceRoleForElasticBeanstalkMaintenance (console)

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

See [eb terminate \(p. 933\)](#) for details about terminating an Elastic Beanstalk environment using the EB CLI.

See [TerminateEnvironment](#) for details about terminating an Elastic Beanstalk environment using the API.

Manually delete the service-linked role

Use the IAM console, the IAM CLI, or the IAM API to delete the AWSServiceRoleForElasticBeanstalkMaintenance service-linked role. For more information, see [Deleting a Service-Linked Role](#) in the *IAM User Guide*.

Supported regions for Elastic Beanstalk service-linked roles

Elastic Beanstalk supports using service-linked roles in all of the regions where the service is available. For more information, see [AWS Elastic Beanstalk Endpoints and Quotas](#).

Managing Elastic Beanstalk user policies

AWS Elastic Beanstalk provides two managed policies that enable you to assign full access or read-only access to all Elastic Beanstalk resources. You can attach the policies to AWS Identity and Access Management (IAM) users or groups.

Managed user policies

- **AWSElasticBeanstalkFullAccess** – Allows the user to create, modify, and delete Elastic Beanstalk applications, application versions, configuration settings, environments, and their underlying resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticbeanstalk:*",
        "ec2:*",
        "ecs:*",
        "ecr:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
        "dynamodb:*",
        "rds:*",
        "sqs:*",
        "logs:*",
        "iam:GetPolicyVersion",
        "iam:GetRole",
        "iam:PassRole",
        "iam:ListRolePolicies",
        "iam:ListAttachedRolePolicies",
        "iam:ListInstanceProfiles",
        "iam:ListRoles",
        "iam:ListServerCertificates",
        "acm:DescribeCertificate",
        "acm:ListCertificates",
        "codebuild:CreateProject",
        "codebuild>DeleteProject",
        "codebuild:BatchGetBuilds",
        "codebuild:StartBuild"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:AddRoleToInstanceProfile",
        "iam:CreateInstanceProfile",
        "iam:CreateRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/aws-elasticbeanstalk*",
        "arn:aws:iam::*:instance-profile/aws-elasticbeanstalk*"
      ]
    }
  ],
  {
    "Effect": "Allow",
```

```

    "Action": [
      "iam:CreateServiceLinkedRole"
    ],
    "Resource": [
      "arn:aws:iam::*:role/aws-service-role/autoscaling.amazonaws.com/
AWSServiceRoleForAutoScaling*"
    ],
    "Condition": {
      "StringLike": {
        "iam:AWSServiceName": "autoscaling.amazonaws.com"
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "iam:CreateServiceLinkedRole"
    ],
    "Resource": [
      "arn:aws:iam::*:role/aws-service-role/elasticbeanstalk.amazonaws.com/
AWSServiceRoleForElasticBeanstalk*"
    ],
    "Condition": {
      "StringLike": {
        "iam:AWSServiceName": "elasticbeanstalk.amazonaws.com"
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "iam:AttachRolePolicy"
    ],
    "Resource": "*",
    "Condition": {
      "StringLike": {
        "iam:PolicyArn": [
          "arn:aws:iam::aws:policy/AWSElasticBeanstalk*",
          "arn:aws:iam::aws:policy/service-role/AWSElasticBeanstalk*"
        ]
      }
    }
  }
]
}

```

- **AWSElasticBeanstalkReadOnlyAccess** – Allows the user to view applications and environments, but not to perform operations on them. It provides read-only access to all Elastic Beanstalk resources. Note that read-only access does not enable actions such as downloading Elastic Beanstalk logs so that you can read them. This is because the logs are staged in the Amazon S3 bucket, where Elastic Beanstalk would require write permission. See the example at the end of this topic for information on how to enable access to Elastic Beanstalk logs.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticbeanstalk:Check*",
        "elasticbeanstalk:Describe*",
        "elasticbeanstalk:List*",
        "elasticbeanstalk:RequestEnvironmentInfo",
        "elasticbeanstalk:RetrieveEnvironmentInfo",

```

```
    "ec2:Describe*",
    "elasticloadbalancing:Describe*",
    "autoscaling:Describe*",
    "cloudwatch:Describe*",
    "cloudwatch:List*",
    "cloudwatch:Get*",
    "s3:Get*",
    "s3:List*",
    "sns:Get*",
    "sns:List*",
    "cloudformation:Describe*",
    "cloudformation:Get*",
    "cloudformation:List*",
    "cloudformation:Validate*",
    "cloudformation:Estimate*",
    "rds:Describe*",
    "sqs:Get*",
    "sqs:List*"
  ],
  "Resource": "*"
}
]
```

Controlling access with managed policies

You can use managed policies to grant full access or read-only access to Elastic Beanstalk. Elastic Beanstalk updates these policies automatically when additional permissions are required to access new features.

To apply a managed policy to IAM users or groups

1. Open the [Policies](#) page in the IAM console.
2. In the search box, type **AWSElasticBeanstalk** to filter the policies.
3. In the list of policies, select the check box next to **AWSElasticBeanstalkReadOnlyAccess** or **AWSElasticBeanstalkFullAccess**.
4. Choose **Policy actions**, and then choose **Attach**.
5. Select one or more users and groups to attach the policy to. You can use the **Filter** menu and the search box to filter the list of principal entities.
6. Choose **Attach policy**.

Creating a custom user policy

You can create your own IAM policy to allow or deny specific Elastic Beanstalk API actions on specific Elastic Beanstalk resources. For more information about attaching a policy to a user or group, see [Working with Policies](#) in *Using AWS Identity and Access Management*.

Note

While you can restrict how a user interacts with Elastic Beanstalk APIs, there is not currently an effective way to prevent users who have permission to create the necessary underlying resources from creating other resources in Amazon EC2 and other services.

Think of these policies as an effective way to distribute Elastic Beanstalk responsibilities, not as a way to secure all underlying resources.

In November 2019, Elastic Beanstalk released support for [Amazon EC2 launch templates](#). This is a new resource type that your environment's Auto Scaling group can use to launch Amazon EC2 instances, and it requires new permissions. Most customers shouldn't be affected, because environments can still use

the legacy resource, launch configurations, if your user policy lacks the required permissions. However, if you're trying to use a new feature that requires Amazon EC2 launch templates, and you have a custom policy, your environment creation or update might fail. In this case, ensure that your custom policy has the following permissions.

Required permissions for Amazon EC2 launch templates

- `EC2:CreateLaunchTemplate`
- `EC2:CreateLaunchTemplateVersions`
- `EC2>DeleteLaunchTemplate`
- `EC2>DeleteLaunchTemplateVersions`
- `EC2:DescribeLaunchTemplate`
- `EC2:DescribeLaunchTemplateVersions`

An IAM policy contains policy statements that describe the permissions that you want to grant. When you create a policy statement for Elastic Beanstalk, you need to understand how to use the following four parts of a policy statement:

- **Effect** specifies whether to allow or deny the actions in the statement.
- **Action** specifies the [API operations](#) that you want to control. For example, use `elasticbeanstalk:CreateEnvironment` to specify the `CreateEnvironment` operation. Certain operations, such as creating an environment, require additional permissions to perform those actions. For more information, see [Resources and conditions for Elastic Beanstalk actions \(p. 782\)](#).

Note

To use the [UpdateTagsForResource](#) API operation, specify one of the following two virtual actions (or both) instead of the API operation name:

`elasticbeanstalk:AddTags`

Controls permission to call `UpdateTagsForResource` and pass a list of tags to add in the `TagsToAdd` parameter.

`elasticbeanstalk:RemoveTags`

Controls permission to call `UpdateTagsForResource` and pass a list of tag keys to remove in the `TagsToRemove` parameter.

- **Resource** specifies the resources that you want to control access to. To specify Elastic Beanstalk resources, list the [Amazon Resource Name \(p. 781\)](#) (ARN) of each resource.
- (optional) **Condition** specifies restrictions on the permission granted in the statement. For more information, see [Resources and conditions for Elastic Beanstalk actions \(p. 782\)](#).

The following sections demonstrate a few cases in which you might consider a custom user policy.

Enabling limited Elastic Beanstalk environment creation

The policy in the following example enables a user to call the `CreateEnvironment` action to create an environment whose name begins with `Test` with the specified application and application version.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CreateEnvironmentPerm",
      "Action": [
        "elasticbeanstalk:CreateEnvironment"
      ],
      "Effect": "Allow",
```

```

    "Resource": [
      "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My First Elastic
      Beanstalk Application/Test*"
    ],
    "Condition": {
      "StringEquals": {
        "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
        east-2:123456789012:application/My First Elastic Beanstalk Application"],
        "elasticbeanstalk:FromApplicationVersion": ["arn:aws:elasticbeanstalk:us-
        east-2:123456789012:applicationversion/My First Elastic Beanstalk Application/First
        Release"]
      }
    }
  },
  {
    "Sid": "AllNonResourceCalls",
    "Action": [
      "elasticbeanstalk:CheckDNSAvailability",
      "elasticbeanstalk:CreateStorageLocation"
    ],
    "Effect": "Allow",
    "Resource": [
      "*"
    ]
  }
]
}

```

The above policy shows how to grant limited access to Elastic Beanstalk operations. In order to actually launch an environment, the user must have permission to create the AWS resources that power the environment as well. For example, the following policy grants access to the default set of resources for a web server environment:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:*",
        "ecs:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
        "sqs:*"
      ],
      "Resource": "*"
    }
  ]
}

```

Enabling access to Elastic Beanstalk logs stored in Amazon S3

The policy in the following example enables a user to pull Elastic Beanstalk logs, stage them in Amazon S3, and retrieve them.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {

```

```

    "Action": [
      "s3:DeleteObject",
      "s3:GetObjectAcl",
      "s3:PutObjectAcl"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:s3:::elasticbeanstalk-*"
  }
]
}

```

Note

To restrict these permissions to only the logs path, use the following resource format.

```

"arn:aws:s3:::elasticbeanstalk-us-east-2-123456789012/resources/environments/logs/*"

```

Enabling management of a specific Elastic Beanstalk application

The policy in the following example enables a user to manage environments and other resources within one specific Elastic Beanstalk application. The policy denies Elastic Beanstalk actions on resources of other applications, and also denies creation and deletion of Elastic Beanstalk applications.

Note

The policy doesn't deny access to any resources through other services. It demonstrates an effective way to distribute responsibilities for managing Elastic Beanstalk applications among different users, not as a way to secure the underlying resources.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "elasticbeanstalk:CreateApplication",
        "elasticbeanstalk>DeleteApplication"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "elasticbeanstalk:CreateApplicationVersion",
        "elasticbeanstalk:CreateConfigurationTemplate",
        "elasticbeanstalk:CreateEnvironment",
        "elasticbeanstalk>DeleteApplicationVersion",
        "elasticbeanstalk>DeleteConfigurationTemplate",
        "elasticbeanstalk>DeleteEnvironmentConfiguration",
        "elasticbeanstalk:DescribeApplicationVersions",
        "elasticbeanstalk:DescribeConfigurationOptions",
        "elasticbeanstalk:DescribeConfigurationSettings",
        "elasticbeanstalk:DescribeEnvironmentResources",
        "elasticbeanstalk:DescribeEnvironments",
        "elasticbeanstalk:DescribeEvents",
        "elasticbeanstalk>DeleteEnvironmentConfiguration",
        "elasticbeanstalk:RebuildEnvironment",
        "elasticbeanstalk:RequestEnvironmentInfo",
        "elasticbeanstalk:RestartAppServer",
        "elasticbeanstalk:RetrieveEnvironmentInfo",
        "elasticbeanstalk:SwapEnvironmentCNAMEs",
        "elasticbeanstalk:TerminateEnvironment",

```

```

    "elasticbeanstalk:UpdateApplicationVersion",
    "elasticbeanstalk:UpdateConfigurationTemplate",
    "elasticbeanstalk:UpdateEnvironment",
    "elasticbeanstalk:RetrieveEnvironmentInfo",
    "elasticbeanstalk:ValidateConfigurationSettings"
  ],
  "Resource": [
    "*"
  ],
  "Condition": {
    "StringNotEquals": {
      "elasticbeanstalk:InApplication": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/myapplication"
      ]
    }
  }
}
]
}

```

Amazon resource name format for Elastic Beanstalk

You specify a resource for an IAM policy using that resource's Amazon Resource Name (ARN). For Elastic Beanstalk, the ARN has the following format.

```
arn:aws:elasticbeanstalk:region:account-id:resource-type/resource-path
```

Where:

- *region* is the region the resource resides in (for example, **us-west-2**).
- *account-id* is the AWS account ID, with no hyphens (for example, **123456789012**).
- *resource-type* identifies the type of the Elastic Beanstalk resource—for example, **environment**. See the table below for a list of all Elastic Beanstalk resource types.
- *resource-path* is the portion that identifies the specific resource. An Elastic Beanstalk resource has a path that uniquely identifies that resource. See the table below for the format of the resource path for each resource type. For example, an environment is always associated with an application. The resource path for the environment **myEnvironment** in the application **myApp** would look like this:

```
myApp/myEnvironment
```

Elastic Beanstalk has several types of resources you can specify in a policy. The following table shows the ARN format for each resource type and an example.

Resource type	Format for ARN
application	<pre>arn:aws:elasticbeanstalk:<i>region</i>:<i>account-id</i>:application/<i>application-name</i></pre> <p>Example: arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App</p>
applicationversion	<pre>arn:aws:elasticbeanstalk:<i>region</i>:<i>account-id</i>:applicationversion/<i>application-name</i>/<i>version-label</i></pre> <p>Example: arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version</p>

Resource type	Format for ARN
configurationtemplate	<p>arn:aws:elasticbeanstalk:<i>region</i>:<i>account-id</i>:configurationtemplate/<i>application-name</i>/<i>template-name</i></p> <p>Example: arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My App/My Template</p>
environment	<p>arn:aws:elasticbeanstalk:<i>region</i>:<i>account-id</i>:environment/<i>application-name</i>/<i>environment-name</i></p> <p>Example: arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/MyEnvironment</p>
platform	<p>arn:aws:elasticbeanstalk:<i>region</i>:<i>account-id</i>:platform/<i>platform-name</i>/<i>platform-version</i></p> <p>Example: arn:aws:elasticbeanstalk:us-east-2:123456789012:platform/MyPlatform/1.0</p>
solutionstack	<p>arn:aws:elasticbeanstalk:<i>region</i>::solutionstack/<i>solutionstack-name</i></p> <p>Example: arn:aws:elasticbeanstalk:us-east-2::solutionstack/32bit Amazon Linux running Tomcat 7</p>

An environment, application version, and configuration template are always contained within a specific application. You'll notice that these resources all have an application name in their resource path so that they are uniquely identified by their resource name and the containing application. Although solution stacks are used by configuration templates and environments, solution stacks are not specific to an application or AWS account and do not have the application or AWS account in their ARNs.

Resources and conditions for Elastic Beanstalk actions

This section describes the resources and conditions that you can use in policy statements to grant permissions that allow specific Elastic Beanstalk actions to be performed on specific Elastic Beanstalk resources.

Conditions enable you to specify permissions to resources that the action needs to complete. For example, when you can call the `CreateEnvironment` action, you must also specify the application version to deploy as well as the application that contains that application name. When you set permissions for the `CreateEnvironment` action, you specify the application and application version that you want the action to act upon by using the `InApplication` and `FromApplicationVersion` conditions.

In addition, you can specify the environment configuration with a solution stack (`FromSolutionStack`) or a configuration template (`FromConfigurationTemplate`). The following policy statement allows the `CreateEnvironment` action to create an environment with the name `myenv` (specified by `Resource`) in the application `My App` (specified by the `InApplication` condition) using the application version `My Version` (`FromApplicationVersion`) with a `32bit Amazon Linux running Tomcat 7` configuration (`FromSolutionStack`):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```

    "Action": [
      "elasticbeanstalk:CreateEnvironment"
    ],
    "Effect": "Allow",
    "Resource": [
      "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"
    ],
    "Condition": {
      "StringEquals": {
        "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"],
        "elasticbeanstalk:FromApplicationVersion": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"],
        "elasticbeanstalk:FromSolutionStack": ["arn:aws:elasticbeanstalk:us-east-2:solutionstack/32bit Amazon Linux running Tomcat 7"]
      }
    }
  }
}

```

Note

Most condition keys mentioned in this topic are specific to Elastic Beanstalk, and their names contain the `elasticbeanstalk:` prefix. For brevity, we omit this prefix from the condition key names when we mention them in the following sections. For example, we mention `InApplication` instead of its full name `elasticbeanstalk:InApplication`. In contrast, we mention a few condition keys used across AWS services, and we include their `aws:` prefix to highlight the exception. Policy examples always show full condition key names, including the prefix.

Sections

- [Policy information for Elastic Beanstalk actions \(p. 783\)](#)
- [Condition keys for Elastic Beanstalk actions \(p. 805\)](#)

Policy information for Elastic Beanstalk actions

The following table lists all Elastic Beanstalk actions, the resource that each action acts upon, and the additional contextual information that can be provided using conditions.

Policy information for Elastic Beanstalk actions, including resources, conditions, examples, and dependencies

Resource	Conditions	Example statement
Action: <code>AbortEnvironmentUpdate</code>		
application environment	aws:ResourceTag/ <i>key-name</i> (Optional) aws:TagKeys (Optional)	The following policy allows a user to abort environment update operations on environments in an application named <code>My App</code> . <pre> { "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:AbortEnvironmentUpdate"], "Effect": "Allow", "Resource": [</pre>

Resource	Conditions	Example statement
		<pre> "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] }] } </pre>
Action: CheckDNSAvailability		
"*"	N/A	<pre> { "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:CheckDNSAvailability"], "Effect": "Allow", "Resource": "*" }] } </pre>
Action: ComposeEnvironments		
application	<p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows a user to compose environments that belong to an application named My App.</p> <pre> { "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:ComposeEnvironments"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App"] }] } </pre>
Action: CreateApplication		

Resource	Conditions	Example statement
application	<p>aws:RequestTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>This example allows the <code>CreateApplication</code> action to create applications whose names begin with DivA:</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:CreateApplication"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/DivA*"] }] }</pre>
Action: CreateApplicationVersion		
applicationversion	<p>InApplication</p> <p>aws:RequestTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>This example allows the <code>CreateApplicationVersion</code> action to create application versions with any name (*) in the application My App:</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:CreateApplicationVersion"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/*"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }</pre>
Action: CreateConfigurationTemplate		

Resource	Conditions	Example statement
configurationtemplate	<p>InApplication</p> <p>FromApplication</p> <p>FromApplicationVersion</p> <p>FromConfigurationTemplate</p> <p>FromEnvironment</p> <p>FromSolutionStack</p> <p>aws:RequestTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the CreateConfigurationTemplate action to create configuration templates whose name begins with My Template (My Template*) in the application My App:</p> <pre> { "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:CreateConfigurationTemplate"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My App/My Template*"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"], "elasticbeanstalk:FromSolutionStack": ["arn:aws:elasticbeanstalk:us-east-2:solutionstack/32bit Amazon Linux running Tomcat 7"] } } }] } </pre>
<p>Action: CreateEnvironment</p>		

Resource	Conditions	Example statement
environment	<p>InApplication</p> <p>FromApplicationVersion</p> <p>FromConfigurationTemplate</p> <p>FromSolutionStack</p> <p>aws:RequestTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the CreateEnvironment action to create an environment whose name is myenv from the application My App and using the solution stack 32bit Amazon Linux running Tomcat 7:</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:CreateEnvironment"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"], "elasticbeanstalk:FromApplicationVersion": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"], "elasticbeanstalk:FromSolutionStack": ["arn:aws:elasticbeanstalk:us-east-2:solutionstack/32bit Amazon Linux running Tomcat 7"] } } }] }</pre>
<p>Action: CreatePlatformVersion</p>		
platform	<p>aws:RequestTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>This example allows the CreatePlatformVersion action to create platform versions targeting the us-east-2 region, whose names begin with us-east-2_:</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:CreatePlatformVersion"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:platform/us-east-2_*"] }] }</pre>

Resource	Conditions	Example statement
Action: CreateStorageLocation		
"*"	N/A	<pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:CreateStorageLocation"], "Effect": "Allow", "Resource": "*" }] }</pre>
Action: DeleteApplication		
application	aws:ResourceTag/ <i>key-name</i> (Optional) aws:TagKeys (Optional)	<p>The following policy allows the DeleteApplication action to delete the application My App:</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:DeleteApplication"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] }] }</pre>
Action: DeleteApplicationVersion		

Resource	Conditions	Example statement
applicationversion	<p>InApplication (Optional)</p> <p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the DeleteApplicationVersion action to delete an application version whose name is My Version in the application My App:</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:DeleteApplicationVersion"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }</pre>
Action: DeleteConfigurationTemplate		
configurationtemplate	<p>InApplication (Optional)</p> <p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the DeleteConfigurationTemplate action to delete a configuration template whose name is My Template in the application My App. Specifying the application name as a condition is optional.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:DeleteConfigurationTemplate"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My App/My Template"] }] }</pre>
Action: DeleteEnvironmentConfiguration		

Resource	Conditions	Example statement
environment	InApplication (Optional)	<p>The following policy allows the DeleteEnvironmentConfiguration action to delete a draft configuration for the environment myenv in the application My App. Specifying the application name as a condition is optional.</p> <pre> { "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:DeleteEnvironmentConfiguration"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"] }] } </pre>
Action: DeletePlatformVersion		
platform	aws:ResourceTag/ <i>key-name</i> (Optional) aws:TagKeys (Optional)	<p>The following policy allows the DeletePlatformVersion action to delete platform versions targeting the us-east-2 region, whose names begin with us-east-2_:</p> <pre> { "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:DeletePlatformVersion"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:platform/us-east-2_*"] }] } </pre>
Action: DescribeApplications		

Resource	Conditions	Example statement
application	<p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the DescribeApplications action to describe the application My App.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:DescribeApplications"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] }] }</pre>
Action: DescribeApplicationVersions		
applicationversion	<p>inApplication (Optional)</p> <p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the DescribeApplicationVersions action to describe the application version My Version in the application My App. Specifying the application name as a condition is optional.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:DescribeApplicationVersions"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"] }] }</pre>
Action: DescribeConfigurationOptions		

Resource	Conditions	Example statement
environment configurationtemplate solutionstack	InApplication (Optional) aws:ResourceTag/ <i>key-name</i> (Optional) aws:TagKeys (Optional)	<p>The following policy allows the DescribeConfigurationOptions action to describe the configuration options for the environment myenv in the application My App. Specifying the application name as a condition is optional.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": "elasticbeanstalk:DescribeConfigurationOptions", "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"] }] }</pre>
Action: DescribeConfigurationSettings		
environment configurationtemplate	InApplication (Optional) aws:ResourceTag/ <i>key-name</i> (Optional) aws:TagKeys (Optional)	<p>The following policy allows the DescribeConfigurationSettings action to describe the configuration settings for the environment myenv in the application My App. Specifying the application name as a condition is optional.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": "elasticbeanstalk:DescribeConfigurationSettings", "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"] }] }</pre>
Action: DescribeEnvironmentHealth		

Resource	Conditions	Example statement
environment	<p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows use of DescribeEnvironmentHealth to retrieve health information for an environment named myenv.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": "elasticbeanstalk:DescribeEnvironmentHealth", "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us- east-2:123456789012:environment/My App/myenv"] }] }</pre>
Action: DescribeEnvironmentResources		
environment	<p>InApplication (Optional)</p> <p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the DescribeEnvironmentResources action to return list of AWS resources for the environment myenv in the application My App. Specifying the application name as a condition is optional.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": "elasticbeanstalk:DescribeEnvironmentResources", "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us- east-2:123456789012:environment/My App/myenv"] }] }</pre>
Action: DescribeEnvironments		

Resource	Conditions	Example statement
environment	<p>InApplication (Optional)</p> <p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the DescribeEnvironments action to describe the environments myenv and myotherenv in the application My App. Specifying the application name as a condition is optional.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": "elasticbeanstalk:DescribeEnvironments", "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv", "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App2/myotherenv"] }] }</pre>
Action: DescribeEvents		
<p>application</p> <p>applicationversion</p> <p>configurationtemplate</p> <p>environment</p>	<p>InApplication</p> <p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the DescribeEvents action to list event descriptions for the environment myenv and the application version My Version in the application My App.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": "elasticbeanstalk:DescribeEvents", "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv", "arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }</pre>
Action: DescribeInstancesHealth		

Resource	Conditions	Example statement
environment	N/A	<p>The following policy allows use of DescribeInstancesHealth to retrieve health information for instances in an environment named myenv.</p> <pre> { "Version": "2012-10-17", "Statement": [{ "Action": "elasticbeanstalk:DescribeInstancesHealth", "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us- east-2:123456789012:environment/My App/myenv"] }] } </pre>
Action: DescribePlatformVersion		
platform	<p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the DescribePlatformVersion action to describe platform versions targeting the us-east-2 region, whose names begin with us-east-2_:</p> <pre> { "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:DescribePlatformVersion"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us- east-2:123456789012:platform/us-east-2_*"] }] } </pre>
Action: ListAvailableSolutionStacks		

Resource	Conditions	Example statement
solutionstack	N/A	<p>The following policy allows the <code>ListAvailableSolutionStacks</code> action to return only the solution stack 32bit Amazon Linux running Tomcat 7.</p> <pre> { "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:ListAvailableSolutionStacks"], "Effect": "Allow", "Resource": "arn:aws:elasticbeanstalk:us-east-2:solutionstack/32bit Amazon Linux running Tomcat 7" }] } </pre>
Action: ListPlatformVersions		
platform	<p><code>aws:RequestTag/<i>key-name</i></code> (Optional)</p> <p><code>aws:TagKeys</code> (Optional)</p>	<p>This example allows the <code>CreatePlatformVersion</code> action to create platform versions targeting the <code>us-east-2</code> region, whose names begin with <code>us-east-2_</code>:</p> <pre> { "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:ListPlatformVersions"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:platform/us-east-2_*"] }] } </pre>
Action: ListTagsForResource		

Resource	Conditions	Example statement
application applicationversion configurationtemplate environment platform	aws:ResourceTag/ <i>key-name</i> (Optional) aws:TagKeys (Optional)	<p>The following policy allows the <code>ListTagsForResource</code> action to list tags of existing resources only if they have a tag named <code>stage</code> with the value <code>test</code>:</p> <pre> { "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:ListTagsForResource"], "Effect": "Allow", "Resource": "*", "Condition": { "StringEquals": { "aws:ResourceTag/stage": ["test"] } } }] } </pre>
<p>Action: RebuildEnvironment</p>		
environment	InApplication aws:ResourceTag/ <i>key-name</i> (Optional) aws:TagKeys (Optional)	<p>The following policy allows the <code>RebuildEnvironment</code> action to rebuild the environment <code>myenv</code> in the application <code>My App</code>.</p> <pre> { "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:RebuildEnvironment"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] } </pre>
<p>Action: RequestEnvironmentInfo</p>		

Resource	Conditions	Example statement
environment	<p>InApplication</p> <p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the RequestEnvironmentInfo action to compile information about the environment myenv in the application My App.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:RequestEnvironmentInfo"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }</pre>
Action: RestartAppServer		
environment	<p>InApplication</p>	<p>The following policy allows the RestartAppServer action to restart the application container server for the environment myenv in the application My App.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:RestartAppServer"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }</pre>
Action: RetrieveEnvironmentInfo		

Resource	Conditions	Example statement
environment	<p>InApplication</p> <p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the RetrieveEnvironmentInfo action to retrieve the compiled information for the environment myenv in the application My App.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:RetrieveEnvironmentInfo"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }</pre>
Action: SwapEnvironmentCNAMEs		
environment	<p>InApplication (Optional)</p> <p>FromEnvironment (Optional)</p>	<p>The following policy allows the SwapEnvironmentCNAMEs action to swap the CNAMEs for the environments mysrcenv and mydestenv.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:SwapEnvironmentCNAMEs"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/mysrcenv", "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/mydestenv"] }] }</pre>
Action: TerminateEnvironment		

Resource	Conditions	Example statement
environment	<p>InApplication</p> <p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the TerminateEnvironment action to terminate the environment myenv in the application My App.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:TerminateEnvironment"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }</pre>
Action: UpdateApplication		
application	<p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the UpdateApplication action to update properties of the application My App.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:UpdateApplication"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] }] }</pre>
Action: UpdateApplicationResourceLifecycle		

Resource	Conditions	Example statement
application	<p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the UpdateApplicationResourceLifecycle action to update lifecycle settings of the application My App.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:UpdateApplicationResourceLifecycle"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] }] }</pre>
Action: UpdateApplicationVersion		
applicationversion	<p>InApplication</p> <p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the UpdateApplicationVersion action to update the properties of the application version My Version in the application My App.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:UpdateApplicationVersion"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }</pre>
Action: UpdateConfigurationTemplate		

Resource	Conditions	Example statement
configurationtemplate	<p>InApplication</p> <p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the UpdateConfigurationTemplate action to update the properties or options of the configuration template My Template in the application My App.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:UpdateConfigurationTemplate"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My App/My Template"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }</pre>
<p>Action: UpdateEnvironment</p>		

Resource	Conditions	Example statement
environment	<p>InApplication</p> <p>FromApplicationVersion</p> <p>FromConfigurationTemplate</p> <p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The following policy allows the <code>UpdateEnvironment</code> action to update the environment <code>myenv</code> in the application <code>My App</code> by deploying the application version <code>My Version</code>.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:UpdateEnvironment"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"], "elasticbeanstalk:FromApplicationVersion": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"] } } }] }</pre>
<p>Action: <code>UpdateTagsForResource</code> – <code>AddTags</code></p>		
<p>application</p> <p>applicationversion</p> <p>configurationtemplate</p> <p>environment</p> <p>platform</p>	<p>aws:ResourceTag/ <i>key-name</i> (Optional)</p> <p>aws:RequestTag/ <i>key-name</i> (Optional)</p> <p>aws:TagKeys (Optional)</p>	<p>The <code>AddTags</code> action is one of two virtual actions associated with the <code>UpdateTagsForResource</code> API.</p> <p>The following policy allows the <code>AddTags</code> action to modify tags of existing resources only if they have a tag named <code>stage</code> with the value <code>test</code>:</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:AddTags"], "Effect": "Allow", "Resource": "*", "Condition": { "StringEquals": { "aws:ResourceTag/stage": ["test"] } } }] }</pre>

Resource	Conditions	Example statement
Action: UpdateTagsForResource – RemoveTags		
application applicationversion configurationtemplate environment platform	aws:ResourceTag/ <i>key-name</i> (Optional) aws:TagKeys (Optional)	<p>The RemoveTags action is one of two virtual actions associated with the UpdateTagsForResource API.</p> <p>The following policy denies the RemoveTags action to request the removal of a tag named stage from existing resources:</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:RemoveTags"], "Effect": "Deny", "Resource": "*", "Condition": { "ForAnyValue:StringEquals": { "aws:TagKeys": ["stage"] } } }] }</pre>
Action: ValidateConfigurationSettings		
template environment	InApplication aws:ResourceTag/ <i>key-name</i> (Optional) aws:TagKeys (Optional)	<p>The following policy allows the ValidateConfigurationSettings action to validate configuration settings against the environment myenv in the application My App.</p> <pre>{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:ValidateConfigurationSettings"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }</pre>

Condition keys for Elastic Beanstalk actions

Keys enable you to specify conditions that express dependencies, restrict permissions, or specify constraints on the input parameters for an action. Elastic Beanstalk supports the following keys.

InApplication

Specifies the application that contains the resource that the action operates on.

The following example allows the `UpdateApplicationVersion` action to update the properties of the application version **My Version**. The `InApplication` condition specifies **My App** as the container for **My Version**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:UpdateApplicationVersion"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My
Version"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
east-2:123456789012:application/My App"]
        }
      }
    }
  ]
}
```

FromApplicationVersion

Specifies an application version as a dependency or a constraint on an input parameter.

The following example allows the `UpdateEnvironment` action to update the environment **myenv** in the application **My App**. The `FromApplicationVersion` condition constrains the `VersionLabel` parameter to allow only the application version **My Version** to update the environment.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:UpdateEnvironment"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
east-2:123456789012:application/My App"],
          "elasticbeanstalk:FromApplicationVersion": ["arn:aws:elasticbeanstalk:us-
east-2:123456789012:applicationversion/My App/My Version"]
        }
      }
    }
  ]
}
```

```
    }  
  }  
]  
}
```

FromConfigurationTemplate

Specifies a configuration template as a dependency or a constraint on an input parameter.

The following example allows the `UpdateEnvironment` action to update the environment **myenv** in the application **My App**. The `FromConfigurationTemplate` condition constrains the `TemplateName` parameter to allow only the configuration template **My Template** to update the environment.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Action": [  
        "elasticbeanstalk:UpdateEnvironment"  
      ],  
      "Effect": "Allow",  
      "Resource": [  
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"  
      ],  
      "Condition": {  
        "StringEquals": {  
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-  
east-2:123456789012:application/My App"],  
          "elasticbeanstalk:FromConfigurationTemplate": ["arn:aws:elasticbeanstalk:us-  
east-2:123456789012:configurationtemplate/My App/My Template"]  
        }  
      }  
    }  
  ]  
}
```

FromEnvironment

Specifies an environment as a dependency or a constraint on an input parameter.

The following example allows the `SwapEnvironmentCNAMEs` action to swap the CNAMEs in **My App** for all environments whose names begin with **mysrcenv** and **mydestenv** but not those environments whose names begin with **mysrcenvPROD*** and **mydestenvPROD***.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Action": [  
        "elasticbeanstalk:SwapEnvironmentCNAMEs"  
      ],  
      "Effect": "Allow",  
      "Resource": [  
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/mysrcenv*",  
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/mydestenv*"  
      ],  
      "Condition": {  
        "StringNotLike": {  
          "elasticbeanstalk:FromEnvironment": [  
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/  
mysrcenvPROD*",  
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/  
mydestenvPROD*"
```

```
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/  
mydestenvPROD*"  
    ]  
  }  
}
```

FromSolutionStack

Specifies a solution stack as a dependency or a constraint on an input parameter.

The following policy allows the `CreateConfigurationTemplate` action to create configuration templates whose name begins with **My Template** (`My Template*`) in the application **My App**. The `FromSolutionStack` condition constrains the `solutionstack` parameter to allow only the solution stack **32bit Amazon Linux running Tomcat 7** as the input value for that parameter.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Action": [  
        "elasticbeanstalk:CreateConfigurationTemplate"  
      ],  
      "Effect": "Allow",  
      "Resource": [  
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My App/  
My Template*"  
      ],  
      "Condition": {  
        "StringEquals": {  
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-  
east-2:123456789012:application/My App"],  
          "elasticbeanstalk:FromSolutionStack": ["arn:aws:elasticbeanstalk:us-  
east-2::solutionstack/32bit Amazon Linux running Tomcat 7"]  
        }  
      }  
    }  
  ]  
}
```

`aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, `aws:TagKeys`

Specify tag-based conditions. For details, see [Using tags to control access to Elastic Beanstalk resources](#) (p. 807).

Using tags to control access to Elastic Beanstalk resources

Conditions in AWS Identity and Access Management (IAM) user policy statements are part of the syntax that you use to specify permissions to resources that Elastic Beanstalk actions need to complete. For details about specifying policy statement conditions, see [Resources and conditions for Elastic Beanstalk actions](#) (p. 782). Using tags in conditions is one way to control access to resources and requests. For information about tagging Elastic Beanstalk resources, see [Tagging Elastic Beanstalk application resources](#) (p. 349). This topic discusses tag-based access control.

When you design IAM policies, you might be setting granular permissions by granting access to specific resources. As the number of resources that you manage grows, this task becomes more difficult. Tagging

resources and using tags in policy statement conditions can make this task easier. You grant access in bulk to any resource with a certain tag. Then you repeatedly apply this tag to relevant resources, during creation or later.

Tags can be attached to the resource or passed in the request to services that support tagging. In Elastic Beanstalk, resources can have tags, and some actions can include tags. When you create an IAM policy, you can use tag condition keys to control:

- Which users can perform actions on an environment, based on tags that it already has.
- What tags can be passed in an action's request.
- Whether specific tag keys can be used in a request.

For the complete syntax and semantics of tag condition keys, see [Controlling Access Using Tags](#) in the *IAM User Guide*.

The following examples demonstrate how to specify tag conditions in policies for Elastic Beanstalk users.

Example 1: Limit actions based on tags in the request

The Elastic Beanstalk **AWSElasticBeanstalkFullAccess** managed user policy gives users unlimited permission to perform any Elastic Beanstalk action on any resource.

The following policy limits this power and denies unauthorized users permission to create Elastic Beanstalk production environments. To do that, it denies the `CreateEnvironment` action if the request specifies a tag named `stage` with one of the values `gamma` or `prod`. In addition, the policy prevents these unauthorized users from tampering with the stage of production environments by not allowing tag modification actions to include these same tag values or to completely remove the `stage` tag. A customer's administrator must attach this IAM policy to unauthorized IAM users, in addition to the managed user policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "elasticbeanstalk:CreateEnvironment",
        "elasticbeanstalk:AddTags"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/stage": ["gamma", "prod"]
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "elasticbeanstalk:RemoveTags"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:TagKeys": ["stage"]
        }
      }
    }
  ]
}
```

Example 2: Limit actions based on resource tags

The Elastic Beanstalk **AWSElasticBeanstalkFullAccess** managed user policy gives users unlimited permission to perform any Elastic Beanstalk action on any resource.

The following policy limits this power and denies unauthorized users permission to perform actions on Elastic Beanstalk production environments. To do that, it denies specific actions if the environment has a tag named `stage` with one of the values `gamma` or `prod`. A customer's administrator must attach this IAM policy to unauthorized IAM users, in addition to the managed user policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "elasticbeanstalk:AddTags",
        "elasticbeanstalk:RemoveTags",
        "elasticbeanstalk:DescribeEnvironments",
        "elasticbeanstalk:TerminateEnvironment",
        "elasticbeanstalk:UpdateEnvironment",
        "elasticbeanstalk:ListTagsForResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/stage": ["gamma", "prod"]
        }
      }
    }
  ]
}
```

Example 3: Allow actions based on tags in the request

The following policy grants users permission to create Elastic Beanstalk development applications.

To do that, it allows the `CreateApplication` and `AddTags` actions if the request specifies a tag named `stage` with the value `development`. The `aws:TagKeys` condition ensures that the user can't add other tag keys. In particular, it ensures case sensitivity of the `stage` tag key. Notice that this policy is useful for IAM users that don't have the Elastic Beanstalk **AWSElasticBeanstalkFullAccess** managed user policy attached. The managed policy gives users unlimited permission to perform any Elastic Beanstalk action on any resource.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticbeanstalk:CreateApplication",
        "elasticbeanstalk:AddTags"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/stage": "development"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["stage"]
        }
      }
    }
  ]
}
```



```
}  
  }  
]  
}
```

Example 4: Allow actions based on resource tags

The following policy grants users permission to perform actions on, and get information about, Elastic Beanstalk development applications.

To do that, it allows specific actions if the application has a tag named `stage` with the value `development`. The `aws:TagKeys` condition ensures that the user can't add other tag keys. In particular, it ensures case sensitivity of the `stage` tag key. Notice that this policy is useful for IAM users that don't have the Elastic Beanstalk **AWSElasticBeanstalkFullAccess** managed user policy attached. The managed policy gives users unlimited permission to perform any Elastic Beanstalk action on any resource.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "elasticbeanstalk:UpdateApplication",  
        "elasticbeanstalk>DeleteApplication",  
        "elasticbeanstalk:DescribeApplications"  
      ],  
      "Resource": "*",  
      "Condition": {  
        "StringEquals": {  
          "aws:ResourceTag/stage": "development"  
        },  
        "ForAllValues:StringEquals": {  
          "aws:TagKeys": ["stage"]  
        }  
      }  
    }  
  ]  
}
```

Example policies based on managed policies

This section demonstrates how to control user access to AWS Elastic Beanstalk and includes example policies that provide the required access for common scenarios. These policies are derived from the Elastic Beanstalk managed policies. For information about attaching managed policies to users and groups, see [Managing Elastic Beanstalk user policies \(p. 775\)](#).

In this scenario, Example Corp. is a software company with three teams responsible for the company website: administrators who manage the infrastructure, developers who build the software for the website, and a QA team that tests the website. To help manage permissions to their Elastic Beanstalk resources, Example Corp. creates three groups to which members of each respective team belong: Admins, Developers, and Testers. Example Corp. wants the Admins group to have full access to all applications, environments, and their underlying resources so that they can create, troubleshoot, and delete all Elastic Beanstalk assets. Developers require permissions to view all Elastic Beanstalk assets and to create and deploy application versions. Developers should not be able to create new applications or environments or terminate running environments. Testers need to view all Elastic Beanstalk resources to monitor and test applications. The Testers should not be able to make changes to any Elastic Beanstalk resources.

The following example policies provide the required permissions for each group.

Example 1: Admins group – All Elastic Beanstalk and related service APIs

The following policy gives users permissions for all actions required to use Elastic Beanstalk. This policy also allows Elastic Beanstalk to provision and manage resources on your behalf in the following services. Elastic Beanstalk relies on these additional services to provision underlying resources when creating an environment.

- Amazon Elastic Compute Cloud
- Elastic Load Balancing
- Auto Scaling
- Amazon CloudWatch
- Amazon Simple Storage Service
- Amazon Simple Notification Service
- Amazon Relational Database Service
- AWS CloudFormation

Note that this policy is an example. It gives a broad set of permissions to the AWS services that Elastic Beanstalk uses to manage applications and environments. For example, `ec2:*` allows an AWS Identity and Access Management (IAM) user to perform any action on any Amazon EC2 resource in the AWS account. These permissions are not limited to the resources that you use with Elastic Beanstalk. As a best practice, you should grant individuals only the permissions they need to perform their duties.

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "rds:*",
        "cloudformation:*"
      ],
      "Resource" : "*"
    }
  ]
}
```

Example 2: Developers group – All but highly privileged operations

The following policy denies permission to create applications and environments, and allows all other Elastic Beanstalk actions.

Note that this policy is an example. It gives a broad set of permissions to the AWS products that Elastic Beanstalk uses to manage applications and environments. For example, `ec2:*` allows an IAM user to perform any action on any Amazon EC2 resource in the AWS account. These permissions are not limited to the resources that you use with Elastic Beanstalk. As a best practice, you should grant individuals only the permissions they need to perform their duties.

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Action" : [
        "elasticbeanstalk:CreateApplication",
        "elasticbeanstalk:CreateEnvironment",
        "elasticbeanstalk>DeleteApplication",
        "elasticbeanstalk:RebuildEnvironment",
        "elasticbeanstalk:SwapEnvironmentCNAMEs",
        "elasticbeanstalk:TerminateEnvironment"],
      "Effect" : "Deny",
      "Resource" : "*"
    },
    {
      "Action" : [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "rds:*",
        "cloudformation:*"],
      "Effect" : "Allow",
      "Resource" : "*"
    }
  ]
}
```

Example 3: Testers – View only

The following policy allows read-only access to all applications, application versions, events, and environments. It doesn't allow performing any actions.

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "elasticbeanstalk:Check*",
        "elasticbeanstalk:Describe*",
        "elasticbeanstalk:List*",
        "elasticbeanstalk:RequestEnvironmentInfo",
        "elasticbeanstalk:RetrieveEnvironmentInfo",
        "ec2:Describe*",
        "elasticloadbalancing:Describe*",
        "autoscaling:Describe*",
        "cloudwatch:Describe*",
        "cloudwatch:List*",
        "cloudwatch:Get*",
        "s3:Get*",
        "s3:List*",
        "sns:Get*",
        "sns:List*",
        "rds:Describe*",
        "cloudformation:Describe*",
        "cloudformation:Get*",
        "cloudformation:List*",
        "cloudformation:Validate*",
        "cloudformation:Estimate*"
      ]
    }
  ]
}
```

```
    ],  
    "Resource" : "*"    
  }  
]  
}
```

Example policies based on resource permissions

This section walks through a use case for controlling user permissions for Elastic Beanstalk actions that access specific Elastic Beanstalk resources. We'll walk through the sample policies that support the use case. For more information policies on Elastic Beanstalk resources, see [Creating a custom user policy \(p. 777\)](#). For information about attaching policies to users and groups, go to [Managing IAM Policies](#) in *Using AWS Identity and Access Management*.

In our use case, Example Corp. is a small consulting firm developing applications for two different customers. John is the development manager overseeing the development of the two Elastic Beanstalk applications, app1 and app2. John does development and some testing on the two applications, and only he can update the production environment for the two applications. These are the permissions that he needs for app1 and app2:

- View application, application versions, environments, and configuration templates
- Create application versions and deploy them to the staging environment
- Update the production environment
- Create and terminate environments

Jill is a tester who needs access to view the following resources in order to monitor and test the two applications: applications, application versions, environments, and configuration templates. However, she should not be able to make changes to any Elastic Beanstalk resources.

Jack is the developer for app1 who needs access to view all resources for app1 and also needs to create application versions for app1 and deploy them to the staging environment.

Judy is the administrator of the AWS account for Example Corp. She has created IAM users for John, Jill, and Jack and attaches the following policies to those users to grant the appropriate permissions to the app1 and app2 applications.

Example 1: John – Development manager for app1, app2

We have broken down John's policy into three separate policies so that they are easier to read and manage. Together, they give John the permissions he needs to perform development, testing, and deployment actions on the two applications.

The first policy specifies actions for Auto Scaling, Amazon S3, Amazon EC2, CloudWatch, Amazon SNS, Elastic Load Balancing, Amazon RDS, and AWS CloudFormation. Elastic Beanstalk relies on these additional services to provision underlying resources when creating an environment.

Note that this policy is an example. It gives a broad set of permissions to the AWS products that Elastic Beanstalk uses to manage applications and environments. For example, `ec2 : *` allows an IAM user to perform any action on any Amazon EC2 resource in the AWS account. These permissions are not limited to the resources that you use with Elastic Beanstalk. As a best practice, you should grant individuals only the permissions they need to perform their duties.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {
```

```

    "Effect": "Allow",
    "Action": [
      "ec2:*",
      "ecs:*",
      "ecr:*",
      "elasticloadbalancing:*",
      "autoscaling:*",
      "cloudwatch:*",
      "s3:*",
      "sns:*",
      "cloudformation:*",
      "dynamodb:*",
      "rds:*",
      "sqs:*",
      "logs:*",
      "iam:GetPolicyVersion",
      "iam:GetRole",
      "iam:PassRole",
      "iam:ListRolePolicies",
      "iam:ListAttachedRolePolicies",
      "iam:ListInstanceProfiles",
      "iam:ListRoles",
      "iam:ListServerCertificates",
      "acm:DescribeCertificate",
      "acm:ListCertificates",
      "codebuild:CreateProject",
      "codebuild:DeleteProject",
      "codebuild:BatchGetBuilds",
      "codebuild:StartBuild"
    ],
    "Resource": "*"
  }
}

```

The second policy specifies the Elastic Beanstalk actions that John is allowed to perform on the app1 and app2 resources. The `AllCallsInApplications` statement allows all Elastic Beanstalk actions (`"elasticbeanstalk:*"`) performed on all resources within app1 and app2 (for example, `elasticbeanstalk:CreateEnvironment`). The `AllCallsOnApplications` statement allows all Elastic Beanstalk actions (`"elasticbeanstalk:*"`) on the app1 and app2 application resources (for example, `elasticbeanstalk:DescribeApplications`, `elasticbeanstalk:UpdateApplication`, etc.). The `AllCallsOnSolutionStacks` statement allows all Elastic Beanstalk actions (`"elasticbeanstalk:*"`) for solution stack resources (for example, `elasticbeanstalk:ListAvailableSolutionStacks`).

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllCallsInApplications",
      "Action": [
        "elasticbeanstalk:*"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": [
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app1",
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app2"
          ]
        }
      }
    }
  ]
}

```

```
    }
  },
  {
    "Sid": "AllCallsOnApplications",
    "Action": [
      "elasticbeanstalk:*"
    ],
    "Effect": "Allow",
    "Resource": [
      "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app1",
      "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app2"
    ]
  },
  {
    "Sid": "AllCallsOnSolutionStacks",
    "Action": [
      "elasticbeanstalk:*"
    ],
    "Effect": "Allow",
    "Resource": [
      "arn:aws:elasticbeanstalk:us-east-2::solutionstack/*"
    ]
  }
]
}
```

The third policy specifies the Elastic Beanstalk actions that the second policy needs permissions to in order to complete those Elastic Beanstalk actions. The `AllNonResourceCalls` statement allows the `elasticbeanstalk:CheckDNSAvailability` action, which is required to call `elasticbeanstalk:CreateEnvironment` and other actions. It also allows the `elasticbeanstalk:CreateStorageLocation` action, which is required for `elasticbeanstalk:CreateApplication`, `elasticbeanstalk:CreateEnvironment`, and other actions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllNonResourceCalls",
      "Action": [
        "elasticbeanstalk:CheckDNSAvailability",
        "elasticbeanstalk:CreateStorageLocation"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Example 2: Jill – Tester for app1, app2

We have broken down Jill's policy into three separate policies so that they are easier to read and manage. Together, they give Jill the permissions she needs to perform testing and monitoring actions on the two applications.

The first policy specifies `Describe*`, `List*`, and `Get*` actions on Auto Scaling, Amazon S3, Amazon EC2, CloudWatch, Amazon SNS, Elastic Load Balancing, Amazon RDS, and AWS CloudFormation (for non-legacy container types) so that the Elastic Beanstalk actions are able to retrieve the relevant information about the underlying resources of the app1 and app2 applications.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:Describe*",
        "elasticloadbalancing:Describe*",
        "autoscaling:Describe*",
        "cloudwatch:Describe*",
        "cloudwatch:List*",
        "cloudwatch:Get*",
        "s3:Get*",
        "s3:List*",
        "sns:Get*",
        "sns:List*",
        "rds:Describe*",
        "cloudformation:Describe*",
        "cloudformation:Get*",
        "cloudformation:List*",
        "cloudformation:Validate*",
        "cloudformation:Estimate*"
      ],
      "Resource": "*"
    }
  ]
}
```

The second policy specifies the Elastic Beanstalk actions that Jill is allowed to perform on the app1 and app2 resources. The AllReadCallsInApplications statement allows her to call the Describe* actions and the environment info actions. The AllReadCallsOnApplications statement allows her to call the DescribeApplications and DescribeEvents actions on the app1 and app2 application resources. The AllReadCallsOnSolutionStacks statement allows viewing actions that involve solution stack resources (ListAvailableSolutionStacks, DescribeConfigurationOptions, and ValidateConfigurationSettings).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllReadCallsInApplications",
      "Action": [
        "elasticbeanstalk:Describe*",
        "elasticbeanstalk:RequestEnvironmentInfo",
        "elasticbeanstalk:RetrieveEnvironmentInfo"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": [
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app1",
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app2"
          ]
        }
      }
    },
    {
      "Sid": "AllReadCallsOnApplications",
      "Action": [
        "elasticbeanstalk:DescribeApplications",

```

```
        "elasticbeanstalk:DescribeEvents"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app1",
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app2"
      ]
    },
    {
      "Sid": "AllReadCallsOnSolutionStacks",
      "Action": [
        "elasticbeanstalk:ListAvailableSolutionStacks",
        "elasticbeanstalk:DescribeConfigurationOptions",
        "elasticbeanstalk:ValidateConfigurationSettings"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2::solutionstack/*"
      ]
    }
  ]
}
```

The third policy specifies the Elastic Beanstalk actions that the second policy needs permissions to in order to complete those Elastic Beanstalk actions. The `AllNonResourceCalls` statement allows the `elasticbeanstalk:CheckDNSAvailability` action, which is required for some viewing actions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllNonResourceCalls",
      "Action": [
        "elasticbeanstalk:CheckDNSAvailability"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Example 3: Jack – Developer for app1

We have broken down Jack's policy into three separate policies so that they are easier to read and manage. Together, they give Jack the permissions he needs to perform testing, monitoring, and deployment actions on the app1 resource.

The first policy specifies the actions on Auto Scaling, Amazon S3, Amazon EC2, CloudWatch, Amazon SNS, Elastic Load Balancing, Amazon RDS, and AWS CloudFormation (for non-legacy container types) so that the Elastic Beanstalk actions are able to view and work with the underlying resources of app1. For a list of supported non-legacy container types, see [the section called "Why are some platform versions marked legacy?"](#) (p. 426)

Note that this policy is an example. It gives a broad set of permissions to the AWS products that Elastic Beanstalk uses to manage applications and environments. For example, `ec2:*` allows an IAM user to perform any action on any Amazon EC2 resource in the AWS account. These permissions are not limited to the resources that you use with Elastic Beanstalk. As a best practice, you should grant individuals only the permissions they need to perform their duties.


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "rds:*",
        "cloudformation:*"
      ],
      "Resource": "*"
    }
  ]
}
```

The second policy specifies the Elastic Beanstalk actions that Jack is allowed to perform on the app1 resource.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllReadCallsAndAllVersionCallsInApplications",
      "Action": [
        "elasticbeanstalk:Describe*",
        "elasticbeanstalk:RequestEnvironmentInfo",
        "elasticbeanstalk:RetrieveEnvironmentInfo",
        "elasticbeanstalk:CreateApplicationVersion",
        "elasticbeanstalk>DeleteApplicationVersion",
        "elasticbeanstalk:UpdateApplicationVersion"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": [
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app1"
          ]
        }
      }
    },
    {
      "Sid": "AllReadCallsOnApplications",
      "Action": [
        "elasticbeanstalk:DescribeApplications",
        "elasticbeanstalk:DescribeEvents"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app1"
      ]
    },
    {
      "Sid": "UpdateEnvironmentInApplications",
      "Action": [
        "elasticbeanstalk:UpdateEnvironment"
      ],

```

```

        "Effect": "Allow",
        "Resource": [
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/app1/app1-
staging*"
        ],
        "Condition": {
            "StringEquals": {
                "elasticbeanstalk:InApplication": [
                    "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app1"
                ]
            },
            "StringLike": {
                "elasticbeanstalk:FromApplicationVersion": [
                    "arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/app1/
*"
                ]
            }
        }
    },
    {
        "Sid": "AllReadCallsOnSolutionStacks",
        "Action": [
            "elasticbeanstalk:ListAvailableSolutionStacks",
            "elasticbeanstalk:DescribeConfigurationOptions",
            "elasticbeanstalk:ValidateConfigurationSettings"
        ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:elasticbeanstalk:us-east-2::solutionstack/*"
        ]
    }
]
}

```

The third policy specifies the Elastic Beanstalk actions that the second policy needs permissions to in order to complete those Elastic Beanstalk actions. The `AllNonResourceCalls` statement allows the `elasticbeanstalk:CheckDNSAvailability` action, which is required to call `elasticbeanstalk:CreateEnvironment` and other actions. It also allows the `elasticbeanstalk:CreateStorageLocation` action, which is required for `elasticbeanstalk:CreateEnvironment`, and other actions.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllNonResourceCalls",
      "Action": [
        "elasticbeanstalk:CheckDNSAvailability",
        "elasticbeanstalk:CreateStorageLocation"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    }
  ]
}

```

Using Elastic Beanstalk with Amazon RDS

AWS Elastic Beanstalk provides support for running [Amazon Relational Database Service \(Amazon RDS\)](#) instances in your Elastic Beanstalk environment. To learn about that, see [the section called "Database" \(p. 506\)](#). This works great for development and testing environments. However, it isn't ideal for a production environment because it ties the lifecycle of the database instance to the lifecycle of your application's environment.

Note

If you haven't used a DB instance with your application before, try adding one to a test environment with the Elastic Beanstalk console first. This lets you verify that your application is able to read environment properties, construct a connection string, and connect to a DB instance before you add [Amazon Virtual Private Cloud \(Amazon VPC\)](#) and security group configuration to the mix. See [Adding a database to your Elastic Beanstalk environment \(p. 506\)](#) for details.

To decouple your database instance from your environment, you can run a database instance in Amazon RDS and configure your application to connect to it on launch. This enables you to connect multiple environments to a database, terminate an environment without affecting the database, and perform seamless updates with blue-green deployments. For a detailed procedure, see [How do I decouple an Amazon RDS instance from an Elastic Beanstalk environment without downtime, database sync issues, or data loss?](#)

To allow the Amazon EC2 instances in your environment to connect to an outside database, you can configure the environment's Auto Scaling group with an additional security group. The security group that you attach to your environment can be the same one that is attached to your database instance, or a separate security group from which the database's security group allows ingress.

Note

You can connect your environment to a database by adding a rule to your database's security group that allows ingress from the autogenerated security group that Elastic Beanstalk attaches to your environment's Auto Scaling group. However, doing so creates a dependency between the two security groups. Subsequently, when you attempt to terminate the environment, Elastic Beanstalk will be unable to delete the environment's security group because the database's security group is dependent on it.

After launching your database instance and configuring security groups, you can pass the connection information (endpoint, password, etc.) to your application by using environment properties. This is the same mechanism that Elastic Beanstalk uses when you run a database instance in your environment.

For additional security, you can store your connection information in Amazon S3, and configure Elastic Beanstalk to retrieve it during deployment. With [configuration files \(.ebextensions\) \(p. 600\)](#), you can configure the instances in your environment to securely retrieve files from Amazon S3 when you deploy your application.

Topics

- [Launching and connecting to an external Amazon RDS instance in a default VPC \(p. 820\)](#)
- [Launching and connecting to an external Amazon RDS instance in EC2 classic \(p. 825\)](#)
- [Storing the connection string in Amazon S3 \(p. 829\)](#)
- [Cleaning up an external Amazon RDS instance \(p. 831\)](#)

Launching and connecting to an external Amazon RDS instance in a default VPC

To use an external database with an application running in Elastic Beanstalk, first launch a DB instance with Amazon RDS. Any instance that you launch with Amazon RDS is completely independent of Elastic

Beanstalk and your Elastic Beanstalk environments, and is not dependent on Elastic Beanstalk for configuration. This means that you can use any DB engine and instance type supported by Amazon RDS, even those not used by Elastic Beanstalk.

The following procedures describe the process for a [default VPC](#). The process is the same if you are using a custom VPC. The only additional requirements are that your environment and DB instance are in the same subnet, or in subnets that are allowed to communicate with each other. See [Using Elastic Beanstalk with Amazon VPC \(p. 834\)](#) for details on configuring a custom VPC for use with Elastic Beanstalk.

To launch an RDS DB instance in a default VPC

1. Open the [RDS console](#).
2. Choose **Databases** in the navigation pane.
3. Choose **Create database**.
4. Choose **Standard Create**.

Important

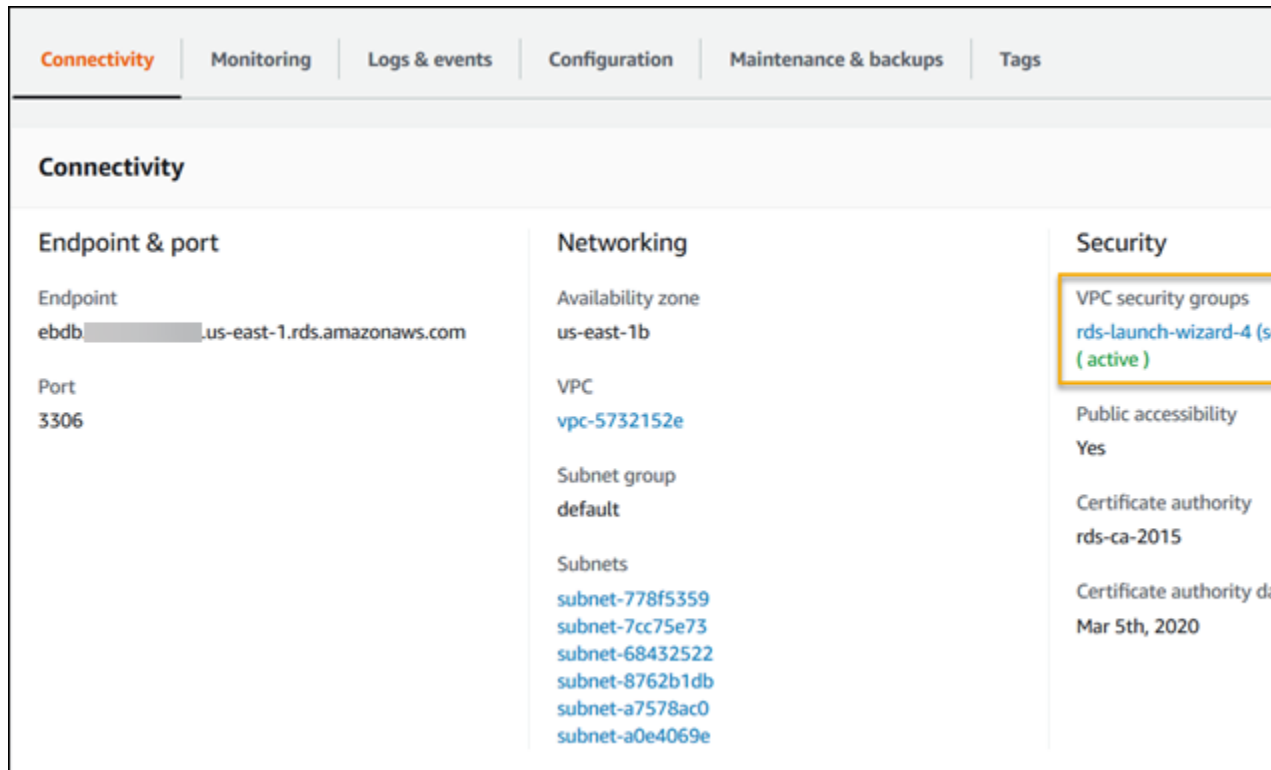
Do not choose **Easy Create**. It does not allow you to configure the necessary settings to launch this RDS DB.

5. Under **Additional configuration**, for **Initial database name**, type **ebdb**.
6. Review the default settings carefully and adjust as necessary. Pay attention to the following options:
 - **DB instance class** – Choose an instance size that has an appropriate amount of memory and CPU power for your workload.
 - **Multi-AZ deployment** – For high availability, set this to **Create an Aurora Replica/Reader node in a different AZ**.
 - **Master username** and **Master password** – The database username and password. Make a note of these settings because you'll use them later.
7. Verify the default settings for the remaining options, and then choose **Create database**.

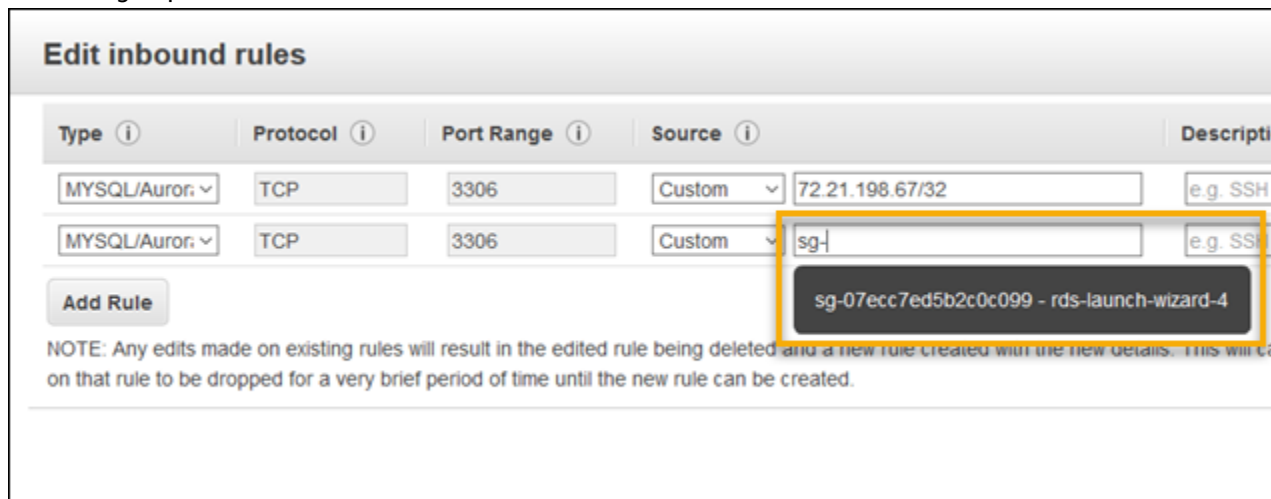
Next, modify the security group attached to your DB instance to allow inbound traffic on the appropriate port. This is the same security group that you will attach to your Elastic Beanstalk environment later, so the rule that you add will grant ingress permission to other resources in the same security group.

To modify the inbound rules on your RDS instance's security group

1. Open the [Amazon RDS console](#).
2. Choose **Databases**.
3. Choose the name of your DB instance to view its details.
4. In the **Connectivity** section, make a note of the **Subnets**, **Security groups**, and **Endpoint** shown on this page so you can use this information later.
5. Under **Security**, you can see the security group associated with the DB instance. Open the link to view the security group in the Amazon EC2 console.



6. In the security group details, choose **Inbound**.
7. Choose **Edit**.
8. Choose **Add Rule**.
9. For **Type**, choose the DB engine that your application uses.
10. For **Source**, type **sg-** to view a list of available security groups. Choose the current security group to allow resources in the security group to receive traffic on the database port from other resources in the same group.



11. Choose **Save**.

Next, add the DB instance's security group to your running environment. This procedure causes Elastic Beanstalk to re-provision all instances in your environment with the additional security group attached.

To add a security group to your environment

- Do one of the following:
 - To add a security group using the Elastic Beanstalk console
 - a. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
 - b. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

- c. In the navigation pane, choose **Configuration**.
 - d. In the **Instances** configuration category, choose **Edit**.
 - e. Under **EC2 security groups**, choose the security group to attach to the instances, in addition to the instance security group that Elastic Beanstalk creates.
 - f. Choose **Apply**.
 - g. Read the warning, and then choose **Confirm**.
- To add a security group using a [configuration file \(p. 600\)](#), use the [securitygroup-addexisting.config](#) example file.

Next, pass the connection information to your environment by using environment properties. When you [add a DB instance to your environment \(p. 506\)](#) with the Elastic Beanstalk console, Elastic Beanstalk uses environment properties like **RDS_HOSTNAME** to pass connection information to your application. You can use the same properties, which will let you use the same application code with both integrated DB instances and external DB instances, or choose your own property names.

To configure environment properties for an Amazon RDS DB instance

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. In the **Environment properties** section, define the variables that your application reads to construct a connection string. For compatibility with environments that have an integrated RDS DB instance, use the following names and values. You can find all values, except for your password, in the [RDS console](#).

Property name	Description	Property value
RDS_HOSTNAME	The hostname of the DB instance.	On the Connectivity & security tab on the Amazon RDS console: Endpoint .
RDS_PORT	The port on which the DB instance accepts connections. The default value varies among DB engines.	On the Connectivity & security tab on the Amazon RDS console: Port .

Property name	Description	Property value
RDS_DB_NAME	The database name, ebdb .	On the Configuration tab on the Amazon RDS console: DB Name .
RDS_USERNAME	The username that you configured for your database.	On the Configuration tab on the Amazon RDS console: Master username .
RDS_PASSWORD	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

Environment Properties

The following properties are passed into the application as environment variables. [Learn more.](#)

Property Name	Property Value
RDS_DB_NAME	ebdb
RDS_HOSTNAME	webapp-db.jxzb5mpan
RDS_PORT	5432
RDS_USERNAME	webapp-admin
RDS_PASSWORD	kUj5uKxmWDMYc403

[Cancel](#)

6. Choose **Apply**.

If you haven't programmed your application to read environment properties and construct a connection string yet, see the following language-specific topics for instructions:

- Java SE – [Connecting to a database \(Java SE platforms\) \(p. 119\)](#)
- Java with Tomcat – [Connecting to a database \(Tomcat platforms\) \(p. 120\)](#)
- Node.js – [Connecting to a database \(p. 227\)](#)
- .NET – [Connecting to a database \(p. 165\)](#)
- PHP – [Connecting to a database with a PDO or MySQLi \(p. 287\)](#)

- Python – [Connecting to a database \(p. 313\)](#)
- Ruby – [Connecting to a database \(p. 331\)](#)

Finally, depending on when your application reads environment variables, you might need to restart the application server on the instances in your environment.

To restart your environment's app servers

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Restart app server(s)**.

Launching and connecting to an external Amazon RDS instance in EC2 classic

If you use EC2 Classic (no VPC) with AWS Elastic Beanstalk, the procedure changes slightly due to differences in how security groups work. In EC2 Classic, DB instances can't use EC2 security groups, so they get a DB security group that works only with Amazon RDS.

You can add rules to a DB security group that allow ingress from EC2 security groups, but you cannot attach a DB security group to your environment's Auto Scaling group. To avoid creating a dependency between the DB security group and your environment, you must create a third security group in Amazon EC2, add a rule in the DB security group to grant ingress to the new security group, and then assign it to the Auto Scaling group in your Elastic Beanstalk environment.

To launch an RDS instance in EC2 classic (no VPC)

1. Open the [RDS management console](#).
2. Choose **Create database**.
3. Proceed through the wizard. Note the values that you enter for the following options:
 - **Master Username**
 - **Master Password**
4. When you reach **Configure advanced settings**, for **Network and Security** settings, choose the following:
 - **VPC – Not in VPC**. If this option isn't available, your account might not support [EC2-Classic](#), or you may have chosen an [instance type that is only available in VPC](#).
 - **Availability Zone – No Preference**
 - **DB Security Group(s) – Create new Security Group**
5. Configure the remaining options and choose **Create database**. Note the values that you enter for the following options:
 - **Database Name**
 - **Database Port**

In EC2-Classic, your DB instance will have a DB security group instead of a VPC security group. You can't attach a DB security group to your Elastic Beanstalk environment, so you need to create a new security

group that you can authorize to access the DB instance and attach to your environment. We will refer to this as a *bridge security group* and name it **webapp-bridge**.

To create a bridge security group

1. Open the [Amazon EC2 console](#).
2. Choose **Security Groups** under **Network & Security** in the navigation sidebar.
3. Choose **Create Security Group**.
4. For **Security group name**, type **webapp-bridge**.
5. For **Description**, type **Provide access to DB instance from Elastic Beanstalk environment instances**.
6. For **VPC**, leave the default selected.
7. Choose **Create**

Next, modify the security group attached to your DB instance to allow inbound traffic from the bridge security group.

To modify the ingress rules on your RDS instance's security group

1. Open the [Amazon RDS console](#).
2. Choose **Databases**.
3. Choose the name of your DB instance to view its details.
4. In the **Connectivity** section, under **Security**, the security group associated with the DB instance is shown. Open the link to view the security group in the Amazon EC2 console.
5. In the security group details, set **Connection Type** to **EC2 Security Group**.
6. Set **EC2 Security Group Name** to the name of the bridge security group that you created.
7. Choose **Authorize**.

Next, add the bridge security group to your running environment. This procedure requires all instances in your environment to be reprovisioned with the additional security group attached.

To add a security group to your environment

- Do one of the following:
 - To add a security group using the Elastic Beanstalk console
 - a. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
 - b. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

- c. In the navigation pane, choose **Configuration**.
 - d. In the **Instances** configuration category, choose **Edit**.
 - e. Under **EC2 security groups**, choose the security group to attach to the instances, in addition to the instance security group that Elastic Beanstalk creates.
 - f. Choose **Apply**.
 - g. Read the warning, and then choose **Confirm**.
- To add a security group using a [configuration file \(p. 600\)](#), use the [securitygroup-addexisting.config](#) example file.

Next, pass the connection information to your environment by using environment properties. When you [add a DB instance to your environment \(p. 506\)](#) with the Elastic Beanstalk console, Elastic Beanstalk uses environment properties like **RDS_HOSTNAME** to pass connection information to your application. You can use the same properties, which will let you use the same application code with both integrated DB instances and external DB instances, or choose your own property names.

To configure environment properties

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. In the **Environment Properties** section, define the variables that your application reads to construct a connection string. For compatibility with environments that have an integrated RDS instance, use the following:
 - **RDS_DB_NAME** – The **DB Name** shown in the Amazon RDS console.
 - **RDS_USERNAME** – The **Master Username** that you enter when you add the database to your environment.
 - **RDS_PASSWORD** – The **Master Password** that you enter when you add the database to your environment.
 - **RDS_HOSTNAME** – The **Endpoint** of the DB instance shown in the Amazon RDS console.
 - **RDS_PORT** – The **Port** shown in the Amazon RDS console.

Environment Properties

The following properties are passed into the application as environment variables. [Learn more.](#)

Property Name	Property Value
RDS_DB_NAME	ebdb
RDS_HOSTNAME	webapp-db.jxzc5mpan
RDS_PORT	5432
RDS_USERNAME	webapp-admin
<input type="text" value="RDS_PASSWORD"/>	<input type="text" value="kUj5uKxmWDMYc403"/>

[Cancel](#)

6. Choose **Apply**

If you haven't programmed your application to read environment properties and construct a connection string yet, see the following language-specific topics for instructions:

- Java SE – [Connecting to a database \(Java SE platforms\) \(p. 119\)](#)
- Java with Tomcat – [Connecting to a database \(Tomcat platforms\) \(p. 120\)](#)
- Node.js – [Connecting to a database \(p. 227\)](#)
- .NET – [Connecting to a database \(p. 165\)](#)
- PHP – [Connecting to a database with a PDO or MySQLi \(p. 287\)](#)
- Python – [Connecting to a database \(p. 313\)](#)
- Ruby – [Connecting to a database \(p. 331\)](#)

Finally, depending on when your application reads environment variables, you might need to restart the application server on the instances in your environment.

To restart your environment's app servers

1. Open the [Elastic Beanstalk console](#), and then, in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note

If you have many environments, use the search bar to filter the environment list.

3. Choose **Environment actions**, and then choose **Restart app server(s)**.

Storing the connection string in Amazon S3

Providing connection information to your application with environment properties is a good way to keep passwords out of your code, but it's not a perfect solution. Environment properties are discoverable in the [environment management console](#) (p. 353), and can be viewed by any user that has permission to [describe configuration settings](#) on your environment. Depending on the platform, environment properties may also appear in [instance logs](#) (p. 732).

You can lock down your connection information by storing it in an Amazon S3 bucket that you control. The basic steps are as follows:

- Upload a file that contains your connection string to an Amazon S3 bucket.
- Grant the EC2 instance profile permission to read the file.
- Configure your application to download the file during deployment.
- Read the file in your application code.

First, create a bucket to store the file that contains your connection string. For this example, we will use a JSON file that has a single key and value. The value is a JDBC connection string for a PostgreSQL DB instance in Amazon RDS.

beanstalk-database.json

```
{
  "connection": "jdbc:postgresql://mydb.b5uacpxznm.us-west-2.rds.amazonaws.com:5432/ebdb?
user=username&password=myspassword"
}
```

The highlighted portions of the URL correspond to the endpoint, port, DB name, user name and password for the database.

To create a bucket and upload a file

1. Open the [Amazon S3 console](#).
2. Choose **Create Bucket**.
3. Type a **Bucket Name**, and then choose a **Region**.
4. Choose **Create**.
5. Open the bucket, and then choose **Upload**
6. Follow the prompts to upload the file.

By default, your account owns the file and has permission to manage it, but IAM users and roles do not unless you grant them access explicitly. Grant the instances in your Elastic Beanstalk environment by adding a policy to the [instance profile](#) (p. 21).

The default instance profile is named `aws-elasticbeanstalk-ec2-role`. If you're not sure what your instance profile is named, you can find it on the **Configuration** page in the [environment management console](#) (p. 451).

To add permissions to the instance profile

1. Open the [IAM console](#).
2. Choose **Roles**.
3. Choose **aws-elasticbeanstalk-ec2-role**.
4. Choose **Add inline policy**.
5. Add a policy that allows the instance to retrieve the file.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "database",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::my-secret-bucket-123456789012/beanstalk-database.json"
      ]
    }
  ]
}
```

Replace the bucket and object names with the names of your bucket and object.

Next, add a [configuration file \(p. 600\)](#) to your source code that tells Elastic Beanstalk to download the file from Amazon S3 during deployment.

```
~/my-app/.ebextensions/database.config
```

```
Resources:
  AWSEBAutoScalingGroup:
    Metadata:
      AWS::CloudFormation::Authentication:
        S3Auth:
          type: "s3"
          buckets: ["my-secret-bucket-123456789012"]
          roleName: "aws-elasticbeanstalk-ec2-role"

files:
  "/tmp/beanstalk-database.json" :
    mode: "000644"
    owner: root
    group: root
    authentication: "S3Auth"
    source: https://s3-us-west-2.amazonaws.com/my-secret-bucket-123456789012/beanstalk-database.json
```

This configuration file does two things. The `Resources` key adds an authentication method to your environment's Auto Scaling group metadata that Elastic Beanstalk can use to access Amazon S3. The `files` key tells Elastic Beanstalk to download the file from Amazon S3 and store it locally in `/tmp/` during deployment.

Deploy your application with the configuration file in `.ebextensions` folder at the root of your source code. If you configured permissions correctly, the deployment will succeed and the file will be downloaded to all of the instances in your environment. If not, the deployment will fail.

Finally, add code to your application to read the JSON file and use the connection string to connect to the database.

Cleaning up an external Amazon RDS instance

When you connect an external Amazon RDS instance to your Elastic Beanstalk environment, the database instance isn't tied to your environment's lifecycle, and isn't deleted when you terminate your environment. To ensure that personal information that you might have stored in the database instance isn't unnecessarily retained, delete any records that you don't need anymore, or delete the database instance.

Using Elastic Beanstalk with Amazon S3

Amazon Simple Storage Service (Amazon S3) provides highly durable, fault-tolerant data storage.

Elastic Beanstalk creates an Amazon S3 bucket named `elasticbeanstalk-region-account-id` for each region in which you create environments. Elastic Beanstalk uses this bucket to store objects, for example temporary configuration files, that are required for the proper operation of your application.

Elastic Beanstalk doesn't turn on default encryption for the Amazon S3 bucket that it creates. This means that by default, objects are stored unencrypted in the bucket (and are accessible only by authorized users). Some applications require all objects to be encrypted when they are stored—on a hard drive, in a database, etc. (also known as *encryption at rest*). If you have this requirement, you can configure your account's buckets for default encryption. For more details, see [Amazon S3 Default Encryption for S3 Buckets](#) in the *Amazon Simple Storage Service Developer Guide*.

Contents of the Elastic Beanstalk Amazon S3 bucket

The following table lists some objects that Elastic Beanstalk stores in your `elasticbeanstalk-*` Amazon S3 bucket. The table also shows which objects have to be deleted manually. To avoid unnecessary storage costs, and to ensure that personal information isn't retained, be sure to manually delete these objects when you no longer need them.

Object	When stored?	When deleted?
Application versions (p. 337)	When you create an environment or deploy your application code to an existing environment, Elastic Beanstalk stores an application version in Amazon S3 and associates it with the environment.	During application deletion, and according to Version lifecycle (p. 339).
Source bundles (p. 337)	When you upload a new application version using the Elastic Beanstalk console or the EB CLI, Elastic Beanstalk stores a copy of it in Amazon S3, and sets it as your environment's source bundle.	<i>Manually.</i> When you delete an application version, you can choose Delete versions from Amazon S3 to also delete the related source bundle. For details, see Managing application versions (p. 337).
Custom platforms (p. 388)	When you create a custom platform, Elastic Beanstalk temporarily stores related data in Amazon S3.	Upon successful completion of the custom platform's creation.
Log files (p. 732)	You can request Elastic Beanstalk to retrieve instance log files (tail or bundle logs) and store them in Amazon S3. You can also enable	Tail and bundle logs: 15 minutes after they are created.

Object	When stored?	When deleted?
	log rotation and configure your environment to publish logs automatically to Amazon S3 after they are rotated.	Rotated logs: <i>Manually.</i>
Saved configurations	<i>Manually.</i> (p. 640)	<i>Manually.</i>

Deleting objects in the Elastic Beanstalk Amazon S3 bucket

When you terminate an environment or delete an application, Elastic Beanstalk deletes most related objects from Amazon S3. To minimize storage costs of a running application, routinely delete objects that your application doesn't need. In addition, pay attention to objects that you have to delete manually, as listed in [Contents of the Elastic Beanstalk Amazon S3 bucket](#) (p. 831). To ensure that private information isn't unnecessarily retained, delete these objects when you don't need them anymore.

- Delete application versions that you don't expect to use in your application anymore. When you delete an application version, you can select **Delete versions from Amazon S3** to also delete the related source bundle—a copy of your application's source code and configurations files, which Elastic Beanstalk uploaded to Amazon S3 when you deployed an application or uploaded an application version. To learn how to delete an application version, see [Managing application versions](#) (p. 337).
- Delete rotated logs that you don't need. Alternatively, download them or move them to Amazon S3 Glacier for further analysis.
- Delete saved configurations that you aren't going to use in any environment anymore.

Deleting the Elastic Beanstalk Amazon S3 bucket

Elastic Beanstalk applies a bucket policy to buckets it creates to allow environments to write to the bucket and prevent accidental deletion. If you need to delete a bucket that Elastic Beanstalk created, first delete the bucket policy from the **Permissions** section of the bucket properties in the Amazon S3 console.

Warning

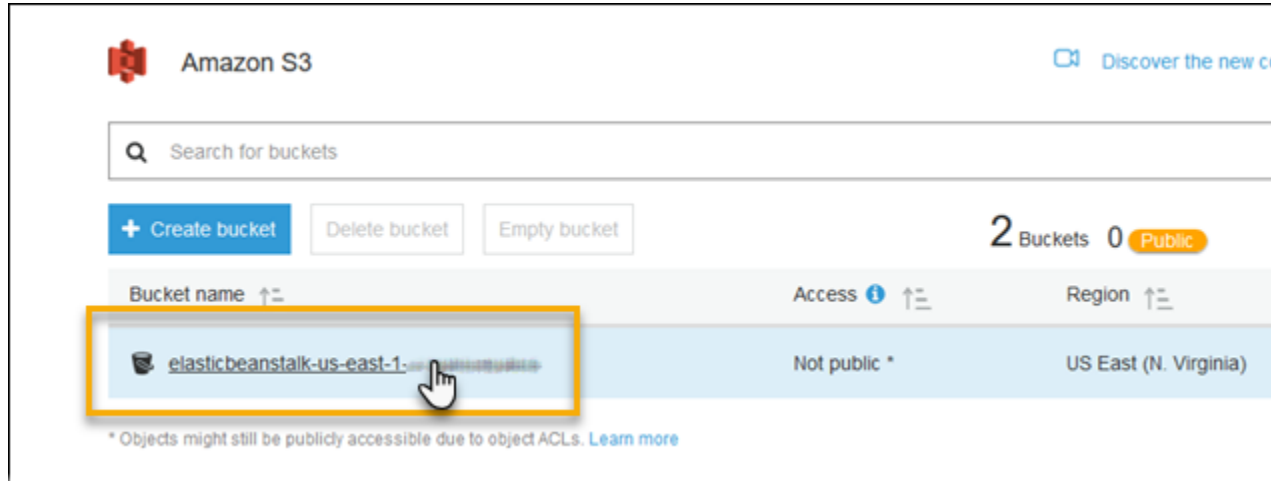
If you delete a bucket that Elastic Beanstalk created in your account, and you still have existing applications and running environments in the corresponding region, your applications might stop working correctly. For example:

- When an environment scales out, Elastic Beanstalk should be able to find the environment's application version in the Amazon S3 bucket and use it to start new Amazon EC2 instances.
- When you create a custom platform, Elastic Beanstalk uses temporary Amazon S3 storage during the creation process.

We recommend that you delete specific unnecessary objects from your Elastic Beanstalk Amazon S3 bucket, instead of deleting the entire bucket.

To delete an Elastic Beanstalk storage bucket (console)

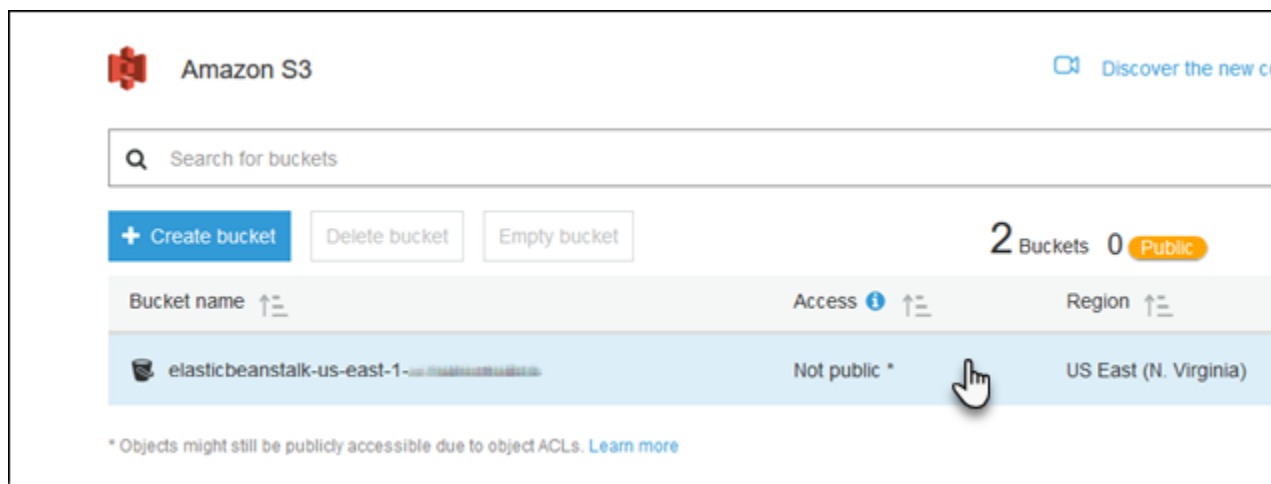
1. Open the [Amazon S3 console](#).
2. Open the Elastic Beanstalk storage bucket's page by choosing the bucket name.



3. Choose the **Permissions** tab.
4. Choose **Bucket Policy**.
5. Choose **Delete**.



6. Go back to the Amazon S3 console's main page, and then select the Elastic Beanstalk storage bucket by clicking its line anywhere outside of the bucket name.



7. Choose **Delete Bucket**.
8. Type the name of the bucket, and then choose **Confirm**.

Using Elastic Beanstalk with Amazon VPC

You can use an [Amazon Virtual Private Cloud](#) (Amazon VPC) to create a secure network for your Elastic Beanstalk application and related AWS resources. When you create your environment, you choose which VPC, subnets, and security groups are used for your application instances and load balancer. You can use any VPC configuration that you like as long as it meets the following requirements.

VPC requirements

- **Internet Access** – Instances can have access to the internet through one of the following methods:
 - **Public Subnet** – Instances have a public IP address and use an internet gateway to access the internet.
 - **Private Subnet** – Instances use a NAT device to access the internet.

Note

If you configure [VPC endpoints](#) (p. 847) in your VPC to connect to both the `elasticbeanstalk` and `elasticbeanstalk-healthd` services, internet access is optional, and is only required if your application specifically needs it. Without VPC endpoints, your VPC must have access to the internet.

The default VPC that Elastic Beanstalk sets up for you provides internet access.

Elastic Beanstalk doesn't support proxy settings like `HTTPS_PROXY` for configuring a web proxy.

- **NTP** – Instances in your Elastic Beanstalk environment use Network Time Protocol (NTP) to synchronize the system clock. If instances are unable to communicate on UDP port 123, the clock may go out of sync, causing issues with Elastic Beanstalk health reporting. Ensure that your VPC security groups and network ACLs allow inbound and outbound UDP traffic on port 123 to avoid these issues.

The [elastic-beanstalk-samples](#) repository provides AWS CloudFormation templates that you can use to create a VPC for use with your Elastic Beanstalk environments.

To create resources with a AWS CloudFormation template

1. Clone the samples repository or download a template using the links in the [README](#).
2. Open the [AWS CloudFormation console](#).

3. Choose **Create stack**.
4. Choose **Upload a template to Amazon S3**.
5. Choose **Upload file** and upload the template file from your local machine.
6. Choose **Next** and follow the instructions to create a stack with the resources in the template.

When stack creation completes, check the **Outputs** tab to find the VPC ID and subnet IDs. Use these to configure the VPC in the new environment wizard [network configuration category](#) (p. 380).

Topics

- [Public VPC](#) (p. 835)
- [Public/private VPC](#) (p. 836)
- [Private VPC](#) (p. 836)
- [Example: Launching an Elastic Beanstalk application in a VPC with bastion hosts](#) (p. 837)
- [Example: Launching an Elastic Beanstalk in a VPC with Amazon RDS](#) (p. 842)
- [Using Elastic Beanstalk with VPC endpoints](#) (p. 847)

Public VPC

AWS CloudFormation template – [vpc-public.yaml](#)

Settings (load balanced)

- **Load balancer visibility** – Public
- **Load balancer subnets** – Both public subnets
- **Instance public IP** – Enabled
- **Instance subnets** – Both public subnets
- **Instance security groups** – Add the default security group

Settings (single instance)

- **Instance subnets** – One of the public subnets
- **Instance security groups** – Add the default security group

A basic *public-only* VPC layout includes one or more public subnets, an internet gateway, and a default security group that allows traffic between resources in the VPC. When you create an environment in the VPC, Elastic Beanstalk creates additional resources that vary depending on the environment type.

VPC resources

- **Single Instance** – Elastic Beanstalk creates a security group for the application instance that allows traffic on port 80 from the internet, and assigns the instance an Elastic IP to give it a public IP address. The environment's domain name resolves to the instance's public IP address.
- **Load Balanced** – Elastic Beanstalk creates a security group for the load balancer that allows traffic on port 80 from the internet, and a security group for the application instances that allows traffic from the load balancer's security group. The environment's domain name resolves to the load balancer's public domain name.

This is similar to the way that Elastic Beanstalk manages networking when you use the default VPC. Security in a public subnet depends on the load balancer and instance security groups created by Elastic Beanstalk. It is the least expensive configuration as it does not require a NAT Gateway.

Public/private VPC

AWS CloudFormation template – [vpc-privatepublic.yaml](#)

Settings (load balanced)

- **Load balancer visibility** – Public
- **Load balancer subnets** – Both public subnets
- **Instance public IP** – Disabled
- **Instance subnets** – Both private subnets
- **Instance security groups** – Add the default security group

For additional security, add private subnets to your VPC to create a *public-private* layout. This layout requires a load balancer and NAT gateway in the public subnets, and lets you run your application instances, database, and any other resources in private subnets. Instances in private subnets can only communicate with the internet through the load balancer and NAT gateway.

Private VPC

AWS CloudFormation template – [vpc-private.yaml](#)

Settings (load balanced)

- **Load balancer visibility** – Private
- **Load balancer subnets** – Both private subnets
- **Instance public IP** – Disabled
- **Instance subnets** – Both private subnets
- **Instance security groups** – Add the default security group

For internal applications that shouldn't have access from the internet, you can run everything in private subnets and configure the load balancer to be internally facing (change **Load balancer visibility** to **Internal**). This template creates a VPC with no public subnets and no internet gateway. Use this layout for applications that should only be accessible from the same VPC or an attached VPN.

Running an Elastic Beanstalk environment in a private VPC

When you create your Elastic Beanstalk environment in a private VPC, the environment doesn't have access to the internet. Your application might need access to the Elastic Beanstalk service or other services. Your environment might use enhanced health reporting, and in this case the environment instances send health information to the enhanced health service. And Elastic Beanstalk code on environment instances sends traffic to other AWS services, and other traffic to non-AWS endpoints (for example, to download dependency packages for your application). Here are some steps you might need to take in this case to ensure that your environment works properly.

- *Configure VPC endpoints for Elastic Beanstalk* – Elastic Beanstalk and its enhanced health service support VPC endpoints, which ensure that traffic to these services stays inside the Amazon network and doesn't require internet access. For more information, see [the section called "VPC endpoints"](#) (p. 847).
- *Configure VPC endpoints for additional services* – Elastic Beanstalk instances send traffic to several other AWS services on your behalf: Amazon Simple Storage Service (Amazon S3), Amazon Simple

Queue Service (Amazon SQS), and AWS CloudFormation. You must configure VPC endpoints for these services too. For detailed information about VPC endpoints, including per-service links, see [VPC Endpoints](#) in the *Amazon VPC User Guide*.

Note

Some AWS services, including Elastic Beanstalk, support VPC endpoints in a limited number of AWS Regions. When you design your private VPC solution, verify that Elastic Beanstalk and the other dependent services mentioned here support VPC endpoints in the AWS Region that you choose.

- *Provide a private Docker image* – In a [Docker \(p. 50\)](#) environment, code on the environment's instances might try to pull your configured Docker image from the internet during environment creation and fail. To avoid this failure, [build a custom Docker image \(p. 58\)](#) on your environment, or use a Docker image stored in [Amazon Elastic Container Registry \(Amazon ECR\)](#) and [configure a VPC endpoint for the Amazon ECR service](#).
- *Enable DNS names* – Elastic Beanstalk code on environment instances sends traffic to all AWS services using their public endpoints. To ensure that this traffic goes through, choose the **Enable DNS name** option when you configure all interface VPC endpoints. This adds a DNS entry in your VPC that maps the public service endpoint to the interface VPC endpoint.

Important

If your VPC isn't private and has public internet access, and if **Enable DNS name** is disabled for any VPC endpoint, traffic to the respective service travels through the public internet. This is probably not what you intend. It's easy to detect this issue with a private VPC, because it prevents this traffic from going through and you receive errors. However, with a public facing VPC, you get no indication.

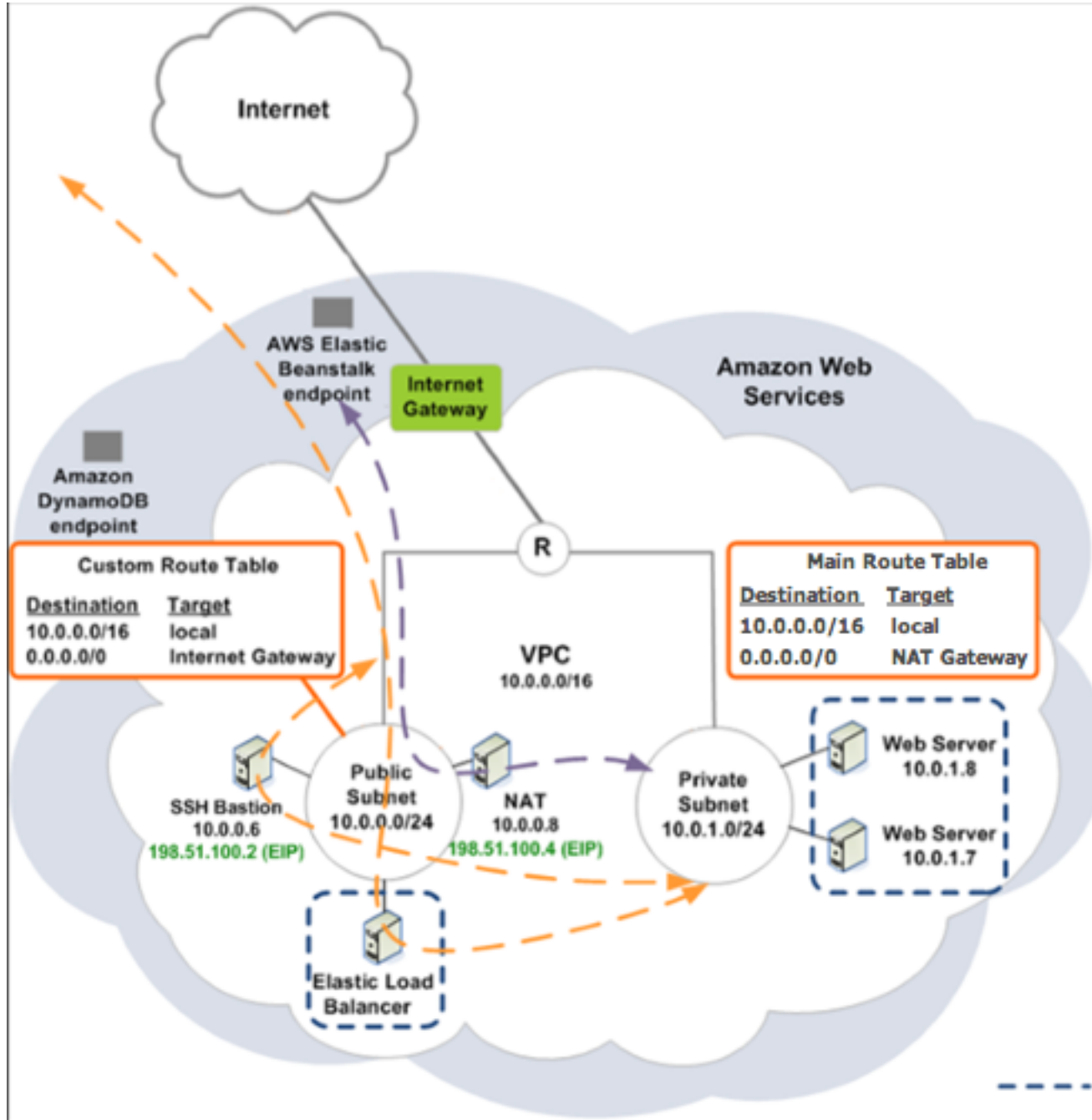
- *Include application dependencies* – If your application has dependencies such as language runtime packages, it might try to download and install them from the internet during environment creation and fail. To avoid this failure, include all dependency packages in your application's source bundle.
- *Use a current platform version* – Be sure that your environment uses a platform version that was released on February 24, 2020 or later. Specifically, use a platform version that was released in or after one of these two updates: [Linux Update 2020-02-28](#), [Windows Update 2020-02-24](#).

Note

The reason for needing an updated platform version is that older versions had an issue that would prevent DNS entries created by the **Enable DNS name** option from working properly for Amazon SQS.

Example: Launching an Elastic Beanstalk application in a VPC with bastion hosts

If your Amazon EC2 instances are located inside a private subnet, you will not be able to connect to them remotely. To connect to your instances, you can set up bastion servers in the public subnet to act as proxies. For example, you can set up SSH port forwarders or RDP gateways in the public subnet to proxy the traffic going to your database servers from your own network. This section provides an example of how to create a VPC with a private and public subnet. The instances are located inside the private subnet, and the bastion host, NAT gateway, and load balancer are located inside the public subnet. Your infrastructure will look similar to the following diagram:



To deploy an Elastic Beanstalk application inside a VPC using a bastion host, complete the steps described in the following subsections.

Steps

- [Create a VPC with a public and private subnet \(p. 839\)](#)
- [Create and configure the bastion host security group \(p. 839\)](#)
- [Update the instance security group \(p. 841\)](#)
- [Create a bastion host \(p. 841\)](#)

Create a VPC with a public and private subnet

Complete all of the procedures in [Public/private VPC \(p. 836\)](#). When deploying the application, you must specify an Amazon EC2 key pair for the instances so you can connect to them remotely. For more information about how to specify the instance key pair, see [Your Elastic Beanstalk environment's Amazon EC2 instances \(p. 451\)](#).

Create and configure the bastion host security group

Create a security group for the bastion host, and add rules that allow inbound SSH traffic from the Internet, and outbound SSH traffic to the private subnet that contains the Amazon EC2 instances.

To create the bastion host security group

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Security Groups**.
3. Choose **Create Security Group**.
4. In the **Create Security Group** dialog box, enter the following and choose **Yes, Create**.

Name tag (Optional)

Enter a name tag for the security group.

Group name

Enter the name of the security group.

Description

Enter a description for the security group.

VPC

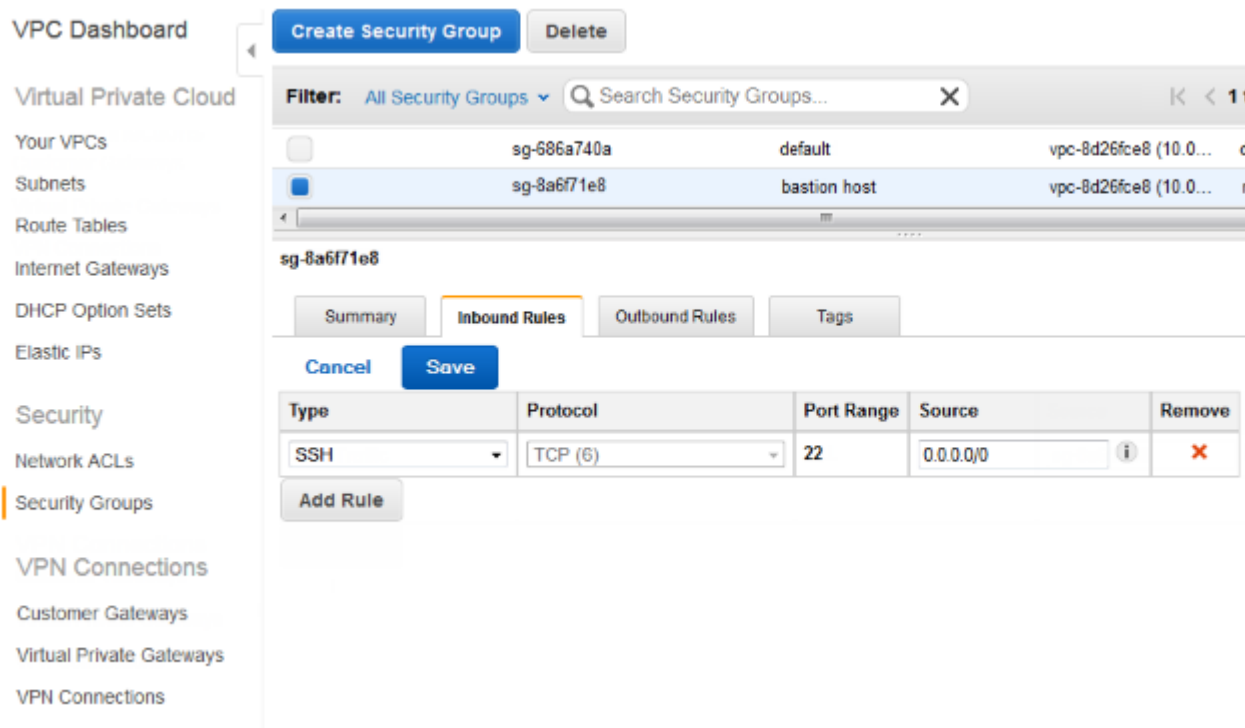
Select your VPC.

The security group is created and appears on the **Security Groups** page. Notice that it has an ID (e.g., `sg-xxxxxxxx`). You might have to turn on the **Group ID** column by clicking **Show/Hide** in the top right corner of the page.

To configure the bastion host security group

1. In the list of security groups, select the check box for the security group you just created for your bastion host.
2. On the **Inbound Rules** tab, choose **Edit**.
3. If needed, choose **Add another rule**.
4. If your bastion host is a Linux instance, under **Type**, select **SSH**.

If your bastion host is a Windows instance, under **Type**, select **RDP**.
5. Enter the desired source CIDR range in the **Source** field and choose **Save**.



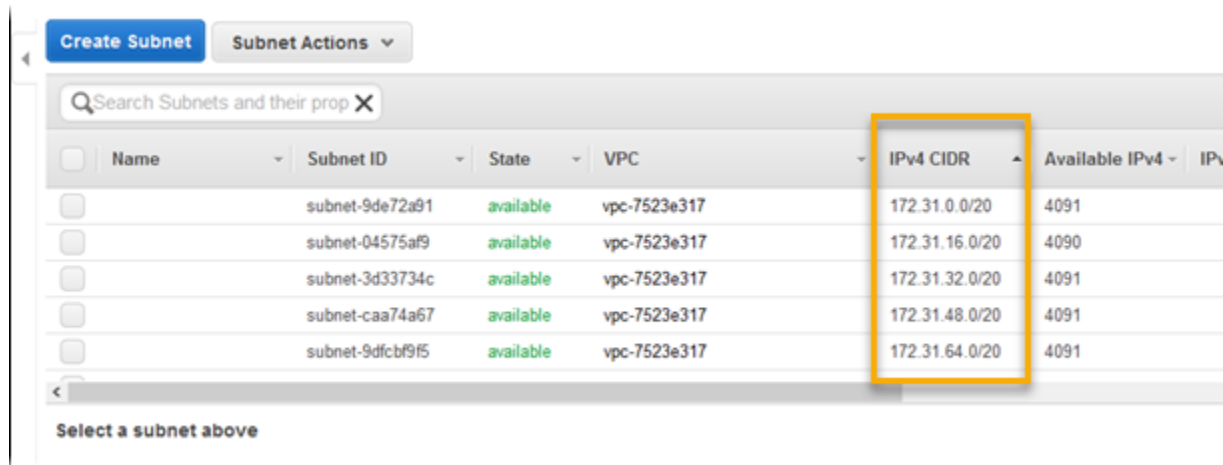
6. On the **Outbound Rules** tab, choose **Edit**.
7. If needed, choose **Add another rule**.
8. Under **Type**, select the type that you specified for the inbound rule.
9. In the **Source** field, enter the CIDR range of the subnet of the hosts in the VPC's private subnet.

To find it:

- a. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
- b. In the navigation pane, choose **Subnets**.
- c. Note the value under **IPv4 CIDR** for each **Availability Zone** in which you have hosts that you want the bastion host to bridge to.

Note

If you have hosts in multiple availability zones, create an outbound rule for each one of these availability zones.



10. Choose **Save**.

Update the instance security group

By default, the security group you created for your instances does not allow incoming traffic. While Elastic Beanstalk will modify the default group for the instances to allow SSH traffic, you must modify your custom instance security group to allow RDP traffic if your instances are Windows instances.

To update the instance security group for RDP

1. In the list of security groups, select the check box for the instance security group.
2. On the **Inbound** tab, choose **Edit**.
3. If needed, choose **Add another rule**.
4. Enter the following values, and choose **Save**.

Type

RDP

Protocol

TCP

Port Range

3389

Source

Enter the ID of the bastion host security group (e.g., sg-8a6f71e8) and choose **Save**.

Create a bastion host

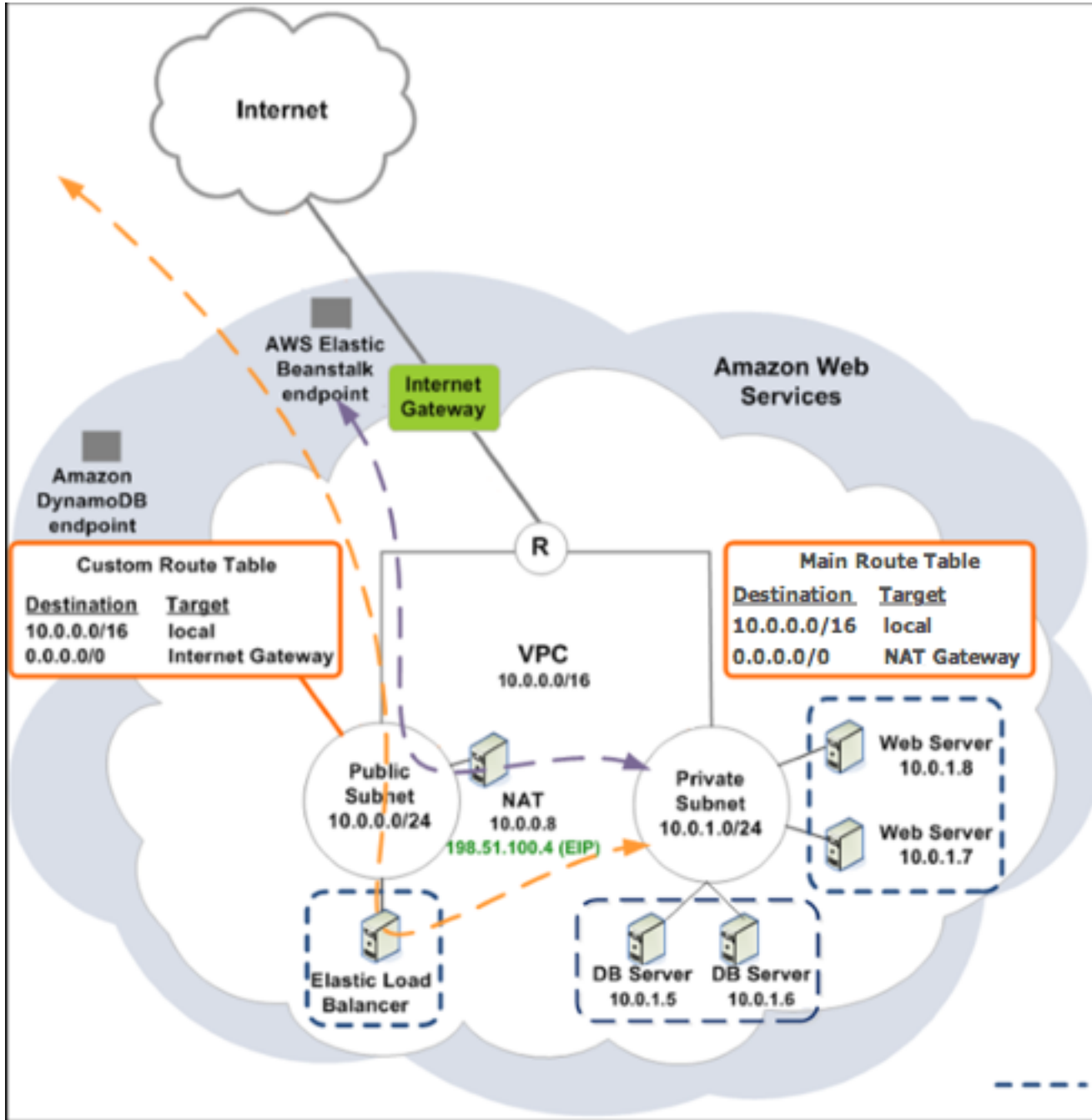
To create a bastion host, you launch an Amazon EC2 instance in your public subnet that will act as the bastion host.

For more information about setting up a bastion host for Windows instances in the private subnet, see [Controlling Network Access to EC2 Instances Using a Bastion Server](#) .

For more information about setting up a bastion host for Linux instances in the private subnet, see [Securely Connect to Linux Instances Running in a Private Amazon VPC](#) .

Example: Launching an Elastic Beanstalk in a VPC with Amazon RDS

This section walks you through the tasks to deploy an Elastic Beanstalk application with Amazon RDS in a VPC using a NAT gateway. Your infrastructure will look similar to the following diagram.



Note

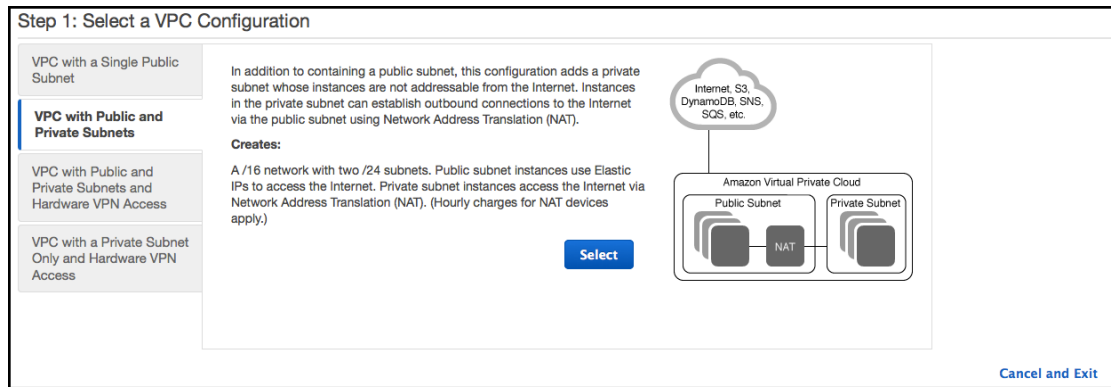
If you haven't used a DB instance with your application before, try [adding one to a test environment \(p. 506\)](#), and [connecting to an external DB instance \(p. 820\)](#) before adding a VPC configuration to the mix.

Create a VPC with a public and private subnet

You can use the [Amazon VPC console](#) to create a VPC.

To create a VPC

1. Sign in to the [Amazon VPC console](#).
2. In the navigation pane, choose **VPC Dashboard**. Then choose **Create VPC**.
3. Choose **VPC with Public and Private Subnets**, and then choose **Select**.



4. Your Elastic Load Balancing load balancer and your Amazon EC2 instances must be in the same Availability Zone so they can communicate with each other. Choose the same Availability Zone from each **Availability Zone** list.

The screenshot shows the 'Step 2: VPC with Public and Private Subnets' configuration screen. It includes fields for:

- IPv4 CIDR block: 10.0.0.0/16 (65531 IP addresses available)
- IPv6 CIDR block: No IPv6 CIDR Block, Amazon provided IPv6 CIDR block
- VPC name: [text input]
- Public subnet's IPv4 CIDR: 10.0.0.0/24 (251 IP addresses available)
- Availability Zone: No Preference (dropdown menu, highlighted with a red box)
- Public subnet name: Public subnet
- Private subnet's IPv4 CIDR: 10.0.1.0/24 (251 IP addresses available)
- Availability Zone: No Preference (dropdown menu, highlighted with a red box)
- Private subnet name: Private subnet

 Below these fields, there is a section for 'Specify the details of your NAT gateway (NAT gateway rates apply)' with an 'Elastic IP Allocation ID' field. At the bottom, there are checkboxes for 'Enable DNS hostnames' (checked), 'Hardware tenancy' (Default), and 'Enable ClassicLink'. 'Cancel and Exit', 'Back', and 'Create VPC' buttons are at the bottom right.

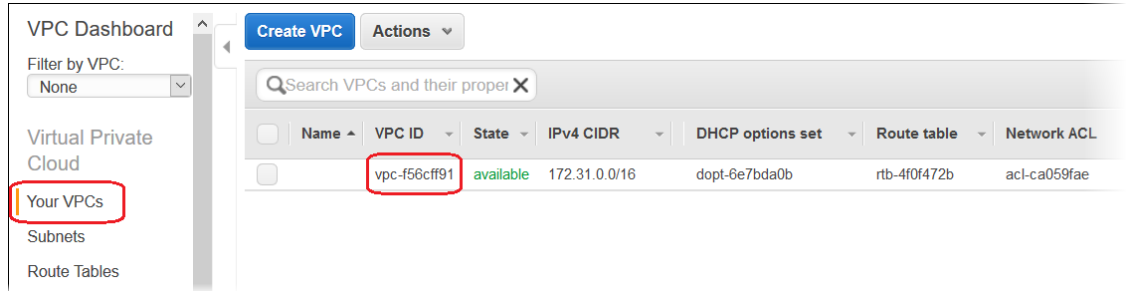
5. Choose an Elastic IP address for your NAT gateway.
6. Choose **Create VPC**.

The wizard begins to create your VPC, subnets, and internet gateway. It also updates the main route table and creates a custom route table. Finally, the wizard creates a NAT gateway in the public subnet.

Note

You can choose to launch a NAT instance in the public subnet instead of a NAT gateway. For more information, see [Scenario 2: VPC with Public and Private Subnets \(NAT\)](#) in the *Amazon VPC User Guide*.

7. After the VPC is successfully created, you get a VPC ID. You need this value for the next step. To view your VPC ID, choose **Your VPCs** in the left pane of the [Amazon VPC console](#).

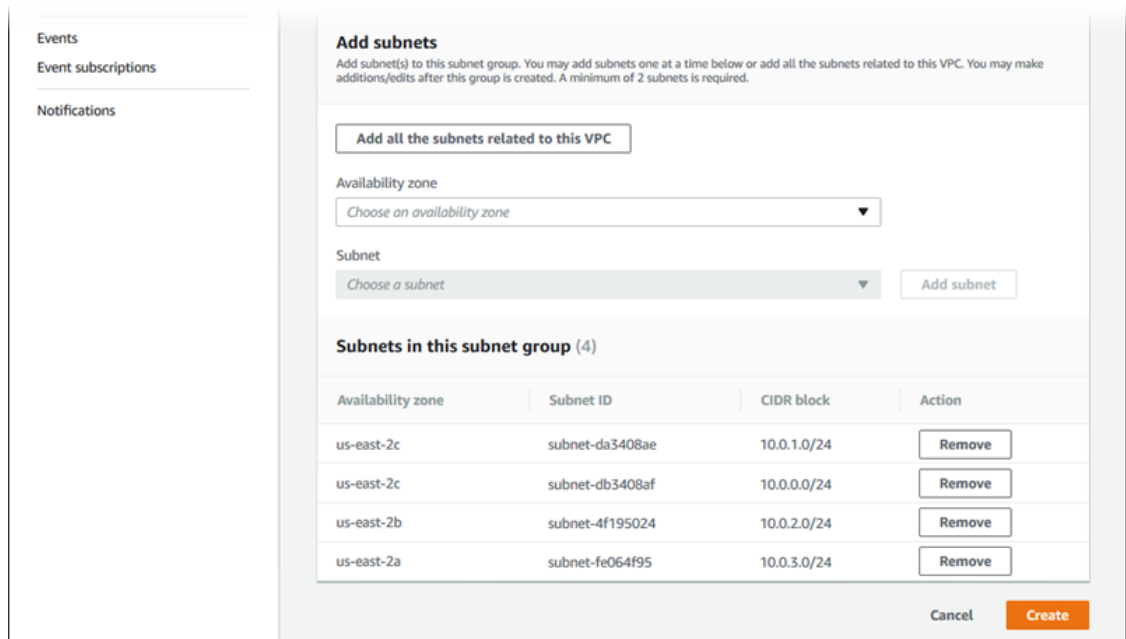


Create a DB subnet group

A DB subnet group for a VPC is a collection of subnets (typically private) that you can designate for your backend RDS DB instances. Each DB subnet group should have at least one subnet for every Availability Zone in a given AWS Region. To learn more, see [Creating a Subnet in Your VPC](#).

Create a DB subnet group

1. Open the [Amazon RDS console](#).
2. In the navigation pane, choose **Subnet groups**.
3. Choose **Create DB Subnet Group**.
4. Choose **Name**, and then type the name of your DB subnet group.
5. Choose **Description**, and then describe your DB subnet group.
6. For **VPC**, choose the ID of the VPC that you created.
7. In **Add subnets**, choose **Add all the subnets related to this VPC**.



8. When you are finished, choose **Create**.

Your new DB subnet group appears in the Subnet groups list of the Amazon RDS console. You can choose it to see details, such as all of the subnets associated with this group, in the details pane at the bottom of the page.

Deploy to Elastic Beanstalk

After you set up your VPC, you can create your environment inside it and deploy your application to Elastic Beanstalk. You can do this using the Elastic Beanstalk console, or you can use the AWS toolkits, AWS CLI, EB CLI, or Elastic Beanstalk API. If you use the Elastic Beanstalk console, you just need to upload your `.war` or `.zip` file and select the VPC settings inside the wizard. Elastic Beanstalk then creates your environment inside your VPC and deploys your application. Alternatively, you can use the AWS toolkits, AWS CLI, EB CLI, or Elastic Beanstalk API to deploy your application. To do this, you need to define your VPC option settings in a configuration file and deploy this file with your source bundle. This topic provides instructions for both methods.

Deploying with the Elastic Beanstalk console

The Elastic Beanstalk console walks you through creating your new environment inside your VPC. You need to provide a `.war` file (for Java applications) or a `.zip` file (for all other applications). On the **VPC Configuration** page of the Elastic Beanstalk environment wizard, you must make the following selections:

VPC

Select your VPC.

VPC security group

Select the instance security group you created above.

ELB visibility

Select `External` if your load balancer should be publicly available, or select `Internal` if the load balancer should be available only within your VPC.

Select the subnets for your load balancer and EC2 instances. Be sure you select the public subnet for the load balancer, and the private subnet for your Amazon EC2 instances. By default, the VPC creation wizard creates the public subnet in `10.0.0.0/24` and the private subnet in `10.0.1.0/24`.

You can view your subnet IDs by choosing **Subnets** in the [Amazon VPC console](#).

The screenshot shows the Amazon VPC console interface. On the left, the 'Subnets' link is highlighted in the navigation menu. The main area displays a table of subnets:

Name	Subnet ID	State	VPC	IPv4 CIDR	Available IPv4	Availability Zone
	subnet-3ba3c75e	available	vpc-f56cff91	172.31.64.0/20	4091	us-east-1a
<input checked="" type="checkbox"/>	subnet-ec18feb4	available	vpc-f56cff91	172.31.16.0/20	4089	us-east-1d

Below the table, the details for the selected subnet `subnet-ec18feb4` are shown:

- Subnet ID:** subnet-ec18feb4
- IPv4 CIDR:** 172.31.16.0/20
- IPv6 CIDR:**
- State:** available
- VPC:** vpc-f56cff91
- Available IPs:** 4089
- Availability Zone:** us-east-1d
- Route table:** rtb-4f0f472b
- Network ACL:** acl-ca059fae
- Default subnet:** yes
- Auto-assign Public IP:** yes
- Auto-assign IPv6 address:** no

Deploying with the AWS toolkits, EB CLI, AWS CLI, or API

When deploying your application to Elastic Beanstalk using the AWS toolkits, EB CLI, AWS CLI, or API, you can specify your VPC option settings in a file and deploy it with your source bundle. See [Advanced environment customization with configuration files \(.ebextensions\)](#) (p. 600) for more information.

When you update the option settings, you need to specify at least the following:

- **VPCId**–Contains the ID of the VPC.
- **Subnets**–Contains the ID of the Auto Scaling group subnet. In this example, this is the ID of the private subnet.
- **ELBSubnets**–Contains the ID of the subnet for the load balancer. In this example, this is the ID of the public subnet.
- **SecurityGroups**–Contains the ID of the security groups.
- **DBSubnets**–Contains the ID of the DB subnets.

Note

When using DB subnets, you need to create additional subnets in your VPC to cover all the Availability Zones in the AWS Region.

Optionally, you can also specify the following information:

- **ELBScheme** – Specify `internal` to create an internal load balancer inside your VPC so that your Elastic Beanstalk application can't be accessed from outside your VPC.

The following is an example of the option settings you could use when deploying your Elastic Beanstalk application inside a VPC. For more information about VPC option settings (including examples for how to specify them, default values, and valid values), see the `aws:ec2:vpc` namespace table in [Configuration options \(p. 536\)](#).

```
option_settings:
- namespace: aws:autoscaling:launchconfiguration
  option_name: EC2KeyName
  value: ec2keypair

- namespace: aws:ec2:vpc
  option_name: VPCId
  value: vpc-170647c

- namespace: aws:ec2:vpc
  option_name: Subnets
  value: subnet-4f195024

- namespace: aws:ec2:vpc
  option_name: ELBSubnets
  value: subnet-fe064f95

- namespace: aws:ec2:vpc
  option_name: DBSubnets
  value: subnet-fg148g78

- namespace: aws:autoscaling:launchconfiguration
  option_name: InstanceType
  value: m1.small

- namespace: aws:autoscaling:launchconfiguration
  option_name: SecurityGroups
  value: sg-7f1ef110
```

Note

When using DB subnets, be sure you have subnets in your VPC to cover all the Availability Zones in the AWS Region.

Using Elastic Beanstalk with VPC endpoints

A *VPC endpoint* enables you to privately connect your VPC to supported AWS services and VPC endpoint services powered by AWS PrivateLink, without requiring an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection.

Instances in your VPC don't require public IP addresses to communicate with resources in the service. Traffic between your VPC and the other service doesn't leave the Amazon network. For complete information about VPC endpoints, see [VPC Endpoints](#) in the *Amazon VPC User Guide*.

AWS Elastic Beanstalk supports AWS PrivateLink, which provides private connectivity to the Elastic Beanstalk service and eliminates exposure of traffic to the public internet. To enable your application to send requests to Elastic Beanstalk using AWS PrivateLink, you configure a type of VPC endpoint known as an *interface VPC endpoint* (interface endpoint). For more information, see [Interface VPC Endpoints \(AWS PrivateLink\)](#) in the *Amazon VPC User Guide*.

Note

Elastic Beanstalk supports AWS PrivateLink and interface VPC endpoints in a limited number of AWS Regions. We're working to extend support to more AWS Regions in the near future.

Setting up a VPC endpoint for Elastic Beanstalk

To create the interface VPC endpoint for the Elastic Beanstalk service in your VPC, follow the [Creating an Interface Endpoint](#) procedure. For **Service Name**, choose `com.amazonaws.region.elasticbeanstalk`.

If your VPC is configured with public internet access, your application can still access Elastic Beanstalk over the internet using the `elasticbeanstalk.region.amazonaws.com` public endpoint. You can prevent this by ensuring that **Enable DNS name** is enabled during endpoint creation (true by default). This adds a DNS entry in your VPC that maps the public service endpoint to the interface VPC endpoint.

Setting up a VPC endpoint for enhanced health

If you enabled [enhanced health reporting](#) (p. 691) for your environment, you can configure enhanced health information to be sent over AWS PrivateLink too. Enhanced health information is sent by the `healthd` daemon, an Elastic Beanstalk component on your environment instances, to a separate Elastic Beanstalk enhanced health service. To create an interface VPC endpoint for this service in your VPC, follow the [Creating an Interface Endpoint](#) procedure. For **Service Name**, choose `com.amazonaws.region.elasticbeanstalk-health`.

Important

The `healthd` daemon sends enhanced health information to the public endpoint, `elasticbeanstalk-health.region.amazonaws.com`. If your VPC is configured with public internet access, and **Enable DNS name** is disabled for the VPC endpoint, enhanced health information travels through the public internet. This is probably not your intention when you set up an enhanced health VPC endpoint. Ensure that **Enable DNS name** is enabled (true by default).

Using VPC endpoints in a private VPC

A private VPC, or a private subnet in a VPC, has no public internet access. You might want to run your Elastic Beanstalk environment in a [private VPC](#) (p. 836) and configure interface VPC endpoints for enhanced security. In this case, be aware that your environment might try to connect to the internet for other reasons in addition to contacting the Elastic Beanstalk service. To learn more about running an environment in a private VPC, see [the section called "Running an Elastic Beanstalk environment in a private VPC"](#) (p. 836).

Using endpoint policies to control access with VPC endpoints

By default, a VPC endpoint allows full access to the service with which it's associated. When you create or modify an endpoint, you can attach an *endpoint policy* to it.

An endpoint policy is an AWS Identity and Access Management (IAM) resource policy that controls access from the endpoint to the specified service. The endpoint policy is specific to the endpoint. It's separate from any user or instance IAM policies that your environment might have and doesn't override or replace them. For details about authoring and using VPC endpoint policies, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.

The following example denies all users the permission to terminate an environment through the VPC endpoint, and allows full access to all other actions.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "elasticbeanstalk:TerminateEnvironment",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": "*"
    }
  ]
}
```

Note

At this time, only the main Elastic Beanstalk service supports attaching an endpoint policy to its VPC endpoint. The enhanced health service doesn't support endpoint policies.

Configuring your development machine for use with Elastic Beanstalk

This page shows you how to set up your local machine for development of an AWS Elastic Beanstalk application. It covers folder structure, source control, and CLI tools.

Topics

- [Creating a project folder \(p. 849\)](#)
- [Setting up source control \(p. 849\)](#)
- [Configuring a remote repository \(p. 850\)](#)
- [Installing the EB CLI \(p. 850\)](#)
- [Installing the AWS CLI \(p. 850\)](#)

Creating a project folder

Create a folder for your project. You can store the folder anywhere on your local disk as long as you have permission to read from and write to it. Creating a folder in your user folder is acceptable. If you plan on working on multiple applications, create your project folders inside another folder named something like `workspace` or `projects` to keep everything organized:

```
workspace/  
|-- my-first-app  
`-- my-second-app
```

The contents of your project folder will vary depending on the web container or framework that your application uses.

Note

Avoid folders and paths with single-quote (') or double-quote (") characters in the folder name or any path element. Some Elastic Beanstalk commands fail when run within a folder with either character in the name.

Setting up source control

Set up source control to protect yourself from accidentally deleting files or code in your project folder, and for a way to revert changes that break your project.

If you don't have a source control system, consider Git, a free and easy-to-use option, and it integrates well with the Elastic Beanstalk Command Line Interface (CLI). Visit the [Git homepage](#) to install Git.

Follow the instructions on the Git website to install and configure Git, and then run `git init` in your project folder to set up a local repository:

```
~/workspace/my-first-app$ git init
```



```
Initialized empty Git repository in /home/local/username/workspace/my-first-app/.git/
```

As you add content to your project folder and update content, commit the changes to your Git repository:

```
~/workspace/my-first-app$ git add default.jsp  
~/workspace/my-first-app$ git commit -m "add default JSP"
```

Every time you commit, you create a snapshot of your project that you can restore later if anything goes wrong. For much more information on Git commands and workflows, see the [Git documentation](#).

Configuring a remote repository

What if your hard drive crashes, or you want to work on your project on a different computer? To back up your source code online and access it from any computer, configure a remote repository to which you can push your commits.

AWS CodeCommit lets you create a private repository in the AWS cloud. CodeCommit is free in the [AWS free tier](#) for up to five AWS Identity and Access Management (IAM) users in your account. For pricing details, see [AWS CodeCommit Pricing](#).

Visit the [AWS CodeCommit User Guide](#) for instructions on getting set up.

GitHub is another popular option for storing your project code online. It lets you create a public online repository for free and also supports private repositories for a monthly charge. Sign up for GitHub at [github.com](#).

After you've created a remote repository for your project, attach it to your local repository with `git remote add`:

```
~/workspace/my-first-app$ git remote add origin ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/my-repo
```

Installing the EB CLI

Use the [EB CLI \(p. 852\)](#) to manage your Elastic Beanstalk environments and monitor health from the command line. See [Install the EB CLI \(p. 853\)](#) for installation instructions.

By default, the EB CLI packages everything in your project folder and uploads it to Elastic Beanstalk as a source bundle. When you use Git and the EB CLI together, you can prevent built class files from being committed to source with `.gitignore` and prevent source files from being deployed with `.ebignore`.

You can also [configure the EB CLI to deploy a build artifact \(p. 863\)](#) (a WAR or ZIP file) instead of the contents of your project folder.

Installing the AWS CLI

The AWS Command Line Interface (AWS CLI) is a unified client for AWS services that provides commands for all public API operations. These commands are lower level than those provided by the EB CLI, so it often takes more commands to do an operation with the AWS CLI. On the other hand, the AWS CLI allows you to work with any application or environment running in your account without setting

up a repository on your local machine. Use the AWS CLI to create scripts that simplify or automate operational tasks.

For more information about supported services and to download the AWS Command Line Interface, see [AWS Command Line Interface](#).

Using the Elastic Beanstalk command line interface (EB CLI)

The EB CLI is a command line interface for AWS Elastic Beanstalk that provides interactive commands that simplify creating, updating and monitoring environments from a local repository. Use the EB CLI as part of your everyday development and testing cycle as an alternative to the Elastic Beanstalk console.

Note

The current version of the EB CLI has a different base set of commands than versions prior to version 3.0. If you are using an older version, see [Migrating to EB CLI 3 and CodeCommit \(p. 937\)](#) for migration information.

After you [install the EB CLI \(p. 853\)](#) and configure a project directory, you can create environments with a single command:

```
~/my-app$ eb create my-env
```

The source code for the EB CLI is an open-source project. It resides in the [aws/aws-elastic-beanstalk-cli](#) GitHub repository. You can participate by reporting issues, making suggestions, and submitting pull requests. We value your contributions! For an environment where you only intend to use the EB CLI as is, we recommend that you install it using one of the EB CLI setup scripts, as detailed in [the section called "Install the EB CLI using setup scripts" \(p. 853\)](#).

Previously, Elastic Beanstalk supported a separate CLI that provided direct access to API operations called the [Elastic Beanstalk API CLI \(p. 937\)](#). This has been replaced with the [AWS CLI \(p. 850\)](#), which provides the same functionality but for all AWS services' APIs.

With the AWS CLI you have direct access to the Elastic Beanstalk API. The AWS CLI is great for scripting, but is not as easy to use from the command line because of the number of commands that you need to run and the number of parameters on each command. For example, creating an environment requires a series of commands:

```
~$ aws elasticbeanstalk check-dns-availability --cname-prefix my-cname
~$ aws elasticbeanstalk create-application-version --application-name my-application --
version-label v1 --source-bundle S3Bucket=my-bucket,S3Key=php-proxy-sample.zip
~$ aws elasticbeanstalk create-environment --cname-prefix my-cname --application-name my-
app --version-label v1 --environment-name my-env --solution-stack-name "64bit Amazon Linux
2015.03 v2.0.0 running Ruby 2.2 (Passenger Standalone)"
```

For information about installing the EB CLI, configuring a repository, and working with environments, see the following topics.

Topics

- [Install the EB CLI \(p. 853\)](#)
- [Configure the EB CLI \(p. 860\)](#)
- [Managing Elastic Beanstalk environments with the EB CLI \(p. 864\)](#)
- [Using the EB CLI with AWS CodeBuild \(p. 868\)](#)

- [Using the EB CLI with Git \(p. 870\)](#)
- [Using the EB CLI with AWS CodeCommit \(p. 871\)](#)
- [Using the EB CLI to monitor environment health \(p. 875\)](#)
- [Managing multiple Elastic Beanstalk environments as a group with the EB CLI \(p. 881\)](#)
- [Troubleshooting issues with the EB CLI \(p. 882\)](#)
- [EB CLI command reference \(p. 884\)](#)
- [EB CLI 2.6 \(retired\) \(p. 936\)](#)
- [Elastic Beanstalk API command line interface \(retired\) \(p. 937\)](#)

Install the EB CLI

The AWS Elastic Beanstalk Command Line Interface (EB CLI) is a command line client that you can use to create, configure, and manage Elastic Beanstalk environments. For more information about the EB CLI, see [EB CLI \(p. 852\)](#).

Topics

- [Install the EB CLI using setup scripts \(p. 853\)](#)
- [Manually install the EB CLI \(p. 853\)](#)

Install the EB CLI using setup scripts

The easiest and recommended way to install the EB CLI is to use the [EB CLI setup scripts](#) available on GitHub. Use the scripts to install the EB CLI on Linux, macOS, or Windows. The scripts install the EB CLI and its dependencies, including Python and `pip`. The scripts also create a virtual environment for the EB CLI. For installation instructions, see the [aws/aws-elastic-beanstalk-cli-setup](#) repository on GitHub.

Manually install the EB CLI

To install the EB CLI, we recommend using the [EB CLI setup scripts](#). If the setup scripts aren't compatible with your development environment, manually install the EB CLI.

The primary distribution method for the EB CLI on Linux, macOS, and Windows is `pip`. This is a package manager for Python that provides an easy way to install, upgrade, and remove Python packages and their dependencies. For macOS, you can also get the latest version of the EB CLI with `Homebrew`.

Compatibility notes

The EB CLI is developed in Python and requires Python version 2.7, 3.4, or later.

Note

Amazon Linux, starting with version 2015.03, comes with Python 2.7 and `pip`.

We recommend using the [EB CLI setup scripts](#) to install the EB CLI and its dependencies. If you manually install the EB CLI, it can be difficult to manage dependency conflicts in your development environment.

The EB CLI and the [AWS Command Line Interface \(AWS CLI\)](#) share a dependency on the `botocore` Python package. Due to a breaking change in `botocore`, different versions of these two CLI tools depend on different versions of `botocore`.

The latest versions of the two CLIs are compatible. If you need to use an earlier version, see the following table for a compatible version to use.

EB CLI version	Compatible AWS CLI version
3.14.5 or earlier	1.16.9 or earlier
3.14.6 or later	1.16.11 or later

Install the EB CLI

If you already have `pip` and a supported version of Python, use the following procedure to install the EB CLI.

If you don't have Python and `pip`, use the procedure for the operating system you're using.

- [Install Python, pip, and the EB CLI on Linux \(p. 855\)](#)
- [Install the EB CLI on macOS \(p. 857\)](#)
- [Install Python, pip, and the EB CLI on Windows \(p. 859\)](#)

To install the EB CLI

1. Run the following command.

```
$ pip install awsebcli --upgrade --user
```

The `--upgrade` option tells `pip` to upgrade any requirements that are already installed. The `--user` option tells `pip` to install the program to a subdirectory of your user directory to avoid modifying libraries that your operating system uses.

Note

If you encounter issues when you try to install the EB CLI with `pip`, you can [install the EB CLI in a virtual environment \(p. 860\)](#) to isolate the tool and its dependencies, or use a different version of Python than you normally do.

2. Add the path to the executable file to your `PATH` variable:
 - On Linux and macOS:

Linux – `~/local/bin`

macOS – `~/Library/Python/3.7/bin`

To modify your `PATH` variable (Linux, Unix, or macOS):

- a. Find your shell's profile script in your user folder. If you are not sure which shell you have, run `echo $SHELL`.

```
$ ls -a -
.  ..  .bash_logout  .bash_profile  .bashrc  Desktop  Documents  Downloads
```

- **Bash** – `.bash_profile`, `.profile`, or `.bash_login`.
 - **Zsh** – `.zshrc`
 - **Tcsh** – `.tcshrc`, `.cshrc` or `.login`.
- b. Add an export command to your profile script. The following example adds the path represented by `LOCAL_PATH` to the current `PATH` variable.

```
export PATH=LOCAL_PATH:$PATH
```

- c. Load the profile script described in the first step into your current session. The following example loads the profile script represented by *PROFILE_SCRIPT*.

```
$ source -/PROFILE_SCRIPT
```

- On Windows:

Python 3.7 – %USERPROFILE%\AppData\Roaming\Python\Python37\Scripts

Python earlier versions – %USERPROFILE%\AppData\Roaming\Python\Scripts

To modify your PATH variable (Windows):

- a. Press the Windows key, and then enter **environment variables**.
- b. Choose **Edit environment variables for your account**.
- c. Choose **PATH**, and then choose **Edit**.
- d. Add paths to the **Variable value** field, separated by semicolons. For example: **C:
\item1\path;C:\item2\path**
- e. Choose **OK** twice to apply the new settings.
- f. Close any running Command Prompt windows, and then reopen a Command Prompt window.

3. Verify that the EB CLI installed correctly by running **eb --version**.

```
$ eb --version  
EB CLI 3.14.8 (Python 3.7)
```

The EB CLI is updated regularly to add functionality that supports [the latest Elastic Beanstalk features](#). To update to the latest version of the EB CLI, run the installation command again.

```
$ pip install awsebcli --upgrade --user
```

If you need to uninstall the EB CLI, use `pip uninstall`.

```
$ pip uninstall awsebcli
```

Install Python, pip, and the EB CLI on Linux

The EB CLI requires Python 2.7, 3.4, or later. If your distribution didn't come with Python, or came with an earlier version, install Python before installing `pip` and the EB CLI.

To install Python 3.7 on Linux

1. Determine whether Python is already installed.

```
$ python --version
```

Note

If your Linux distribution came with Python, you might need to install the Python developer package to get the headers and libraries required to compile extensions and install the EB CLI. Use your package manager to install the developer package (typically named `python-dev` or `python-devel`).

2. If Python 2.7 or later isn't installed, install Python 3.7 using your distribution's package manager. The command and package name vary:

- On Debian derivatives, such as Ubuntu, use `APT`.

```
$ sudo apt-get install python3.7
```

- On Red Hat and derivatives, use `yum`.

```
$ sudo yum install python37
```

- On SUSE and derivatives, use `zypper`.

```
$ sudo zypper install python3-3.7
```

3. To verify that Python installed correctly, open a terminal or shell and run the following command.

```
$ python3 --version  
Python 3.7.3
```

Install `pip` by using the script provided by the Python Packaging Authority, and then install the EB CLI.

To install `pip` and the EB CLI

1. Download the installation script from [pypa.io](https://bootstrap.pypa.io).

```
$ curl -O https://bootstrap.pypa.io/get-pip.py
```

The script downloads and installs the latest version of `pip` and another required package named `setuptools`.

2. Run the script with Python.

```
$ python3 get-pip.py --user  
Collecting pip  
  Downloading pip-8.1.2-py2.py3-none-any.whl (1.2MB)  
Collecting setuptools  
  Downloading setuptools-26.1.1-py2.py3-none-any.whl (464kB)  
Collecting wheel  
  Downloading wheel-0.29.0-py2.py3-none-any.whl (66kB)  
Installing collected packages: pip, setuptools, wheel  
Successfully installed pip setuptools wheel
```

Invoking Python version 3 directly by using the `python3` command instead of `python` ensures that `pip` is installed in the proper location, even if an earlier version of Python is present on your system.

3. Add the executable path, `~/local/bin`, to your `PATH` variable.

To modify your `PATH` variable (Linux, Unix, or macOS):

- a. Find your shell's profile script in your user folder. If you are not sure which shell you have, run `echo $SHELL`.

```
$ ls -a ~  
.  ..  .bash_logout  .bash_profile  .bashrc  Desktop  Documents  Downloads
```

- **Bash** – `.bash_profile`, `.profile`, or `.bash_login`.

- **Zsh** – `.zshrc`
 - **Tcsh** – `.tcshrc`, `.cshrc` or `.login`.
- b. Add an export command to your profile script. The following example adds the path represented by `LOCAL_PATH` to the current `PATH` variable.

```
export PATH=LOCAL_PATH:$PATH
```

- c. Load the profile script described in the first step into your current session. The following example loads the profile script represented by `PROFILE_SCRIPT`.

```
$ source -/PROFILE_SCRIPT
```

4. Verify that `pip` is installed correctly.

```
$ pip --version  
pip 8.1.2 from ~/.local/lib/python3.7/site-packages (python 3.7)
```

5. Use `pip` to install the EB CLI.

```
$ pip install awsebcli --upgrade --user
```

6. Verify that the EB CLI installed correctly.

```
$ eb --version  
EB CLI 3.14.8 (Python 3.7)
```

To upgrade to the latest version, run the installation command again.

```
$ pip install awsebcli --upgrade --user
```

Install the EB CLI on macOS

If you use the Homebrew package manager, you can install the EB CLI by using the `brew` command. You can also install Python and `pip`, and then use `pip` to install the EB CLI.

Install the EB CLI with homebrew

The latest version of the EB CLI is typically available from Homebrew a couple of days after it appears in `pip`.

To install the EB CLI with Homebrew

1. Ensure you have the latest version of Homebrew.

```
$ brew update
```

2. Run `brew install awsebcli`.

```
$ brew install awsebcli
```

3. Verify that the EB CLI is installed correctly.

```
$ eb --version
```



```
EB CLI 3.14.8 (Python 3.7)
```

Install Python, pip, and the EB CLI on macOS

You can install the latest version of Python and pip and then use them to install the EB CLI.

To install the EB CLI on macOS

1. Download and install Python from the [downloads page](#) of [Python.org](#). We use version 3.7 to demonstrate.

Note

The EB CLI requires Python 2 version 2.7, or Python 3 version in the range of 3.4 to 3.7.

2. Install pip with the script that the Python Packaging Authority provides.

```
$ curl -O https://bootstrap.pypa.io/get-pip.py
$ python3 get-pip.py --user
```

3. Use pip to install the EB CLI.

```
$ pip3 install awsebcli --upgrade --user
```

4. Add the executable path, `~/Library/Python/3.7/bin`, to your PATH variable.

To modify your PATH variable (Linux, Unix, or macOS):

- a. Find your shell's profile script in your user folder. If you are not sure which shell you have, run `echo $SHELL`.

```
$ ls -a ~
.  ..  .bash_logout  .bash_profile  .bashrc  Desktop  Documents  Downloads
```

- **Bash** – `.bash_profile`, `.profile`, or `.bash_login`.
- **Zsh** – `.zshrc`
- **Tcsh** – `.tcshrc`, `.cshrc` or `.login`.

- b. Add an export command to your profile script. The following example adds the path represented by `LOCAL_PATH` to the current PATH variable.

```
export PATH=LOCAL_PATH:$PATH
```

- c. Load the profile script described in the first step into your current session. The following example loads the profile script represented by `PROFILE_SCRIPT`.

```
$ source ~/PROFILE_SCRIPT
```

5. Verify that the EB CLI is installed correctly.

```
$ eb --version
EB CLI 3.14.8 (Python 3.7)
```

To upgrade to the latest version, run the installation command again.

```
$ pip3 install awsebcli --upgrade --user
```

Install Python, pip, and the EB CLI on Windows

The Python Software Foundation provides installers for Windows that include pip.

To install Python 3.7 and pip (Windows)

1. Download the Python 3.7 Windows x86-64 executable installer from the [downloads page](#) of [Python.org](#).
2. Run the installer.
3. Choose **Add Python 3.7 to PATH**.
4. Choose **Install Now**.

The installer installs Python in your user folder and adds its executable directories to your user path.

To install the AWS CLI with pip (Windows)

1. From the Start menu, open a Command Prompt window.
2. Verify that Python and pip are both installed correctly by using the following commands.

```
C:\Windows\System32> python --version
Python 3.7.3
C:\Windows\System32> pip --version
pip 9.0.1 from c:\users\myname\appdata\local\programs\python\python37\lib\site-packages
(python 3.7)
```

3. Install the EB CLI using pip.

```
C:\Windows\System32> pip install awsebcli --upgrade --user
```

4. Add the executable path, %USERPROFILE%\AppData\roaming\Python\Python37\scripts, to your PATH environment variable. The location might be different, depending on whether you install Python for one user or all users.

To modify your PATH variable (Windows):

- a. Press the Windows key, and then enter **environment variables**.
 - b. Choose **Edit environment variables for your account**.
 - c. Choose **PATH**, and then choose **Edit**.
 - d. Add paths to the **Variable value** field, separated by semicolons. For example: **C:\item1\path;C:\item2\path**
 - e. Choose **OK** twice to apply the new settings.
 - f. Close any running Command Prompt windows, and then reopen a Command Prompt window.
5. Restart a new command shell for the new PATH variable to take effect.
 6. Verify that the EB CLI is installed correctly.

```
C:\Windows\System32> eb --version
EB CLI 3.14.8 (Python 3.7)
```

To upgrade to the latest version, run the installation command again.

```
C:\Windows\System32> pip install awsebcli --upgrade --user
```

Install the EB CLI in a virtual environment

You can avoid version requirement conflicts with other `pip` packages by installing the EB CLI in a virtual environment.

To install the EB CLI in a virtual environment

1. Install `virtualenv` with `pip`.

```
$ pip install --user virtualenv
```

2. Create a virtual environment.

```
$ virtualenv ~/eb-ve
```

To use a Python executable other than the default, use the `-p` option.

```
$ virtualenv -p /usr/bin/python3.7 ~/eb-ve
```

3. Activate the virtual environment.

Linux, Unix, or macOS

```
$ source ~/eb-ve/bin/activate
```

Windows

```
$ %USERPROFILE%\eb-ve\Scripts\activate
```

4. Install the EB CLI.

```
(eb-ve)-$ pip install awsebcli --upgrade
```

5. Verify that the EB CLI is installed correctly.

```
$ eb --version  
EB CLI 3.14.8 (Python 3.7)
```

You can use the `deactivate` command to exit the virtual environment. Whenever you start a new session, run the activation command again.

To upgrade to the latest version, run the installation command again.

```
(eb-ve)-$ pip install awsebcli --upgrade
```

Configure the EB CLI

After [installing the EB CLI \(p. 853\)](#), you are ready to configure your project directory and the EB CLI by running `eb init`.

The following example shows the configuration steps when running `eb init` for the first time in a project folder named `eb`.

To initialize an EB CLI project

1. First, the EB CLI prompts you to select a region. Type the number that corresponds to the region that you want to use, and then press **Enter**.

```
~/eb $ eb init
Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
4) eu-west-1 : Europe (Ireland)
5) eu-central-1 : Europe (Frankfurt)
6) ap-south-1 : Asia Pacific (Mumbai)
7) ap-southeast-1 : Asia Pacific (Singapore)
...
(default is 3): 3
```

2. Next, provide your access key and secret key so that the EB CLI can manage resources for you. Access keys are created in the AWS Identity and Access Management console. If you don't have keys, see [How Do I Get Security Credentials?](#) in the *Amazon Web Services General Reference*.

```
You have not yet set up your credentials or your credentials are incorrect.
You must provide your credentials.
(aws-access-id): AKIAJOUAASEXAMPLE
(aws-secret-key): 5ZRIrtTM4ciIAvd4EXAMPLEDtm+PiPSzpoK
```

3. An application in Elastic Beanstalk is a resource that contains a set of application versions (source), environments, and saved configurations that are associated with a single web application. Each time you deploy your source code to Elastic Beanstalk using the EB CLI, a new application version is created and added to the list.

```
Select an application to use
1) [ Create new Application ]
(default is 1): 1
```

4. The default application name is the name of the folder in which you run **eb init**. Enter any name that describes your project.

```
Enter Application Name
(default is "eb"): eb
Application eb has been created.
```

5. Select a platform that matches the language or framework that your web application is developed in. If you haven't started developing an application yet, choose a platform that you're interested in. You will see how to launch a sample application shortly, and you can always change this setting later.

```
Select a platform.
1) Node.js
2) PHP
3) Python
4) Ruby
5) Tomcat
6) IIS
7) Docker
8) Multi-container Docker
9) GlassFish
10) Go
11) Java
(default is 1): 1
```

6. Choose **yes** to assign an SSH key pair to the instances in your Elastic Beanstalk environment. This allows you to connect directly to them for troubleshooting.

```
Do you want to set up SSH for your instances?  
(y/n): y
```

7. Choose an existing key pair or create a new one. To use **eb init** to create a new key pair, you must have **ssh-keygen** installed on your local machine and available from the command line. The EB CLI registers the new key pair with Amazon EC2 for you and stores the private key locally in a folder named `.ssh` in your user directory.

```
Select a keypair.  
1) [ Create new KeyPair ]  
(default is 1): 1
```

Your EB CLI installation is now configured and ready to use. See [Managing Elastic Beanstalk environments with the EB CLI \(p. 864\)](#) for instructions on creating and working with an Elastic Beanstalk environment.

Advanced Configuration

- [Ignoring files using .ebignore \(p. 862\)](#)
- [Using named profiles \(p. 862\)](#)
- [Deploying an artifact instead of the project folder \(p. 863\)](#)
- [Configuration settings and precedence \(p. 863\)](#)
- [Instance metadata \(p. 864\)](#)

Ignoring files using .ebignore

You can tell the EB CLI to ignore certain files in your project directory by adding the file `.ebignore` to the directory. This file works like a `.gitignore` file. When you deploy your project directory to Elastic Beanstalk and create a new application version, the EB CLI doesn't include files specified by `.ebignore` in the source bundle that it creates.

If `.ebignore` isn't present, but `.gitignore` is, the EB CLI ignores files specified in `.gitignore`. If `.ebignore` is present, the EB CLI doesn't read `.gitignore`.

When `.ebignore` is present, the EB CLI doesn't use git commands to create your source bundle. This means that EB CLI ignores files specified in `.ebignore`, and includes all other files. In particular, it includes uncommitted source files.

Note

In Windows, adding `.ebignore` causes the EB CLI to follow symbolic links and include the linked file when creating a source bundle. This is a known issue and will be fixed in a future update.

Using named profiles

If you store your credentials as a named profile in a `credentials` or `config` file, you can use the `--profile` (p. 935) option to explicitly specify a profile. For example, the following command creates a new application using the `user2` profile.

```
$ eb init --profile user2
```

You can also change the default profile by setting the `AWS_EB_PROFILE` environment variable. When this variable is set, the EB CLI reads credentials from the specified profile instead of `default` or `eb-cli`.

Linux, macOS, or Unix

```
$ export AWS_EB_PROFILE=user2
```

Windows

```
> set AWS_EB_PROFILE=user2
```

Deploying an artifact instead of the project folder

You can tell the EB CLI to deploy a ZIP file or WAR file that you generate as part of a separate build process by adding the following lines to `.elasticbeanstalk/config.yml` in your project folder.

```
deploy:
  artifact: path/to/buildartifact.zip
```

If you configure the EB CLI in your [Git repository \(p. 870\)](#), and you don't commit the artifact to source, use the `--staged` option to deploy the latest build.

```
~/eb$ eb deploy --staged
```

Configuration settings and precedence

The EB CLI uses a *provider chain* to look for AWS credentials in a number of different places, including system or user environment variables and local AWS configuration files.

The EB CLI looks for credentials and configuration settings in the following order:

1. **Command line options** – Specify a named profile by using `--profile` to override default settings.
2. **Environment variables** – `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
3. **The AWS credentials file** – Located at `~/.aws/credentials` on Linux and OS X systems, or at `C:\Users\USERNAME\.aws\credentials` on Windows systems. This file can contain multiple named profiles in addition to a default profile.
4. **The AWS CLI configuration file** – Located at `~/.aws/config` on Linux and OS X systems or `C:\Users\USERNAME\.aws\config` on Windows systems. This file can contain a default profile, [named profiles](#), and AWS CLI-specific configuration parameters for each.
5. **Legacy EB CLI configuration file** – Located at `~/.elasticbeanstalk/config` on Linux and OS X systems or `C:\Users\USERNAME\.elasticbeanstalk\config` on Windows systems.
6. **Instance profile credentials** – These credentials can be used on Amazon EC2 instances with an assigned instance role, and are delivered through the Amazon EC2 metadata service. The [instance profile \(p. 21\)](#) must have permission to use Elastic Beanstalk.

If the credentials file contains a named profile with the name "eb-cli", the EB CLI will prefer that profile over the default profile. If no profiles are found, or a profile is found but does not have permission to use Elastic Beanstalk, the EB CLI prompts you to enter keys.

Instance metadata

To use the EB CLI from an Amazon EC2 instance, create a role that has access to the resources needed and assign that role to the instance when it is launched. Launch the instance and install the EB CLI by using `pip`.

```
~$ sudo pip install awsebcli
```

`pip` comes preinstalled on Amazon Linux.

The EB CLI reads credentials from the instance metadata. For more information, see [Granting Applications that Run on Amazon EC2 Instances Access to AWS Resources](#) in *IAM User Guide*.

Managing Elastic Beanstalk environments with the EB CLI

After [installing the EB CLI \(p. 853\)](#) and [configuring your project directory \(p. 860\)](#), you are ready to create an Elastic Beanstalk environment using the EB CLI, deploy source and configuration updates, and pull logs and events.

Note

Creating environments with the EB CLI requires a [service role \(p. 20\)](#). You can create a service role by creating an environment in the Elastic Beanstalk console. If you don't have a service role, the EB CLI attempts to create one when you run `eb create`.

The EB CLI returns a zero (0) exit code for all successful commands, and a non-zero exit code when it encounters any error.

The following examples use an empty project folder named `eb` that was initialized with the EB CLI for use with a sample Docker application.

Basic Commands

- [Eb create \(p. 864\)](#)
- [Eb status \(p. 865\)](#)
- [Eb health \(p. 865\)](#)
- [Eb events \(p. 866\)](#)
- [Eb logs \(p. 866\)](#)
- [Eb open \(p. 866\)](#)
- [Eb deploy \(p. 867\)](#)
- [Eb config \(p. 867\)](#)
- [Eb terminate \(p. 868\)](#)

Eb create

To create your first environment, run [eb create \(p. 894\)](#) and follow the prompts. If your project directory has source code in it, the EB CLI will bundle it up and deploy it to your environment. Otherwise, a sample application will be used.

```
~/eb$ eb create
Enter Environment Name
(default is eb-dev): eb-dev
Enter DNS CNAME prefix
```

```
(default is eb-dev): eb-dev
WARNING: The current directory does not contain any source code. Elastic Beanstalk is
launching the sample application instead.
Environment details for: elasticBeanstalkExa-env
  Application name: elastic-beanstalk-example
  Region: us-west-2
  Deployed Version: Sample Application
  Environment ID: e-j3pmc8tscn
  Platform: 64bit Amazon Linux 2015.03 v1.4.3 running Docker 1.6.2
  Tier: WebServer-Standard
  CNAME: eb-dev.elasticbeanstalk.com
  Updated: 2015-06-27 01:02:24.813000+00:00
Printing Status:
INFO: createEnvironment is starting.
-- Events -- (safe to Ctrl+C) Use "eb abort" to cancel the command.
```

Your environment can take several minutes to become ready. Press **Ctrl+C** to return to the command line while the environment is created.

Eb status

Run **eb status** to see the current status of your environment. When the status is ready, the sample application is available at elasticbeanstalk.com and the environment is ready to be updated.

```
~/eb$ eb status
Environment details for: elasticBeanstalkExa-env
  Application name: elastic-beanstalk-example
  Region: us-west-2
  Deployed Version: Sample Application
  Environment ID: e-gbzqc3jcra
  Platform: 64bit Amazon Linux 2015.03 v1.4.3 running Docker 1.6.2
  Tier: WebServer-Standard
  CNAME: elasticbeanstalkexa-env.elasticbeanstalk.com
  Updated: 2015-06-30 01:47:45.589000+00:00
  Status: Ready
  Health: Green
```

Eb health

Use the **eb health** command to view [health information \(p. 691\)](#) about the instances in your environment and the state of your environment overall. Use the **--refresh** option to view health in an interactive view that updates every 10 seconds.

```
~/eb$ eb health
api                               Ok                               2016-09-15 18:39:04
WebServer                         Java 8
total      ok      warning  degraded  severe  info  pending  unknown
  3         3         0         0         0         0         0         0

instance-id      status  cause  health
Overall          Ok
i-0ef05ec54918bf567  Ok
i-001880c1187493460  Ok
i-04703409d90d7c353  Ok

instance-id      r/sec  %2xx  %3xx  %4xx  %5xx  p99  p90  p75
p50      p10
Overall          8.6    100.0  0.0   0.0   0.0   0.083*  0.065  0.053
0.040  0.019
i-0ef05ec54918bf567  2.9    29     0     0     0     0.069*  0.066  0.057
0.050  0.023
```



```

i-001880c1187493460 2.9 29 0 0 0 0.087* 0.069 0.056
0.050 0.034
i-04703409d90d7c353 2.8 28 0 0 0 0.051* 0.027 0.024
0.021 0.015

instance-id      type      az  running  load_1  load_5  user%  nice%
system% idle%  iowait%
i-0ef05ec54918bf567 t2.micro 1c  23 mins  0.19   0.05   3.0    0.0
0.3  96.7    0.0
i-001880c1187493460 t2.micro 1a  23 mins  0.0    0.0    3.2    0.0
0.3  96.5    0.0
i-04703409d90d7c353 t2.micro 1b  1 day   0.0    0.0    3.6    0.0
0.2  96.2    0.0

instance-id      status  id  version  ago
deployments
i-0ef05ec54918bf567 Deployed 28  app-bc1b-160915_181041 20 mins
i-001880c1187493460 Deployed 28  app-bc1b-160915_181041 20 mins
i-04703409d90d7c353 Deployed 28  app-bc1b-160915_181041 27 mins

```

Eb events

Use **eb events** to see a list of events output by Elastic Beanstalk.

```

~/eb$ eb events
2015-06-29 23:21:09 INFO createEnvironment is starting.
2015-06-29 23:21:10 INFO Using elasticbeanstalk-us-east-2-EXAMPLE as Amazon S3
storage bucket for environment data.
2015-06-29 23:21:23 INFO Created load balancer named: awseb-e-g-AWSEBLoa-EXAMPLE
2015-06-29 23:21:42 INFO Created security group named: awseb-e-gbzqc3jcra-stack-
AWSEBSecurityGroup-EXAMPLE
...

```

Eb logs

Use **eb logs** to pull logs from an instance in your environment. By default, **eb logs** pull logs from the first instance launched and displays them in standard output. You can specify an instance ID with the **--instance** option to get logs from a specific instance.

The **--all** option pulls logs from all instances and saves them to subdirectories under `.elasticbeanstalk/logs`.

```

~/eb$ eb logs --all
Retrieving logs...
Logs were saved to /home/local/ANT/mwunderl/ebcli/environments/test/.elasticbeanstalk/
logs/150630_201410
Updated symlink at /home/local/ANT/mwunderl/ebcli/environments/test/.elasticbeanstalk/logs/
latest

```

Eb open

To open your environment's website in a browser, use **eb open**:

```
~/eb$ eb open
```

In a windowed environment, your default browser will open in a new window. In a terminal environment, a command line browser (e.g. `w3m`) will be used if available.

Eb deploy

Once the environment is up and ready, you can update it using **eb deploy**.

This command works better with some source code to bundle up and deploy, so for this example we've created a `Dockerfile` in the project directory with the following content:

`~/eb/Dockerfile`

```
FROM ubuntu:12.04

RUN apt-get update
RUN apt-get install -y nginx zip curl

RUN echo "daemon off;" >> /etc/nginx/nginx.conf
RUN curl -o /usr/share/nginx/www/master.zip -L https://codeload.github.com/gabrielecirulli/2048/zip/master
RUN cd /usr/share/nginx/www/ && unzip master.zip && mv 2048-master/* . && rm -rf 2048-master master.zip

EXPOSE 80

CMD ["/usr/sbin/nginx", "-c", "/etc/nginx/nginx.conf"]
```

This `Dockerfile` deploys an image of Ubuntu 12.04 and installs the game 2048. Run **eb deploy** to upload the application to your environment:

```
~/eb$ eb deploy
Creating application version archive "app-150630_014338".
Uploading elastic-beanstalk-example/app-150630_014338.zip to S3. This may take a while.
Upload Complete.
INFO: Environment update is starting.
-- Events -- (safe to Ctrl+C) Use "eb abort" to cancel the command.
```

When you run **eb deploy**, the EB CLI bundles up the contents of your project directory and deploys it to your environment.

Note

If you have initialized a git repository in your project folder, the EB CLI will always deploy the latest commit, even if you have pending changes. Commit your changes prior to running **eb deploy** to deploy them to your environment.

Eb config

Take a look at the configuration options available for your running environment with the **eb config** command:

```
~/eb$ eb config
ApplicationName: elastic-beanstalk-example
DateUpdated: 2015-06-30 02:12:03+00:00
EnvironmentName: elasticBeanstalkExa-env
SolutionStackName: 64bit Amazon Linux 2015.03 v1.4.3 running Docker 1.6.2
settings:
  AWSEBAutoScalingScaleDownPolicy.aws:autoscaling:trigger:
    LowerBreachScaleIncrement: '-1'
  AWSEBAutoScalingScaleUpPolicy.aws:autoscaling:trigger:
    UpperBreachScaleIncrement: '1'
  AWSEBCloudwatchAlarmHigh.aws:autoscaling:trigger:
    UpperThreshold: '6000000'
...
```

This command populates a list of available configuration options in a text editor. Many of the options shown have a null value, these are not set by default but can be modified to update the resources in your environment. See [Configuration options \(p. 536\)](#) for more information about these options.

Eb terminate

If you are done using the environment for now, use **eb terminate** to terminate it.

```
~/eb$ eb terminate
The environment "eb-dev" and all associated instances will be terminated.
To confirm, type the environment name: eb-dev
INFO: terminateEnvironment is starting.
INFO: Deleted CloudWatch alarm named: awseb-e-jc8t3pmscn-stack-
AWSEBCloudwatchAlarmHigh-1XLMU7DNCBV6Y
INFO: Deleted CloudWatch alarm named: awseb-e-jc8t3pmscn-stack-
AWSEBCloudwatchAlarmLow-8IVI04W2SCXS
INFO: Deleted Auto Scaling group policy named: arn:aws:autoscaling:us-
east-2:123456789012:scalingPolicy:1753d43e-ae87-4df6-
a405-11d31f4c8f97:autoScalingGroupName/awseb-e-jc8t3pmscn-stack-
AWSEBAutoScalingGroup-90TTS2ZL4MXV:policyName/awseb-e-jc8t3pmscn-stack-
AWSEBAutoScalingScaleUpPolicy-A070H1BMUQAJ
INFO: Deleted Auto Scaling group policy named: arn:aws:autoscaling:us-
east-2:123456789012:scalingPolicy:1fd24ea4-3d6f-4373-
affc-4912012092ba:autoScalingGroupName/awseb-e-jc8t3pmscn-stack-
AWSEBAutoScalingGroup-90TTS2ZL4MXV:policyName/awseb-e-jc8t3pmscn-stack-
AWSEBAutoScalingScaleDownPolicy-LSWFUMZ46H1V
INFO: Waiting for EC2 instances to terminate. This may take a few minutes.
-- Events -- (safe to Ctrl+C)
```

For a full list of available EB CLI commands, check out the [EB CLI command reference \(p. 884\)](#).

Using the EB CLI with AWS CodeBuild

[AWS CodeBuild](#) compiles your source code, runs unit tests, and produces artifacts that are ready to deploy. You can use CodeBuild together with the EB CLI to automate building your application from its source code. Environment creation and each deployment thereafter start with a build step, and then deploy the resulting application.

Note

Some regions don't offer CodeBuild. The integration between Elastic Beanstalk and CodeBuild doesn't work in these regions.

For information about the AWS services offered in each region, see [Region Table](#).

Creating an application

To create an Elastic Beanstalk application that uses CodeBuild

1. Include a CodeBuild build specification file, `buildspec.yml`, in your application folder.
2. Add an `eb_codebuild_settings` entry with options specific to Elastic Beanstalk to the file.
3. Run **eb init (p. 907)** in the folder.

Elastic Beanstalk extends the [CodeBuild build specification file format](#) to include the following additional settings:

```
eb_codebuild_settings:
  CodeBuildServiceRole: role-name
```

```
ComputeType: size  
Image: image  
Timeout: minutes
```

CodeBuildServiceRole

The ARN or name of the AWS Identity and Access Management (IAM) service role that CodeBuild can use to interact with dependent AWS services on your behalf. This value is required. If you omit it, any subsequent **eb create** or **eb deploy** command fails.

To learn more about creating a service role for CodeBuild, see [Create a CodeBuild Service Role](#) in the *AWS CodeBuild User Guide*.

Note

You also need permissions to perform actions in CodeBuild itself. The Elastic Beanstalk `AWSElasticBeanstalkFullAccess` managed user policy includes all the required CodeBuild action permissions. If you're not using the managed policy, be sure to allow the following permissions in your user policy.

```
"codebuild:CreateProject",  
"codebuild:DeleteProject",  
"codebuild:BatchGetBuilds",  
"codebuild:StartBuild"
```

For details, see [Managing Elastic Beanstalk user policies \(p. 775\)](#).

ComputeType

The amount of resources used by the Docker container in the CodeBuild build environment. Valid values are `BUILD_GENERAL1_SMALL`, `BUILD_GENERAL1_MEDIUM`, and `BUILD_GENERAL1_LARGE`.

Image

The name of the Docker Hub or Amazon ECR image that CodeBuild uses for the build environment. This Docker image should contain all the tools and runtime libraries required to build your code, and should match your application's target platform. CodeBuild manages and maintains a set of images specifically meant to be used with Elastic Beanstalk. It is recommended that you use one of them. For details, see [Docker Images Provided by CodeBuild](#) in the *AWS CodeBuild User Guide*.

The `Image` value is optional. If you omit it, the **eb init** command attempts to choose an image that best matches your target platform. In addition, if you run **eb init** in interactive mode and it fails to choose an image for you, it prompts you to choose one. At the end of a successful initialization, **eb init** writes the chosen image into the `buildspec.yml` file.

Timeout

The duration, in minutes, that the CodeBuild build runs before timing out. This value is optional. For details about valid and default values, see [Create a Build Project in CodeBuild](#).

Note

This timeout controls the maximum duration for a CodeBuild run, and the EB CLI also respects it as part of its first step to create an application version. It's distinct from the value you can specify with the `--timeout` option of the **eb create** (p. 894) or **eb deploy** (p. 903) commands. The latter value controls the maximum duration that for EB CLI to wait for environment creation or update.

Building and deploying your application code

Whenever your application code needs to be deployed, the EB CLI uses CodeBuild to run a build, then deploys the resulting build artifacts to your environment. This happens when you create an Elastic

Beanstalk environment for your application using the **eb create** (p. 894) command, and each time you later deploy code changes to the environment using the **eb deploy** (p. 903) command.

If the CodeBuild step fails, environment creation or deployment doesn't start.

Using the EB CLI with Git

The EB CLI provides integration with Git. This section provides an overview of how to use Git with the EB CLI.

To install Git and initialize your Git repository

1. Download the most recent version of Git by visiting <http://git-scm.com>.
2. Initialize your Git repository by typing the following:

```
~/eb$ git init
```

EB CLI will now recognize that your application is set up with Git.

3. If you haven't already run **eb init**, do that now:

```
~/eb$ eb init
```

Associating Elastic Beanstalk environments with Git branches

You can associate a different environment with each branch of your code. When you checkout a branch, changes are deployed to the associated environment. For example, you can type the following to associate your production environment with your master branch, and a separate development environment with your development branch:

```
~/eb$ git checkout master
~/eb$ eb use prod
~/eb$ git checkout develop
~/eb$ eb use dev
```

Deploying changes

By default, the EB CLI deploys the latest commit in the current branch, using the commit ID and message as the application version label and description, respectively. If you want to deploy to your environment without committing, you can use the `--staged` option to deploy changes that have been added to the staging area.

To deploy changes without committing

1. Add new and changed files to the staging area:

```
~/eb$ git add .
```

2. Deploy the staged changes with **eb deploy**:

```
~/eb$ eb deploy --staged
```

If you have configured the EB CLI to [deploy an artifact \(p. 863\)](#), and you don't commit the artifact to your git repository, use the `--staged` option to deploy the latest build.

Using Git submodules

Some code projects benefit from having Git submodules — repositories within the top-level repository. When you deploy your code using **eb create** or **eb deploy**, the EB CLI can include submodules in the application version zip file and upload them with the rest of the code.

You can control the inclusion of submodules by using the `include_git_submodules` option in the `global` section of the EB CLI configuration file, `.elasticbeanstalk/config.yml` in your project folder.

To include submodules, set this option to `true`:

```
global:
  include_git_submodules: true
```

When the `include_git_submodules` option is missing or set to `false`, EB CLI does not include submodules in the uploaded zip file.

See [Git Tools - Submodules](#) for more details about Git submodules.

Default behavior

When you run **eb init** to configure your project, the EB CLI adds the `include_git_submodules` option and sets it to `true`. This ensures that any submodules you have in your project are included in your deployments.

The EB CLI did not always support including submodules. To avoid an accidental and undesirable change to projects that had existed before we added submodule support, the EB CLI does not include submodules when the `include_git_submodules` option is missing. If you have one of these existing projects and you want to include submodules in your deployments, add the option and set it to `true` as explained in this section.

CodeCommit behavior

Elastic Beanstalk's integration with [CodeCommit \(p. 871\)](#) doesn't support submodules at this time. If you enabled your environment to integrate with CodeCommit, submodules are not included in your deployments.

Assigning Git tags to your application version

You can use a Git tag as your version label to identify what application version is running in your environment. For example, type the following:

```
~/eb$ git tag -a v1.0 -m "My version 1.0"
```

Using the EB CLI with AWS CodeCommit

You can use the EB CLI to deploy your application directly from your AWS CodeCommit repository. With CodeCommit, you can upload only your changes to the repository when you deploy, instead of uploading your entire project. This can save you time and bandwidth if you have a large project or limited Internet connectivity. The EB CLI pushes your local commits and uses them to create application versions when you use **eb create** or **eb deploy**.

To deploy your changes, CodeCommit integration requires you to commit changes first. However, as you develop or debug, you might not want to push changes that you haven't confirmed are working. You can avoid committing your changes by staging them and using **eb deploy --staged** (which performs a

standard deployment). Or commit your changes to a development or testing branch first, and merge to your master branch only when your code is ready. With **eb use**, you can configure the EB CLI to deploy to one environment from your development branch, and to a different environment from your master branch.

Note

Some regions don't offer CodeCommit. The integration between Elastic Beanstalk and CodeCommit doesn't work in these regions.

For information about the AWS services offered in each region, see [Region Table](#).

Sections

- [Prerequisites \(p. 872\)](#)
- [Creating a CodeCommit repository with the EB CLI \(p. 872\)](#)
- [Deploying from your CodeCommit repository \(p. 873\)](#)
- [Configuring additional branches and environments \(p. 874\)](#)
- [Using an existing CodeCommit repository \(p. 875\)](#)

Prerequisites

To use CodeCommit with AWS Elastic Beanstalk, you need a local Git repository (either one you have already or a new one you create) with at least one commit, [permission to use CodeCommit](#), and an Elastic Beanstalk environment in a region that CodeCommit supports. Your environment and repository must be in the same region.

To initialize a Git repository

1. Run `git init` in your project folder.

```
~/my-app$ git init
```

2. Stage your project files with `git add`.

```
~/my-app$ git add .
```

3. Commit changes with `git commit`.

```
~/my-app$ git commit -m "Elastic Beanstalk application"
```

Creating a CodeCommit repository with the EB CLI

To get started with CodeCommit, run [eb init \(p. 907\)](#). During repository configuration, the EB CLI prompts you to use CodeCommit to store your code and speed up deployments. Even if you previously configured your project with **eb init**, you can run it again to configure CodeCommit.

To create a CodeCommit repository with the EB CLI

1. Run **eb init** in your project folder. During configuration, the EB CLI asks if you want to use CodeCommit to store your code and speed up deployments. If you previously configured your project with **eb init**, you can still run it again to configure CodeCommit. Type **y** at the prompt to set up CodeCommit.

```
~/my-app$ eb init
Note: Elastic Beanstalk now supports AWS CodeCommit; a fully-managed source control
service. To learn more, see Docs: https://aws.amazon.com/codecommit/
```

```
Do you wish to continue with CodeCommit? (y/n)(default is n): y
```

2. Choose **Create new Repository**.

```
Select a repository
1) my-repo
2) [ Create new Repository ]
(default is 2): 2
```

3. Type a repository name or press **Enter** to accept the default name.

```
Enter Repository Name
(default is "codecommit-origin"): my-app
Successfully created repository: my-app
```

4. Choose an existing branch for your commits, or use the EB CLI to create a new branch.

```
Enter Branch Name
***** Must have at least one commit to create a new branch with CodeCommit *****
(default is "master"): ENTER
Successfully created branch: master
```

Deploying from your CodeCommit repository

When you configure CodeCommit with your EB CLI repository, the EB CLI uses the contents of the repository to create source bundles. When you run **eb deploy** or **eb create**, the EB CLI pushes new commits and uses the HEAD revision of your branch to create the archive that it deploys to the EC2 instances in your environment.

To use CodeCommit integration with the EB CLI

1. Create a new environment with **eb create**.

```
~/my-app$ eb create my-app-env
Starting environment deployment via CodeCommit
--- Waiting for application versions to be pre-processed ---
Finished processing application version app-ac1ea-161010_201918
Setting up default branch
Environment details for: my-app-env
  Application name: my-app
  Region: us-east-2
  Deployed Version: app-ac1ea-161010_201918
  Environment ID: e-pm5mvvkfnd
  Platform: 64bit Amazon Linux 2016.03 v2.1.6 running Java 8
  Tier: WebServer-Standard
  CNAME: UNKNOWN
  Updated: 2016-10-10 20:20:29.725000+00:00
Printing Status:
INFO: createEnvironment is starting.
...
```

The EB CLI uses the latest commit in the tracked branch to create the application version that is deployed to the environment.

2. When you have new local commits, use **eb deploy** to push the commits and deploy to your environment.

```
~/my-app$ eb deploy
```



```
Starting environment deployment via CodeCommit
INFO: Environment update is starting.
INFO: Deploying new version to instance(s).
INFO: New application version was deployed to running EC2 instances.
INFO: Environment update completed successfully.
```

3. To test changes before you commit them, use the `--staged` option to deploy changes that you added to the staging area with `git add`.

```
~/my-app$ git add new-file
~/my-app$ eb deploy --staged
```

Deploying with the `--staged` option performs a standard deployment, bypassing CodeCommit.

Configuring additional branches and environments

CodeCommit configuration applies to a single branch. You can use `eb use` and `eb codesource` to configure additional branches or modify the current branch's configuration.

To configure CodeCommit integration with the EB CLI

1. To change the remote branch, use the `eb use` (p. 935) command's `--source` option.

```
~/my-app$ eb use test-env --source my-app/test
```

2. To create a new branch and environment, check out a new branch, push it to CodeCommit, create the environment, and then use `eb use` to connect the local branch, remote branch, and environment.

```
~/my-app$ git checkout -b production
~/my-app$ git push --set-upstream production
~/my-app$ eb create production-env
~/my-app$ eb use --source my-app/production production-env
```

3. To configure CodeCommit interactively, use `eb codesource codecommit` (p. 890).

```
~/my-app$ eb codesource codecommit
Current CodeCommit setup:
  Repository: my-app
  Branch: test
Do you wish to continue (y/n): y

Select a repository
1) my-repo
2) my-app
3) [ Create new Repository ]
(default is 2): 2

Select a branch
1) master
2) test
3) [ Create new Branch with local HEAD ]
(default is 1): 1
```

4. To disable CodeCommit integration, use `eb codesource local` (p. 890).

```
~/my-app$ eb codesource local
Current CodeCommit setup:
  Repository: my-app
```

```
Branch: master
Default set to use local sources
```

Using an existing CodeCommit repository

If you already have a CodeCommit repository and want to use it with Elastic Beanstalk, run **eb init** at the root of your local Git repository.

To use an existing CodeCommit repository with the EB CLI

1. Clone your CodeCommit repository.

```
~$ git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/my-app
```

2. Check out and push a branch to use for your Elastic Beanstalk environment.

```
~/my-app$ git checkout -b dev-env
~/my-app$ git push --set-upstream origin dev-env
```

3. Run **eb init**. Choose the same region, repository, and branch name that you are currently using.

```
~/my-app$ eb init
Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
4) eu-west-1 : Europe (Ireland)
5) eu-central-1 : Europe (Frankfurt)
6) ap-south-1 : Asia Pacific (Mumbai)
7) ap-southeast-1 : Asia Pacific (Singapore)
...
(default is 3): 1
...
Note: Elastic Beanstalk now supports AWS CodeCommit; a fully-managed source control
service. To learn more, see Docs: https://aws.amazon.com/codecommit/
Do you wish to continue with CodeCommit? (y/n)(default is n): y

Select a repository
1) my-app
2) [ Create new Repository ]
(default is 1): 1

Select a branch
1) master
2) dev-env
3) [ Create new Branch with local HEAD ]
(default is 2): 2
```

For more information about using **eb init**, see [Configure the EB CLI \(p. 860\)](#).

Using the EB CLI to monitor environment health

The [Elastic Beanstalk Command Line Interface \(p. 852\)](#) (EB CLI) is a command line tool for managing AWS Elastic Beanstalk environments. You also can use the EB CLI to monitor your environment's health in real time and with more granularity than is currently available in the Elastic Beanstalk console

After [installing \(p. 853\)](#) and [configuring \(p. 860\)](#) the EB CLI, you can [launch a new environment \(p. 864\)](#) and deploy your code to it with the **eb create** command. If you already have an environment that you created in the Elastic Beanstalk console, you can attach the EB CLI to it by running **eb init** in a project folder and following the prompts (the project folder can be empty).

Important

Ensure that you are using the latest version of the EB CLI by running `pip install` with the `--upgrade` option:

```
$ sudo pip install --upgrade awsebcli
```

For complete EB CLI installation instructions, see [Install the EB CLI \(p. 853\)](#).

To use the EB CLI to monitor your environment's health, you must first configure a local project folder by running **eb init** and following the prompts. For complete instructions, see [Configure the EB CLI \(p. 860\)](#).

If you already have an environment running in Elastic Beanstalk and want to use the EB CLI to monitor its health, follow these steps to attach it to the existing environment.

To attach the EB CLI to an existing environment

1. Open a command line terminal and navigate to your user folder.
2. Create and open a new folder for your environment.
3. Run the **eb init** command, and then choose the application and environment whose health you want to monitor. If you have only one environment running the application you choose, the EB CLI will select it automatically and you won't need to choose the environment, as shown in the following example.

```
~/project$ eb init
Select an application to use
1) elastic-beanstalk-example
2) [ Create new Application ]
(default is 2): 1
Select the default environment.
You can change this later by typing "eb use [environment_name]".
1) elasticBeanstalkEx2-env
2) elasticBeanstalkExa-env
(default is 1): 1
```

To monitor health by using the EB CLI

1. Open a command line and navigate to your project folder.
2. Run the **eb health** command to display the health status of the instances in your environment. In this example, there are five instances running in a Linux environment.

```
~/project $ eb health
elasticBeanstalkExa-env                               Ok
2015-07-08 23:13:20
WebServer
Ruby 2.1 (Puma)
total          ok          warning    degraded    severe      info      pending    unknown
5             5             0           0           0           0         0         0

instance-id   status      cause
health
Overall       Ok
i-d581497d    Ok
i-d481497c    Ok
i-136e00c0    Ok
```

AWS Elastic Beanstalk Developer Guide
Monitoring health

```

i-126e00c1    Ok
i-8b2cf575    Ok

instance-id  r/sec    %2xx    %3xx    %4xx    %5xx    p99    p90    p75    p50
p10
Overall      671.8    100.0    0.0     0.0     0.0     0.003  0.002  0.001  0.001
0.000
i-d581497d   143.0    1430     0        0        0        0.003  0.002  0.001  0.001
0.000
i-d481497c   128.8    1288     0        0        0        0.003  0.002  0.001  0.001
0.000
i-136e00c0   125.4    1254     0        0        0        0.004  0.002  0.001  0.001
0.000
i-126e00c1   133.4    1334     0        0        0        0.003  0.002  0.001  0.001
0.000
i-8b2cf575   141.2    1412     0        0        0        0.003  0.002  0.001  0.001
0.000

instance-id  type     az    running    load 1  load 5    user%  nice%  system%
idle%  iowait%
i-d581497d   t2.micro  1a    12 mins    0.0     0.04     6.2    0.0    1.0
92.5      0.1
i-d481497c   t2.micro  1a    12 mins    0.01    0.09     5.9    0.0    1.6
92.4      0.1
i-136e00c0   t2.micro  1b    12 mins    0.15    0.07     5.5    0.0    0.9
93.2      0.0
i-126e00c1   t2.micro  1b    12 mins    0.17    0.14     5.7    0.0    1.4
92.7      0.1
i-8b2cf575   t2.micro  1c    1 hour     0.19    0.08     6.5    0.0    1.2
92.1      0.1

instance-id  status    id    version    ago
deployments
i-d581497d   Deployed  1     Sample Application  12 mins
i-d481497c   Deployed  1     Sample Application  12 mins
i-136e00c0   Deployed  1     Sample Application  12 mins
i-126e00c1   Deployed  1     Sample Application  12 mins
i-8b2cf575   Deployed  1     Sample Application  1 hour

```

In this example, there is a single instance running in a Windows environment.

```

~/project $ eb health
WindowsSampleApp-env                               Ok
2018-05-22 17:33:19
WebServer                                           IIS 10.0 running on 64bit
Windows Server 2016/2.2.0
total      ok      warning  degraded  severe    info    pending  unknown
1          1          0          0          0          0          0          0

instance-id    status    cause
health
Overall        Ok
i-065716fba0e08a351  Ok

instance-id    r/sec    %2xx    %3xx    %4xx    %5xx    p99    p90    p75
p50    p10
Overall        13.7    100.0    0.0     0.0     0.0     1.403  0.970  0.710
0.413  0.079
i-065716fba0e08a351  2.4    100.0    0.0     0.0     0.0     1.102*  0.865  0.601
0.413  0.091

instance-id    type     az    running    % user time    % privileged time %
idle time     cpu

```

```
i-065716fba0e08a351 t2.large 1b 4 hours 0.2 0.1
99.7

instance-id status id version ago
deployments
i-065716fba0e08a351 Deployed 2 Sample Application 4 hours
```

Reading the output

The output displays the name of the environment, the environment's overall health, and the current date at the top of the screen.

```
elasticbeanstalkExa-env Ok
2015-07-08 23:13:20
```

The next three lines display the type of environment ("WebServer" in this case), the configuration (Ruby 2.1 with Puma), and a breakdown of how many instances are in each of the seven states.

```
WebServer Ruby
2.1 (Puma)
total ok warning degraded severe info pending unknown
5 5 0 0 0 0 0 0
```

The rest of the output is split into four sections. The first displays the *status* and the *cause* of the status for the environment overall, and then for each instance. The following example shows two instances in the environment with a status of *Info* and a cause indicating that a deployment has started.

```
instance-id status cause health
Overall Ok
i-d581497d Info Performing application deployment (running for 3 seconds)
i-d481497c Info Performing application deployment (running for 3 seconds)
i-136e00c0 Ok
i-126e00c1 Ok
i-8b2cf575 Ok
```

For information about health statuses and colors, see [Health colors and statuses \(p. 706\)](#).

The **requests** section displays information from the web server logs on each instance. In this example, each instance is taking requests normally and there are no errors.

```
instance-id r/sec %2xx %3xx %4xx %5xx p99 p90 p75 p50
p10 requests
Overall 13.7 100.0 0.0 0.0 0.0 1.403 0.970 0.710 0.413
0.079
i-d581497d 2.4 100.0 0.0 0.0 0.0 1.102* 0.865 0.601 0.413
0.091
i-d481497c 2.7 100.0 0.0 0.0 0.0 0.842* 0.788 0.480 0.305
0.062
i-136e00c0 4.1 100.0 0.0 0.0 0.0 1.520* 1.088 0.883 0.524
0.104
i-126e00c1 2.2 100.0 0.0 0.0 0.0 1.334* 0.791 0.760 0.344
0.197
i-8b2cf575 2.3 100.0 0.0 0.0 0.0 1.162* 0.867 0.698 0.477
0.076
```

The **cpu** section shows operating system metrics for each instance. The output differs by operating system. Here is the output for Linux environments.

instance-id	type	az	running	load 1	load 5	user%	nice%	system%	idle
% iowait%			cpu						
i-d581497d	t2.micro	1a	12 mins	0.0	0.03	0.2	0.0	0.0	
99.7	0.1								
i-d481497c	t2.micro	1a	12 mins	0.0	0.03	0.3	0.0	0.0	
99.7	0.0								
i-136e00c0	t2.micro	1b	12 mins	0.0	0.04	0.1	0.0	0.0	
99.9	0.0								
i-126e00c1	t2.micro	1b	12 mins	0.01	0.04	0.2	0.0	0.0	
99.7	0.1								
i-8b2cf575	t2.micro	1c	1 hour	0.0	0.01	0.2	0.0	0.1	
99.6	0.1								

Here is the output for Windows environments.

instance-id	type	az	running	% user time	% privileged time	% idle time
i-065716fba0e08a351	t2.large	1b	4 hours	0.2		0.0
99.8						

For information about the server and operating system metrics shown, see [Instance metrics \(p. 708\)](#).

The final section, **deployments**, shows the deployment status of each instance. If a rolling deployment fails, you can use the deployment ID, status, and version label shown to identify instances in your environment that are running the wrong version.

instance-id	status	id	version	ago
			deployments	
i-d581497d	Deployed	1	Sample Application	12 mins
i-d481497c	Deployed	1	Sample Application	12 mins
i-136e00c0	Deployed	1	Sample Application	12 mins
i-126e00c1	Deployed	1	Sample Application	12 mins
i-8b2cf575	Deployed	1	Sample Application	1 hour

Interactive health view

The **eb health** command displays a snapshot of your environment's health. To refresh the displayed information every 10 seconds, use the `--refresh` option.

```
$ eb health --refresh
elasticBeanstalkExa-env                               Ok
2015-07-09 22:10:04 (1 secs)
WebServer
  Ruby 2.1 (Puma)
  total      ok      warning  degraded  severe    info    pending  unknown
  5          5          0         0         0         0       0         0

instance-id  status  cause
health
Overall      Ok
i-bb65c145   Ok      Application deployment completed 35 seconds ago and took 26
seconds
i-ba65c144   Ok      Application deployment completed 17 seconds ago and took 25
seconds
i-f6a2d525   Ok      Application deployment completed 53 seconds ago and took 26
seconds
i-e8a2d53b   Ok      Application deployment completed 32 seconds ago and took 31
seconds
i-e81cca40   Ok
```

```

instance-id  r/sec  %2xx  %3xx  %4xx  %5xx  p99  p90  p75  p50
p10
Overall      671.8  100.0  0.0   0.0   0.0   0.003 0.002 0.001 0.001
0.000
i-bb65c145   143.0  1430   0     0     0     0.003 0.002 0.001 0.001
0.000
i-ba65c144   128.8  1288   0     0     0     0.003 0.002 0.001 0.001
0.000
i-f6a2d525   125.4  1254   0     0     0     0.004 0.002 0.001 0.001
0.000
i-e8a2d53b   133.4  1334   0     0     0     0.003 0.002 0.001 0.001
0.000
i-e81cca40   141.2  1412   0     0     0     0.003 0.002 0.001 0.001
0.000

instance-id  type      az  running  load 1  load 5  user%  nice%  system%  idle
% iowait%
i-bb65c145   t2.micro  1a  12 mins  0.0    0.03   0.2    0.0    0.0
99.7 0.1
i-ba65c144   t2.micro  1a  12 mins  0.0    0.03   0.3    0.0    0.0
99.7 0.0
i-f6a2d525   t2.micro  1b  12 mins  0.0    0.04   0.1    0.0    0.0
99.9 0.0
i-e8a2d53b   t2.micro  1b  12 mins  0.01   0.04   0.2    0.0    0.0
99.7 0.1
i-e81cca40   t2.micro  1c  1 hour   0.0    0.01   0.2    0.0    0.1
99.6 0.1

instance-id  status  id  version  deployments  ago
i-bb65c145   Deployed  1  Sample Application  12 mins
i-ba65c144   Deployed  1  Sample Application  12 mins
i-f6a2d525   Deployed  1  Sample Application  12 mins
i-e8a2d53b   Deployed  1  Sample Application  12 mins
i-e81cca40   Deployed  1  Sample Application  1 hour

(Commands: Help,Quit, # # # #)

```

This example shows an environment that has recently been scaled up from one to five instances. The scaling operation succeeded, and all instances are now passing health checks and are ready to take requests. In interactive mode, the health status updates every 10 seconds. In the upper-right corner, a timer ticks down to the next update.

In the lower-left corner, the report displays a list of options. To exit interactive mode, press **Q**. To scroll, press the arrow keys. To see a list of additional commands, press **H**.

Interactive health view options

When viewing environment health interactively, you can use keyboard keys to adjust the view and tell Elastic Beanstalk to replace or reboot individual instances. To see a list of available commands while viewing the health report in interactive mode, press **H**.

```

up,down,home,end  Scroll vertically
left,right         Scroll horizontally
F                 Freeze/unfreeze data
X                 Replace instance
B                 Reboot instance
<,>              Move sort column left/right
-,+              Sort order descending/ascending
P                 Save health snapshot data file
Z                 Toggle color/mono mode
Q                 Quit this program

```

```
Views
1      All tables/split view
2      Status Table
3      Request Summary Table
4      CPU%/Load Table
H      This help menu

(press Q or ESC to return)
```

Managing multiple Elastic Beanstalk environments as a group with the EB CLI

You can use the EB CLI to create groups of AWS Elastic Beanstalk environments, each running a separate component of a service-oriented architecture application. The EB CLI manages such groups by using the [ComposeEnvironments](#) API.

Note

Environment groups are different than multiple containers in a Multicontainer Docker environment. With environment groups, each component of your application runs in a separate Elastic Beanstalk environment, with its own dedicated set of Amazon EC2 instances. Each component can scale separately. With Multicontainer Docker, you combine several components of an application into a single environment. All components share the same set of Amazon EC2 instances, with each instance running multiple Docker containers. Choose one of these architectures according to your application's needs.

For details about Multicontainer Docker, see [Multicontainer Docker environments \(p. 58\)](#).

Organize your application components into the following folder structure:

```
~/project-name
|-- component-a
|   |-- env.yaml
|-- component-b
|   |-- env.yaml
```

Each subfolder contains the source code for an independent component of an application that will run in its own environment and an environment definition file named `env.yaml`. For details on the `env.yaml` format, see [Environment manifest \(env.yaml\) \(p. 645\)](#).

To use the `Compose Environments` API, first run `eb init` from the project folder, specifying each component by the name of the folder that contains it with the `--modules` option:

```
~/workspace/project-name$ eb init --modules component-a component-b
```

The EB CLI prompts you to [configure each component \(p. 860\)](#), and then creates the `.elasticbeanstalk` directory in each component folder. EB CLI doesn't create configuration files in the parent directory.

```
~/project-name
|-- component-a
|   |-- .elasticbeanstalk
|   |-- env.yaml
|-- component-b
|   |-- .elasticbeanstalk
|   |-- env.yaml
```


Next, run the **eb create** command with a list of environments to create, one for each component:

```
~/workspace/project-name$ eb create --modules component-a component-b --env-group-suffix group-name
```

This command creates an environment for each component. The names of the environments are created by concatenating the `EnvironmentName` specified in the `env.yml` file with the group name, separated by a hyphen. The total length of these two options and the hyphen must not exceed the maximum allowed environment name length of 23 characters.

To update the environment, use the **eb deploy** command:

```
~/workspace/project-name$ eb deploy --modules component-a component-b
```

You can update each component individually or you can update them as a group. Specify the components that you want to update with the `--modules` option.

The EB CLI stores the group name that you used with **eb create** in the `branch-defaults` section of the EB CLI configuration file under `/.elasticbeanstalk/config.yml`. To deploy your application to a different group, use the `--env-group-suffix` option when you run **eb deploy**. If the group does not already exist, the EB CLI will create a new group of environments:

```
~/workspace/project-name$ eb deploy --modules component-a component-b --env-group-suffix group-2-name
```

To terminate environments, run **eb terminate** in the folder for each module. By default, the EB CLI will show an error if you try to terminate an environment that another running environment is dependent on. Terminate the dependent environment first, or use the `--ignore-links` option to override the default behavior:

```
~/workspace/project-name/component-b$ eb terminate --ignore-links
```

Troubleshooting issues with the EB CLI

This topic lists common error messages encountered when using the EB CLI and possible solutions. If you encounter an error message not shown here, use the *Feedback* links to let us know about it.

ERROR: An error occurred while handling git command. Error code: 128 Error: fatal: Not a valid object name HEAD

Cause: This error message is shown when you have initialized a Git repository but have not yet committed. The EB CLI looks for the HEAD revision when your project folder contains a Git repository.

Solution: Add the files in your project folder to the staging area and commit:

```
~/my-app$ git add .  
~/my-app$ git commit -m "First commit"
```

ERROR: This branch does not have a default environment. You must either specify an environment by typing "eb status my-env-name" or set a default environment by typing "eb use my-env-name".

Cause: When you create a new branch in git, it is not attached to an Elastic Beanstalk environment by default.

Solution: Run **eb list** to see a list of available environments. Then run **eb use env-name** to use one of the available environments.

ERROR: 2.0+ Platforms require a service role. You can provide one with --service-role option

Cause: If you specify an environment name with **eb create** (for example, **eb create my-env**), the EB CLI will not attempt to create a service role for you. If you don't have the default service role, the above error is shown.

Solution: Run **eb create** without an environment name and follow the prompts to create the default service role.

Troubleshooting deployments

If your Elastic Beanstalk deployment didn't go quite as smoothly as planned, you may get a 404 (if your application failed to launch) or 500 (if your application fails during runtime) response, instead of seeing your website. To troubleshoot many common issues, you can use the EB CLI to check the status of your deployment, view its logs, gain access to your EC2 instance with SSH, or to open the AWS Management Console page for your application environment.

To use the EB CLI to help troubleshoot your deployment

1. Run **eb status** to see the status of your current deployment and health of your EC2 hosts. For example:

```
$ eb status --verbose
Environment details for: python_eb_app
Application name: python_eb_app
Region: us-west-2
Deployed Version: app-150206_035343
Environment ID: e-wa8u6rrmqy
Platform: 64bit Amazon Linux 2014.09 v1.1.0 running Python 2.7
Tier: WebServer-Standard-
CNAME: python_eb_app.elasticbeanstalk.com
Updated: 2015-02-06 12:00:08.557000+00:00
Status: Ready
Health: Green
Running instances: 1
    i-8000528c: InService
```

Note

Using the **--verbose** switch provides information about the status of your running instances. Without it, **eb status** will print only general information about your environment.

2. Run **eb health** to view health information about your environment:

```
$ eb health --refresh
elasticBeanstalkExa-env                               Degraded
2016-03-28 23:13:20
WebServer
Ruby 2.1 (Puma)
total          ok      warning  degraded  severe   info    pending  unknown
5              2       0         2         1        0       0        0

instance-id   status   cause
Overall      Degraded Incorrect application version found on 3 out of 5 instances.
Expected version "Sample Application" (deployment 1).
i-d581497d    Degraded Incorrect application version "v2" (deployment 2). Expected
version "Sample Application" (deployment 1).
i-d481497c    Degraded Incorrect application version "v2" (deployment 2). Expected
version "Sample Application" (deployment 1).
i-136e00c0    Severe   Instance ELB health has not been available for 5 minutes.
i-126e00c1    Ok
```

```

i-8b2cf575    Ok

instance-id  r/sec    %2xx    %3xx    %4xx    %5xx    p99    p90    p75    p50
p10
Overall      646.7    100.0    0.0     0.0     0.0     0.003  0.002  0.001  0.001
0.000
i-dac3f859   167.5    1675    0       0       0       0.003  0.002  0.001  0.001
0.000
i-05013a81   161.2    1612    0       0       0       0.003  0.002  0.001  0.001
0.000
i-04013a80   0.0      -       -       -       -       -       -       -       -       -
-
i-3ab524a1   155.9    1559    0       0       0       0.003  0.002  0.001  0.001
0.000
i-bf300d3c   162.1    1621    0       0       0       0.003  0.002  0.001  0.001
0.000

instance-id  type     az    running    load 1  load 5    user%  nice%  system%
idle%  iowait%
i-d581497d   t2.micro 1a    25 mins    0.16    0.1       7.0    0.0    1.7
91.0    0.1
i-d481497c   t2.micro 1a    25 mins    0.14    0.1       7.2    0.0    1.6
91.1    0.0
i-136e00c0   t2.micro 1b    25 mins    0.0     0.01     0.0    0.0    0.0
99.9    0.1
i-126e00c1   t2.micro 1b    25 mins    0.03    0.08     6.9    0.0    2.1
90.7    0.1
i-8b2cf575   t2.micro 1c    1 hour     0.05    0.41     6.9    0.0    2.0
90.9    0.0

instance-id  status    id  version    ago
deployments
i-d581497d   Deployed  2   v2          9 mins
i-d481497c   Deployed  2   v2          7 mins
i-136e00c0   Failed   2   v2          5 mins
i-126e00c1   Deployed  1   Sample Application 25 mins
i-8b2cf575   Deployed  1   Sample Application 1 hour

```

The above example shows an environment with five instances where the deployment of version "v2" failed on the third instance. After a failed deployment, the expected version is reset to the last version that succeeded, which in this case is "Sample Application" from the first deployment. See [Using the EB CLI to monitor environment health \(p. 875\)](#) for more information.

3. Run **eb logs** to download and view the logs associated with your application deployment.

```
$ eb logs
```

4. Run **eb ssh** to connect with the EC2 instance that's running your application and examine it directly. On the instance, your deployed application can be found in the `/opt/python/current/app` directory, and your Python environment will be found in `/opt/python/run/venv/`.
5. Run **eb console** to view your application environment on the [AWS Management Console](#). You can use the web interface to easily examine various aspects of your deployment, including your application's configuration, status, events, logs. You can also download the current or past application versions that you've deployed to the server.

EB CLI command reference

You can use the Elastic Beanstalk command line interface (EB CLI) to perform a variety of operations to deploy and manage your Elastic Beanstalk applications and environments. The EB CLI integrates with Git if you want to deploy application source code that is under Git source control. For more information,

see [Using the Elastic Beanstalk command line interface \(EB CLI\) \(p. 852\)](#) and [Using the EB CLI with Git \(p. 870\)](#).

Commands

- [eb abort \(p. 885\)](#)
- [eb appversion \(p. 886\)](#)
- [eb clone \(p. 888\)](#)
- [eb codesource \(p. 890\)](#)
- [eb config \(p. 891\)](#)
- [eb console \(p. 893\)](#)
- [eb create \(p. 894\)](#)
- [eb deploy \(p. 903\)](#)
- [eb events \(p. 904\)](#)
- [eb health \(p. 905\)](#)
- [eb init \(p. 907\)](#)
- [eb labs \(p. 910\)](#)
- [eb list \(p. 910\)](#)
- [eb local \(p. 911\)](#)
- [eb logs \(p. 913\)](#)
- [eb open \(p. 916\)](#)
- [eb platform \(p. 916\)](#)
- [eb printenv \(p. 923\)](#)
- [eb restore \(p. 924\)](#)
- [eb scale \(p. 924\)](#)
- [eb setenv \(p. 925\)](#)
- [eb ssh \(p. 926\)](#)
- [eb status \(p. 928\)](#)
- [eb swap \(p. 929\)](#)
- [eb tags \(p. 930\)](#)
- [eb terminate \(p. 933\)](#)
- [eb upgrade \(p. 934\)](#)
- [eb use \(p. 935\)](#)
- [Common options \(p. 935\)](#)

eb abort

Description

Cancels an upgrade when environment configuration changes to instances are still in progress.

Note

If you have more than two environments that are undergoing a update, you are prompted to select the name of the environment for which you want to roll back changes.

Syntax

eb abort

eb abort *environment-name*

Options

Name	Description
Common options (p. 935)	

Output

The command shows a list of environments currently being updated and prompts you to choose the update that you want to abort. If only one environment is currently being updated, you do not need to specify the environment name. If successful, the command reverts environment configuration changes. The rollback process continues until all instances in the environment have the previous environment configuration or until the rollback process fails.

Example

The following example cancels the platform upgrade.

```
$ eb abort
Aborting update to environment "tmp-dev".
<list of events>
```

eb appversion

Description

Manages your Elastic Beanstalk application versions, including deleting a version of the application or creating the application version lifecycle policy. If you invoke the command without any options, it goes into [interactive mode \(p. 887\)](#).

Use the `--delete` option to delete a version of the application.

Use the `lifecycle` option to display or create the application version lifecycle policy. Learn more at [the section called "Version lifecycle" \(p. 339\)](#).

Syntax

eb appversion

eb appversion [-d | --delete] *version-label*

eb appversion lifecycle [-p | --print]

Options

Name	Description
-d <i>version-label</i> or --delete <i>version-label</i>	Delete version <i>version-label</i> of the application.

Name	Description
lifecycle	Invoke the default editor to create a new application version lifecycle policy. Use this policy to avoid hitting the application version quota .
lifecycle -p or lifecycle --print	Display the current application lifecycle policy.
Common options (p. 935)	

Using the command interactively

The command without any arguments displays the versions of the application, from most recent to oldest. See the **Examples** section for examples of what the screen looks like. Note the status line at the bottom of the display. It displays context-sensitive information that you can use to guide you.

Press **d** to delete an application version, press **l** to manage the lifecycle policy for your application, or press **q** to quit without making any changes.

Note

If the version is deployed to any environment, you cannot delete that version.

Output

The command with the `--delete version-label` option displays a message confirming that the application version was deleted.

Examples

The following example shows the interactive window for an application with no deployments.

```

No Environment Specified
Environment Status: Unknown Health Unknown
Current version # deployed: None

#  Version Label  Date Created  Age  Description
3  v4             2016/12/22 13:28  56 secs  new features
2  v3             2016/12/22 13:27  1 min    important update
1  v1             2016/12/15 23:51  6 days   wow

(Commands: Quit, Delete, Lifecycle, ▼▲◀▶)

```

The following example shows the interactive window for an application with the fourth version, with version label **Sample Application**, deployed.

```

Sample-env Applica
Environment Status: Launching Health Green
Current version # deployed: 4

#   Version Label   Date Created   Age   Description
4   Sample Application 2016/12/22 13:30 2 mins -
3   v4                2016/12/22 13:28 4 mins new features
2   v3                2016/12/22 13:27 5 mins important updat
1   v1                2016/12/15 23:51 6 days wow

(Commands: Quit, Delete, Lifecycle, ▼▲◀▶)

```

The following example shows the output from an **eb appversion lifecycle -p** command, where **ACCOUNT-ID** is the user's account ID:

```

Application details for: lifecycle
Region: sa-east-1
Description: Application created from the EB CLI using "eb init"
Date Created: 2016/12/20 02:48 UTC
Date Updated: 2016/12/20 02:48 UTC
Application Versions: ['Sample Application']
Resource Lifecycle Config(s):
VersionLifecycleConfig:
  MaxCountRule:
    DeleteSourceFromS3: False
    Enabled: False
    MaxCount: 200
  MaxAgeRule:
    DeleteSourceFromS3: False
    Enabled: False
    MaxAgeInDays: 180
ServiceRole: arn:aws:iam::ACCOUNT-ID:role/aws-elasticbeanstalk-service-role

```

eb clone

Description

Clones an environment to a new environment so that both have identical environment settings.

Note

By default, regardless of the solution stack version of the environment from which you create the clone, the **eb clone** command creates the clone environment with the most recent solution stack. You can suppress this by including the **--exact** option when you run the command.

Syntax

eb clone

eb clone *environment-name*

Options

Name	Description
<p><code>-n <i>string</i></code></p> <p>or</p> <p><code>--clone_name <i>string</i></code></p>	Desired name for the cloned environment.
<p><code>-c <i>string</i></code></p> <p>or</p> <p><code>--cname <i>string</i></code></p>	Desired CNAME prefix for the cloned environment.
<code>--envvars</code>	<p>Environment properties in a comma-separated list with the format <i>name=value</i>.</p> <p>Type: String</p> <p>Constraints:</p> <ul style="list-style-type: none"> • Key-value pairs must be separated by commas. • Keys and values can contain any alphabetic character in any language, any numeric character, white space, invisible separator, and the following symbols: <code>_ . : / + \ - @</code> • Keys can contain up to 128 characters. Values can contain up to 256 characters. • Keys and values are case sensitive. • Values cannot match the environment name. • Values cannot include either <code>aws:</code> or <code>elasticbeanstalk:</code>. • The combined size of all environment properties cannot exceed 4096 bytes.
<code>--exact</code>	Prevents Elastic Beanstalk from updating the solution stack version for the new clone environment to the most recent version available (for the original environment's platform).
<code>--scale <i>number</i></code>	The number of instances to run in the clone environment when it is launched.
<code>--tags <i>name=value</i></code>	Tags (p. 513) for the resources in your environment in a comma-separated list with the format <i>name=value</i> .
<code>--timeout</code>	The number of minutes before the command times out.
Common options (p. 935)	

Output

If successful, the command creates an environment that has the same settings as the original environment or with modifications to the environment as specified by any **eb clone** options.

Example

The following example clones the specified environment.

```
$ eb clone
Enter name for Environment Clone
(default is tmp-dev-clone):
Enter DNS CNAME prefix
(default is tmp-dev-clone):
Environment details for: tmp-dev-clone
  Application name: tmp
  Region: us-west-2
  Deployed Version: app-141029_144740
  Environment ID: e-vjvrqnn5pv
  Platform: 64bit Amazon Linux 2014.09 v1.0.9 running PHP 5.5
  Tier: WebServer-Standard-1.0
  CNAME: tmp-dev-clone.elasticbeanstalk.com
  Updated: 2014-10-29 22:00:23.008000+00:00
Printing Status:
2018-07-11 21:04:20    INFO: createEnvironment is starting.
2018-07-11 21:04:21    INFO: Using elasticbeanstalk-us-west-2-888888888888 as Amazon S3
  storage bucket for environment data.
...
2018-07-11 21:07:10    INFO: Successfully launched environment: tmp-dev-clone
```

eb codesource

Description

Configures the EB CLI to [deploy from a CodeCommit repository \(p. 871\)](#), or disables CodeCommit integration and uploads the source bundle from your local machine.

Note

Some AWS Regions don't offer CodeCommit. The integration between Elastic Beanstalk and CodeCommit doesn't work in these Regions.

For information about the AWS services offered in each Region, see [Region Table](#).

Syntax

eb codesource

eb codesource codecommit

eb codesource local

Options

Name	Description
Common options (p. 935)	

Output

eb codesource prompts you to choose between CodeCommit integration and standard deployments.

eb codesource codecommit initiates interactive repository configuration for CodeCommit integration.

eb codesource local shows the original configuration and disables CodeCommit integration.

Examples

Use **eb codesource codecommit** to configure CodeCommit integration for the current branch.

```
~/my-app$ eb codesource codecommit
Select a repository
1) my-repo
2) my-app
3) [ Create new Repository ]
(default is 1): 1

Select a branch
1) master
2) test
3) [ Create new Branch with local HEAD ]
(default is 1): 1
```

Use **eb codesource local** to disable CodeCommit integration for the current branch.

```
~/my-app$ eb codesource local
Current CodeCommit setup:
  Repository: my-app
  Branch: master
Default set to use local sources
```

eb config

Description

Changes the environment configuration settings. This command saves the environment configuration settings as well as uploads, downloads, or lists saved configurations.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command also changes the builder configuration settings, based on the values set in `platform.yaml`.

Note

eb config does not show environment properties. To set environment properties that you can read from within your application, use [eb setenv \(p. 552\)](#).

Syntax

eb config

eb config *environment-name*

The following describes the syntax for using the **eb config** command to work with saved configurations. For examples, see the [Examples \(p. 892\)](#) section later in this topic.

- **eb config delete *filename*** – Deletes the named saved configuration.
- **eb config get *filename*** – Downloads the named saved configuration.

- **eb config list** – Lists the saved configurations that you have in Amazon S3.
- **eb config put *filename*** – Uploads the named saved configuration to an Amazon S3 bucket. The *filename* must have the file extension `.cfg.yml`. To specify the file name without a path, you can save the file to the `.elasticbeanstalk` folder or to the `.elasticbeanstalk/saved_configs/` folder before you run the command. Alternatively, you can specify the *filename* by providing the full path.
- **eb config save** – Saves the environment configuration settings for the current running environment to `.elasticbeanstalk/saved_configs/` with the filename `[configuration-name].cfg.yml`. By default, the EB CLI saves the configuration settings with a *configuration-name* based on the environment name. You can specify a different configuration name by including the `--cfg` option with your desired configuration name when you run the command.

You can tag your saved configuration using the `--tags` option.

Options

Name	Description
<code>--cfg <i>config-name</i></code>	The name to use for a saved configuration (which you can later specify to create or update an environment from a saved configuration). This option works with eb config save only.
<code>--tags <i>key1=value1[,key2=value2]</i></code>	Tags to add to your saved configuration. Tags are specified as a comma-separated list of <code>key=value</code> pairs. For more details, see Tagging saved configurations (p. 644) . This option works with eb config save only.
<code>--timeout <i>timeout</i></code>	The number of minutes before the command times out.
Common options (p. 935)	

Output

If the command runs successfully with no parameters, the command displays your current option settings in the text editor that you configured as the `EDITOR` environment variable. (If you have not configured an `EDITOR` environment variable, then EB CLI displays your option settings in your computer's default editor for YAML files.) When you save changes to the file and close the editor, the environment is updated with the option settings in the file.

If the command runs successfully with the `get` parameter, the command displays the location of the local copy that you downloaded.

If the command runs successfully with the `save` parameter, the command displays the location of the saved file.

Examples

This section describes how to change the text editor that you use to view and edit your option settings file.

For Linux/UNIX, the following example changes the editor to vim:

```
$ export EDITOR=vim
```

For Linux/UNIX, the following example changes the editor to what is installed at `/usr/bin/kate`.

```
$ export EDITOR=/usr/bin/kate
```

For Windows, the following example changes the editor to Notepad++.

```
> set EDITOR="C:\Program Files\Notepad++\Notepad++.exe
```

This section provides examples for the **eb config** command when it is run with parameters.

The following example deletes the saved configuration named `app-tmp`.

```
$ eb config delete app-tmp
```

The following example downloads the saved configuration with the name `app-tmp` from your Amazon S3 bucket.

```
$ eb config get app-tmp
```

The following example lists the names of saved configurations that are stored in your Amazon S3 bucket.

```
$ eb config list
```

The following example uploads the local copy of the saved configuration named `app-tmp` to your Amazon S3 bucket.

```
$ eb config put app-tmp
```

The following example saves configuration settings from the current running environment. If you do not provide a name to use for the saved configuration, then Elastic Beanstalk names the configuration file according to the environment name. For example, an environment named `tmp-dev` would be called `tmp-dev.cfg.yml`. Elastic Beanstalk saves the file to the folder `/.elasticbeanstalk/saved_configs/`.

```
$ eb config save
```

The following example shows how to use the `--cfg` option to save the configuration settings from the environment `tmp-dev` to a file called `v1-app-tmp.cfg.yml`. Elastic Beanstalk saves the file to the folder `/.elasticbeanstalk/saved_configs/`. If you do not specify an environment name, Elastic Beanstalk saves configuration settings from the current running environment.

```
$ eb config save tmp-dev --cfg v1-app-tmp
```

eb console

Description

Opens a browser to display the environment configuration dashboard in the Elastic Beanstalk Management Console.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command also displays the builder environment configuration, as specified in `platform.yaml`, in the Elastic Beanstalk Management Console.

Syntax

eb console

eb console *environment-name*

Options

Name	Description
Common options (p. 935)	

eb create

Description

Creates a new environment and deploys an application version to it.

Note

- To use **eb create** on a .NET application, you must create a deployment package as described in [Creating a source bundle for a .NET application \(p. 347\)](#), then set up the CLI configuration to deploy the package as an artifact as described in [Deploying an artifact instead of the project folder \(p. 863\)](#).
- Creating environments with the EB CLI requires a [service role \(p. 20\)](#). You can create a service role by creating an environment in the Elastic Beanstalk console. If you don't have a service role, the EB CLI attempts to create one when you run `eb create`.

You can deploy the application version from a few sources:

- By default: from the application source code in the local project directory.
- Using the `--version` option: from an application version that already exists in your application.
- When your project directory doesn't have application code, or when using the `--sample` option: from a sample application, specific to your environment's platform.

Syntax

eb create

eb create *environment-name*

An environment name must be between 4 and 40 characters in length, and can only contain letters, numbers, and hyphens. An environment name can't begin or end with a hyphen.

If you include an environment name in the command, the EB CLI doesn't prompt you to make any selections or create a service role.

If you run the command without an environment name argument, it runs in an interactive flow, and prompts you to enter or select values for some settings. In this interactive flow, in case you are deploying

a sample application, the EB CLI also asks you if you want to download this sample application to your local project directory. This enables you to use the EB CLI with the new environment later to run operations that require the application's code, like [eb deploy](#) (p. 903).

Options

None of these options are required. If you run **eb create** without any options, the EB CLI prompts you to enter or select a value for each setting.

Name	Description
-d or --branch_default	Set the environment as the default environment for the current repository.
--cfg <i>config-name</i>	Use platform settings from a saved configuration (p. 545) in <code>.elasticbeanstalk/saved_configs/</code> or your Amazon S3 bucket. Specify the name of the file only, without the <code>.cfg.yml</code> extension.
-c <i>subdomain-name</i> or --cname <i>subdomain-name</i>	The subdomain name to prefix the CNAME DNS entry that routes to your website. Type: String Default: The environment name
-db or --database	Attaches a database to the environment. If you run eb create with the <code>--database</code> option, but without the <code>--database.username</code> and <code>--database.password</code> options, EB CLI prompts you for the master database user name and password.
-db.engine <i>engine</i> or --database.engine <i>engine</i>	The database engine type. If you run eb create with this option, then EB CLI launches the environment with a database attached even if you didn't run the command with the <code>--database</code> option. Type: String Valid values: <code>mysql</code> , <code>oracle-se1</code> , <code>postgres</code> , <code>sqlserver-ex</code> , <code>sqlserver-web</code> , <code>sqlserver-se</code>
-db.i <i>instance_type</i> or --database.instance <i>instance_type</i>	The type of Amazon EC2 instance to use for the database. If you run eb create with this option, then EB CLI launches the environment with a database attached even if you didn't run the command with the <code>--database</code> option. Type: String Valid values: See Option Values .
-db.pass <i>password</i> or --database.password <i>password</i>	The password for the database. If you run eb create with this option, then EB CLI launches the environment with a database attached even if you didn't run the command with the <code>--database</code> option.

Name	Description
<p><code>-db.size</code> <i>number_of_gigabytes</i></p> <p>or</p> <p><code>--database.size</code> <i>number_of_gigabytes</i></p>	<p>The number of gigabytes (GB) to allocate for database storage. If you run eb create with this option, then EB CLI launches the environment with a database attached even if you didn't run the command with the <code>--database</code> option.</p> <p>Type: Number</p> <p>Valid values:</p> <ul style="list-style-type: none"> • MySQL – 5 to 1024. The default is 5. • Postgres – 5 to 1024. The default is 5. • Oracle – 10 to 1024. The default is 10. • Microsoft SQL Server Express Edition – 30. • Microsoft SQL Server Web Edition – 30. • Microsoft SQL Server Standard Edition – 200.
<p><code>-db.user</code> <i>username</i></p> <p>or</p> <p><code>--database.username</code> <i>username</i></p>	<p>The user name for the database. If you run eb create with this option, then EB CLI launches the environment with a database attached even if you didn't run the command with the <code>--database</code> option. If you run eb create with the <code>--database</code> option, but without the <code>--database.username</code> and <code>--database.password</code> options, then EB CLI prompts you for the master database user name and password.</p>
<p><code>-db.version</code> <i>version</i></p> <p>or</p> <p><code>--database.version</code> <i>version</i></p>	<p>Allows you to specify the database engine version. If this flag is present, the environment will launch with a database with the specified version number, even if the <code>--database</code> flag is not present.</p>
<p><code>--elb-type</code> <i>type</i></p>	<p>The load balancer type (p. 470).</p> <p>Type: String</p> <p>Valid values: <code>classic</code>, <code>application</code>, <code>network</code></p> <p>Default: <code>application</code></p>
<p><code>-es</code></p> <p>or</p> <p><code>--enable-spot</code></p>	<p>Enable Spot Instance requests for your environment. For more details, see Auto Scaling group (p. 457).</p> <p>Related options:</p> <ul style="list-style-type: none"> • <code>--instance-types</code> • <code>--on-demand-base-capacity</code> • <code>--on-demand-above-base-capacity</code> • <code>--spot-max-price</code>
<p><code>--env-group-suffix</code> <i>groupname</i></p>	<p>Group name to append to the environment name. Only for use with Compose Environments (p. 881).</p>
<p><code>--envvars</code></p>	<p>Environment properties (p. 516) in a comma-separated list with the format <i>name=value</i>. See Configuring environment properties (p. 518) for limits.</p>

Name	Description
<p><code>-ip <i>profile_name</i></code> or <code>--instance-profile <i>profile_name</i></code></p>	<p>The instance profile with the IAM role with the temporary security credentials that your application needs to access AWS resources.</p>
<p><code>-it</code> or <code>--instance-types <i>type1</i> [, <i>type2</i> ...]</code></p>	<p>A comma-separated list of Amazon EC2 instance types you want your environment to use. If you don't specify this option, Elastic Beanstalk provides default instance types.</p> <p>For more details, see Amazon EC2 instances (p. 451) and Auto Scaling group (p. 457).</p>
<p><code>-i</code> or <code>--instance_type</code></p>	<p>The Amazon EC2 instance type you want your environment to use. If you don't specify this option, Elastic Beanstalk provides a default instance type.</p> <p>For more details, see Amazon EC2 instances (p. 451).</p> <p>Note The <code>--instance_type</code> option is obsolete. It's replaced by the newer and more powerful <code>--instance-types</code> option. The new option allows you to specify a list of one or more instance types for your environment. The first value on that list is equivalent to the value of the <code>--instance_type</code> option. The recommended way to specify instance types is by using the new option.</p>
<p><code>-k <i>key_name</i></code> or <code>--keyname <i>key_name</i></code></p>	<p>The name of the Amazon EC2 key pair to use with the Secure Shell (SSH) client to securely log in to the Amazon EC2 instances running your Elastic Beanstalk application. If you include this option with the eb create command, the value you provide overwrites any key name that you might have specified with eb init.</p> <p>Valid values: An existing key name that is registered with Amazon EC2</p>
<p><code>-im <i>number-of-instances</i></code> or <code>--min-instances <i>number-of-instances</i></code></p>	<p>The minimal number of Amazon EC2 instances you require your environment to have.</p> <p>Type: Number (integer)</p> <p>Default: 1</p> <p>Valid values: 1 to 10000</p>
<p><code>-ix <i>number-of-instances</i></code> or <code>--max-instances <i>number-of-instances</i></code></p>	<p>The maximal number of Amazon EC2 instances you allow your environment to have.</p> <p>Type: Number (integer)</p> <p>Default: 4</p> <p>Valid values: 1 to 10000</p>

Name	Description
<code>--modules <i>component-a</i> <i>component-b</i></code>	<p>List of component environments to create. Only for use with Compose Environments (p. 881).</p>
<code>-sb</code> or <code>--on-demand-base-capacity</code>	<p>The minimal number of On-Demand Instances that your Auto Scaling group provisions before considering Spot Instances as your environment scales up.</p> <p>This option can only be specified with the <code>--enable-spot</code> option. For more details, see Auto Scaling group (p. 457).</p> <p>Type: Number (integer)</p> <p>Default: 0</p> <p>Valid values: 0 to <code>--max-instances</code> (when absent: <code>MaxSize</code> option in aws:autoscaling:asg (p. 555) namespace)</p>
<code>-sp</code> or <code>--on-demand-above-base-capacity</code>	<p>The percentage of On-Demand Instances as part of additional capacity that your Auto Scaling group provisions beyond the number of instances specified by the <code>--on-demand-base-capacity</code> option.</p> <p>This option can only be specified with the <code>--enable-spot</code> option. For more details, see Auto Scaling group (p. 457).</p> <p>Type: Number (integer)</p> <p>Default: 0 for a Single Instance environment; 70 for a Load Balanced environment</p> <p>Valid values: 0 to 100</p>

Name	Description
<p><code>-p <i>platform-version</i></code> or <code>--platform <i>platform-version</i></code></p>	<p>The platform version (p. 29) to use. You can specify a platform, a platform and version, a platform branch, a solution stack name, or a solution stack ARN. For example:</p> <ul style="list-style-type: none"> • <code>php</code>, <code>PHP</code>, <code>node.js</code>—The latest platform version for the specified platform • <code>php-7.2</code>, <code>"PHP 7.2"</code>—The recommended (typically latest) PHP 7.2 platform version • <code>"PHP 7.2 running on 64bit Amazon Linux"</code>—The recommended (typically latest) PHP platform version in this platform branch • <code>"64bit Amazon Linux 2017.09 v2.6.3 running PHP 7.1"</code>—The PHP platform version specified by this solution stack name • <code>"arn:aws:elasticbeanstalk:us-east-2::platform/PHP 7.1 running on 64bit Amazon Linux/2.6.3"</code>—The PHP platform version specified by this solution stack ARN <p>Use eb platform list (p. 916) to get a list of available configurations.</p> <p>If you specify the <code>--platform</code> option, it overrides the value that was provided during <code>eb init</code>.</p>
<p><code>-pr</code> or <code>--process</code></p>	<p>Preprocess and validate the environment manifest and configuration files in the source bundle. Validating configuration files can identify issues prior to deploying the application version to an environment.</p>
<p><code>-r <i>region</i></code> or <code>--region <i>region</i></code></p>	<p>The AWS Region in which you want to deploy the application.</p> <p>For the list of values you can specify for this option, see AWS Elastic Beanstalk Endpoints and Quotas in the <i>AWS General Reference</i>.</p>
<p><code>--sample</code></p>	<p>Deploy the sample application to the new environment instead of the code in your repository.</p>
<p><code>--scale <i>number-of-instances</i></code></p>	<p>Launch with the specified number of instances</p>
<p><code>--service-role <i>servicerole</i></code></p>	<p>Assign a non-default service role to the environment.</p> <p>Note Do not enter an ARN, just the role name. Elastic Beanstalk prefixes the role name with the correct values to create the resulting ARN internally.</p>

Name	Description
<code>--single</code>	<p>Create the environment with a single Amazon EC2 instance and without a load balancer.</p> <p>Warning A single-instance environment isn't production ready. If the instance becomes unstable during deployment, or Elastic Beanstalk terminates and restarts the instance during a configuration update, your application can be unavailable for a period of time. Use single-instance environments for development, testing, or staging. Use load-balanced environments for production.</p>
<code>-sm</code> or <code>--spot-max-price</code>	<p>The maximum price per unit hour, in US\$, that you're willing to pay for a Spot Instance.</p> <p>This option can only be specified with the <code>--enable-spot</code> option. For more details, see Auto Scaling group (p. 457).</p> <p>Type: Number (float)</p> <p>Default: The On-Demand price</p> <p>Valid values: 0.001 to 20.0</p>
<code>--tags</code> <i>key1=value1[,key2=value2]</i>	<p>Tag the resources in your environment. Tags are specified as a comma-separated list of <code>key=value</code> pairs.</p> <p>For more details, see Tagging environments (p. 513).</p>
<code>-t worker</code> or <code>--tier worker</code>	<p>Create a worker environment. Omit this option to create a web server environment.</p>
<code>--timeout</code> <i>minutes</i>	<p>Set number of minutes before the command times out.</p>
<code>--version</code> <i>version_label</i>	<p>Specifies the application version that you want deployed to the environment instead of the application source code in the local project directory.</p> <p>Type: String</p> <p>Valid values: An existing application version label</p>
<code>--vpc</code>	<p>Configure a VPC for your environment. When you include this option, the EB CLI prompts you to enter all required settings prior to launching the environment.</p>
<code>--vpc.dbsubnets</code> <i>subnet1,subnet2</i>	<p>Specifies subnets for database instances in a VPC. Required when <code>--vpc.id</code> is specified.</p>
<code>--vpc.ec2subnets</code> <i>subnet1,subnet2</i>	<p>Specifies subnets for Amazon EC2 instances in a VPC. Required when <code>--vpc.id</code> is specified.</p>

Name	Description
<code>--vpc.elbpublic</code>	Launches your Elastic Load Balancing load balancer in a public subnet in your VPC. You can't specify this option with the <code>--tier worker</code> or <code>--single</code> options.
<code>--vpc.elbsubnets</code> <i>subnet1, subnet2</i>	Specifies subnets for the Elastic Load Balancing load balancer in a VPC. You can't specify this option with the <code>--tier worker</code> or <code>--single</code> options.
<code>--vpc.id</code> <i>ID</i>	Launches your environment in the specified VPC.
<code>--vpc.publicip</code>	Launches your Amazon EC2 instances in a public subnet in your VPC. You can't specify this option with the <code>--tier worker</code> option.
<code>--vpc.securitygroups</code> <i>securitygroup1, securitygroup2</i>	Specifies security group IDs. Required when <code>--vpc.id</code> is specified.
Common options (p. 935)	

Output

If successful, the command prompts you with questions and then returns the status of the create operation. If there were problems during the launch, you can use the [eb events \(p. 904\)](#) operation to get more details.

If you enabled CodeBuild support in your application, **eb create** displays information from CodeBuild as your code is built. For information about CodeBuild support in Elastic Beanstalk, see [Using the EB CLI with AWS CodeBuild \(p. 868\)](#).

Examples

The following example creates an environment in interactive mode.

```
$ eb create
Enter Environment Name
(default is tmp-dev): ENTER
Enter DNS CNAME prefix
(default is tmp-dev): ENTER
Select a load balancer type
1) classic
2) application
3) network
(default is 2): ENTER
Environment details for: tmp-dev
  Application name: tmp
  Region: us-east-2
  Deployed Version: app-141029_145448
  Environment ID: e-um3yfrzq22
  Platform: 64bit Amazon Linux 2014.09 v1.0.9 running PHP 5.5
  Tier: WebServer-Standard-1.0
  CNAME: tmp-dev.elasticbeanstalk.com
```

```
Updated: 2014-10-29 21:54:51.063000+00:00
Printing Status:
...
```

The following example also creates an environment in interactive mode. In this example, your project directory doesn't have application code. The command deploys a sample application and downloads it to your local project directory.

```
$ eb create
Enter Environment Name
(default is tmp-dev): ENTER
Enter DNS CNAME prefix
(default is tmp-dev): ENTER
Select a load balancer type
1) classic
2) application
3) network
(default is 2): ENTER
NOTE: The current directory does not contain any source code. Elastic Beanstalk is
launching the sample application instead.
Do you want to download the sample application into the current directory?
(Y/n): ENTER
INFO: Downloading sample application to the current directory.
INFO: Download complete.
Environment details for: tmp-dev
  Application name: tmp
  Region: us-east-2
  Deployed Version: Sample Application
  Environment ID: e-um3yfrzq22
  Platform: 64bit Amazon Linux 2014.09 v1.0.9 running PHP 5.5
  Tier: WebServer-Standard-1.0
  CNAME: tmp-dev.elasticbeanstalk.com
  Updated: 2017-11-08 21:54:51.063000+00:00
Printing Status:
...
```

The following command creates an environment without displaying any prompts.

```
$ eb create dev-env
Creating application version archive "app-160312_014028".
Uploading test/app-160312_014028.zip to S3. This may take a while.
Upload Complete.
Application test has been created.
Environment details for: dev-env
  Application name: test
  Region: us-east-2
  Deployed Version: app-160312_014028
  Environment ID: e-6fgpkjxyyi
  Platform: 64bit Amazon Linux 2015.09 v2.0.8 running PHP 5.6
  Tier: WebServer-Standard
  CNAME: UNKNOWN
  Updated: 2016-03-12 01:40:33.614000+00:00
Printing Status:
...
```

The following command creates an environment in a custom VPC.

```
$ eb create dev-vpc --vpc.id vpc-0ce8dd99 --vpc.elbsubnets subnet-b356d7c6,subnet-02f74b0c
--vpc.ec2subnets subnet-0bb7f0cd,subnet-3b6697c1 --vpc.securitygroup sg-70cff265
Creating application version archive "app-160312_014309".
Uploading test/app-160312_014309.zip to S3. This may take a while.
```

```
Upload Complete.
Environment details for: dev-vpc
  Application name: test
  Region: us-east-2
  Deployed Version: app-160312_014309
  Environment ID: e-pqkqip3mns
  Platform: 64bit Amazon Linux 2015.09 v2.0.8 running Java 8
  Tier: WebServer-Standard
  CNAME: UNKNOWN
  Updated: 2016-03-12 01:43:14.057000+00:00
Printing Status:
...
```

eb deploy

Description

Deploys the application source bundle from the initialized project directory to the running application.

If `git` is installed, EB CLI uses the `git archive` command to create a `.zip` file from the contents of the most recent `git commit` command.

However, when `.ebignore` is present in your project directory, the EB CLI doesn't use `git` commands and semantics to create your source bundle. This means that EB CLI ignores files specified in `.ebignore`, and includes all other files. In particular, it includes uncommitted source files.

Note

You can configure the EB CLI to deploy an artifact from your build process instead of creating a ZIP file of your project folder. See [Deploying an artifact instead of the project folder \(p. 863\)](#) for details.

Syntax

eb deploy

eb deploy *environment-name*

Options

Name	Description
<code>-l</code> <i>version_label</i> or <code>--label</code> <i>version_label</i>	Specify a label to use for the version that the EB CLI creates. If the label has already been used, the EB CLI redeploys the previous version with that label. Type: String
<code>--env-group-suffix</code> <i>groupname</i>	Group name to append to the environment name. Only for use with Compose Environments (p. 881) .
<code>-m</code> " <i>version_description</i> " or <code>--message</code> " <i>version_description</i> "	The description for the application version, enclosed in double quotation marks. Type: String

Name	Description
<code>--modules <i>component-a component-b</i></code>	List of components to update. Only for use with Compose Environments (p. 881) .
<code>-p</code> or <code>--process</code>	Preprocess and validate the environment manifest and configuration files in the source bundle. Validating configuration files can identify issues prior to deploying the application version to an environment.
<code>--source codecommit/<i>repository-name/branch-name</i></code>	CodeCommit repository and branch. See Using the EB CLI with AWS CodeCommit (p. 871) .
<code>--staged</code>	Deploy files staged in the git index instead of the HEAD commit.
<code>--timeout <i>minutes</i></code>	The number of minutes before the command times out.
<code>--version <i>version_label</i></code>	An existing application version to deploy. Type: String
Common options (p. 935)	

Output

If successful, the command returns the status of the `deploy` operation.

If you enabled CodeBuild support in your application, **eb deploy** displays information from CodeBuild as your code is built. For information about CodeBuild support in Elastic Beanstalk, see [Using the EB CLI with AWS CodeBuild \(p. 868\)](#).

Example

The following example deploys the current application.

```
$ eb deploy
2018-07-11 21:05:22 INFO: Environment update is starting.
2018-07-11 21:05:27 INFO: Deploying new version to instance(s).
2018-07-11 21:05:53 INFO: New application version was deployed to running EC2 instances.
2018-07-11 21:05:53 INFO: Environment update completed successfully.
```

eb events

Description

Returns the most recent events for the environment.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command also returns the most recent events for the builder environment.

Syntax

eb events

eb events *environment-name*

Options

Name	Description
-f	Streams events. To cancel, press CTRL+C.
or	
--follow	

Output

If successful, the command returns recent events.

Example

The following example returns the most recent events.

```
$ eb events
2014-10-29 21:55:39 INFO createEnvironment is starting.
2014-10-29 21:55:40 INFO Using elasticbeanstalk-us-west-2-111122223333 as Amazon S3
storage bucket for environment data.
2014-10-29 21:55:57 INFO Created load balancer named: awseb-e-r-AWSEBLoa-
NSKUOK5X6Z9J
2014-10-29 21:56:16 INFO Created security group named: awseb-e-rxgrhjr9bx-stack-
AWSEBSecurityGroup-1UUHU5LZ20ZY7
2014-10-29 21:57:18 INFO Waiting for EC2 instances to launch. This may take a few
minutes.
2014-10-29 21:57:18 INFO Created Auto Scaling group named: awseb-e-rxgrhjr9bx-stack-
AWSEBAutoScalingGroup-1TE320ZCJ9RPD
2014-10-29 21:57:22 INFO Created Auto Scaling group policy named:
arn:aws:autoscaling:us-east-2:11122223333:scalingPolicy:2cced9e6-859b-421a-
be63-8ab34771155a:autoScalingGroupName/awseb-e-rxgrhjr9bx-stack-
AWSEBAutoScalingGroup-1TE320ZCJ9RPD:policyName/awseb-e-rxgrhjr9bx-stack-
AWSEBAutoScalingScaleUpPolicy-1I2ZSNVU4APRY
2014-10-29 21:57:22 INFO Created Auto Scaling group policy named:
arn:aws:autoscaling:us-east-2:11122223333:scalingPolicy:1f08b863-
bf65-415a-b584-b7fa3a69a0d5:autoScalingGroupName/awseb-e-rxgrhjr9bx-stack-
AWSEBAutoScalingGroup-1TE320ZCJ9RPD:policyName/awseb-e-rxgrhjr9bx-stack-
AWSEBAutoScalingScaleDownPolicy-1E3G7PZKZPSOG
2014-10-29 21:57:25 INFO Created CloudWatch alarm named: awseb-e-rxgrhjr9bx-stack-
AWSEBCloudwatchAlarmLow-VF5EJ549FZBL
2014-10-29 21:57:25 INFO Created CloudWatch alarm named: awseb-e-rxgrhjr9bx-stack-
AWSEBCloudwatchAlarmHigh-LA9YEW306WJO
2014-10-29 21:58:50 INFO Added EC2 instance 'i-c7ee492d' to Auto ScalingGroup
'awseb-e-rxgrhjr9bx-stack-AWSEBAutoScalingGroup-1TE320ZCJ9RPD'.
2014-10-29 21:58:53 INFO Successfully launched environment: tmp-dev
2014-10-29 21:59:14 INFO Environment health has been set to GREEN
2014-10-29 21:59:43 INFO Adding instance 'i-c7ee492d' to your environment.
```

eb health

Description

Returns the most recent health for the environment.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command also returns the most recent health for the builder environment.

Syntax

eb health

eb health *environment-name*

Options

Name	Description
-r or --refresh	Show health information interactively and update every 10 seconds as new information is reported.
--mono	Don't display color in output.

Output

If successful, the command returns recent health.

Example

The following example returns the most recent health information for a Linux environment.

```
~/project $ eb health
elasticBeanstalkExa-env                               Ok
2015-07-08 23:13:20
WebServer                                             Ruby
2.1 (Puma)
total          ok    warning  degraded  severe  info  pending  unknown
5             5      0        0         0       0     0        0

instance-id   status    cause
health
Overall       Ok
i-d581497d   Ok
i-d481497c   Ok
i-136e00c0   Ok
i-126e00c1   Ok
i-8b2cf575   Ok

instance-id   r/sec    %2xx    %3xx    %4xx    %5xx    p99    p90    p75    p50
p10
Overall       671.8   100.0   0.0    0.0    0.0    0.003  0.002  0.001  0.001
0.000
i-d581497d   143.0   1430    0      0      0      0.003  0.002  0.001  0.001
0.000
i-d481497c   128.8   1288    0      0      0      0.003  0.002  0.001  0.001
0.000
i-136e00c0   125.4   1254    0      0      0      0.004  0.002  0.001  0.001
0.000
i-126e00c1   133.4   1334    0      0      0      0.003  0.002  0.001  0.001
0.000
i-8b2cf575   141.2   1412    0      0      0      0.003  0.002  0.001  0.001
0.000

instance-id   type     az    running  load 1  load 5    user%  nice%  system%  idle
% iowait%
cpu
```

i-d581497d	t2.micro	1a	12 mins	0.0	0.04	6.2	0.0	1.0
92.5	0.1							
i-d481497c	t2.micro	1a	12 mins	0.01	0.09	5.9	0.0	1.6
92.4	0.1							
i-136e00c0	t2.micro	1b	12 mins	0.15	0.07	5.5	0.0	0.9
93.2	0.0							
i-126e00c1	t2.micro	1b	12 mins	0.17	0.14	5.7	0.0	1.4
92.7	0.1							
i-8b2cf575	t2.micro	1c	1 hour	0.19	0.08	6.5	0.0	1.2
92.1	0.1							
instance-id	status	id	version	ago				
			deployments					
i-d581497d	Deployed	1	Sample Application	12 mins				
i-d481497c	Deployed	1	Sample Application	12 mins				
i-136e00c0	Deployed	1	Sample Application	12 mins				
i-126e00c1	Deployed	1	Sample Application	12 mins				
i-8b2cf575	Deployed	1	Sample Application	1 hour				

eb init

Description

Sets default values for Elastic Beanstalk applications created with EB CLI by prompting you with a series of questions.

Note

The values you set with `init` apply only to the current directory and repository.

Syntax

eb init

eb init *application-name*

Options

If you run **eb init** without specifying the `--platform` option, the EB CLI prompts you to enter a value for each setting.

Note

To use **eb init** to create a new key pair, you must have `ssh-keygen` installed on your local machine and available from the command line.

Name	Description
-i	Forces EB CLI to prompt you to provide a value for every eb init command option.
--interactive	
	<p>Note</p> <p>The <code>init</code> command prompts you to provide values for eb init command options that do not have a (default) value. After the first time you run the eb init command in a directory, EB CLI might not prompt you about any command options. Therefore, use the <code>--interactive</code> option when you want to change a setting that you previously set.</p>

Name	Description	
<p><code>-k <i>keyname</i></code> <code>--keyname <i>keyname</i></code></p>	<p>The name of the Amazon EC2 key pair to use with the Secure Shell (SSH) client to securely log in to the Amazon EC2 instances running your Elastic Beanstalk application.</p>	
<p><code>--modules <i>folder-1</i></code> <code><i>folder-2</i></code></p>	<p>List of child directories to initialize. Only for use with Compose Environments (p. 881).</p>	
<p><code>-p <i>platform-version</i></code> <code>--platform <i>platform-version</i></code></p>	<p>The platform version (p. 29) to use. You can specify a platform, a platform and version, a platform branch, a solution stack name, or a solution stack ARN. For example:</p> <ul style="list-style-type: none"> <code>php</code>, <code>PHP</code>, <code>node.js</code>—The latest platform version for the specified platform <code>php-7.2</code>, <code>"PHP 7.2"</code>—The recommended (typically latest) PHP 7.2 platform version <code>"PHP 7.2 running on 64bit Amazon Linux"</code>—The recommended (typically latest) PHP platform version in this platform branch <code>"64bit Amazon Linux 2017.09 v2.6.3 running PHP 7.1"</code>—The PHP platform version specified by this solution stack name <code>"arn:aws:elasticbeanstalk:us-east-2::platform/PHP 7.1 running on 64bit Amazon Linux/2.6.3"</code>—The PHP platform version specified by this solution stack ARN <p>Use eb platform list (p. 916) to get a list of available configurations.</p> <p>Specify the <code>--platform</code> option to skip interactive configuration.</p> <p>Note When you specify this option, then EB CLI does not prompt you for values for any other options. Instead, it assumes default values for each option. You can specify options for anything for which you do not want to use default values.</p>	
<p><code>--source</code> <code>codecommit/</code> <code><i>repository-</i></code> <code><i>name/branch-name</i></code></p>	<p>CodeCommit repository and branch. See Using the EB CLI with AWS CodeCommit (p. 871).</p>	
<p><code>--tags <i>key1=value1</i> [, <i>key2=value2</i>]</code></p>	<p>Tag your application. Tags are specified as a comma-separated list of <code>key=value</code> pairs.</p> <p>For more details, see Tagging applications (p. 349).</p>	
<p>Common options (p. 935)</p>		

CodeBuild support

If you run **eb init** in a folder that contains a `buildspec.yml` file, Elastic Beanstalk parses the file for an **eb_codebuild_settings** entry with options specific to Elastic Beanstalk. For information about CodeBuild support in Elastic Beanstalk, see [Using the EB CLI with AWS CodeBuild \(p. 868\)](#).

Output

If successful, the command guides you through setting up a new Elastic Beanstalk application through a series of prompts.

Example

The following example request initializes EB CLI and prompts you to enter information about your application. Replace *placeholder* text with your own values.

```
$ eb init -i
Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
4) eu-west-1 : Europe (Ireland)
5) eu-central-1 : Europe (Frankfurt)
6) ap-south-1 : Asia Pacific (Mumbai)
7) ap-southeast-1 : Asia Pacific (Singapore)
...
(default is 3): 3

Select an application to use
1) HelloWorldApp
2) NewApp
3) [ Create new Application ]
(default is 3): 3

Enter Application Name
(default is "tmp"):
Application tmp has been created.

It appears you are using PHP. Is this correct?
(y/n): y

Select a platform branch.
1) PHP 7.2 running on 64bit Amazon Linux
2) PHP 7.1 running on 64bit Amazon Linux (Deprecated)
3) PHP 7.0 running on 64bit Amazon Linux (Deprecated)
4) PHP 5.6 running on 64bit Amazon Linux (Deprecated)
5) PHP 5.5 running on 64bit Amazon Linux (Deprecated)
6) PHP 5.4 running on 64bit Amazon Linux (Deprecated)
(default is 1): 1

Do you want to set up SSH for your instances?
(y/n): y

Select a keypair.
1) aws-eb
2) [ Create new KeyPair ]
(default is 2): 1
```

eb labs

Description

Subcommands of **eb labs** support work-in-progress or experimental functionality. These commands may be removed or reworked in future versions of the EB CLI and are not guaranteed to be forward compatible.

For a list of available subcommands and descriptions, run **eb labs --help**.

eb list

Description

Lists all environments in the current application or all environments in all applications, as specified by the `--all` option.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command also lists the builder environments.

Syntax

eb list

Options

Name	Description
-a or --all	Lists all environments from all applications.
-v or --verbose	Provides more detailed information about all environments, including instances.
Common options (p. 935)	

Output

If successful, the command returns a list of environment names in which your current environment is marked with an asterisk (*).

Example 1

The following example lists your environments and indicates that `tmp-dev` is your default environment.

```
$ eb list
```

```
* tmp-dev
```

Example 2

The following example lists your environments with additional details.

```
$ eb list --verbose
Region: us-west-2
Application: tmp
  Environments: 1
    * tmp-dev : ['i-c7ee492d']
```

eb local

Description

Use **eb local run** to run your application's containers locally in Docker. Check the application's container status with **eb local status**. Open the application in a web browser with **eb local open**. Retrieve the location of the application's logs with **eb local logs**.

eb local setenv and **eb local printenv** let you set and view environment variables that are provided to the Docker containers that you run locally with **eb local run**.

You must run all **eb local** commands in the project directory of a Docker application that has been initialized as an EB CLI repository by using **eb init**.

Note

Use **eb local** on a local computer running Linux or macOS. The command doesn't support Windows.

Before using the command on macOS, install Docker for Mac, and ensure that boot2docker isn't installed (or isn't in the execution path). The **eb local** command tries to use boot2docker if it's present, but doesn't work well with it on macOS.

Syntax

eb local run

eb local status

eb local open

eb local logs

eb local setenv

eb local printenv

Options

eb local run

Name	Description
<code>--envvars</code> <code>key1=value1, key2=value2</code>	Sets environment variables that the EB CLI will pass to the local Docker containers. In multicontainer environments, all variables are passed to all containers.

Name	Description
<code>--port <i>hostport</i></code>	Maps a port on the host to the exposed port on the container. If you don't specify this option, the EB CLI uses the same port on both host and container. This option works only with single container applications.
Common options (p. 935)	

eb local status

eb local open

eb local logs

eb local setenv

eb local printenv

Name	Description
Common options (p. 935)	

Output

eb local run

Status messages from Docker. Remains active as long as application is running. Press **Ctrl+C** to stop the application.

eb local status

The status of each container used by the application, running or not.

eb local open

Opens the application in a web browser and exits.

eb local logs

The location of the logs generated in your project directory by applications running locally under **eb local run**.

eb local setenv

None

eb local printenv

The name and values of environment variables set with **eb local setenv**.

Examples

eb local run

```
~/project$ eb local run
Creating elasticbeanstalk_phpapp_1...
Creating elasticbeanstalk_nginxproxy_1...
```

```
Attaching to elasticbeanstalk_phpapp_1, elasticbeanstalk_nginxproxy_1
phpapp_1      | [23-Apr-2015 23:24:25] NOTICE: fpm is running, pid 1
phpapp_1      | [23-Apr-2015 23:24:25] NOTICE: ready to handle connections
```

eb local status

View the status of your local containers:

```
~/project$ eb local status
Platform: 64bit Amazon Linux 2014.09 v1.2.1 running Multi-container Docker 1.3.3 (Generic)
Container name: elasticbeanstalk_nginxproxy_1
Container ip: 127.0.0.1
Container running: True
Exposed host port(s): 80
Full local URL(s): 127.0.0.1:80

Container name: elasticbeanstalk_phpapp_1
Container ip: 127.0.0.1
Container running: True
Exposed host port(s): None
Full local URL(s): None
```

eb local logs

View the log path for the current project:

```
~/project$ eb local logs
Elastic Beanstalk will write logs locally to /home/user/project/.elasticbeanstalk/logs/local.
Logs were most recently created 3 minutes ago and written to /home/user/project/.elasticbeanstalk/logs/local/150420_234011665784.
```

eb local setenv

Set environment variables for use with **eb local run**.

```
~/project$ eb local setenv PARAM1=value
```

Print environment variables set with **eb local setenv**.

```
~/project$ eb local printenv
Environment Variables:
PARAM1=value
```

eb logs

Description

The **eb logs** command has two distinct purposes: to enable or disable log streaming to CloudWatch Logs, and to retrieve instance logs or CloudWatch Logs logs. With the `--cloudwatch-logs (-cw)` option, the command enables or disables log streaming. Without this option, it retrieves logs.

When retrieving logs, specify the `--all`, `--zip`, or `--stream` option to retrieve complete logs. If you don't specify any of these options, Elastic Beanstalk retrieves tail logs.

The command processes logs for the specified or default environment. Relevant logs vary by container type. If the root directory contains a `platform.yaml` file specifying a custom platform, this command also processes logs for the builder environment.

For more information, see [the section called “CloudWatch Logs” \(p. 743\)](#).

Syntax

To enable or disable log streaming to CloudWatch Logs:

```
eb logs --cloudwatch-logs [enable | disable] [--cloudwatch-log-source instance |
environment-health | all] [environment-name]
```

To retrieve instance logs:

```
eb logs [-all | --zip | --stream] [--cloudwatch-log-source instance] [--instance instance-
id] [--log-group log-group] [environment-name]
```

To retrieve environment health logs:

```
eb logs [-all | --zip | --stream] --cloudwatch-log-source environment-health [environment-
name]
```

Options

Name	Description
-cw [enable disable] or --cloudwatch-logs [enable disable]	Enables or disables log streaming to CloudWatch Logs. If no argument is supplied, log streaming is enabled. If the --cloudwatch-log-source (-cls) option isn't specified in addition, instance log streaming is enabled or disabled.
-cls instance environment-health all or --cloudwatch-log-source instance environment-health all	Specifies the source of logs when working with CloudWatch Logs. With the enable or disable form of the command, these are the logs for which to enable or disable CloudWatch Logs streaming. With the retrieval form of the command, these are the logs to retrieve from CloudWatch Logs. Valid values: <ul style="list-style-type: none"> • With --cloudwatch-logs (enable or disable) – instance environment-health all • Without --cloudwatch-logs (retrieve) – instance environment-health Value meanings: <ul style="list-style-type: none"> • instance (default) – Instance logs • environment-health – Environment health logs (supported only when enhanced health is enabled in the environment) • all – Both log sources
-a or --all	Retrieves complete logs and saves them to the .elasticbeanstalk/logs directory.

Name	Description
-z or --zip	Retrieves complete logs, compresses them into a .zip file, and then saves the file to the .elasticbeanstalk/logs directory.
--stream	Streams (continuously outputs) complete logs. With this option, the command keeps running until you interrupt it (press Ctrl+C).
-i <i>instance-id</i> or --instance <i>instance-id</i>	Retrieves logs for the specified instance only.
-g <i>log-group</i> or --log-group <i>log-group</i>	<p>Specifies the CloudWatch Logs log group from which to retrieve logs. The option is valid only when instance log streaming to CloudWatch Logs is enabled.</p> <p>If instance log streaming is enabled, and you don't specify the --log-group option, the default log group is one of the following:</p> <ul style="list-style-type: none"> Linux platforms – /aws/elasticbeanstalk/<i>environment-name</i>/var/log/eb-activity.log Windows platforms – /aws/elasticbeanstalk/<i>environment-name</i>/EBDeploy-Log <p>For information about the log group corresponding to each log file, see How Elastic Beanstalk sets up CloudWatch Logs (p. 746).</p>
Common options (p. 935)	

Output

By default, displays the logs directly in the terminal. Uses a paging program to display the output. Press **q** or **q** to exit.

With --stream, shows existing logs in the terminal and keeps running. Press **Ctrl+C** to exit.

With --all and --zip, saves the logs to local files and displays the file location.

Examples

The following example enables instance log streaming to CloudWatch Logs.

```
$ eb logs -cw enable
Enabling instance log streaming to CloudWatch for your environment
After the environment is updated you can view your logs by following the link:
https://console.aws.amazon.com/cloudwatch/home?region=us-east-1#logs:prefix=/aws/
elasticbeanstalk/environment-name/
Printing Status:
2018-07-11 21:05:20      INFO: Environment update is starting.
2018-07-11 21:05:27      INFO: Updating environment environment-name's configuration
      settings.
2018-07-11 21:06:45      INFO: Successfully deployed new configuration to environment.
```

The following example retrieves instance logs into a `.zip` file.

```
$ eb logs --zip
Retrieving logs...
Logs were saved to /home/workspace/environment/.elasticbeanstalk/logs/150622_173444.zip
```

eb open

Description

Opens the public URL of your website in the default browser.

Syntax

eb open

eb open *environment-name*

Options

Name	Description
Common options (p. 935)	

Output

The command **eb open** does not have output. Instead, it opens the application in a browser window.

eb platform

Description

This command supports two different workspaces:

[Platform \(p. 916\)](#)

Use this workspace to manage custom platforms.

[Environment \(p. 921\)](#)

Use this workspace to select a default platform or show information about the current platform.

Elastic Beanstalk provides the shortcut **ebp** for **eb platform**.

Note

Windows PowerShell uses **ebp** as a command alias. If you're running the EB CLI in Windows PowerShell, use the long form of this command — **eb platform**.

Using eb platform for custom platforms

Lists the versions of the current platform and enables you to manage custom platforms.

Syntax

eb platform create [*version*] [*options*]

eb platform delete [*version*] [*options*]

eb platform events [*version*] [*options*]

eb platform init [*platform*] [*options*]

eb platform list [*options*]

eb platform logs [*version*] [*options*]

eb platform status [*version*] [*options*]

eb platform use [*platform*] [*options*]

Options

Name	Description
create [<i>version</i>] [<i>options</i>]	Build a new version of the platform. Learn more (p. 918) .
delete <i>version</i> [<i>options</i>]	Delete a platform version. Learn more (p. 919) .
events [<i>version</i>] [<i>options</i>]	Display the events from a platform version. Learn more (p. 919) .
init [<i>platform</i>] [<i>options</i>]	Initialize a platform repository. Learn more (p. 919) .
list [<i>options</i>]	List the versions of the current platform. Learn more (p. 920) .
logs [<i>version</i>] [<i>options</i>]	Display logs from the builder environment for a platform version. Learn more (p. 921) .
status [<i>version</i>] [<i>options</i>]	Display the status of the a platform version. Learn more (p. 921) .
use [<i>platform</i>] [<i>options</i>]	Select a different platform from which new versions are built. Learn more (p. 921) .
Common options (p. 935)	

Common options

All **eb platform** commands include the following common options.

Name	Description
-h OR --help	Shows a help message and exits.
--debug	Shows additional debugging output.
--quiet	Suppresses all output.

Name	Description
-v OR --verbose	Shows additional output.
--profile <i>PROFILE</i>	Uses the specified <i>PROFILE</i> from your credentials.
-r <i>REGION</i> OR --region <i>REGION</i>	Use the region <i>REGION</i> .
--no-verify-ssl	Do not verify AWS SSL certificates.

Eb platform create

Builds a new version of the platform and returns the ARN for the new version. If there is no builder environment running in the current region, this command launches one. The *version* and increment options (-M, -m, and -p) are mutually exclusive.

Options

Name	Description
<i>version</i>	If <i>version</i> isn't specified, creates a new version based on the most-recent platform with the patch version (N in n.n.N) incremented.
-M OR --major-increment	Increments the major version number (the N in N.n.n).
-m OR --minor-increment	Increments the minor version number (the N in n.N.n).
-p OR --patch-increment	Increments the patch version number (the N in n.n.N).
-i <i>INSTANCE_TYPE</i> OR --instance-type <i>INSTANCE_TYPE</i>	Use <i>INSTANCE_TYPE</i> as the instance type, such as t1.micro .
-ip <i>INSTANCE_PROFILE</i> OR	Use <i>INSTANCE_PROFILE</i> as the instance profile when creating AMIs for a custom platform.

Name	Description
<code>--instance-profile</code> <i>INSTANCE_PROFILE</i>	If the <code>-ip</code> option isn't specified, creates the instance profile <code>aws-elasticbeanstalk-custom-platform-ec2-role</code> and uses it for the custom platform.
<code>--tags</code> <i>key1=value1[,key2=value2]</i>	Tags your custom platform version. Tags are specified as a comma-separated list of <code>key=value</code> pairs. For more details, see Tagging custom platform versions (p. 48) .
<code>--timeout</code> <i>minutes</i>	Set number of minutes before the command times out.
<code>--vpc.id</code> <i>VPC_ID</i>	The ID of the VPC in which Packer builds.
<code>--vpc.subnets</code> <i>VPC_SUBNETS</i>	The VPC subnets in which Packer builds.
<code>--vpc.publicip</code>	Associates public IPs to EC2 instances launched.

Eb platform delete

Delete a platform version. The version isn't deleted if an environment is using that version.

Options

Name	Description
<i>version</i>	The version to delete. This value is required.
<code>--cleanup</code>	Remove all platform versions in the <code>Failed</code> state.
<code>--all-platforms</code>	If <code>--cleanup</code> is specified, remove all platform versions in the <code>Failed</code> state for all platforms.
<code>--force</code>	Do not require confirmation when deleting a version.

Eb platform events

Display the events from a platform version. If *version* is specified, display the events from that version, otherwise display the events from the current version.

Options

Name	Description
<i>version</i>	The version for which events are displayed. This value is required.
<code>-f</code> OR <code>--follow</code>	Continue to display events as they occur.

Eb platform init

Initialize a platform repository.

Options

Name	Description
<i>platform</i>	The name of the platform to initialize. This value is required, unless <code>-i</code> (interactive mode) is enabled.
<code>-i</code> OR <code>--interactive</code>	Use interactive mode.
<code>-k</code> <i>KEYNAME</i> OR <code>--keyname</code> <i>KEYNAME</i>	The default EC2 key name.

You can run this command in a directory that has been previously initialized, although you cannot change the workspace type if run in a directory that has been previously initialized.

To re-initialize with different options, use the `-i` option.

Eb platform list

List the versions of the platform associated with a workspace (directory) or a region.

The command returns different results depending on the type of workspace you run it in, as follows:

- In a platform workspace (a directory initialized by `eb platform init`), the command returns a list of all platform versions of the custom platform defined in the workspace. Add the `--all-platforms` or `--verbose` option to get a list of all platform versions of all custom platforms your account has in the region associated with the workspace.
- In an application workspace (a directory initialized by `eb init`), the command returns a list of all platform versions, both for platforms managed by Elastic Beanstalk and for your account's custom platforms. The list uses short platform version names, and some platform version variants might be combined. Add the `--verbose` option to get a detailed list with full names and all variants listed separately.
- In an uninitialized directory, the command only works with the `--region` option. It returns a list of all Elastic Beanstalk-managed platform versions supported in the region. The list uses short platform version names, and some platform version variants might be combined. Add the `--verbose` option to get a detailed list with full names and all variants listed separately.

Options

Name	Description
<code>-a</code> OR <code>--all-platforms</code>	Valid only in an initialized workspace (a directory initialized by <code>eb platform init</code> or <code>eb init</code>). Lists the platform versions of all custom platforms associated with your account.
<code>-s</code> <i>STATUS</i>	List only the platforms matching <i>STATUS</i> :

Name	Description
OR <code>--status <i>STATUS</i></code>	<ul style="list-style-type: none"> • Ready • Failed • Deleting • Creating

Eb platform logs

Display logs from the builder environment for a platform version.

Options

Name	Description
<code><i>version</i></code>	The version of the platform for which logs are displayed. If omitted, display logs from the current version.
<code>--stream</code>	Stream deployment logs that were set up with CloudWatch.

Eb platform status

Display the status of the a platform version.

Options

Name	Description
<code><i>version</i></code>	The version of the platform for which the status is retrieved. If omitted, display the status of the current version.

Eb platform use

Select a different platform from which new versions are built.

Options

Name	Description
<code><i>platform</i></code>	Specifies <i>platform</i> as the active version for this workspace. This value is required.

Using eb platform for environments

Lists supported platforms and enables you to set the default platform and platform version to use when you launch an environment. Use **eb platform list** to view a list of all supported platforms. Use **eb platform select** to change the platform for your project. Use **eb platform show** to view your project's selected platform.

Syntax

eb platform list

eb platform select

eb platform show

Options

Name	Description
list	List the version of the current platform.
select	Select the default platform.
show	Show information about the current platform.

Example 1

The following example lists the names of all configurations for all platforms that Elastic Beanstalk supports.

```
$ eb platform list
docker-1.5.0
glassfish-4.0-java-7-(preconfigured-docker)
glassfish-4.1-java-8-(preconfigured-docker)
go-1.3-(preconfigured-docker)
go-1.4-(preconfigured-docker)
iis-7.5
iis-8
iis-8.5
multi-container-docker-1.3.3-(generic)
node.js
php-5.3
php-5.4
php-5.5
python
python-2.7
python-3.4
python-3.4-(preconfigured-docker)
ruby-1.9.3
ruby-2.0-(passenger-standalone)
ruby-2.0-(puma)
ruby-2.1-(passenger-standalone)
ruby-2.1-(puma)
ruby-2.2-(passenger-standalone)
ruby-2.2-(puma)
tomcat-6
tomcat-7
tomcat-7-java-6
tomcat-7-java-7
tomcat-8-java-8
```

Example 2

The following example prompts you to choose from a list of platforms and the version that you want to deploy for the specified platform.

```
$ eb platform select
Select a platform.
1) PHP
2) Node.js
3) IIS
```

```
4) Tomcat
5) Python
6) Ruby
7) Docker
8) Multi-container Docker
9) GlassFish
10) Go
(default is 1): 5

Select a platform version.
1) Python 2.7
2) Python
3) Python 3.4 (Preconfigured - Docker)
```

Example 3

The following example shows information about the current default platform.

```
$ eb platform show
Current default platform: Python 2.7
New environments will be running: 64bit Amazon Linux 2014.09 v1.2.0 running Python 2.7

Platform info for environment "tmp-dev":
Current: 64bit Amazon Linux 2014.09 v1.2.0 running Python
Latest: 64bit Amazon Linux 2014.09 v1.2.0 running Python
```

eb printenv

Description

Prints all the environment properties in the command window.

Syntax

eb printenv

eb printenv *environment-name*

Options

Name	Description
Common options (p. 935)	

Output

If successful, the command returns the status of the `printenv` operation.

Example

The following example prints environment properties for the specified environment.

```
$ eb printenv
Environment Variables:
  PARAM1 = Value1
```

eb restore

Description

Rebuilds a terminated environment, creating a new environment with the same name, ID, and configuration. The environment name, domain name, and application version must be available for use in order for the rebuild to succeed.

Syntax

eb restore

eb restore *environment_id*

Options

Name	Description
Common options (p. 935)	

Output

The EB CLI displays a list of terminated environments that are available to restore.

Example

```
$ eb restore
Select a terminated environment to restore

#   Name      ID              Application Version   Date Terminated      Ago
3   gamma     e-s7mimej8e9   app-77e3-161213_211138 2016/12/14 20:32 PST  13 mins
2   beta      e-sj28uu2wia   app-77e3-161213_211125 2016/12/14 20:32 PST  13 mins
1   alpha     e-gia8mphu6q   app-77e3-161213_211109 2016/12/14 16:21 PST  4 hours

(Commands: Quit, Restore, # #)

Selected environment alpha
Application:    scorekeep
Description:    Environment created from the EB CLI using "eb create"
CNAME:         alpha.h23tbtbm92.us-east-2.elasticbeanstalk.com
Version:       app-77e3-161213_211109
Platform:      64bit Amazon Linux 2016.03 v2.1.6 running Java 8
Terminated:    2016/12/14 16:21 PST
Restore this environment? [y/n]: y

2018-07-11 21:04:20   INFO: restoreEnvironment is starting.
2018-07-11 21:04:39   INFO: Created security group named: sg-e2443f72
...
```

eb scale

Description

Scales the environment to always run on a specified number of instances, setting both the minimum and maximum number of instances to the specified number.

Syntax

eb scale *number-of-instances*

eb scale *number-of-instances environment-name*

Options

Name	Description
--timeout	The number of minutes before the command times out.
Common options (p. 935)	

Output

If successful, the command updates the number of minimum and maximum instances to run to the specified number.

Example

The following example sets the number of instances to 2.

```
$ eb scale 2
2018-07-11 21:05:22 INFO: Environment update is starting.
2018-07-11 21:05:27 INFO: Updating environment tmp-dev's configuration settings.
2018-07-11 21:08:53 INFO: Added EC2 instance 'i-5fce3d53' to Auto Scaling Group 'awseb-
e-2cpfjbra9a-stack-AWSEBAutoScalingGroup-7AXY7U13ZQ6E'.
2018-07-11 21:08:58 INFO: Successfully deployed new configuration to environment.
2018-07-11 21:08:59 INFO: Environment update completed successfully.
```

eb setenv

Description

Sets [environment properties \(p. 516\)](#) for the default environment.

Syntax

eb setenv *key=value*

You can include as many properties as you want, but the total size of all properties cannot exceed 4096 bytes. You can delete a variable by leaving the value blank. See [Configuring environment properties \(p. 518\)](#) for limits.

Note

If the `value` contains a [special character](#), you must escape that character by preceding it with a `\` character.

Options

Name	Description
--timeout	The number of minutes before the command times out.

Name	Description
Common options (p. 935)	

Output

If successful, the command displays that the environment update succeeded.

Example

The following example sets the environment variable ExampleVar.

```
$ eb setenv ExampleVar=ExampleValue
2018-07-11 21:05:25 INFO: Environment update is starting.
2018-07-11 21:05:29 INFO: Updating environment tmp-dev's configuration settings.
2018-07-11 21:06:50 INFO: Successfully deployed new configuration to environment.
2018-07-11 21:06:51 INFO: Environment update completed successfully.
```

The following command sets multiple environment properties. It adds the environment property named `foo` and sets its value to `bar`, changes the value of the `JDBC_CONNECTION_STRING` property, and deletes the `PARAM4` and `PARAM5` properties.

```
$ eb setenv foo=bar JDBC_CONNECTION_STRING=hello PARAM4= PARAM5=
```

eb ssh

Description

Note

This command does not work with environments running Windows Server instances.

Connect to a Linux Amazon EC2 instance in your environment using Secure Shell (SSH). If an environment has multiple running instances, EB CLI prompts you to specify which instance you want to connect to. To use this command, SSH must be installed on your local machine and available from the command line. Private key files must be located in a folder named `.ssh` under your user directory, and the EC2 instances in your environment must have public IP addresses.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command also connects to instances in the custom environment.

SSH keys

If you have not previously configured SSH, you can use the EB CLI to create a key when running `eb init`. If you have already run `eb init`, run it again with the `--interactive` option and select **Yes** and **Create New Keypair** when prompted to set up SSH. Keys created during this process will be stored in the proper folder by the EB CLI.

This command temporarily opens port 22 in your environment's security group for incoming traffic from 0.0.0.0/0 (all IP addresses) if no rules for port 22 are already in place. If you have configured your environment's security group to open port 22 to a restricted CIDR range for increased security, the EB CLI will respect that setting and forgo any changes to the security group. To override this behavior and force the EB CLI to open port 22 to all incoming traffic, use the `--force` option.

See [Security groups \(p. 455\)](#) for information on configuring your environment's security group.

Syntax

eb ssh

eb ssh *environment-name*

Options

Name	Description
-i or --instance	Specifies the instance ID of the instance to which you connect. We recommend that you use this option.
-n or --number	Specify the instance to connect to by number.
-o or --keep_open	Leave port 22 open on the security group after the SSH session ends.
--command	Execute a shell command on the specified instance instead of starting an SSH session.
--custom	Specify an SSH command to use instead of 'ssh -i keyfile'. Do not include the remote user and hostname.
--setup	Change the key pair assigned to the environment's instances (requires instances to be replaced).
--force	Open port 22 to incoming traffic from 0.0.0.0/0 in the environment's security group, even if the security group is already configured for SSH. Use this option if your environment's security group is configured to open port 22 to a restricted CIDR range that does not include the IP address that you are trying to connect from.
--timeout <i>minutes</i>	Set number of minutes before the command times out. Can only be used with the --setup argument.
Common options (p. 935)	

Output

If successful, the command opens an SSH connection to the instance.

Example

The following example connects you to the specified environment.

```

$ eb ssh
Select an instance to ssh into
1) i-96133799
2) i-5931e053
(default is 1): 1
INFO: Attempting to open port 22.
INFO: SSH port 22 open.
The authenticity of host '54.191.45.125 (54.191.45.125)' can't be established.
RSA key fingerprint is ee:69:62:df:90:f7:63:af:52:7c:80:60:1b:3b:51:a9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '54.191.45.125' (RSA) to the list of known hosts.

  _|  _|_ )
 _| (  /   Amazon Linux AMI
  _|\__|__|

https://aws.amazon.com/amazon-linux-ami/2014.09-release-notes/
No packages needed for security; 1 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-8-185 ~]$ ls
[ec2-user@ip-172-31-8-185 ~]$ exit
logout
Connection to 54.191.45.125 closed.
INFO: Closed port 22 on ec2 instance security group

```

eb status

Description

Provides information about the status of the environment.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command also provides information about the builder environment.

Syntax

eb status

eb status *environment-name*

Options

Name	Description
-v or --verbose	Provides more information about individual instances, such as their status with the Elastic Load Balancing load balancer.
Common options (p. 935)	

Output

If successful, the command returns the following information about the environment:

- Environment name
- Application name
- Deployed application version
- Environment ID
- Platform
- Environment tier
- CNAME
- Time the environment was last updated
- Status
- Health

If you use verbose mode, EB CLI also provides you with the number of running Amazon EC2 instances.

Example

The following example shows the status for the environment tmp-dev.

```
$ eb status
Environment details for: tmp-dev
  Application name: tmp
  Region: us-west-2
  Deployed Version: None
  Environment ID: e-2cpfjbra9a
  Platform: 64bit Amazon Linux 2014.09 v1.0.9 running PHP 5.5
  Tier: WebServer-Standard-1.0
  CNAME: tmp-dev.elasticbeanstalk.com
  Updated: 2014-10-29 21:37:19.050000+00:00
  Status: Launching
  Health: Grey
```

eb swap

Description

Swaps the environment's CNAME with the CNAME of another environment (for example, to avoid downtime when you update your application version).

Note

If you have more than two environments, you are prompted to select the name of the environment that is currently using your desired CNAME from a list of environments. To suppress this, you can specify the name of the environment to use by including the `-n` option when you run the command.

Syntax

eb swap

eb swap *environment-name*

Note

The *environment-name* is the environment for which you want a different CNAME. If you don't specify *environment-name* as a command line parameter when you run **eb swap**, EB CLI updates the CNAME of the default environment.

Options

Name	Description
-n or --destination_name	Specifies the name of the environment with which you want to swap CNAMEs. If you run eb swap without this option, then EB CLI prompts you to choose from a list of your environments.
Common options (p. 935)	

Output

If successful, the command returns the status of the swap operation.

Examples

The following example swaps the environment tmp-dev with live-env.

```
$ eb swap
Select an environment to swap with.
1) staging-dev
2) live-env
(default is 1): 2
2018-07-11 21:05:25 INFO: swapEnvironmentCNAMEs is starting.
2018-07-11 21:05:26 INFO: Swapping CNAMEs for environments 'tmp-dev' and 'live-env'.
2018-07-11 21:05:30 INFO: 'tmp-dev.elasticbeanstalk.com' now points to 'awseb-e-j-
AWSEBLoa-M7U21VXNLWHN-487871449.us-west-2.elb.amazonaws.com'.
2018-07-11 21:05:30 INFO: Completed swapping CNAMEs for environments 'tmp-dev' and
'live-env'.
```

The following example swaps the environment tmp-dev with the environment live-env but does not prompt you to enter or select a value for any settings.

```
$ eb swap tmp-dev --destination_name live-env
2018-07-11 21:18:12 INFO: swapEnvironmentCNAMEs is starting.
2018-07-11 21:18:13 INFO: Swapping CNAMEs for environments 'tmp-dev' and 'live-env'.
2018-07-11 21:18:17 INFO: 'tmp-dev.elasticbeanstalk.com' now points to 'awseb-e-j-
AWSEBLoa-M7U21VXNLWHN-487871449.us-west-2.elb.amazonaws.com'.
2018-07-11 21:18:17 INFO: Completed swapping CNAMEs for environments 'tmp-dev' and
'live-env'.
```

eb tags

Description

Add, delete, update, and list tags of an Elastic Beanstalk resource.

For details about resource tagging in Elastic Beanstalk, see [Tagging Elastic Beanstalk application resources \(p. 349\)](#).

Syntax

```
eb tags [environment-name] [--resource ARN] -l | --list
```

```
eb tags [environment-name] [--resource ARN] -a | --add key1=value1[,key2=value2 ...]
```

eb tags [*environment-name*] [--resource *ARN*] -u | --update *key1=value1* [, *key2=value2* ...]

eb tags [*environment-name*] [--resource *ARN*] -d | --delete *key1* [, *key2* ...]

You can combine the --add, --update, and --delete subcommand options in a single command. At least one of them is required. You can't combined any of these three subcommand options with --list.

Without any additional arguments, all of these commands list or modify tags of the default environment in the current directory's application. With an *environment-name* argument, the commands list or modify tags of that environment. With the --resource option, the commands list or modify tags of any Elastic Beanstalk resource – an application, an environment, an application version, a saved configuration, or a custom platform version. Specify the resource by its Amazon Resource Name (ARN).

Options

None of these options are required. If you run **eb create** without any options, you are prompted to enter or select a value for each setting.

Name	Description
-l or --list	List all tags that are currently applied to the resource.
-a <i>key1=value1</i> [, <i>key2=value2</i> ...] or --add <i>key1=value1</i> [, <i>key2=value2</i> ...]	Apply new tags to the resource. Specify tags as a comma-separated list of <i>key=value</i> pairs. You can't specify keys of existing tags. Valid values: See Tagging resources (p. 349) .
-u <i>key1=value1</i> [, <i>key2=value2</i> ...] or --update <i>key1=value1</i> [, <i>key2=value2</i> ...]	Update the values of existing resource tags. Specify tags as a comma-separated list of <i>key=value</i> pairs. You must specify keys of existing tags. Valid values: See Tagging resources (p. 349) .
-d <i>key1</i> [, <i>key2</i> ...] or --delete <i>key1</i> [, <i>key2</i> ...]	Delete existing resource tags. Specify tags as a comma-separated list of keys. You must specify keys of existing tags. Valid values: See Tagging resources (p. 349) .
-r <i>region</i> or --region <i>region</i>	The AWS Region in which your resource exists. Default: the configured default region. For the list of values you can specify for this option, see AWS Elastic Beanstalk Endpoints and Quotas in the <i>AWS General Reference</i> .
--resource <i>ARN</i>	The ARN of the resource that the command modifies or lists tags for. If not specified, the command refers to the (default or specified) environment in the current directory's application. Valid values: See one of the sub-topic of Tagging resources (p. 349) that is specific to the resource you're interested in. These topics show how the resource's ARN is

Name	Description
	constructed and explain how to get a list of this resource's ARNs that exist for your application or account.

Output

The `--list` subcommand option displays a list of the resource's tags. The output shows both the tags that Elastic Beanstalk applies by default and your custom tags.

```
$ eb tags --list
Showing tags for environment 'MyApp-env':

Key                               Value
Name                               MyApp-env
elasticbeanstalk:environment-id    e-63cmxwjaut
elasticbeanstalk:environment-name  MyApp-env
mytag                               tagvalue
tag2                                2nd value
```

The `--add`, `--update`, and `--delete` subcommand options, when successful, don't have any output. You can add the `--verbose` option to see detailed output of the command's activity.

```
$ eb tags --verbose --update "mytag=tag value"
Updated Tags:

Key                               Value
mytag                             tag value
```

Examples

The following command successfully adds a tag with the key `tag1` and the value `value1` to the application's default environment, and at the same time deletes the tag `tag2`.

```
$ eb tags --add tag1=value1 --delete tag2
```

The following command successfully adds a tag to a saved configuration within an application.

```
$ eb tags --add tag1=value1 \
  --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-
id:configurationtemplate/my-app/my-template"
```

The following command fails because it tries to update a nonexistent tag.

```
$ eb tags --update tag3=newval
ERROR: Tags with the following keys can't be updated because they don't exist:

tag3
```

The following command fails because it tries to update and delete the same key.

```
$ eb tags --update mytag=newval --delete mytag
ERROR: A tag with the key 'mytag' is specified for both '--delete' and '--update'. Each tag
can be either deleted or updated in a single operation.
```

eb terminate

Description

Terminates the running environment so that you do not incur charges for unused AWS resources.

Using the `--all` option, deletes the application that the current directory was initialized to using [eb init](#) (p. 907). The command terminates all environments in the application, terminates the application's [application versions](#) (p. 337) and [saved configurations](#) (p. 640), and then deletes the application.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command terminates the running custom environment.

Note

You can always launch a new environment using the same version later. If you have data from an environment that you would like to preserve, create a snapshot of your current database instance before you terminate the environment. You can later use it as the basis for new DB instance when you create a new environment. For more information, see [Creating a DB Snapshot](#) in the *Amazon Relational Database Service User Guide*.

Syntax

eb terminate

eb terminate *environment-name*

Options

Name	Description
<code>--all</code>	Terminates all environments in the application, the application's application versions (p. 337), and its saved configurations (p. 640), and then deletes the application.
<code>--force</code>	Terminate the environment without prompting for confirmation.
<code>--ignore-links</code>	Terminate the environment even if there are dependent environments with links to it. See Compose Environments (p. 881).
<code>--timeout</code>	The number of minutes before the command times out.

Output

If successful, the command returns the status of the `terminate` operation.

Example

The following example request terminates the environment `tmp-dev`.

```
$ eb terminate
The environment "tmp-dev" and all associated instances will be terminated.
To confirm, type the environment name: tmp-dev
2018-07-11 21:05:25 INFO: terminateEnvironment is starting.
2018-07-11 21:05:40 INFO: Deleted CloudWatch alarm named: awseb-e-2cpfjbra9a-stack-
AWSEBCloudwatchAlarmHigh-16V08YOF2KQ7U
```

```

2018-07-11 21:05:41 INFO: Deleted CloudWatch alarm named: awseb-e-2cpfjbra9a-stack-
AWSEBCloudwatchAlarmLow-6ZAWH9F20P7C
2018-07-11 21:06:42 INFO: Deleted Auto Scaling group policy named:
arn:aws:autoscaling:us-east-2:11122223333:scalingPolicy:5d7d3e6b-
d59b-47c5-b102-3e11fe3047be:autoScalingGroupName/awseb-e-2cpfjbra9a-stack-
AWSEBAutoScalingGroup-7AXY7U13ZQ6E:policyName/awseb-e-2cpfjbra9a-stack-AWSEBAutoSca
lingScaleUpPolicy-1876U27JEC34J
2018-07-11 21:06:43 INFO: Deleted Auto Scaling group policy named:
arn:aws:autoscaling:us-east-2:11122223333:scalingPolicy:29c6e7c7-7ac8-46fc-91f5-
cfabb65b985b:autoScalingGroupName/awseb-e-2cpfjbra9a-stack-
AWSEBAutoScalingGroup-7AXY7U13ZQ6E:policyName/awseb-e-2cpfjbra9a-stack-AWSEBAutoSca
lingScaleDownPolicy-SL4LHODMOMU
2018-07-11 21:06:48 INFO: Waiting for EC2 instances to terminate. This may take a few
minutes.
2018-07-11 21:08:55 INFO: Deleted Auto Scaling group named: awseb-e-2cpfjbra9a-stack-
AWSEBAutoScalingGroup-7AXY7U13ZQ6E
2018-07-11 21:09:10 INFO: Deleted security group named: awseb-e-2cpfjbra9a-stack-
AWSEBSecurityGroup-XT4YYGFL7I99
2018-07-11 21:09:40 INFO: Deleted load balancer named: awseb-e-2-AWSEBLoa-AK6RRYFQVV3S
2018-07-11 21:09:42 INFO: Deleting SNS topic for environment tmp-dev.
2018-07-11 21:09:52 INFO: terminateEnvironment completed successfully.

```

eb upgrade

Description

Upgrades the platform of your environment to the most recent version of the platform on which it is currently running.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command upgrades the environment to the most recent version of the custom platform on which it is currently running.

Syntax

eb upgrade

eb upgrade *environment-name*

Options

Name	Description
<code>--force</code>	Upgrades without requiring you to confirm the environment name before starting the upgrade process.
<code>--noroll</code>	Updates all instances without using rolling updates to keep some instances in service during the upgrade.
Common options (p. 935)	

Output

The command shows an overview of the change and prompts you to confirm the upgrade by typing the environment name. If successful, your environment is updated and then launched with the most recent version of the platform.

Example

The following example upgrades the current platform version of the specified environment to the most recently available platform version.

```
$ eb upgrade
Current platform: 64bit Amazon Linux 2014.09 v1.0.9 running Python 2.7
Latest platform: 64bit Amazon Linux 2014.09 v1.2.0 running Python 2.7

WARNING: This operation replaces your instances with minimal or zero downtime. You may
cancel the upgrade after it has started by typing "eb abort".
You can also change your platform version by typing "eb clone" and then "eb swap".

To continue, type the environment name:
```

eb use

Description

Sets the specified environment as the default environment.

When using Git, **eb use** sets the default environment for the current branch. Run this command once in each branch that you want to deploy to Elastic Beanstalk.

Syntax

eb use *environment-name*

Options

Name	Description
<code>--source</code> <code>codecommit/<i>repository-</i></code> <code><i>name/branch-name</i></code>	CodeCommit repository and branch. See Using the EB CLI with AWS CodeCommit (p. 871) .
<code>-r</code> <i>region</i> <code>--region</code> <i>region</i>	Change the region in which you create environments.
Common options (p. 935)	

Common options

You can use the following options with all EB CLI commands.

Name	Description
<code>--debug</code>	Print information for debugging.
<code>-h, --help</code>	Show the Help message.

Name	Description
	Type: String Default: None
<code>--no-verify-ssl</code>	Skip SSL certificate verification. Use this option if you have issues using the CLI with a proxy.
<code>--profile</code>	Use a specific profile from your AWS credentials file.
<code>--quiet</code>	Suppress all output from the command.
<code>--region</code>	Use the specified region.
<code>-v, --verbose</code>	Display verbose information.

EB CLI 2.6 (retired)

This version of the EB CLI and its documentation have been replaced with version 3 (in this section, EB CLI 3 represents version 3 and later of the EB CLI). For information on the new version, see [Using the Elastic Beanstalk command line interface \(EB CLI\) \(p. 852\)](#).

You should migrate to the latest version of EB CLI 3. It can manage environments that you launched using EB CLI 2.6 or earlier versions of EB CLI.

Differences from version 3 of EB CLI

EB is a command line interface (CLI) tool for Elastic Beanstalk that you can use to deploy applications quickly and more easily. The latest version of EB was introduced by Elastic Beanstalk in EB CLI 3. EB CLI automatically retrieves settings from an environment created using EB if the environment is running. Note that EB CLI 3 does not store option settings locally, as in earlier versions.

EB CLI introduces the commands **eb create**, **eb deploy**, **eb open**, **eb console**, **eb scale**, **eb setenv**, **eb config**, **eb terminate**, **eb clone**, **eb list**, **eb use**, **eb printenv**, and **eb ssh**. In EB CLI 3.1 or later, you can also use the **eb swap** command. In EB CLI 3.2 only, you can use the **eb abort**, **eb platform**, and **eb upgrade** commands. In addition to these new commands, EB CLI 3 commands differ from EB CLI 2.6 commands in several cases:

- **eb init** – Use **eb init** to create an `.elasticbeanstalk` directory in an existing project directory and create a new Elastic Beanstalk application for the project. Unlike with previous versions, EB CLI 3 and later versions do not prompt you to create an environment.
- **eb start** – EB CLI 3 does not include the command **eb start**. Use **eb create** to create an environment.
- **eb stop** – EB CLI 3 does not include the command **eb stop**. Use **eb terminate** to completely terminate an environment and clean up.
- **eb push** and **git aws .push** – EB CLI 3 does not include the commands **eb push** or `git aws .push`. Use **eb deploy** to update your application code.
- **eb update** – EB CLI 3 does not include the command **eb update**. Use **eb config** to update an environment.
- **eb branch** – EB CLI 3 does not include the command **eb branch**.

For more information about using EB CLI 3 commands to create and manage an application, see [EB CLI command reference \(p. 884\)](#). For a walkthrough of how to deploy a sample application using EB CLI 3, see [Managing Elastic Beanstalk environments with the EB CLI \(p. 864\)](#).

Migrating to EB CLI 3 and CodeCommit

Elastic Beanstalk has not only retired EB CLI 2.6, but has also removed some 2.6 functionality. The most significant change from 2.6 is that EB CLI no longer natively supports incremental code updates (**eb push**, `git aws.push`) or branching (**eb branch**). This section describes how to migrate from EB CLI 2.6 to the latest version of EB CLI and use CodeCommit as your code repository.

If you have not done so already, create a code repository in CodeCommit, as described in [Migrate to CodeCommit](#).

Once you have [installed](#) (p. 853) and [configured](#) (p. 860) EB CLI, you have two opportunities to associate your application with your CodeCommit repository, including a specific branch.

- When executing **eb init**, such in the following example where *myRepo* is the name of your CodeCommit repository and *myBranch* is the branch in CodeCommit.

```
eb init --source codecommit/myRepo/myBranch
```

- When executing **eb deploy**, such in the following example where *myRepo* is the name of your CodeCommit repository and *myBranch* is the branch in CodeCommit.

```
eb deploy --source codecommit/myRepo/myBranch
```

For further information, including how to deploy incremental code updates to an Elastic Beanstalk environment without having to re-upload your entire project, see [Using the EB CLI with AWS CodeCommit](#) (p. 871).

Elastic Beanstalk API command line interface (retired)

This tool, the Elastic Beanstalk API command line interface (API CLI), has been replaced by the AWS CLI, which provides API equivalent commands for all AWS services. See the AWS Command Line Interface User Guide to get started with the AWS CLI. Also try the [EB CLI](#) (p. 852) for a simplified, higher-level command line experience.

Converting Elastic Beanstalk API CLI scripts

Convert your old EB API CLI scripts to use the AWS CLI or Tools for Windows PowerShell to get access to the latest Elastic Beanstalk APIs. The following table lists the Elastic Beanstalk API-based CLI commands and their equivalent commands in the AWS CLI and Tools for Windows PowerShell.

Elastic Beanstalk API CLI	AWS CLI	AWS Tools for Windows PowerShell
<code>elastic-beanstalk-check-dns-availability</code>	<code>check-dns-availability</code>	<code>Get-EBDNSAvailability</code>
<code>elastic-beanstalk-create-application</code>	<code>create-application</code>	<code>New-EBApplication</code>

Elastic Beanstalk API CLI	AWS CLI	AWS Tools for Windows PowerShell
elastic-beanstalk-create-application-version	create-application-version	New-EBApplicationVersion
elastic-beanstalk-create-configuration-template	create-configuration-template	New-EBConfigurationTemplate
elastic-beanstalk-create-environment	create-environment	New-EBEnvironment
elastic-beanstalk-create-storage-location	create-storage-location	New-EBStorageLocation
elastic-beanstalk-delete-application	delete-application	Remove-EBApplication
elastic-beanstalk-delete-application-version	delete-application-version	Remove-EBApplicationVersion
elastic-beanstalk-delete-configuration-template	delete-configuration-template	Remove-EBConfigurationTemplate
elastic-beanstalk-delete-environment-configuration	delete-environment-configuration	Remove-EBEnvironmentConfiguration
elastic-beanstalk-describe-application-versions	describe-application-versions	Get-EBApplicationVersion
elastic-beanstalk-describe-applications	describe-applications	Get-EBApplication
elastic-beanstalk-describe-configuration-options	describe-configuration-options	Get-EBConfigurationOption
elastic-beanstalk-describe-configuration-settings	describe-configuration-settings	Get-EBConfigurationSetting
elastic-beanstalk-describe-environment-resources	describe-environment-resources	Get-EBEnvironmentResource
elastic-beanstalk-describe-environments	describe-environments	Get-EBEnvironment
elastic-beanstalk-describe-events	describe-events	Get-EBEvent

Elastic Beanstalk API CLI	AWS CLI	AWS Tools for Windows PowerShell
elastic-beanstalk-list-available-solution-stacks	list-available-solution-stacks	Get-EBAvailableSolutionStack
elastic-beanstalk-rebuild-environment	rebuild-environment	Start-EBEnvironmentRebuild
elastic-beanstalk-request-environment-info	request-environment-info	Request-EBEnvironmentInfo
elastic-beanstalk-restart-app-server	restart-app-server	Restart-EBAppServer
elastic-beanstalk-retrieve-environment-info	retrieve-environment-info	Get-EBEnvironmentInfo
elastic-beanstalk-swap-environment-cnames	swap-environment-cnames	Set-EBEnvironmentCNAME
elastic-beanstalk-terminate-environment	terminate-environment	Stop-EBEnvironment
elastic-beanstalk-update-application	update-application	Update-EBApplication
elastic-beanstalk-update-application-version	update-application-version	Update-EBApplicationVersion
elastic-beanstalk-update-configuration-template	update-configuration-template	Update-EBConfigurationTemplate
elastic-beanstalk-update-environment	update-environment	Update-EBEnvironment
elastic-beanstalk-validate-configuration-settings	validate-configuration-settings	Test-EBConfigurationSetting

AWS Elastic Beanstalk security

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [Shared Responsibility Model](#) describes this as *Security of the Cloud* and *Security in the Cloud*.

Security of the Cloud – AWS is responsible for protecting the infrastructure that runs all of the services offered in the AWS Cloud and providing you with services that you can use securely. Our security responsibility is the highest priority at AWS, and the effectiveness of our security is regularly tested and verified by third-party auditors as part of the [AWS Compliance Programs](#). Review the [AWS Services in Scope of AWS assurance programs](#) for information as it relates to Elastic Beanstalk.

Security in the Cloud – Your responsibility is determined by the AWS service you are using, and other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations. This documentation is intended to help you understand how to apply the Shared Responsibility Model when using Elastic Beanstalk.

Use the following security topics to learn more about the security tasks Elastic Beanstalk is responsible for, and the security configurations you should consider when using Elastic Beanstalk to meet your security and compliance objectives.

Topics

- [Data protection in Elastic Beanstalk \(p. 940\)](#)
- [Identity and access management for Elastic Beanstalk \(p. 942\)](#)
- [Logging and monitoring in Elastic Beanstalk \(p. 942\)](#)
- [Compliance validation for Elastic Beanstalk \(p. 943\)](#)
- [Resilience in Elastic Beanstalk \(p. 944\)](#)
- [Infrastructure security in Elastic Beanstalk \(p. 944\)](#)
- [Configuration and vulnerability analysis in Elastic Beanstalk \(p. 945\)](#)
- [Security best practices for Elastic Beanstalk \(p. 945\)](#)

Data protection in Elastic Beanstalk

AWS Elastic Beanstalk conforms to the AWS [shared responsibility model](#), which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). Give each user only the permissions necessary to do their jobs. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use Secure Sockets Layer (SSL)/Transport Layer Security (TLS) to communicate with AWS resources.

- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions and all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as **Name** fields. This includes when you work with Elastic Beanstalk or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into Elastic Beanstalk or other services might get included in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

For more information about data protection, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the *AWS Security Blog*.

For other Elastic Beanstalk security topics, see [AWS Elastic Beanstalk security](#) (p. 940).

Topics

- [Protecting data using encryption](#) (p. 941)
- [Internetwork traffic privacy](#) (p. 942)

Protecting data using encryption

Elastic Beanstalk stores various objects in an Amazon Simple Storage Service (Amazon S3) bucket that it creates for each AWS Region in which you create environments. For details, see [the section called "Amazon S3"](#) (p. 831).

You provide some of the stored objects and send them to Elastic Beanstalk, for example, application versions and source bundles. Elastic Beanstalk generates other objects, for example, log files. In addition to the data that Elastic Beanstalk stores, your application can transfer and/or store data as part of its operation.

Data protection refers to protecting data while *in transit* (as it travels to and from Elastic Beanstalk) and *at rest* (while it is stored in AWS data centers).

Encryption in transit

You can achieve data protection in transit in two ways: encrypt the connection using Secure Sockets Layer (SSL), or use client-side encryption (where the object is encrypted before it is sent). Both methods are valid for protecting your application data. To secure the connection, encrypt it using SSL whenever your application, its developers and administrators, and its end users send or receive any objects. For details about encrypting web traffic to and from your application, see [the section called "HTTPS"](#) (p. 652).

Client-side encryption isn't a valid method for protecting your source code in application versions and source bundles that you upload. Elastic Beanstalk needs access to these objects, so they can't be encrypted. Therefore, be sure to secure the connection between your development or deployment environment and Elastic Beanstalk.

Encryption at Rest

To protect your application's data at rest, learn about data protection in the storage service that your application uses. For example, see [Data Protection in Amazon RDS](#) in the *Amazon RDS User Guide*, [Data Protection in Amazon S3](#) in the *Amazon Simple Storage Service Developer Guide*, or [Encrypting Data and Metadata in EFS](#) in the *Amazon Elastic File System User Guide*.

Elastic Beanstalk doesn't turn on default encryption for the Amazon S3 bucket that it creates. This means that by default, objects are stored unencrypted in the bucket (and are accessible only by users authorized to read the bucket). If your application requires encryption at rest, you can configure your account's buckets for default encryption. For more information, see [Amazon S3 Default Encryption for S3 Buckets](#) in the *Amazon Simple Storage Service Developer Guide*.

For more information about data protection, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the *AWS Security Blog*.

For other Elastic Beanstalk security topics, see [AWS Elastic Beanstalk security \(p. 940\)](#).

Internetwork traffic privacy

You can use Amazon Virtual Private Cloud (Amazon VPC) to create boundaries between resources in your Elastic Beanstalk application and control traffic between them, your on-premises network, and the internet. For details, see [the section called "Amazon VPC" \(p. 834\)](#).

For more information about Amazon VPC security, see [Security](#) in the *Amazon VPC User Guide*.

For more information about data protection, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the *AWS Security Blog*.

For other Elastic Beanstalk security topics, see [AWS Elastic Beanstalk security \(p. 940\)](#).

Identity and access management for Elastic Beanstalk

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS Elastic Beanstalk resources. IAM is an AWS service that you can use with no additional charge.

For details on working with IAM, see [Using Elastic Beanstalk with AWS Identity and Access Management \(p. 759\)](#).

For other Elastic Beanstalk security topics, see [AWS Elastic Beanstalk security \(p. 940\)](#).

Logging and monitoring in Elastic Beanstalk

Monitoring is important for maintaining the reliability, availability, and performance of AWS Elastic Beanstalk and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multipoint failure if one occurs. AWS provides several tools for monitoring your Elastic Beanstalk resources and responding to potential incidents.

For more information about monitoring, see [Monitoring an environment \(p. 685\)](#).

For other Elastic Beanstalk security topics, see [AWS Elastic Beanstalk security \(p. 940\)](#).

Enhanced health reporting

Enhanced health reporting is a feature that you can enable on your environment to allow Elastic Beanstalk to gather additional information about resources in your environment. Elastic Beanstalk

analyzes the information to provide a better picture of overall environment health and help identify issues that can cause your application to become unavailable. For more information, see [Enhanced health reporting and monitoring \(p. 691\)](#).

Amazon EC2 instance logs

The Amazon EC2 instances in your Elastic Beanstalk environment generate logs that you can view to troubleshoot issues with your application or configuration files. Logs created by the web server, application server, Elastic Beanstalk platform scripts, and AWS CloudFormation are stored locally on individual instances. You can easily retrieve them by using the [environment management console \(p. 353\)](#) or the EB CLI. You can also configure your environment to stream logs to Amazon CloudWatch Logs in real time. For more information, see [Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment \(p. 732\)](#).

Environment notifications

You can configure your Elastic Beanstalk environment to use Amazon Simple Notification Service (Amazon SNS) to notify you of important events that affect your application. Specify an email address during or after environment creation to receive emails from AWS when an error occurs, or when your environment's health changes. For more information, see [Elastic Beanstalk environment notifications with Amazon SNS \(p. 526\)](#).

Amazon CloudWatch alarms

Using CloudWatch alarms, you watch a single metric over a time period that you specify. If the metric exceeds a given threshold, a notification is sent to an Amazon SNS topic or AWS Auto Scaling policy. CloudWatch alarms don't invoke actions because they are in a particular state. Instead, alarms invoke actions when the state changed and was maintained for a specified number of periods. For more information, see [Using Elastic Beanstalk with Amazon CloudWatch \(p. 742\)](#).

AWS CloudTrail logs

CloudTrail provides a record of actions taken by a user, role, or an AWS service in Elastic Beanstalk. Using the information collected by CloudTrail, you can determine the request that was made to Elastic Beanstalk, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, see [Logging Elastic Beanstalk API calls with AWS CloudTrail \(p. 741\)](#).

AWS X-Ray debugging

X-Ray is an AWS service that gathers data about the requests that your application serves, and uses it to construct a service map that you can use to identify issues with your application and opportunities for optimization. You can use the AWS Elastic Beanstalk console or a configuration file to run the X-Ray daemon on the instances in your environment. For more information, see [Configuring AWS X-Ray debugging \(p. 521\)](#).

Compliance validation for Elastic Beanstalk

The security and compliance of AWS Elastic Beanstalk is assessed by third-party auditors as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others. AWS provides a frequently updated list of AWS services in scope of specific compliance programs at [AWS Services in Scope by Compliance Program](#).

Third-party audit reports are available for you to download using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

For more information about AWS compliance programs, see [AWS Compliance Programs](#).

Your compliance responsibility when using Elastic Beanstalk is determined by the sensitivity of your data, your organization's compliance objectives, and applicable laws and regulations. If your use of Elastic Beanstalk is subject to compliance with standards such as HIPAA, PCI, or FedRAMP, AWS provides resources to help:

- [Security and Compliance Quick Start Guides](#) – Deployment guides that discuss architectural considerations and provide steps for deploying security-focused and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – A whitepaper that describes how companies can use AWS to create HIPAA-compliant applications.
- [AWS Compliance Resources](#) – A collection of compliance workbooks and guides that might apply to your industry and location.
- [AWS Config](#) – A service that assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – A comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

For other Elastic Beanstalk security topics, see [AWS Elastic Beanstalk security \(p. 940\)](#).

Resilience in Elastic Beanstalk

The AWS global infrastructure is built around AWS Regions and Availability Zones.

AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking.

With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

AWS Elastic Beanstalk manages and automates the use of the AWS global infrastructure on your behalf. When using Elastic Beanstalk, you benefit from the availability and fault tolerance mechanisms that AWS offers.

For other Elastic Beanstalk security topics, see [AWS Elastic Beanstalk security \(p. 940\)](#).

Infrastructure security in Elastic Beanstalk

As a managed service, AWS Elastic Beanstalk is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use AWS published API calls to access Elastic Beanstalk through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS), such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern platforms such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service \(AWS STS\)](#) to generate temporary security credentials to sign requests.

For other Elastic Beanstalk security topics, see [AWS Elastic Beanstalk security \(p. 940\)](#).

Configuration and vulnerability analysis in Elastic Beanstalk

AWS and our customers share responsibility for achieving a high level of software component security and compliance. AWS Elastic Beanstalk helps you perform your side of the shared responsibility model by providing a *managed updates* feature. This feature automatically applies patch and minor updates for an Elastic Beanstalk supported platform version.

For more information, see [Shared responsibility model for Elastic Beanstalk platform maintenance \(p. 27\)](#).

For other Elastic Beanstalk security topics, see [AWS Elastic Beanstalk security \(p. 940\)](#).

Security best practices for Elastic Beanstalk

AWS Elastic Beanstalk provides several security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations, not prescriptions.

For other Elastic Beanstalk security topics, see [AWS Elastic Beanstalk security \(p. 940\)](#).

Preventive security best practices

Preventive security controls attempt to prevent incidents before they occur.

Implement least privilege access

Elastic Beanstalk provides AWS Identity and Access Management (IAM) managed policies for [instance profiles \(p. 760\)](#), [service roles \(p. 764\)](#), and [IAM users \(p. 775\)](#). These managed policies specify all permissions that might be necessary for the correct operation of your environment and application.

Your application might not require all the permissions in our managed policies. You can customize them and grant only the permissions that are required for your environment's instances, the Elastic Beanstalk service, and your users to perform their tasks. This is particularly relevant to user policies, where different user roles might have different permission needs. Implementing least privilege access is fundamental in reducing security risk and the impact that could result from errors or malicious intent.

Update your platforms regularly

Elastic Beanstalk regularly releases new platform versions to update all of its platforms. New platform versions provide operating system, runtime, application server, and web server updates, and updates to Elastic Beanstalk components. Many of these platform updates include important security fixes. Ensure that your Elastic Beanstalk environments are running on a supported platform version (typically the latest version for your platform). For details, see [Updating your Elastic Beanstalk environment's platform version \(p. 415\)](#).

The easiest way to keep your environment's platform up to date is to configure the environment to use [managed platform updates](#) (p. 420).

Detective security best practices

Detective security controls identify security violations after they have occurred. They can help you detect a potential security threat or incident.

Implement monitoring

Monitoring is an important part of maintaining the reliability, security, availability, and performance of your Elastic Beanstalk solutions. AWS provides several tools and services to help you monitor your AWS services.

The following are some examples of items to monitor:

- *Amazon CloudWatch metrics for Elastic Beanstalk* – Set alarms for key Elastic Beanstalk metrics and for your application's custom metrics. For details, see [Using Elastic Beanstalk with Amazon CloudWatch](#) (p. 742).
- *AWS CloudTrail entries* – Track actions that might impact availability, like `UpdateEnvironment` or `TerminateEnvironment`. For details, see [Logging Elastic Beanstalk API calls with AWS CloudTrail](#) (p. 741).

Enable AWS Config

AWS Config provides a detailed view of the configuration of AWS resources in your account. You can see how resources are related, get a history of configuration changes, and see how relationships and configurations change over time.

You can use AWS Config to define rules that evaluate resource configurations for data compliance. AWS Config rules represent the ideal configuration settings for your Elastic Beanstalk resources. If a resource violates a rule and is flagged as *noncompliant*, AWS Config can alert you using an Amazon Simple Notification Service (Amazon SNS) topic. For details, see [Finding and tracking Elastic Beanstalk resources with AWS Config](#) (p. 752).

Troubleshooting

This chapter provides a table of the most common Elastic Beanstalk issues and how to resolve or work around them. Error messages can appear as events on the environment management page in the console, in logs, or on the health page.

If the health of your environment changes to red, try the following:

- Review recent environment [events \(p. 728\)](#). Messages from Elastic Beanstalk about deployment, load, and configuration issues often appear here.
- [Pull logs \(p. 732\)](#) to view recent log file entries. Web server logs contain information about incoming requests and errors.
- [Connect to an instance \(p. 730\)](#) and check system resources.
- [Roll back \(p. 397\)](#) to a previous, working version of the application.
- Undo recent configuration changes or restore a [saved configuration \(p. 540\)](#).
- Deploy a new environment. If it appears healthy, perform a [CNAME swap \(p. 405\)](#) to route traffic to the new environment and continue to debug the old one.

Topics

- [Connectivity \(p. 947\)](#)
- [Environment creation and instance launches \(p. 948\)](#)
- [Deployments \(p. 948\)](#)
- [Health \(p. 948\)](#)
- [Configuration \(p. 949\)](#)
- [Troubleshooting Docker containers \(p. 949\)](#)
- [FAQ \(p. 950\)](#)

Connectivity

Issue: *Unable to connect to Amazon RDS from Elastic Beanstalk.*

To connect RDS to your Elastic Beanstalk application, do the following:

- Make sure RDS is in the same Region as your Elastic Beanstalk application.
- Make sure the RDS security group for your instance has an authorization for the Amazon EC2 security group you are using for your Elastic Beanstalk environment. For instructions on how to find the name of your EC2 security group using the AWS Management Console, see [Security groups \(p. 455\)](#). For more information about configuring your EC2 security group, go to the "Authorizing Network Access to an Amazon EC2 Security Group" section of [Working with DB Security Groups](#) in the *Amazon Relational Database Service User Guide*.
- For Java, make sure the MySQL JAR file is in your WEB-INF/lib. See [Adding an Amazon RDS DB instance to your Java application environment \(p. 118\)](#) for more details.

Issue: *Servers that were created in the Elastic Beanstalk console do not appear in the Toolkit for Eclipse*

You can manually import servers by following the instructions at [Importing existing environments into Eclipse \(p. 124\)](#).

Environment creation and instance launches

Event: *Failed to Launch Environment*

This event occurs when Elastic Beanstalk attempts to launch an environment and encounters failures along the way. Previous events on the **Events** page will alert you to the root cause of this issue.

Event: *Create environment operation is complete, but with command timeouts. Try increasing the timeout period.*

Your application may take a long time to deploy if you use configuration files that run commands on the instance, download large files, or install packages. Increase the [command timeout \(p. 401\)](#) to give your application more time to start running during deployments.

Event: *The following resource(s) failed to create: [AWSEBInstanceLaunchWaitCondition]*

This message indicates that your environment's Amazon EC2 instances did not communicate to Elastic Beanstalk that they were launched successfully. This can occur if the instances do not have Internet connectivity. If you configured your environment to launch instances in a private VPC subnet, [ensure that the subnet has a NAT \(p. 834\)](#) to allow the instances to connect to Elastic Beanstalk.

Event: *A Service Role is required in this region. Please add a Service Role option to the environment.*

Elastic Beanstalk uses a service role to monitor the resources in your environment and support [managed platform updates \(p. 420\)](#). See [Managing Elastic Beanstalk service roles \(p. 764\)](#) for more information.

Deployments

Issue: *Application becomes unavailable during deployments*

Because Elastic Beanstalk uses a drop-in upgrade process, there might be a few seconds of downtime. Use [rolling deployments \(p. 400\)](#) to minimize the effect of deployments on your production environments.

Event: *Failed to create the AWS Elastic Beanstalk application version*

Your application source bundle may be too large, or you may have reached the [application version quota \(p. 337\)](#).

Event: *Update environment operation is complete, but with command timeouts. Try increasing the timeout period.*

Your application may take a long time to deploy if you use configuration files that run commands on the instance, download large files, or install packages. Increase the [command timeout \(p. 401\)](#) to give your application more time to start running during deployments.

Health

Event: *CPU Utilization Exceeds 95.00%*

Try [running more instances \(p. 457\)](#), or [choose a different instance type \(p. 451\)](#).

Event: *Elastic Load Balancer awseb-myapp Has Zero Healthy Instances*

If your application appears to be working, make sure that your application's health check URL is configured correctly. If not, check the Health screen and environment logs for more information.

Event: *Elastic Load Balancer awseb-myapp Cannot Be Found*

Your environment's load balancer may have been removed out-of-band. Only make changes to your environment's resources with the configuration options and [extensibility \(p. 600\)](#) provided by Elastic Beanstalk. Rebuild your environment or launch a new one.

Event: *EC2 Instance Launch Failure. Waiting for a New EC2 Instance to Launch...*

Availability for your environment's instance type may be low, or you may have reached the instance quota for your account. Check the [service health dashboard](#) to ensure that the Elastic Compute Cloud (Amazon EC2) service is green, or [request a quota increase](#).

Configuration

Event: *You cannot configure an Elastic Beanstalk environment with values for both the Elastic Load Balancing Target option and Application Healthcheck URL option*

The `Target` option in the `aws:elb:healthcheck` namespace is deprecated. Remove the `Target` option namespace) from your environment and try updating again.

Event: *ELB cannot be attached to multiple subnets in the same AZ.*

This message can be seen if you try to move a load balancer between subnets in the same Availability Zone. Changing subnets on the load balancer requires moving it out of the original availability zone(s) and then back into the original with the desired subnets. During the process, all of your instances will be migrated between AZs, causing significant downtime. Instead, consider creating a new environment and [perform a CNAME swap \(p. 405\)](#).

Troubleshooting Docker containers

Event: *Failed to pull Docker image :latest: Invalid repository name (), only [a-z0-9-_.] are allowed. Tail the logs for more details.*

Check the syntax of the `dockerrun.aws.json` file using a JSON validator. Also verify the `dockerfile` contents against the requirements described in [Single Container Docker configuration \(p. 54\)](#)

Event: *No EXPOSE directive found in Dockerfile, abort deployment*

The `Dockerfile` or the `dockerrun.aws.json` file does not declare the container port. Use the `EXPOSE` instruction (`Dockerfile`) or `Ports` block (`dockerrun.aws.json` file) to expose a port for incoming traffic.

Event: *Failed to download authentication credentials repository from bucket name*

The `dockerrun.aws.json` provides an invalid EC2 key pair and/or S3 bucket for the `.dockercfg` file. Or, the instance profile does not have `GetObject` authorization for the S3 bucket. Verify that the `.dockercfg` file contains a valid S3 bucket and EC2 key pair. Grant permissions for the action `s3:GetObject` to the IAM role in the instance profile. For details, go to [Managing Elastic Beanstalk instance profiles \(p. 760\)](#)

Event: *Activity execution failed, because: WARNING: Invalid auth configuration file*

Your authentication file (`config.json`) is not formatted correctly. See [Using images from a private repository \(p. 76\)](#)

FAQ

Question: *How can I change my application URL from `myapp.us-west-2.elasticbeanstalk.com` to `www.myapp.com`?*

In a DNS server, register a CNAME record such as `www.mydomain.com CNAME mydomain.elasticbeanstalk.com`.

Question: *How do I specify a specific Availability Zone for my Elastic Beanstalk application?*

You can pick specific Availability Zones using the APIs, CLI, Eclipse plugin, or Visual Studio plugin. For instructions about using the Elastic Beanstalk console to specify an Availability Zone, see [Auto Scaling group for your Elastic Beanstalk environment \(p. 457\)](#).

Question: *How do I change my environment's instance type?*

On the environment configuration page, choose **Edit** in the **Instances** configuration category. Select a new instance type and choose **Apply** to update your environment. Elastic Beanstalk will terminate all running instances and replace them with new ones.

Question: *Can I prevent Amazon EBS volumes from being deleted when instances are terminated?*

Instances in your environment use Amazon EBS for storage; however, the root volume is deleted when an instance is terminated by Auto Scaling. It is not recommended to store state or other data on your instances. If needed, you can prevent volumes from being deleted with the AWS CLI: `$ aws ec2 modify-instance-attribute --instance-id <instance-id> --block-device-mappings [{"DeviceName": "/dev/sdc", "Ebs": {"VolumeId": "<vol-id>", "DeleteOnTermination": false}}]` as described in the [AWS CLI Reference](#).

Question: *How do I delete personal information from my Elastic Beanstalk application?*

AWS resources that your Elastic Beanstalk application uses might store personal information. When you terminate an environment, Elastic Beanstalk terminates the resources that it created. Resources you added using [configuration files \(p. 600\)](#) are also terminated. However, if you created AWS resources outside of your Elastic Beanstalk environment and associated them with your application, you might need to manually ensure that personal information that your application might have stored isn't unnecessarily retained. Throughout this developer guide, wherever we discuss the creation of additional resources, we also mention when you should consider deleting them.

Elastic Beanstalk resources

The following related resources can help you as you work with this service.

- [Elastic Beanstalk API Reference](#) A comprehensive description of all SOAP and Query APIs. Additionally, it contains a list of all SOAP data types.
- [Elastic Beanstalk Sample Code and Libraries](#) – A link to the command line tool as well as a sample Java web application. See the links below for additional sample applications.
- [elastic-beanstalk-samples on GitHub](#) – A GitHub repository with Elastic Beanstalk sample configuration files (.ebextensions). The repository's `README.md` file has links to additional GitHub repositories with sample applications.
- [Elastic Beanstalk Technical FAQ](#) – The top questions developers have asked about this product.
- [AWS Elastic Beanstalk Release Notes](#) – Details about new features, updates, and fixes in Elastic Beanstalk service, platform, console, and EB CLI releases.

- [Classes & Workshops](#) – Links to role-based and specialty courses as well as self-paced labs to help sharpen your AWS skills and gain practical experience.
- [AWS Developer Tools](#) – Links to developer tools, SDKs, IDE toolkits, and command line tools for developing and managing AWS applications.
- [AWS Whitepapers](#) – Links to a comprehensive list of technical AWS whitepapers, covering topics such as architecture, security, and economics and authored by AWS Solutions Architects or other technical experts.
- [AWS Support Center](#) – The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
- [AWS Support](#) – The primary web page for information about AWS Support, a one-on-one, fast-response support channel to help you build and run applications in the cloud.
- [Contact Us](#) – A central contact point for inquiries concerning AWS billing, account, events, abuse, and other issues.
- [AWS Site Terms](#) – Detailed information about our copyright and trademark; your account, license, and site access; and other topics.

Sample applications

The following are download links to the sample applications that are deployed as part of [Getting started using Elastic Beanstalk](#) (p. 3).

Note

Some samples use features that may have been released since the release of the platform you are using. If the sample fails to run, try updating your platform to a current version, as described in [the section called “Supported platforms”](#) (p. 29).

- **Single Container Docker** – [docker-singlecontainer-v1.zip](#)
- **Multicontainer Docker** – [docker-multicontainer-v2.zip](#)
- **Preconfigured Docker (Glassfish)** – [docker-glassfish-v1.zip](#)
- **Go** – [go-v1.zip](#)
- **Java SE** – [java-se-jetty-gradle-v3.zip](#)
- **Tomcat (default)** – [java-tomcat-v3.zip](#)

- **Tomcat 7** – [java7-tomcat7.zip](#)
- **.NET** – [dotnet-asp-v1.zip](#)
- **Node.js** – [nodejs-v1.zip](#)
- **PHP** – [php-v1.zip](#)
- **Python** – [python-v1.zip](#)
- **Ruby (Passenger Standalone)** – [ruby-passenger-v3.zip](#)
- **Ruby (Puma)** – [ruby-puma-v3.zip](#)

Platform history

AWS Elastic Beanstalk platform history has moved. See [Platform History](#) in the *AWS Elastic Beanstalk Platforms* document.