Best Practices for API Testing

RapidAPI

What is API Testing?

APIs are the glue between back-end infrastructure and end-user applications, which makes an API a great insertion point for testing. API testing allows teams to quickly uncover usability and performance issues before their users are impacted. This document will highlight the best practices for API testing. To begin, Wikipedia defines API testing in the following way:

API testing is a type of software testing that involves testing application programming interfaces (APIs) directly and as part of integration testing to determine if they meet expectations for functionality, reliability, performance, and security.

To understand how to begin to test an API, make sure you can connect to the API every time. That is part of direct testing. Next, you need to create some logic to mimic how third-party applications will be using the API. That covers functionality. Along the way, you can measure the performance of the responses. Also you will need to make sure you cannot access or change data that you do not have authorization for. That addresses security expectations.

In this document, we will examine some of the best practices for testing APIs.

Best Practices for API Testing

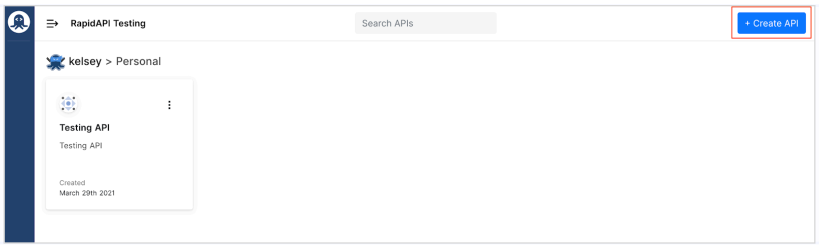
1. Choose an API Testing Tool
2. Use an OpenAPI Specification file to create an API
3. Use visual editor to create tests
4. Create test variables for parameter values to parameterize your tests
5. Create groups of tests
6. Use if/else and looping logic to execute tests only under certain conditions
7. Test from different locations to ensure consistent performance
8. Trigger tests from API calls and review results from another URL. You can use triggers from a Jenkins playbook, GitHub action, or any other CI/CD tool
9. Use email or SMS alerts to notify you of test failures

1. Choose an API Testing Tool

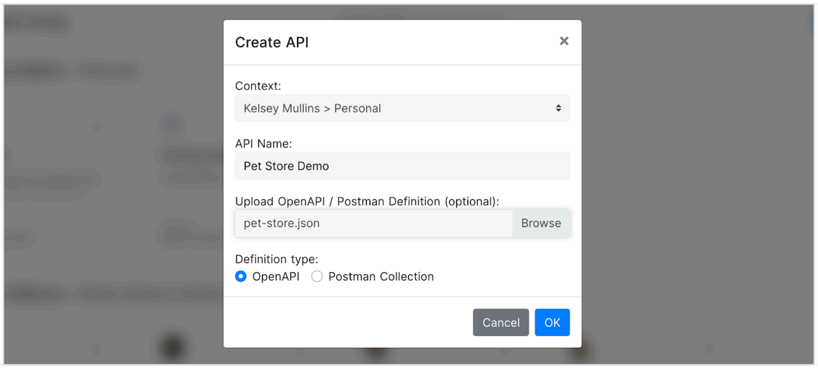
It is important to pick the right API testing tool for your environment. Fortunately, the RapidAPI team has already compiled a list of the best API testing tools. One option is to select RapidAPI Testing. RapidAPI Testing was created with developers in mind so it will be ideal for testing your APIs.

2. Use an OpenAPI Specification file to create an API

You can upload an existing API specification to RapidAPI Testing by clicking the “Create API” button in the top right corner.



Name the API and select the file to upload.


Create API

Context: Kelsey Mullins > Personal

API Name: Pet Store Demo

Upload OpenAPI / Postman Definition (optional): pet-store.json

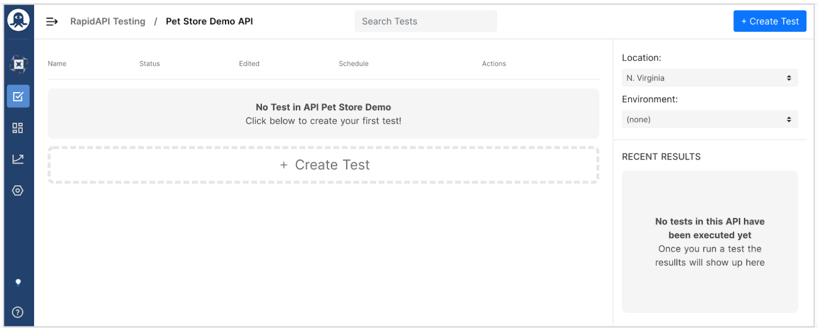
Definition type: [✓ OpenAPI] Postman Collection

Once you import the specification, all the endpoints are created for you and you can begin testing!

Note: You can also add an API via the RapidAPI Provider Dashboard. Any existing APIs or new APIs you add though the Provider Dashboard will also show up on RapidAPI Testing.

3. Use visual editor to create tests

From the RapidAPI Testing dashboard, select the API you wish to test. Then, click on "Create Test" and specify a name for your test.


RapidAPI Testing / Pet Store Demo API

Name	Status	Edited	Schedule	Actions

No Test in API Pet Store Demo
Click below to create your first test!

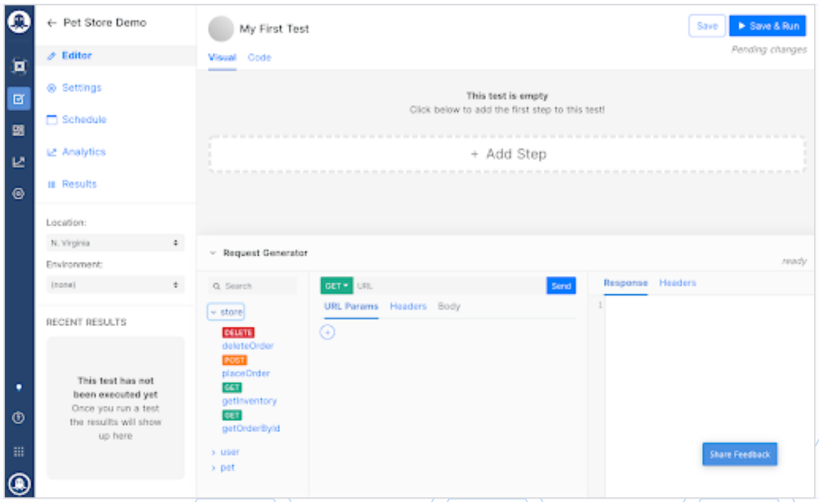
Location: N. Virginia

Environment: (none)

RECENT RESULTS

No tests in this API have been executed yet
Once you run a test the resullts will show up here

Once the test is created the request generator appears on the bottom of the screen:


Pet Store Demo

My First Test

Pending changes

This test is empty
Click below to add the first step to this test!

Location: N. Virginia

Environment: (none)

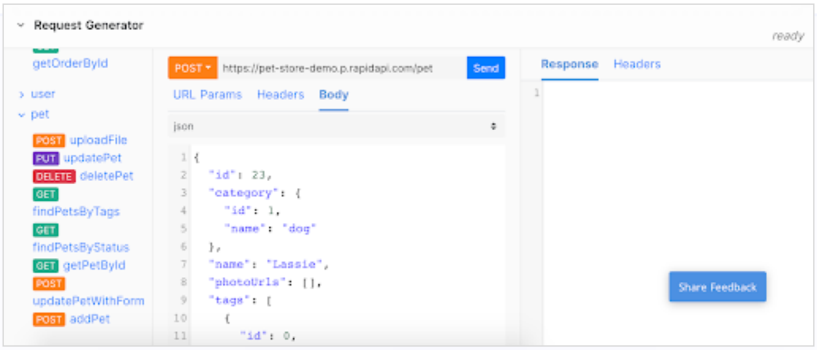
RECENT RESULTS

This test has not been executed yet
Once you run a test the results will show up here

Request Generator

ready

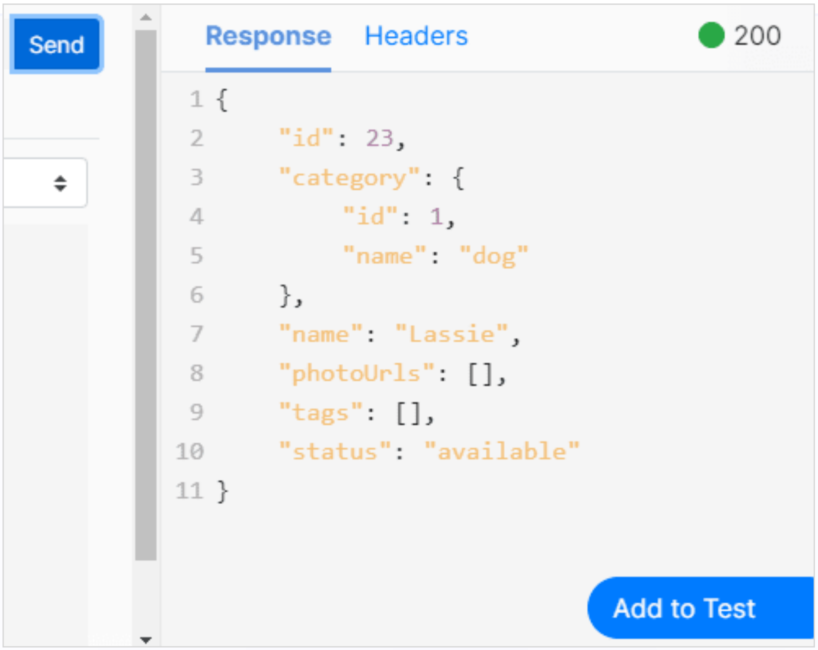
Within the request generator you can select an endpoint and specify parameters to send in the request.



https://pet-store-demo.p.rapidapi.com/pet

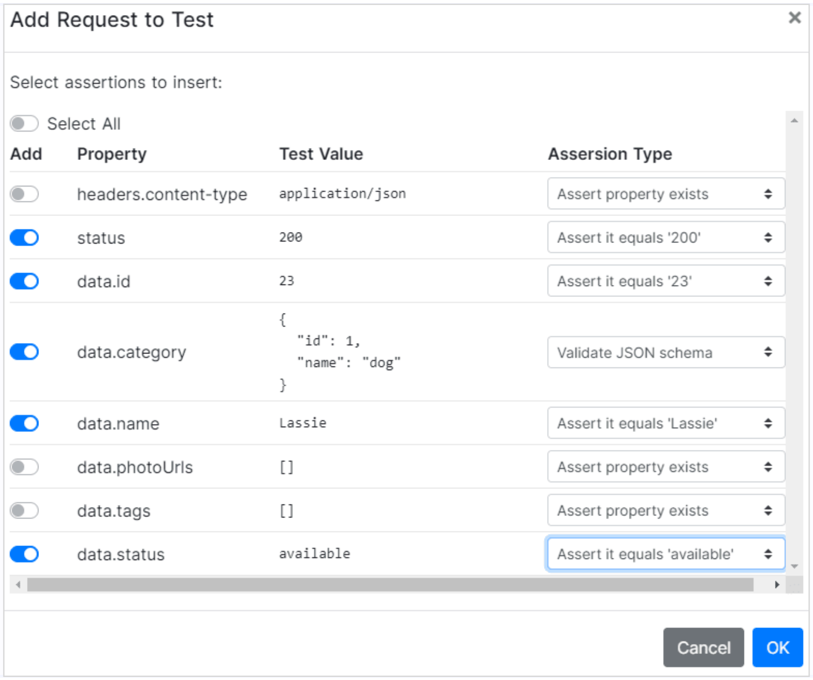
```
{
  "id": 23,
  "category": {
    "id": 1,
    "name": "dog"
  },
  "name": "Lassie",
  "photoUrls": [],
  "tags": [
    {
      "id": 0,
```

In order to add this API call to your test, first click the Send button. Then you'll see what results came back and a button will appear to "Add to Test".



```
{
    "id": 23,
    "category": {
        "id": 1,
        "name": "dog"
    },
    "name": "Lassie",
    "photoUrls": [],
    "tags": [],
    "status": "available"
}
```

When you click the **Add to Test** button, you can then choose which response values to test and what to check for:



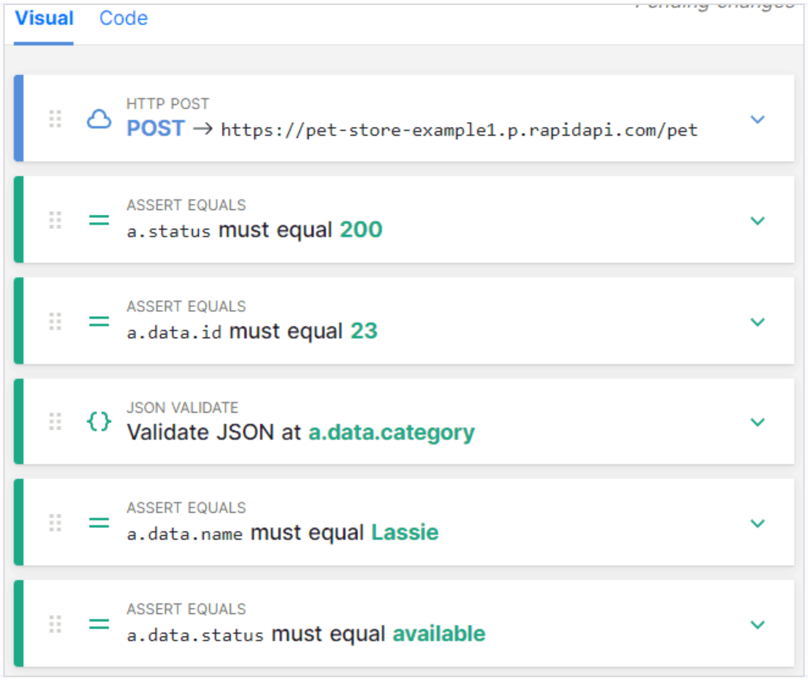
Add Request to Test

Select assertions to insert:

Select All

Add	Property	Test Value	Assersion Type
off	headers.content-type	application/json	Assert property exists
on	status	200	Assert it equals '200'
on	data.id	23	Assert it equals '23'
on	data.category	{ "id": 1, "name": "dog" }	Validate JSON schema
on	data.name	Lassie	Assert it equals 'Lassie'
off	data.photoUrls	[]	Assert property exists
off	data.tags	[]	Assert property exists
on	data.status	available	Assert it equals 'available'

Using the visual editor lets you create tests very fast. Once you click OK, you then have these steps added to your test:



HTTP POST
POST → https://pet-store-example1.p.rapidapi.com/pet

ASSERT EQUALS
a.status must equal 200

ASSERT EQUALS
a.data.id must equal 23

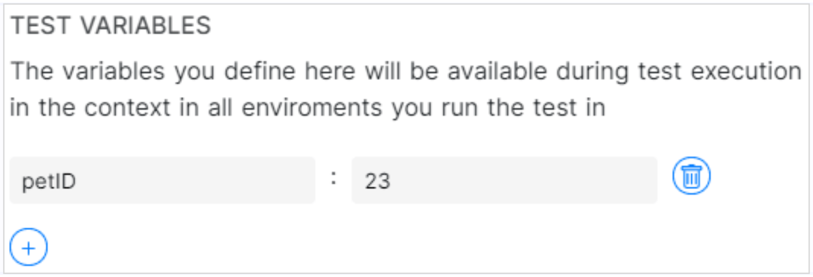
JSON VALIDATE
Validate JSON at a.data.category

ASSERT EQUALS
a.data.name must equal Lassie

ASSERT EQUALS
a.data.status must equal available

4. Create test variables for parameter values to parameterize your tests

Following the example above, switch to the test settings page. Do this by clicking "Settings" under the "Editor" link. But before you switch, make sure you save your test steps or they might not be there when you go back to the editor. There you have the ability to create test parameters for all your test steps:

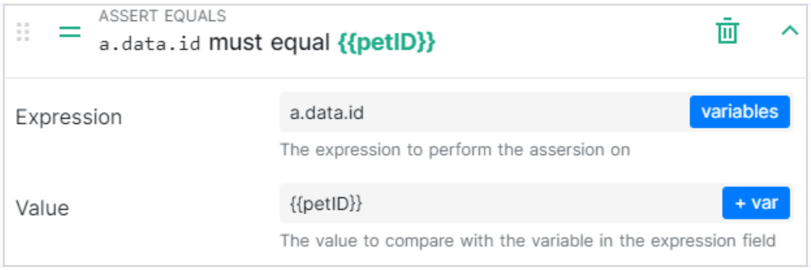


TEST VARIABLES

The variables you define here will be available during test execution in the context in all enviroments you run the test in

petID : 23

Use these variables in your test steps like this:



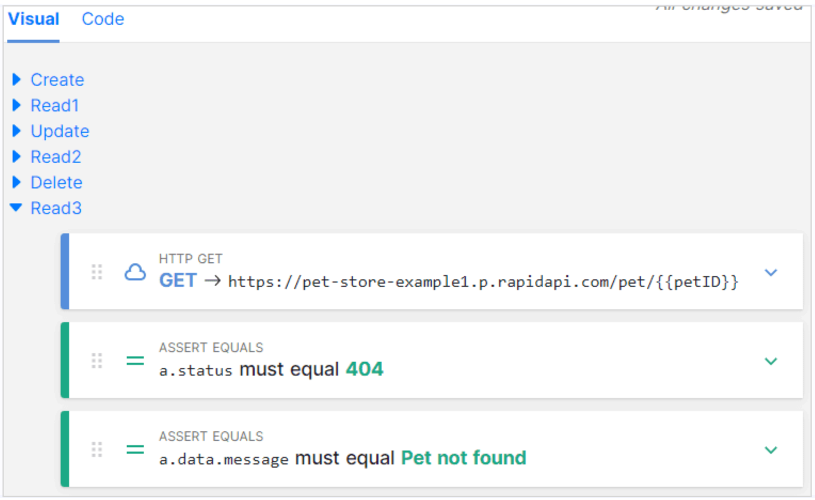
ASSERT EQUALS
`a.data.id` must equal {{petID}}

Expression: a.data.id
The expression to perform the assersion on

Value: {{petID}}
The value to compare with the variable in the expression field

5. Create groups of tests

Once you have several tests and test steps, you will have a long list which may be difficult to read or keep track of. To help with that, you can create groups which look and feel like folders. Each group can expand or collapse independently:



- Create
- Read1
- Update
- Read2
- Delete
- Read3

HTTP GET
GET → `https://pet-store-example1.p.rapidapi.com/pet/{{petID}}`

ASSERT EQUALS
`a.status` must equal 404

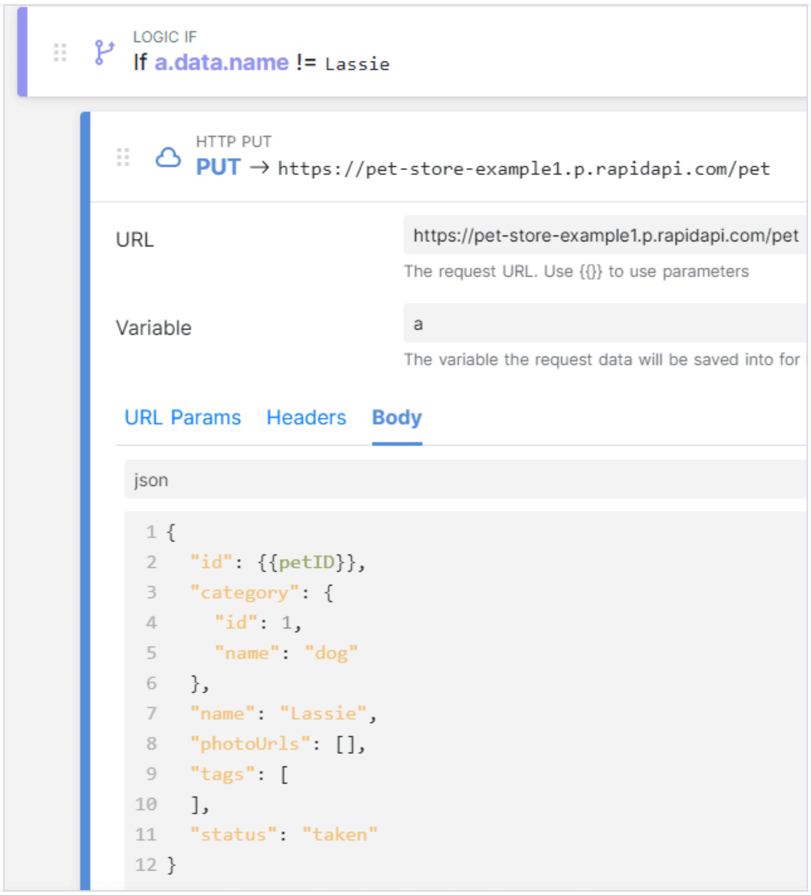
ASSERT EQUALS
`a.data.message` must equal Pet not found

6. Use if/else and looping logic to execute tests only under certain conditions

I learned something important about the pet store example API. If you send a put request to update a pet you will lose any data you do not send in that request. For example, after creating a pet named Lassie with a category of dog, I sent this data in a PUT request:

```
{
 "id": {{petID}},
 "status": "taken"
}
```

Then when I pulled the data for the same petID, the name and category were missing! In this context, the behavior is repeatable. If you have an API which returns variable results, you can use logic to help test it. For the example above, I was able to make my tests pass using an IF statement. I did that by adding another PUT request IF the name of the pet was not Lassie:



LOGIC IF
If a.data.name != Lassie

HTTP PUT
PUT → https://pet-store-example1.p.rapidapi.com/pet

URL: https://pet-store-example1.p.rapidapi.com/pet
The request URL. Use {{}} to use parameters

Variable: a
The variable the request data will be saved into for

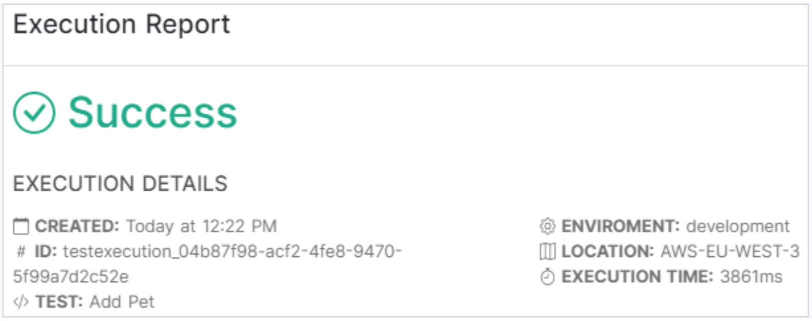
```
{
  "id": {{petID}},
  "category": {
    "id": 1,
    "name": "dog"
  },
  "name": "Lassie",
  "photoUrls": [],
  "tags": [
  ],
  "status": "taken"
}
```

7. Test from different locations to ensure consistent performance

Let's say you have a Pet Store located in Paris. So you want your shop website hosted on a server in Paris. And your API should have good response time in Paris. You can choose Paris from the dropdown of locations to run your test from. I tried that with my test which performs the following steps (ie API calls):

1. Add a pet with an ID defined by the petID variable named Lassie with the category of dog and the status of available.
2. Get the pet with the same petID to confirm the data is correct
3. Update the status to be "taken" for the same petID
4. Get the pet again, and if it's name is not Lassie send another update call. This time pass all the other parameters and give it a status of taken
5. Delete the pet using the same petID
6. Get the pet again and confirm it is not found

The above process covers the basic Create Read Update Delete (CRUD) operations. There is also an extra update and two extra reads. Here is the summary from the execution report:


Execution Report

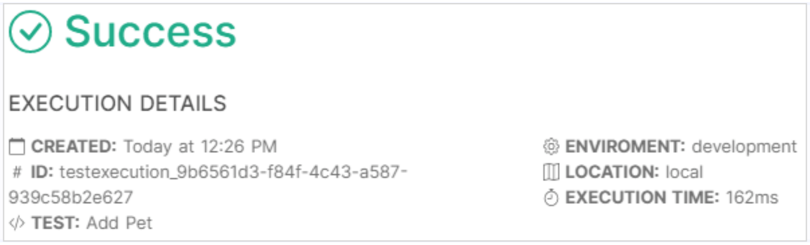
Success

EXECUTION DETAILS

CREATED: Today at 12:22 PM
ID: testexecution_04b87f98-acf2-4fe8-9470-5f99a7d2c52e
TEST: Add Pet

ENVIROMENT: development
LOCATION: AWS-EU-WEST-3
EXECUTION TIME: 3861ms

The execution time is almost 4 seconds, because the time for each API call averaged over half a second. Compare that to a local test which was 30x faster:

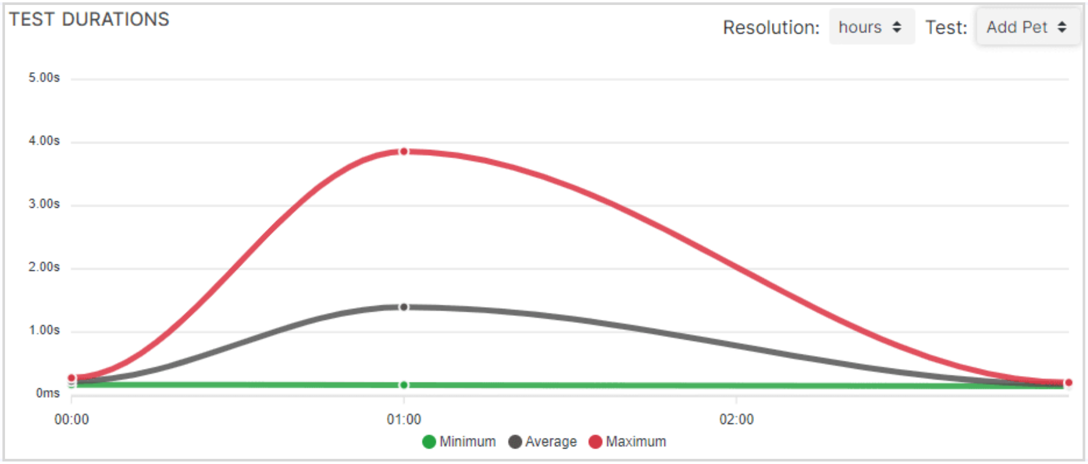

Success

EXECUTION DETAILS

CREATED: Today at 12:26 PM
ID: testexecution_9b6561d3-f84f-4c43-a587-939c58b2e627
TEST: Add Pet

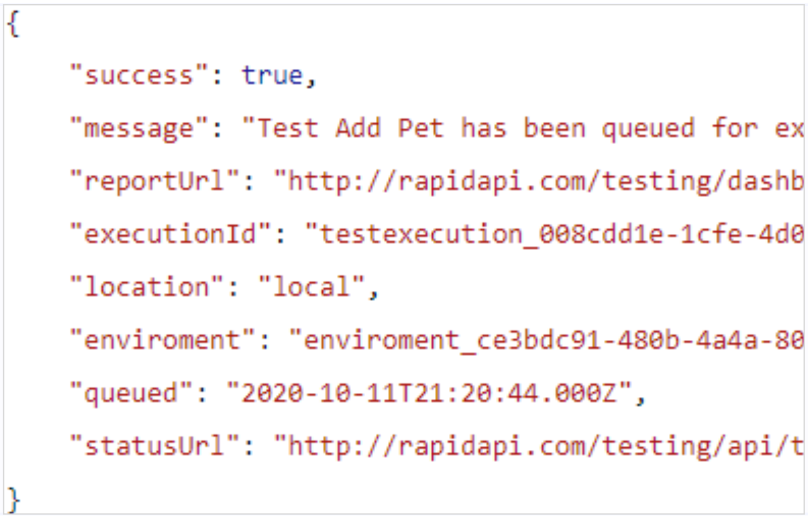
ENVIROMENT: development
LOCATION: local
EXECUTION TIME: 162ms

In real life, API calls are only "local" if they retrieve data from the same server which calls the API. For a pet store, I could imagine that scenario. But when you scale up that "local speed," luxury is often traded. It is usually traded for a distribution of tasks and resources across many servers. If you want to track variability in your API response times, you can look at the charts in the testing dashboard. For example, this chart shows the impact of switching the location. The location was changed from local to Paris and back again over a three-hour period:



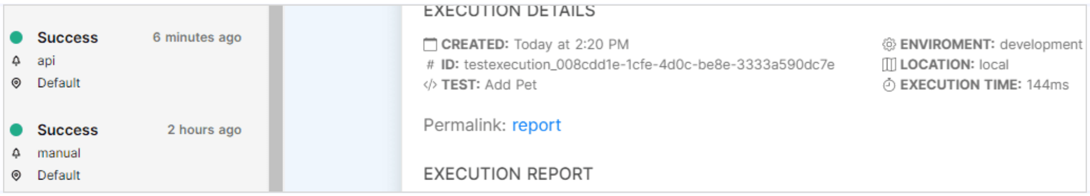
8. Trigger tests from API calls and review results from another URL. You can use triggers from a Jenkins playbook, GitHub action, or any other CI/CD tool

To integrate tests with an automated CI/CD tool, you will need a trigger. In the RapidAPI testing environment, each test is assigned a distinct URL - which will run the test. If you call that URL (put it into a browser to try it out), you'll see a JSON formatted object returned. There are two key elements returned, the status URL and the report URL:



```
{
    "success": true,
    "message": "Test Add Pet has been queued for ex
    "reportUrl": "http://rapidapi.com/testing/dashb
    "executionId": "testexecution_008cdd1e-1cfe-4d0
    "location": "local",
    "enviroment": "enviroment_ce3bdc91-480b-4a4a-80
    "queued": "2020-10-11T21:20:44.000Z",
    "statusUrl": "http://rapidapi.com/testing/api/t
}
```

The report URL takes you back to a page within your testing environment. In the list of results, you'll be able to tell what triggered each test run. The difference is in the picture below. The results entry with the word "api" underneath "Success" is an example using the URL trigger. The entry with "manual" in that position was initiated by a click of the Save and Run button:


Success 6 minutes ago
api
Default

Success 2 hours ago
manual
Default

EXECUTION DETAILS

CREATED: Today at 2:20 PM
ID: testexecution_008cdd1e-1cfe-4d0c-be8e-3333a590dc7e
TEST: Add Pet

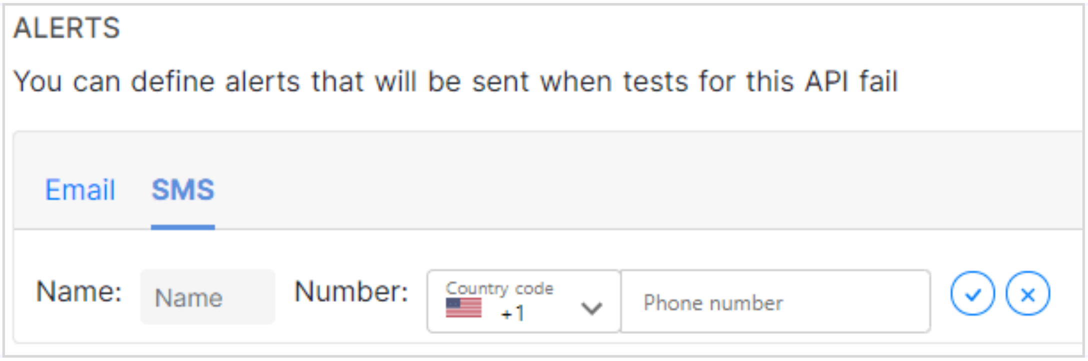
ENVIROMENT: development
LOCATION: local
EXECUTION TIME: 144ms

Permalink: report

EXECUTION REPORT

The status URL can be useful if you have long running tests. By the time I looked at my test status it was already completed. The status of the test can be fed into a CI/CD flow to automate other processes based on the results.

9. Use email or SMS alerts to notify you of test failures


ALERTS

You can define alerts that will be sent when tests for this API fail

If you want to know when your tests fail you don't need a separate CI/CD system. This would be useful if you schedule a test to run every 5 minutes or 5 hours and want to know if something went wrong. On the main settings page you can define emails and/or SMS enabled phone numbers. These will receive alerts whenever there is a failure.

RapidAPI Testing

The best practices for API Testing that have been highlighted through this document have been demonstrated using the RapidAPI Testing environment. However, these best practices can also be used in any testing environment or with any software. RapidAPI provides a framework that will enable you to speed up test development. It also increases test coverage and provides extensive metrics.

RapidAPI Testing is a functional API testing and monitoring solution that enables users and organizations to create and manage comprehensive API tests from development through deployment. RapidAPI Testing supports any API type including REST, SOAP, and GraphQL, simplifies monitoring across multiple geographies, and integrates with existing tooling throughout the development lifecycle. Additionally, RapidAPI Testing automatically syncs with the RapidAPI Marketplace and RapidAPI Enterprise Hub to ensure test coverage across all of an organization's APIs.

RapidAPI Testing enables users and enterprises to:

- **Ensure API Functionality** – Easily create intricate functional tests for deep validation of APIs
- **Centralize Monitoring** – Monitor and manage API tests across multiple geographies
- **Improve Efficiency** – Integrate to the CI/CD pipeline, collaborate across teams, and natively integrate with the RapidAPI Marketplace and Enterprise Hub

05/21

Global HQ
2 Shaw Alley, Fourth Floor
San Francisco, CA 94105

Contact
info@rapidapi.com
www.rapidapi.com

RapidAPI empowers millions of developers to build modern software with a Next-Generation API Platform comprising the world's largest marketplace and a fully-integrated toolset for API discovery, testing, publishing, consumption, and more.