

Testing an API with Hoppscotch, Javascript and Python

Table of Contents

- [Prerequisites and references](#)
- [Hoppscotch](#)
 - [Create an environment](#)
 - [Create a collection](#)
 - [Create requests](#)
 - [Task list](#)
 - [Create tasks](#)
 - [Additional requests](#)
 - [Other features](#)
- [Scripting requests in Javascript](#)
- [Scripting requests with Python](#)
- [Inspect CPU usage](#)

Once you have an API up and running you will want to test it to make sure that everything is working as expected. One way of testing an API is with a web client that access the API, but it is frequently useful to be able to test and debug an API directly.

In this practical we'll look at testing APIs with a GUI application (Hoppscotch) and with short custom scripts in Javascript and Python.



Prerequisites and references

- [Hoppscotch](#)  (<https://hoppscotch.com/download>).
- [Python Requests module](#)  (<https://docs.python-requests.org/en/latest/index.html>).

We'll need an API to test. The rest of this practical assumes that you have the Task API from week 3 up and running on an EC2 instance.

Hoppscotch

Hoppscotch is a GUI program that allows you to interact with APIs in a convenient way. You can use it two ways:

- Install Hoppscotch on your computer [Hoppscotch Download](#)  (<https://hoppscotch.com/download>).
- Run in your browser [Hoppscotch web app](#)  (<https://hoppscotch.io/>).

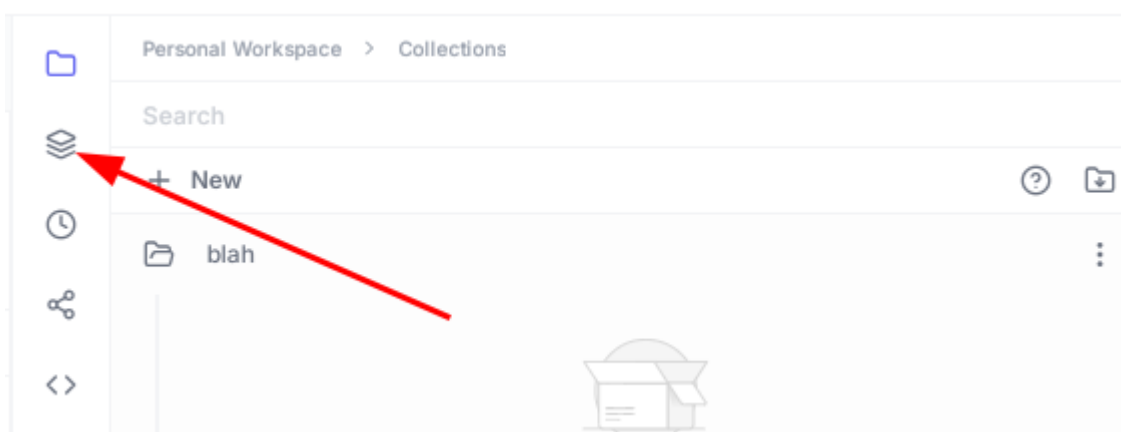
Keep in mind that the web version can only be used to test APIs that are accessible from the internet. So this won't work for testing a server running locally on a machine that is not accessible from the internet (eg. your laptop/desktop on a typical residential internet connection.)

Start up Hoppscotch.

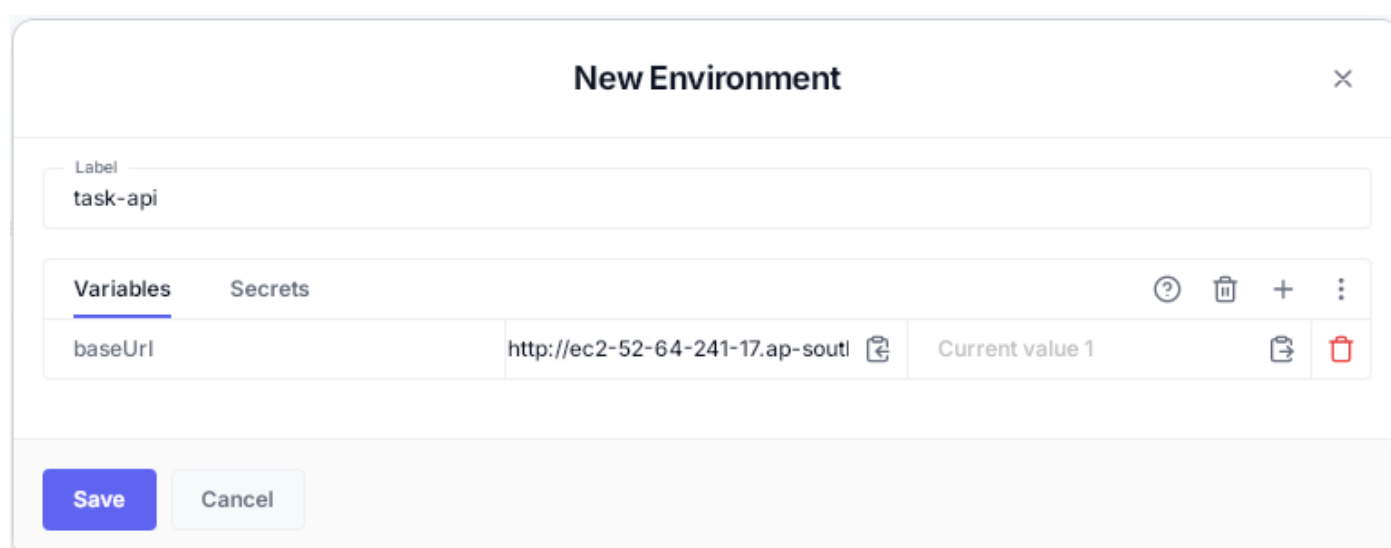
Create an environment

Environments define variables, such as a base URL, that are common to multiple API requests that you will use for testing.

- On the right in Hoppscotch, go to the Environments tab:



- Click *Add new* to create a new environment. This will pop up a window that allows us to configure the environment.
- Add a label, such as `task-api`.
- In the *Variables* tab click *Add new*
- Fill in the variable name as `baseUrl` and value to `http://<your EC2 dns name>:3000/tasks`

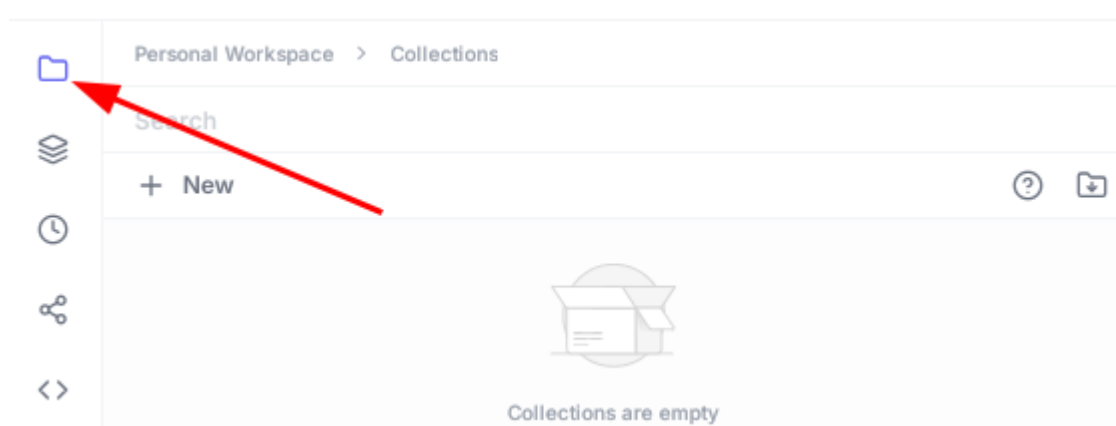


We won't need any more variables now, but keep in mind that you can add more variables if you find yourself needing to change parameters in requests.

Create a collection

In Hoppscotch a collection is a set of related requests. We'll define a collection for the requests required to test our API.

- Go to the collections tab



- Click *Add new* to create a new collection. Give it a name, such as *task-api requests*

Create requests

Task list

- Hover over the name of the collection you just created. Several icons will appear to the right. Click the one with tooltip *Add a request*. Give the request the name *list tasks*.
- You should now see a tab with label *GET list tasks*. This tab should already be selected.
- In the URL bar, type `<<baseUrl>>/` This refers to the environment variable we set up previously.
- Click *Send*. In the bottom half of the window you should see indications of a successful request, such as *Status: 200* and `[]` in the response body. At this point there are no tasks, so we just get an empty list.

Create tasks

- In the collection, create a new request called *Create task*
- In the URL type `<<baseUrl>>/`
- To the left of the URL, change *GET* to *POST*
- Examining the code for the Task API server, we see that the request to create a task requires a single parameter, which is the title. This is supplied in JSON format, i.e. `{ "title": "My first task" }`
- Go to the *Body* tab in the request. Change the *Content Type* to `application/json`.
- In the *Raw Request Body* pane, type `{ "title": "My first task" }`
- Click *Send*
- In the response Body you should see a JSON object with fields for ID, title, and completion status

Additional requests

We leave it to you to examine the code to find the remaining requests that the server allows and create requests for them. You'll need to create requests for getting a task by ID, updating a task, and deleting a task.

Other features

Explore some of the features of Hoppscotch which can be useful:

- Running all requests in a collection
- Javascript scripts to run before and after a request
- Generate code for a request to be used in Javascript/Python/command line

Scripting requests in Javascript

Sometimes we want to run a request multiple times, or script interaction with multiple requests (eg. testing creating and then deleting an item). The following script shows how to make multiple requests in Javascript. The script also outputs the average time required per request. Be sure to update the `endpoint` variable to point to your EC2 instance.

```
const endpoint = 'http://ec2-instance-dns-name:3000/tasks/';
const numberOfRequests = 100;
let totalResponseTime = 0;

async function loadTest() {
  for (let i = 0; i < numberOfRequests; i++) {
    const startTime = performance.now();

    try {
      const response = await fetch(endpoint, {
        method: 'GET',
      });

      if (!response.ok) {
        console.error(`Request ${i + 1} failed with status: ${response.status}`);
        continue;
      }

      const endTime = performance.now();
      const responseTime = endTime - startTime;
      totalResponseTime += responseTime;

      console.log(`Request ${i + 1} completed in ${responseTime.toFixed(2)}ms`);
    } catch (error) {
      console.error(`Request ${i + 1} encountered an error: ${error.message}`);
    }
  }

  const averageResponseTime = totalResponseTime / numberOfRequests;
  console.log(`Average response time: ${averageResponseTime.toFixed(2)}ms`);
}
```

```
loadTest();
```

Do not run this load-testing script on your EC2 instance. It should be ran elsewhere. You could install NodeJs on your personal computer and run it as a script.

Scripting requests with Python

The following script demonstrates similar functionality to the Javascript script above. You can run this on your personal computer if you have Python installed. You will need to install the `requests` library with `pip3 install requests`. Be sure to update the `url` variable to point to your EC2 instance.

```
import requests
import time

url = "http://ec2-instance-dns-name:3000/tasks/"
numberOfRequests = 500

totalTime = 0

def time_ms():
    return round(time.time() * 1000)

for i in range(numberOfRequests):
    startTime = time_ms()
    response = requests.get(url)
    request_time = time_ms() - startTime
    totalTime += request_time

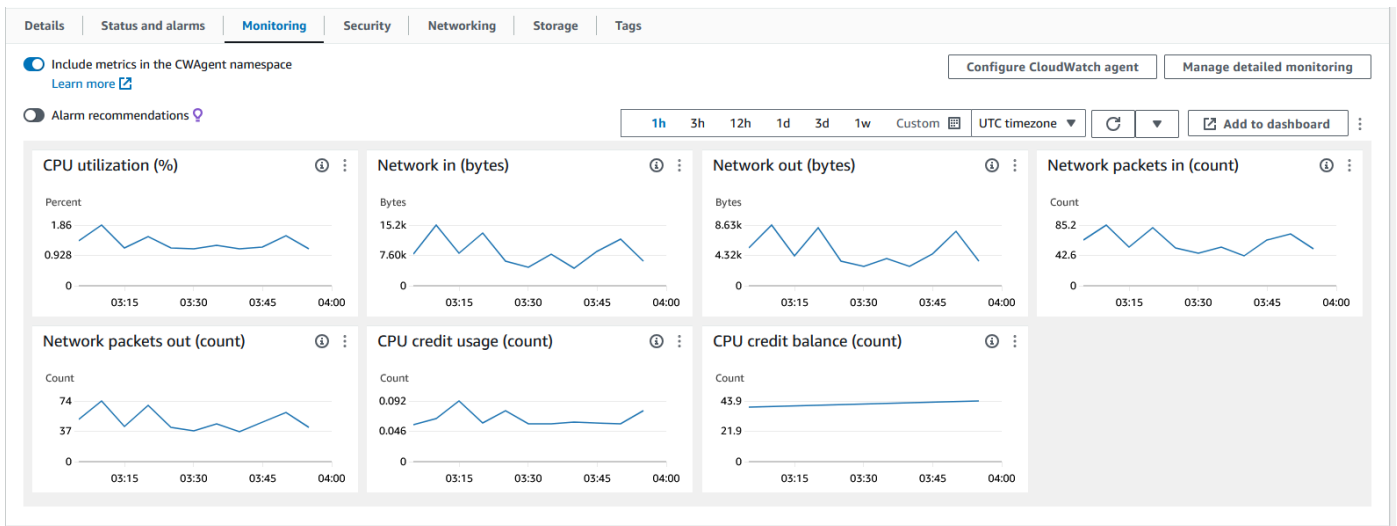
    print(f'Request {i + 1} returned with status: {response.status_code} in {request_time}ms')

print(f'Average time {totalTime / numberOfRequests}')
```

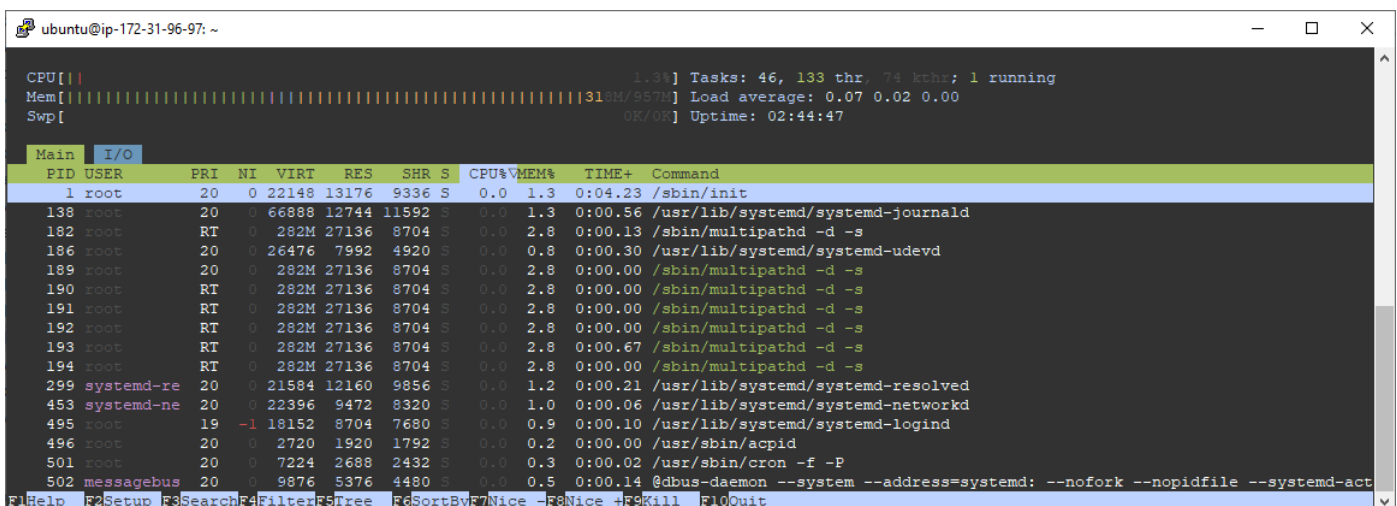
Inspect CPU usage

To inspect CPU usage on an EC2 instance, you can use two approaches demonstrated in the screenshots.

The first approach uses AWS CloudWatch monitoring, which provides a detailed dashboard showing various metrics like CPU utilisation, network in/out, and CPU credit usage. This method offers a high-level overview of the instance's performance over different time ranges. In the EC2 console, find your instance then switch from the Details tab to the Monitoring tab.



The second approach utilises the `htop` command within the instance's terminal. `htop` offers a real-time, detailed view of CPU usage per process, along with memory usage and other system resources. This method is useful for identifying specific processes consuming high CPU resources. Both methods together provide comprehensive insights into CPU performance and can help in diagnosing performance issues effectively. Run `htop` from the Ubuntu terminal of your instance.



TEQSA PRV12079 | CRICOS 00213J | ABN 83 791 724 622