K6 and Grafana K6 Integration Steps

Effortlessly Integrating K6 for Seamless Performance Testing

Neeharica Madanu

Studio-Ghibli

31 March 2024

# K6

K6 is a performance testing software that enables developers to write test scripts in JavaScript and execute them to simulate various levels of load on their system. It offers scripting in JavaScript, performance testing, integrations, and metrics and analysis. K6 is commonly utilised for load testing and performance testing of APIs, microservices, and web applications. It assists developers in ensuring that their systems and applications perform optimally under diverse load conditions. Overall, K6 is a flexible and potent tool for developers to conduct performance testing.

# Grafana K6

Grafana and k6 are typically utilised concurrently for performance monitoring and testing. Grafana is an open-source platform that empowers users to query, visualise, and comprehend metrics from a variety of data sources, such as Graphite, Prometheus, and InfluxDB. It is frequently employed to create dashboards for monitoring diverse aspects of systems and applications. k6, on the other hand, is an open-source load testing tool that enables users to write test scripts in JavaScript and execute tests from the command line. It provides comprehensive performance metrics and integrates with various systems for result analysis. When utilised together, Grafana can be integrated with k6 to visualise and analyse the performance test results generated by k6. This integration allows users to create custom dashboards in Grafana to display the performance metrics collected during load tests conducted with k6. To summarise, Grafana is a monitoring and visualisation tool, while k6 is a load testing tool, and when used in conjunction, they can be employed to monitor and visualise the performance of applications and systems.

# K6 Integration Steps

Step 1: Install k6

To begin using k6, we will first need to install it. We can download and install k6 from the official website or utilise a package manager like Homebrew or Chocolatey. For instance, to install k6 using Homebrew on macOS, we can use the following command: brew install k6

Step 2: Create a Test Script Create a separate file for our k6 test script.

Here's an example of a simple test script using k6 to make a GET request to a sample API:

import http from 'k6/http';

import { check } from 'k6';

export default function () {

let response = http.get(‘<https://api.example.com>');

check(response, { 'status is 200': (r) => r.status === 200, });

}

In this script, we're making a GET request to <https://api.example.com> and verifying if the response status is 200.

Step 3: Run the Test After creating the test script, we can execute it using the k6 CLI.

Navigate to the directory where the test script is located and run the following command:

k6 run script.js

Replace script.js with the actual name of the test script file.

Step 4: View Test Results After running the test.

k6 will provide us with comprehensive test results, including response times, throughput, and error rates. We can view these results in the console or generate HTML reports for more in-depth analysis.

Step 5: Integrate Tests into CI/CD

We can also incorporate k6 tests into our CI/CD pipeline to automate performance testing. For example, we can use tools like Jenkins, CircleCI, or GitHub Actions to execute k6 tests as part of your deployment process.

# Testing Scenarios Integration Steps

**Sequential Testing**

Script Development: Create a test script in JavaScript using k6 to design a sequential testing scenario.

Utilise the following code:

```javascript

import http from 'k6/http';

import { sleep } from 'k6';

export default function () {

[http://{Host}]-get([URL, HTTP\_METHOD], [HEADERS]); [http://{Host}]-resp([HTTP\_STATUS\_CODE]);

sleep(1);

}

```

Execute the Test: Run the test script with k6 from the command line to perform the sequential load testing.

```bash

k6 run sequential\_test.js

```

Collect Data: Retrieve the performance metrics generated by k6 during the test execution.

**Batch Testing**

Script Development: Develop a test script in JavaScript to simulate batch testing with k6, incorporating the essential modules from 'k6/http' and 'k6/sleep'. The script will simulate batch requests through iterating ten times, making a GET request to '<https://example.com/>' and pausing for one second after each request using the 'k6/sleep' module.

```

javascript

import http from 'k6/http';

import { sleep } from 'k6';

export default function () {

// Simulate batch requests

for (let i = 0; i < 10; i++) {

http.get('https://example.com/');

sleep(1);

}

}

```

Script Execution: Execute the batch testing script using k6 to simulate the performance under a batch load by running the command 'k6 run batch\_test.js'.

Metrics Collection: Collect the performance metrics generated by k6 for batch testing.

**Average Load Testing**

Develop a test script in JavaScript to simulate average load testing using k6.

Script Development:

```

javascript

import http from 'k6/http';

import { sleep } from 'k6';

export default function () {

// Simulate average load

http.get('https://example.com/');

sleep(10);

}

```

Execute Test: To initiate the average load test, execute the k6 script from the command line using the following command:

```

sql

k6 run average\_load\_test.js

```

**Stress Testing**

Develop a JavaScript stress testing script using k6 to apply stress to the system.

```javascript

import http from 'k6/http';

import { sleep } from 'k6';

export default function () {

// Simulate stress by making multiple requests

http.get('https://example.com/');

http.get('https://example.com/');

http.get('https://example.com/');

sleep(1);

}

```

// Execute the stress testing script using k6 to simulate system performance under stress conditions

```

k6 run stress\_test.js

```

**Smoke Testing**

Develop a JavaScript test script utilising k6 to design a smoke testing scenario that encompasses a fundamental request to the homepage of a website.

import http from 'k6/http';

export default function () {

// Construct a smoke testing scenario that incorporates a request to the API

http.get([(‘https://example.com/'), ('<https://example.com/%27>' )])

}

Test Execution:

Execute the smoke testing script using k6 to carry out a swift evaluation of the system's efficiency.

k6 run smoke\\_test.js

**Spike Testing**

Script Creation: Develop a JavaScript test script for spike testing using k6.

```javascript

import http from 'k6/http';

import { sleep } from 'k6';

export default function () {

// Mimic sudden load spikes

http.get('https://example.com/');

sleep(0.1);

http.get('https://example.com/');

sleep(0.1);

http.get('https://example.com/');

}

Test Execution: Execute the spike testing script with k6 to observe the system's behaviour under sudden load spikes.

k6 run spike\_test.js

```

**Soak Testing**

Create a JavaScript test script utilising k6 for soak testing, with the objective of evaluating system stability over an extended period.

import http from 'k6/http';

import { sleep } from 'k6';

export default function () {

// Utilise k6 to simulate continuous load over time for soak testing

http.get('https://example.com/');

sleep(5);

}

Test Execution: Initiate soak testing by running the k6 script, which simulates continuous load over time to evaluate system stability.

k6 run soak\\_test.js

# Grafana K6 Integration Steps

To create a Grafana Cloud account, navigate to Grafana Cloud and sign up for a free account.

After creating the account, proceed to the "Grafana K6" section and start testing.

To create a new project in K6 Cloud, navigate to the left pane and select "Create new project."

Once we have created the project, we can access the test case in the project scope. This will demonstrate how to run k6 from our local machine and upload the results to the cloud.

To install k6, use the command "brew install k6" on our local machine.

To login to Grafana Cloud from our local machine, use the command "k6 login cloud --token <token>."

Create a code snippet in your local machine with the following code:

```javascript

import http from 'k6/http';

import { sleep } from 'k6';

export const options = {

vus: 10,

duration: '30s',

ext: {

loadimpact: {

// Project: Ghibli

projectID: <id>,

// Test runs with the same name groups test runs together.

name: 'Test (31/03/2024-23:49:29)'

}

}

};

export default function() {

http.get('https://test.k6.io');

sleep(1);

}

```

To run the local K6 command that uploads the results to the cloud, use the command " k6 run --out=cloud script.js."

To view the results, navigate to the "Dashboards" section in Grafana and select the dashboard that corresponds to your datasource.