PERFORMANCE TESTING

K6

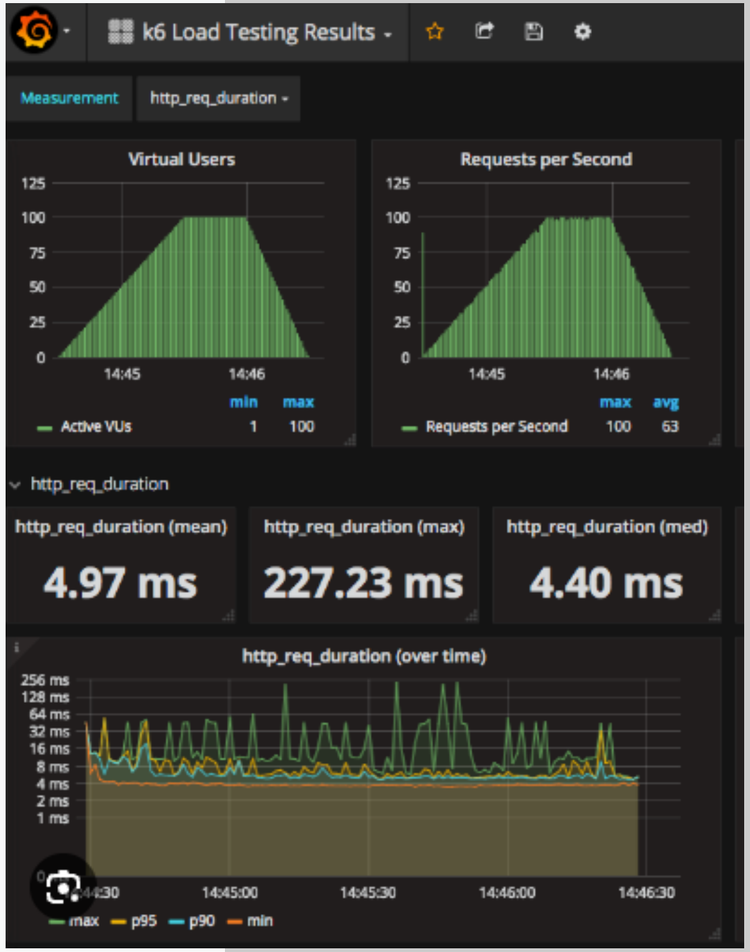
CONTENT

WHAT IS K6?

An open source load testing tool released by Load Impact in 2017

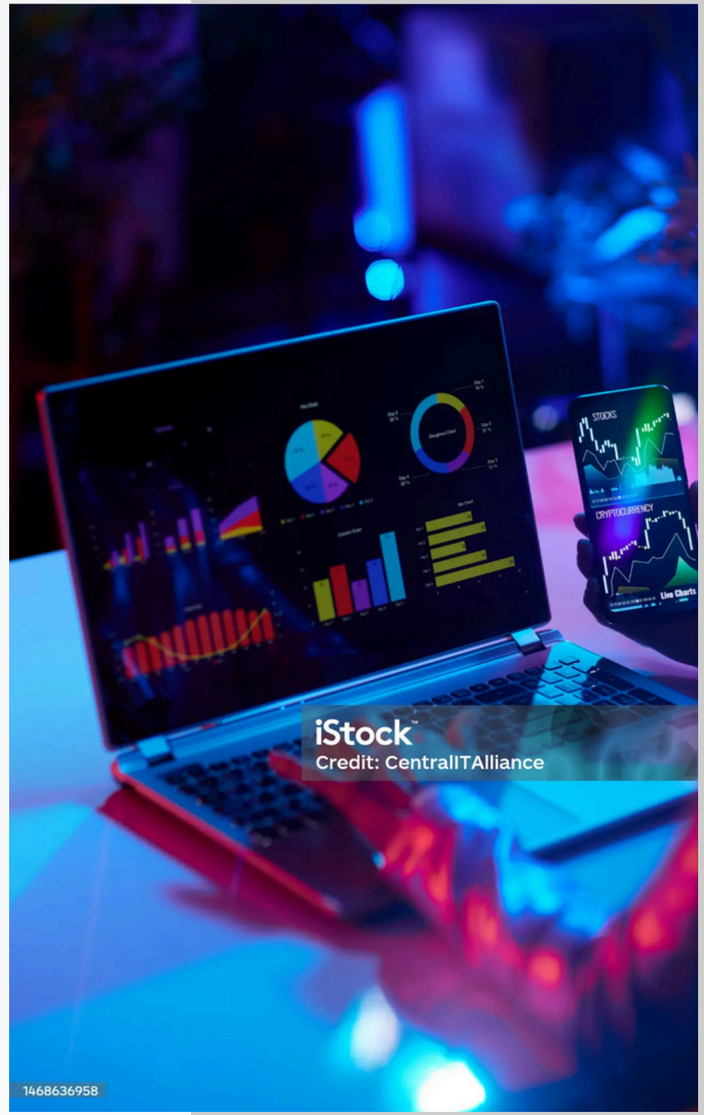
CLI tool with developer-friendly APIs. k6 is written in Go, but scripts are written in JavaScript

A package that can be installed either in bare metal (Windows, Linux, Macos), or in a docker environment with the help of the official docker image provided by the Grafana k6 team.



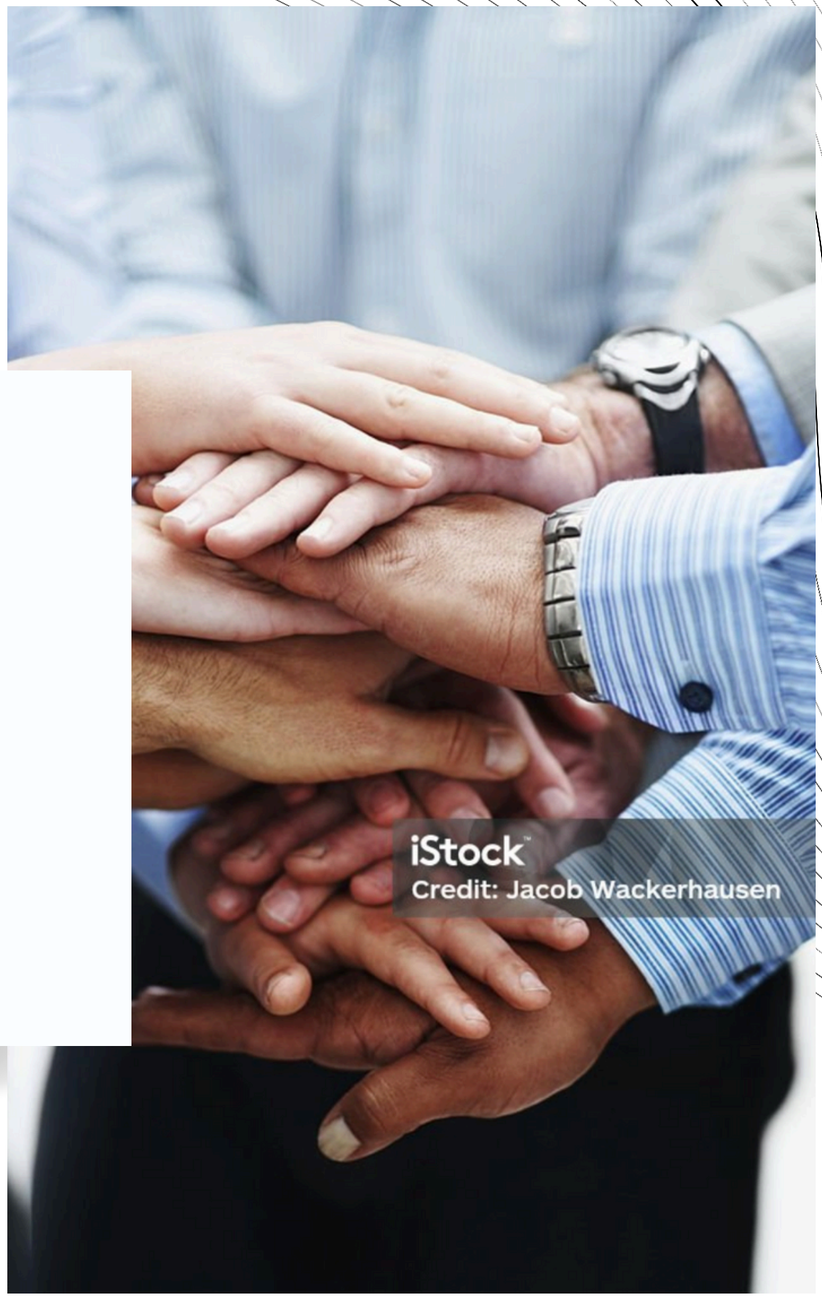
WHY K6?

- **Easy to use:** k6 is written in JavaScript and has a simple CLI (command-line interface) that is easy to learn.
- **Powerful:** k6 can simulate a high volume of traffic and can be used to test a wide variety of web applications and APIs.
- **Open-source:** k6 is a free and open-source tool that can be customized and extended.



BENEFITS OF USING K6

- Identify performance bottlenecks.
- Improve application performance.
- Ensure application reliability.
- Save money.



EASY TO START!

1. Install k6 on your machine:

```
brew install k6
```

2. Run a simple local script:

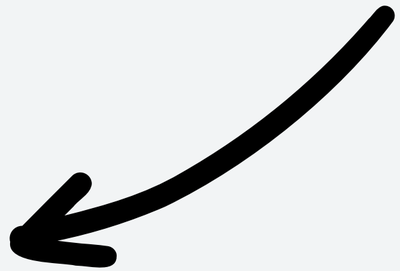
```
k6 new
```

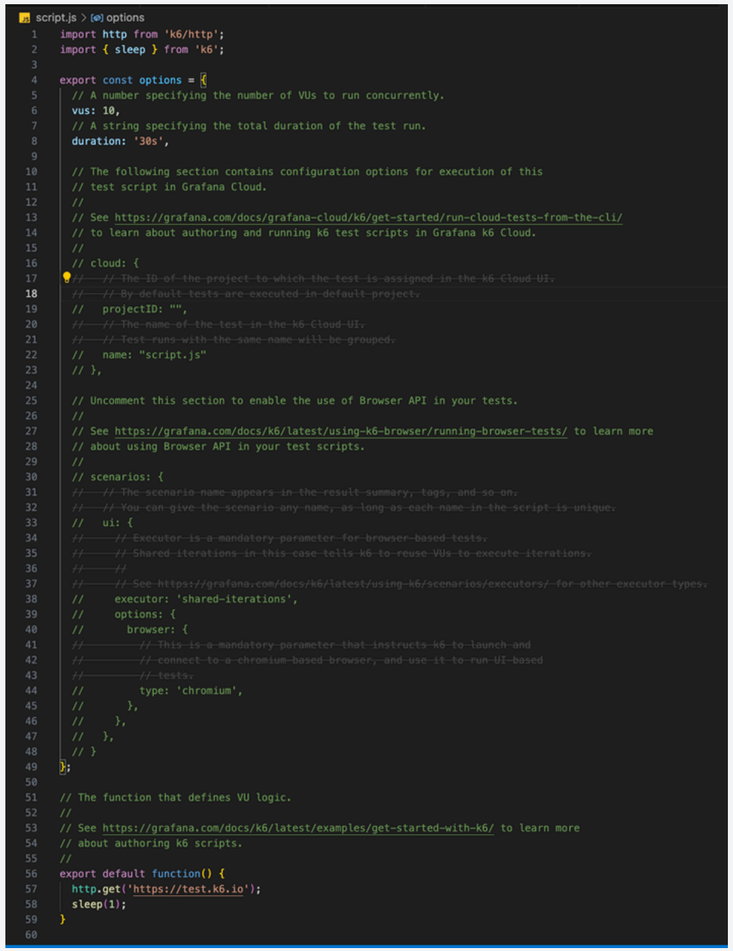
3. Run k6 with the following command:

```
k6 run script.js
```

script.js > ...

```js
import http from 'k6/http';
import { sleep } from 'k6';

export const options = {
  // A number specifying the number of VUs to run concurrently.
  vus: 10,
  // A string specifying the total duration of the test run.
  duration: '30s',
};

// The function that defines VU logic.
export default function() {
  http.get('https://test.k6.io');
  sleep(1);
}
```





TYPES OF K6 TESTS

k6 directly sends HTTP requests and analyzes the responses, simulating user interactions at a lower level. This is useful for testing API endpoints and backend functionality under load.

PROTOCOL-LEVEL TESTS

Simulate real user interactions with a web application through a headless browser. k6 controls a browser window in the background, mimicking user actions like clicking buttons, filling out forms, and navigating pages. This is ideal for testing the overall user experience under load.

BROWSER-LEVEL TESTS

Combines both browser-level and protocol-level testing within a single script. This allows you to simulate realistic user scenarios that involve both user interaction and server communication. You can have k6 perform browser actions and then follow up with specific API calls to test your application's integrated behavior.

HYBRID TESTS

STRATEGIES

TEST FOR FUNCTIONAL BEHAVIOR

- **What**: Checks if your app works as expected (login, forms, etc.)
- **How**: Simulates user actions and verifies results with assertions.

JavaScript

```
// Import necessary modules
import { check } from 'k6';
import http from 'k6/http';

export default function () {
  // define URL and request body
  const url = 'https://test-api.k6.io/auth/basic/login/';
  const payload = JSON.stringify({
    username: 'test_case',
    password: '1234',
  });
  const params = {
    headers: {
      'Content-Type': 'application/json',
    },
  };

  // send a post request and save response as a variable
  const res = http.post(url, payload, params);

  // check that response is 200
  check(res, {
    'response code was 200': (res) => res.status == 200,
  });
}
```

TEST FOR PERFORMANCE

- **What:** Measures how your app handles high traffic (speed, errors).
- **How:** Simulates many users and monitors performance metrics (response time, errors).

COMPONENTS

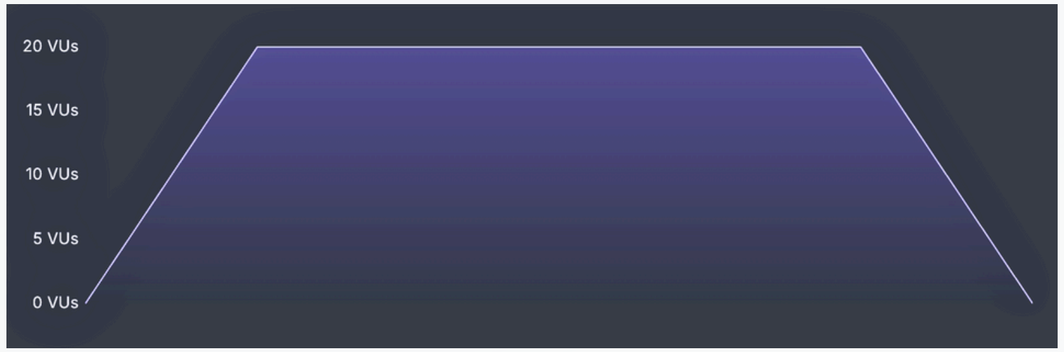
- HTTP Requests
- Options
- Checks and Thresholds
- Modules
- Metrics
- Groups and Tags

OPTIONS

- VUs
- Iterations
- Duration
- Stages

```
export let options = {
  vus: 10,
  iterations: 40,
};
```

```
  vus: 10,
  duration: '2m'
```




```
stages: [
  { target: 20, duration: '1m' },
  { target: 20, duration: '3m30s' },
  { target: 0, duration: '1m' },
],
```

CHECKS

- Check for HTTP response code
- Check for text in response body
- Check for response body size

k6chaijs

A library to provide BDD assertions in k6 based on ChaiJS. You can use k6chaijs as an alternative to check and group.

JavaScript Copy

```javascript
import http from 'k6/http';
import { describe, expect } from 'https://jslib.k6.io/k6chaijs/4.3.4.3/index.js';

export default function testSuite() {
  describe('Fetch a list of public crocodiles', () => {
    const response = http.get('https://test-api.k6.io/public/crocodiles');

    expect(response.status, 'response status').to.equal(200);
    expect(response).to.have.validJsonBody();
    expect(response.json().length, 'number of crocs').to.be.above(4);
  });
}
```

JavaScript

```javascript
import { check } from 'k6';
import http from 'k6/http';

export default function () {
  const res = http.get('http://test.k6.io/');
  check(res, {
    'is status 200': (r) => r.status === 200,
    'body size is 11,105 bytes': (r) => r.body.length == 11105,
  });
}
```

bash

```
$ k6 run checks.js

  ...
    ✓ is status 200
    ✓ body size is 11,105 bytes

  ...
  checks.....................: 100.00% ✓ 2        ✗ 0
  data_received..............: 11 kB   20 kB/s
```

THRESHOLDS

- Thresholds are the pass/fail criteria that you define for your test metrics.

- If the performance of the system under test (SUT) does not meet the conditions of your threshold, the test finishes with a failed status.

The most common types of thresholds you can set:

- Error rate
- Response time
- Checks

Testing best practice: Use error rate, response time, and checks thresholds in your tests where possible.

```
import http from 'k6/http';
import { check, sleep } from 'k6';

export let options = {
  stages: [
    { duration: '30m', target: 100 },
    { duration: '1h', target: 100 },
    { duration: '5m', target: 0 },
  ],
  thresholds: {
    http_req_failed: [{
      threshold: 'rate<=0.05',
      abortOnFail: true,
    }],
    http_req_duration: ['p(95)<=100'],
    checks: ['rate>=0.99'],
  },
};

export default function() {
  let url = 'https://httpbin.test.k6.io/post';
  let response = http.post(url, 'Hello world!');
  check(response, {
      'Application says hello': (r) => r.body.includes('Hello world!')
  });

  sleep(Math.random() * 5);
}
```

MODULES

```
import http from 'k6/http';
```

It’s common to import modules, or parts of modules, to use in your test scripts. In k6, you can import different kinds of modules:

- Built-in modules
- Local modules
- Remote modules
- Extension modules

JavaScript

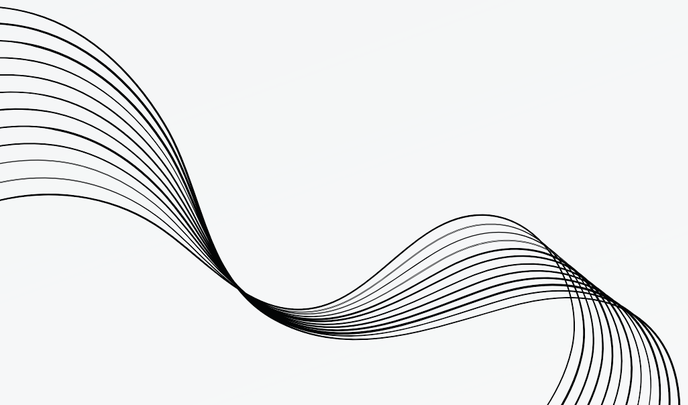
```
//my-test.js
import { someHelper } from './helpers.js';

export default function () {
  someHelper();
}
```

JavaScript

```
//helpers.js
export function someHelper() {
  // ...
}
```

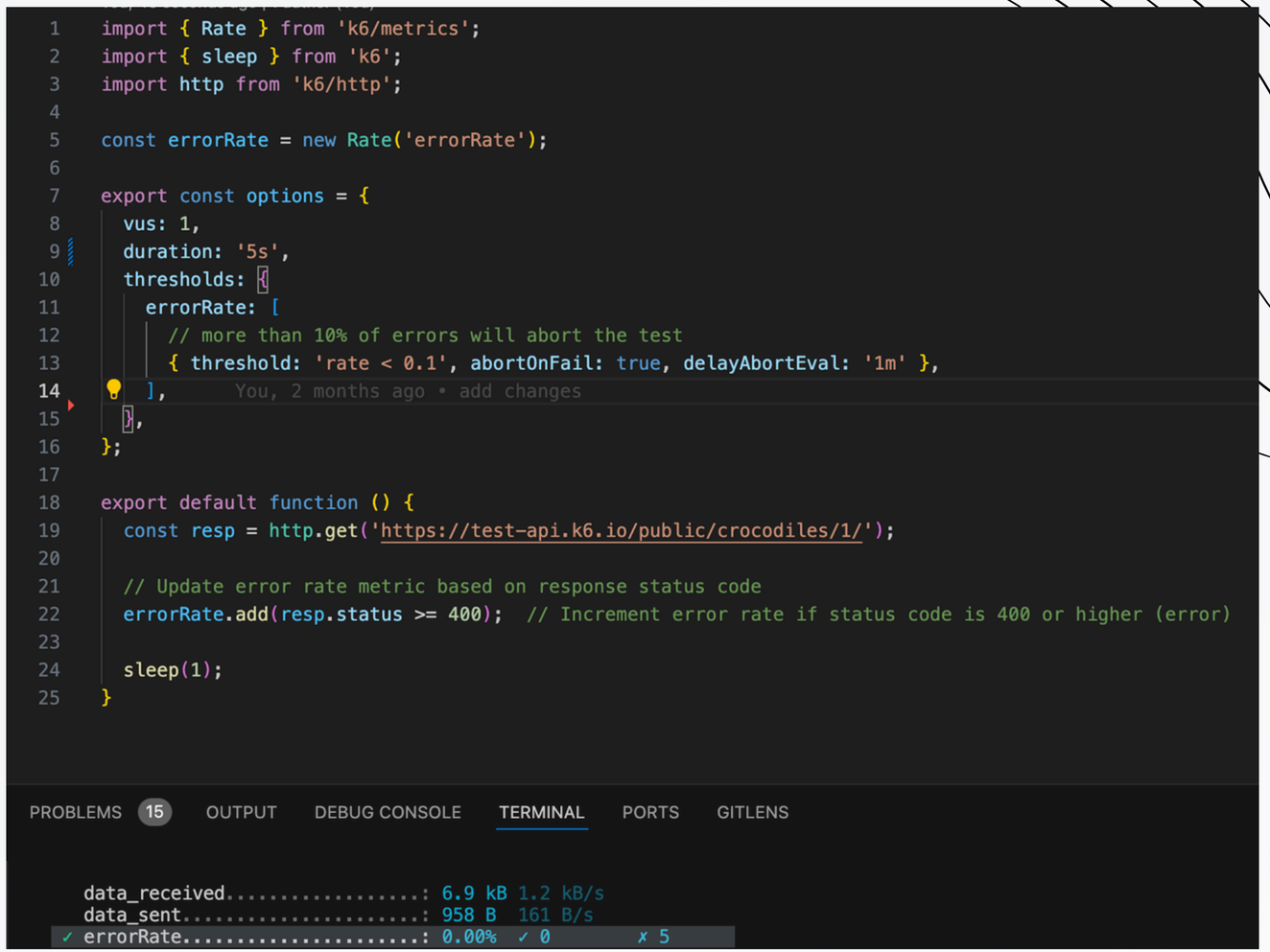
JavaScript

```
import { randomItem } from 'https://jslib.k6.io/k6-utils/1.2.0/index.js';

export default function () {
  randomItem();
}
```



METRICS

- **Counters** sum values.
- **Gauges** track the smallest, largest, and latest values.
- **Rates** track how frequently a non-zero value occurs.
- **Trends** calculates statistics for multiple values (like mean, mode or percentile).

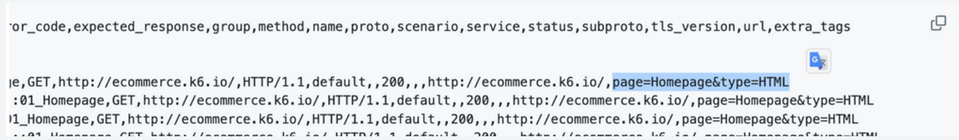


TAGS AND GROUPS

To help you visualize, sort, and filter your test results, k6 adds the following to your results.

- **Tags** categorize your checks, thresholds, custom metrics, and requests for in-depth filtering.
- **Groups** apply tags to the script's functions.

```
import { sleep, group } from 'k6'
import http from 'k6/http'

export default function () {
  let response

  group('01_Homepage', function () {
    response = http.get('http://ecommerce.k6.io/', {
      tags: {
        page: 'Homepage',
        type: 'HTML',
      }
    })

        // ... (other requests in the group)

        // ... (other requests in the group)

  })
}
```




```
import { group } from 'k6';

export default function () {
  group('01_VisitHomepage', function () {
    // ...
  });
  group('02_ClickOnProduct', function () {
    // ...
  });
  group('03_AddProductToCart', function () {
    // ...
  });
  group('04_ViewCart', function () {
    // ...
  });
  group('05_ProceedToCheckout', function () {
    // ...
  });
}
```

USING K6 BROWSER

```
import { browser } from 'k6/browser';
import { check } from 'k6';

export const options = {
  scenarios: {
    ui: {
      executor: 'shared-iterations',
      options: {
        browser: {
          type: 'chromium',
        },
      },
    },
  },
  thresholds: {
    checks: ['rate==1.0'],
  },
};
```

```
export default async function () {
  const context = await browser.newContext();
  const page = await context.newPage();

  try {
    await page.goto("https://test.k6.io/my_messages.php");

    await page.locator('input[name="login"]').type("admin");
    await page.locator('input[name="password"]').type("123");

    await Promise.all([
      page.waitForNavigation(),
      page.locator('input[type="submit"]').click(),
    ]);

    const header = await page.locator("h2").textContent();
    check(header, {
      header: (h) => h == "Welcome, admin!",
    });
  } finally {
    await page.close();
  }
}
```

HYBRID PERFORMANCE

```
import { browser } from 'k6/browser';
import { check } from 'k6';
import http from 'k6/http';

export const options = {
  scenarios: {
    browser: {
      executor: 'constant-vus',
      exec: 'browserTest',
      vus: 1,
      duration: '10s',
      options: {
        browser: {
          type: 'chromium',
        },
      },
    },
    news: {
      executor: 'constant-vus',
      exec: 'news',
      vus: 20,
      duration: '1m',
    },
  },
};
```

```
export async function browserTest() {
  const page = await browser.newPage();

  try {
    await page.goto('https://test.k6.io/browser.php');

    await page.locator('#checkbox1').check();

    const info = await page.locator('#counter-button').textContent();
    check(info, {
      'checkbox is checked': (info) => info === 'Thanks for checking the box',
    });
  } finally {
    await page.close();
  }
}

export function news() {
  const res = http.get('https://test.k6.io/news.php');

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

RESULTS

The end-of-test summary reports details and aggregated statistics for the primary aspects of the test:

- Summary statistics about each built-in and custom metric (e.g. mean, median, p95, etc).
- A list of the test's groups and scenarios
- The pass/fail results of the test's thresholds and checks.

```
Ramp_Up ✓ [======================================] 00/20 VUs  30s
     ■ GET home - https://example.com/

       ✓ status equals 200

     ■ Create resource - https://example.com/create

       ✗ status equals 201
        ↳  0% — ✓ 0 / ✗ 45

     checks.........................: 50.00% ✓ 45       ✗ 45
     data_received..................: 1.3 MB 31 kB/s
     data_sent......................: 81 kB  2.0 kB/s
     group_duration.................: avg=6.45s    min=4.01s    med=6.78s    max=10.15s   p(90)=9.29s    p(95)=
     http_req_blocked...............: avg=57.62ms  min=7µs      med=12.25µs  max=1.35s    p(90)=209.41ms p(95)=
     http_req_connecting............: avg=20.51ms  min=0s       med=0s       max=1.1s     p(90)=100.76ms p(95)=
   ✗ http_req_duration..............: avg=144.56ms min=104.11ms med=110.47ms max=1.14s    p(90)=203.54ms p(95)=
       { expected_response:true }...: avg=144.56ms min=104.11ms med=110.47ms max=1.14s    p(90)=203.54ms p(95)=
     http_req_failed................: 0.00%  ✓ 0        ✗ 180
     http_req_receiving.............: avg=663.96µs min=128.46µs med=759.82µs max=1.66ms   p(90)=1.3ms    p(95)=
     http_req_sending...............: avg=88.01µs  min=43.07µs  med=78.03µs  max=318.81µs p(90)=133.15µs p(95)=
     http_req_tls_handshaking.......: avg=29.25ms  min=0s       med=0s       max=458.71ms p(90)=108.31ms p(95)=
     http_req_waiting...............: avg=143.8ms  min=103.5ms  med=109.5ms  max=1.14s    p(90)=203.19ms p(95)=
     http_reqs......................: 180    4.36938/s
     iteration_duration.............: avg=12.91s   min=12.53s   med=12.77s   max=14.35s   p(90)=13.36s   p(95)=
     iterations.....................: 45     1.092345/s
     vus............................: 1      min=1      max=19
     vus_max........................: 20     min=20     max=20

ERRO[0044] some thresholds have failed
```

CUSTOM SUMMARY

With handleSummary(), you can completely customize your end-of-test summary.

```js
import http from 'k6/http';

export default function () {
  http.get('https://test.k6.io');
}

export function handleSummary(data) {
  return {
    'summary.json': JSON.stringify(data), //the default data object
  };
}
```

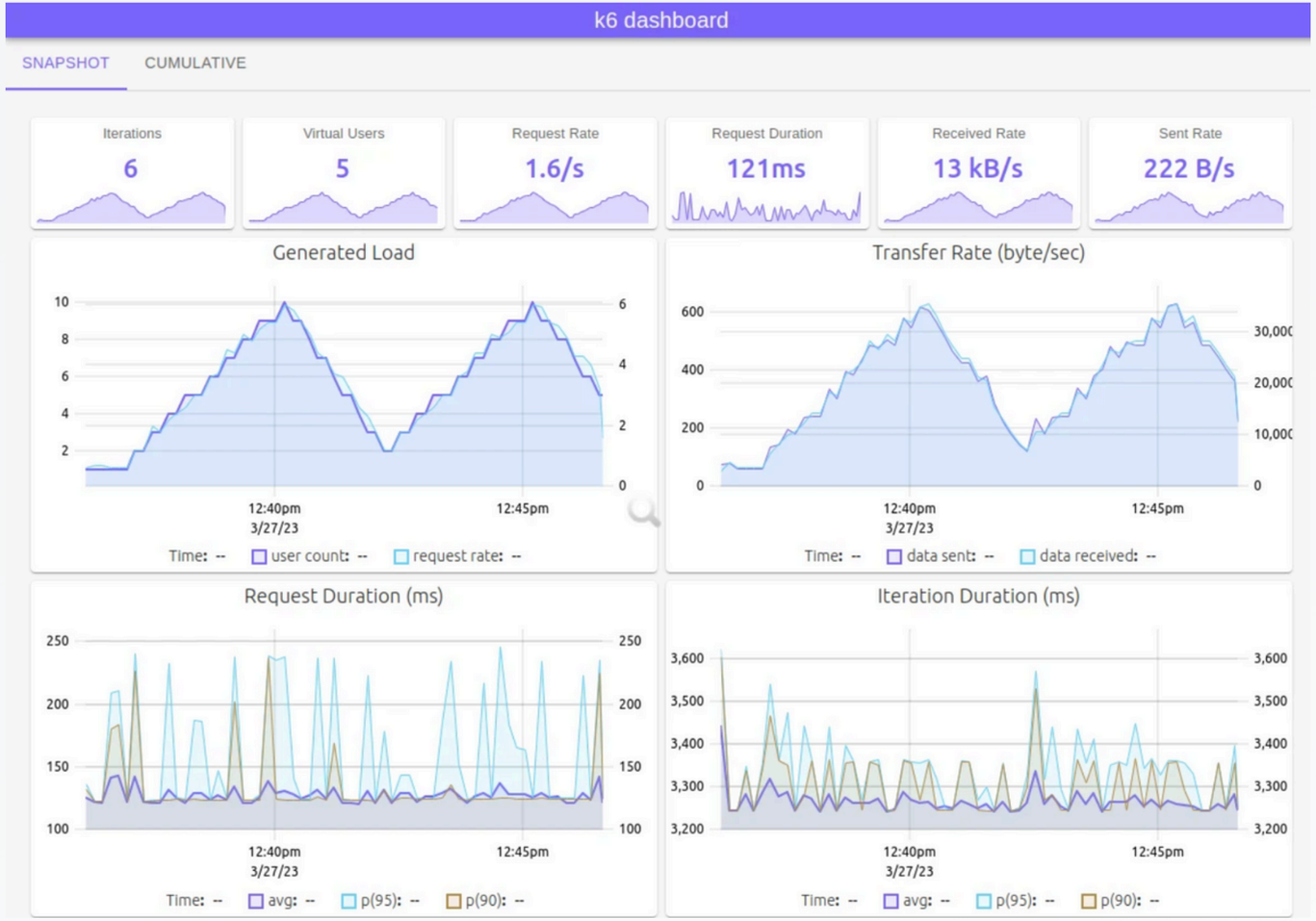
```js
export function handleSummary(data) {
  const med_latency = data.metrics.iteration_duration.values.med;
  const latency_message = `The median latency was ${med_latency}\n`;

  return {
    stdout: latency_message,
  };
}
```

```
          /\      |‾‾| /‾‾/   /‾‾/
     /\  /  \     |  |/  /   /  /
    /  \/    \    |     (   /   ‾‾\
   /          \   |  |\  \ |  (‾)  |
  / __________ \  |__| \__\ \_____/ .io

     execution: local
        script: api-tests/summary.js
        output: -

     scenarios: (100.00%) 1 scenario, 1 max VUs, 10m30s max duration (incl. graceful stop):
              * default: 1 iterations for each of 1 VUs (maxDuration: 10m0s, gracefulStop: 30s)

The median latency was 461.045209

running (00m00.5s), 0/1 VUs, 1 complete and 0 interrupted iterations
default ✓ [======================================] 1 VUs  00m00.5s/10m0s  1/1 iters, 1 per VU
```

HOW TO VISUALIZE K6 RESULTS

The argument takes the form of <key>=<value>, where the key is one of the output types, and the value is the file path or remote destination. You can pass multiple outputs in one script.

```
k6 run script.js \
--out json=test.json \
--out influxdb=http://localhost:8086/k6
```

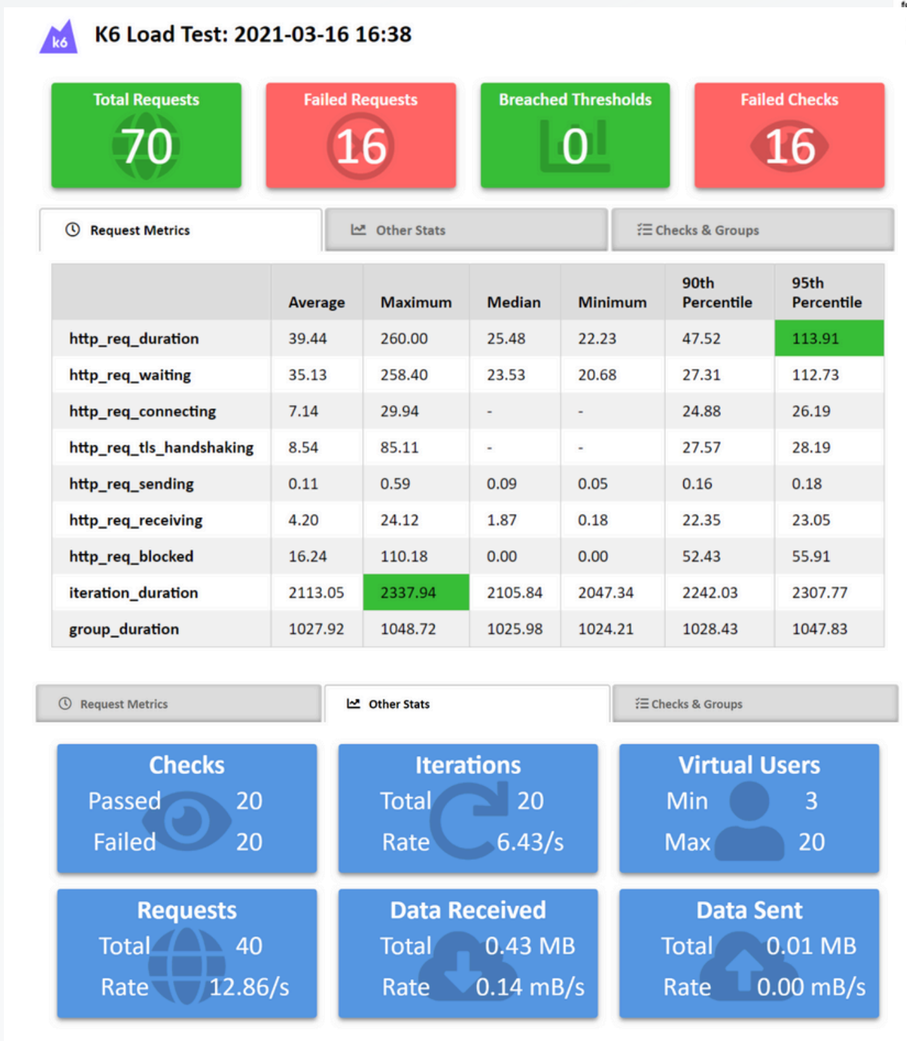



```
K6_WEB_DASHBOARD=true k6 run script.js
```

K6 HTML REPORT

```
import { chromium } from 'k6/experimental/browser';
import { check } from 'k6';
import { htmlReport } from "https://raw.githubusercontent.com/benc-uk/k6-reporter/2.4.0/dist/bundle.js";

export let options = {
    vus: 5,
    iterations: 10

}

export default async function () {
    const browser = chromium.launch({ headless: false });
    const context = browser.newContext();
    const page = context.newPage();

    page.goto('https://ecommerce-playground.lambdatest.io/index.php?route=account/login');
    page.screenshot({ path: 'screenshots/browserTestScreenshot.png' });

    page.locator('#input-email').type('lambdatest.cypress@disposable.com');
    page.locator('#input-password').type('Cypress123!!');
    const submitButton = page.locator('input[value="Login"]');
    await Promise.all([page.waitForNavigation(), submitButton.click()]);

    check(page, {
        'Verify user is logged In': () =>
            page.locator('.breadcrumb-item.active').textContent() == 'Account',
    });
    check(page, {
        'Verify the text': () =>
            page.locator('.breadcrumb-item.active').textContent() == 'Test',
    });


    page.close()
    browser.close();
}

export function handleSummary(data) {
    return {
        'TestSummaryReport.html': htmlReport(data, { debug: true })
    };
}
```

```
K6_BROWSER_ENABLED=true k6 run browserTest.js
```



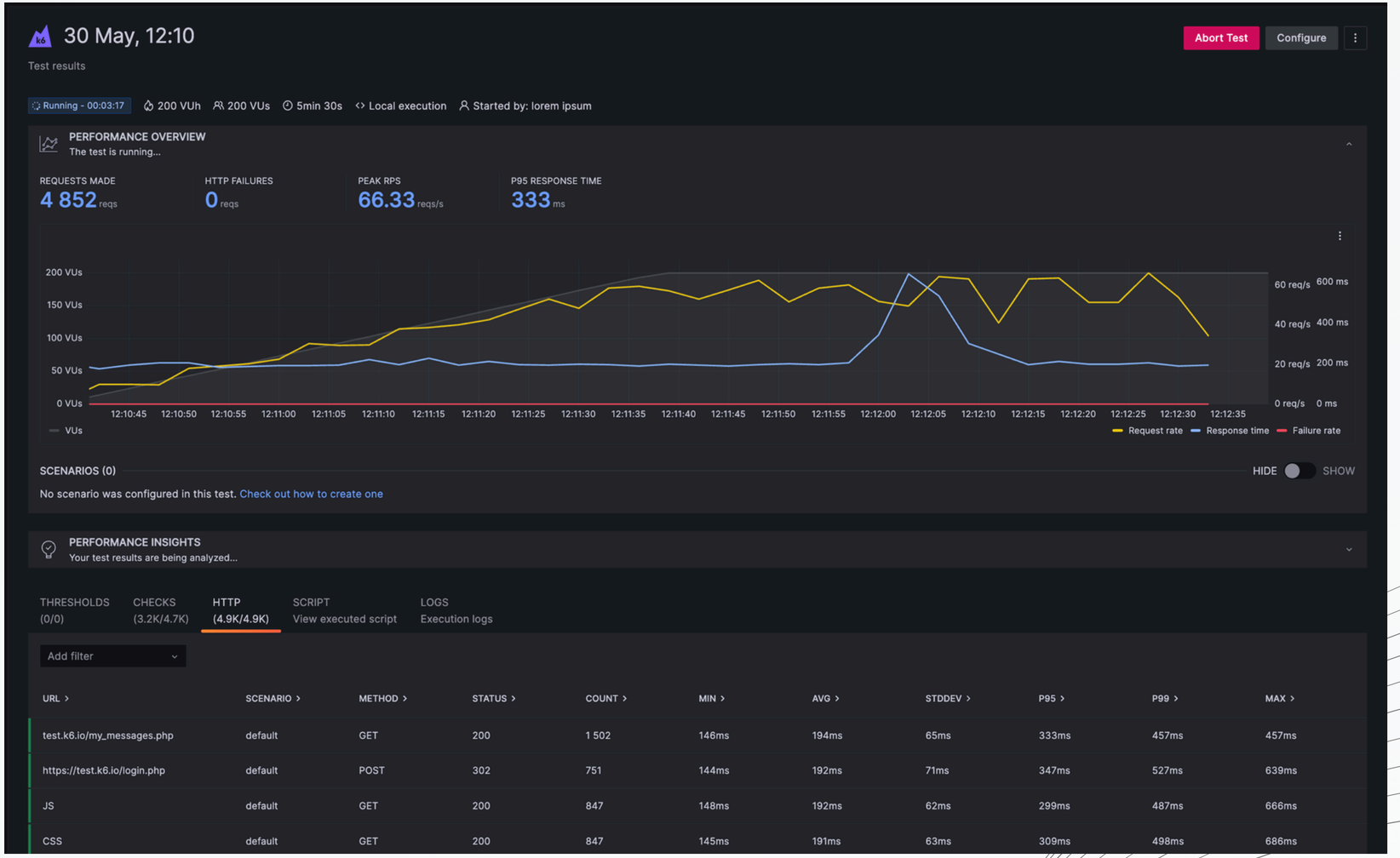
GRAFANA CLOUD K6

To execute the test **locally** and send the output to k6 Grafana Cloud, use the following command:

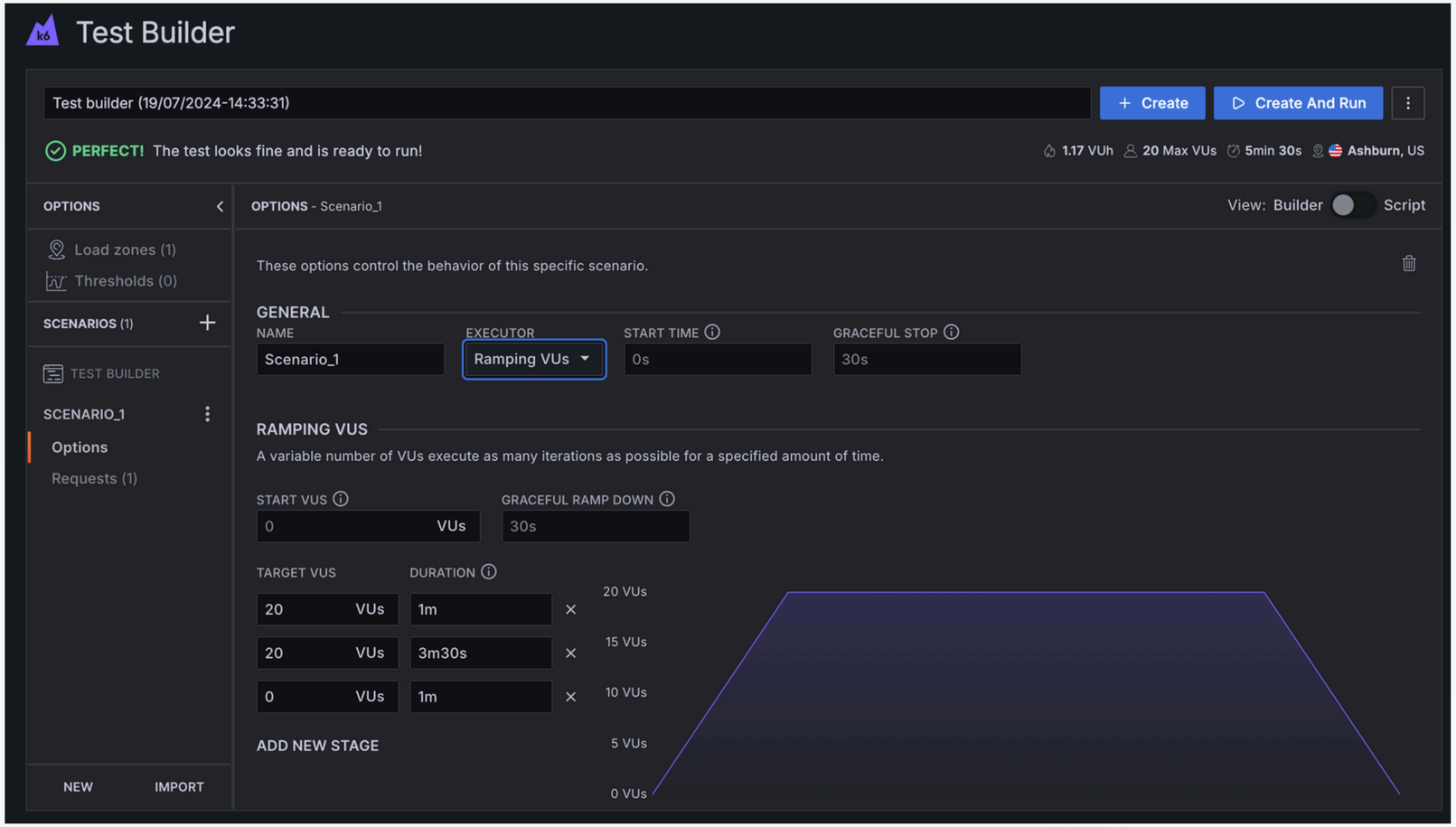
`k6 run <fileName>.js — out cloud`

To execute the script on **cloud**, use the following command:

`k6 cloud <fileName>.js`



CREATE A TEST WITH THE TEST BUILDER



https://olha22.grafana.net/a/k6-app/projects/3699789/tests/908501/edit#s=aR7i4LI1aYxwqPBm Gfmdm

PERFORMING K6 TESTING ON CI/CD: GITHUB ACTIONS

THANK'S FOR YOUR ATTENTION