

# Knowing the Limit of Your System

Allen



# Content

- What is Performance Testing?
- What is K6?
- A brief description of the various functions of the K6

# What is Performance Testing?

Detecting the limits of a system by applying pressure to the system and observing the system's operating conditions


Performance testing covers a wide range of areas, which can be subdivided into:

- Load Testing
- Spike Testing
- Stress Testing
- Soak Testing / Endurance Testing
- Scalability Testing

# What is K6?

## K6 is a load testing tool introduced and maintained by Grafana

You can write test scripts in JavaScript and do a load test to the system



[OPEN SOURCE](#)
[GRAFANA CLOUD K6](#)
[PRICING](#)
[DOCUMENTATION](#)
[ABOUT ▾](#)

[SIGN IN ▾](#)
[SIGN UP](#)

# The best developer experience for load testing

Open source and SaaS for engineering teams

[OPEN SOURCE >](#)
[GRAFANA CLOUD K6 >](#)



# Write a Load Test Script

First, let's create a `test.js` file and write the following contents

**vus**: Virtual users, which means how many users are simulated to access the system

**duration**: Duration of the test

The main logic of the test

```
import http from "k6/http";

export let options = {
  vus: 10,
  duration: "10s",
};

export default function () {
  http.get("https://docfunc.com");
}
```



# Run the Test Script

Run the `test.js` script you just wrote with K6



```
#!/bin/bash
```

```
k6 run test.js
```

# Test Results

```
execution: local
  script: 01-test.js
  output: -
```

```
scenarios: (100.00%) 1 scenario, 10 max VUs, 40s max duration (incl. graceful stop):
    * default: 10 looping VUs for 10s (gracefulStop: 30s)
```

```

data_received.....: 17 MB 1.7 MB/s
data_sent.....: 56 kB 5.4 kB/s
http_req_blocked.....: avg=62.34ms min=0s med=1µs max=1.2s p(90)=3µs p(95)=586.15ms
http_req_connecting.....: avg=3.19ms min=0s med=0s max=80.27ms p(90)=0s p(95)=22.82ms
http_req_duration.....: avg=470.43ms min=255.97ms med=423.93ms max=1.09s p(90)=663.75ms p(95)=752.57ms
  { expected_response:true }...: avg=470.43ms min=255.97ms med=423.93ms max=1.09s p(90)=663.75ms p(95)=752.57ms
http_req_failed.....: 0.00% ✓ 0 × 191
http_req_receiving.....: avg=23.43ms min=2.54ms med=6.68ms max=283.74ms p(90)=9.45ms p(95)=191.09ms
http_req_sending.....: avg=69.84µs min=29µs med=65µs max=315µs p(90)=93µs p(95)=116.49µs
http_req_tls_handshaking.....: avg=4.39ms min=0s med=0s max=102.6ms p(90)=0s p(95)=31.32ms
http_req_waiting.....: avg=446.92ms min=248.48ms med=409.72ms max=1.08s p(90)=533.08ms p(95)=699.51ms
http_reqs.....: 191 18.484516/s
iteration_duration.....: avg=533.17ms min=256.27ms med=424.24ms max=2.29s p(90)=664.06ms p(95)=1.34s
iterations.....: 191 18.484516/s
vus.....: 10 min=10 max=10
vus_max.....: 10 min=10 max=10

```

```
running (10.3s), 00/10 VUs, 191 complete and 0 interrupted iterations
default ✓ [=====] 10 VUs 10s
```

# console.log()

You can use `console.log()` to print out the values in the code



```
import http from "k6/http";

export let options = {
  vus: 10,
  duration: "10s",
};

export default function () {
  const res = http.get("https://docfunc.com");

  console.log(res.status);
}
```



# Checks

You can use `check()` to check if the HTTP response was successful. At the end of the test, an additional checks result will be added to the test result

```
import { check } from "k6";
import http from "k6/http";

export let options = {
  vus: 10,
  duration: "10s",
};

export default function () {
  const res = http.get("https://docfunc.com");

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

✓ status is 200

checks.....	100.00%	✓ 215	× 0
data_received.....	20 MB	1.9 MB/s	
data_sent.....	58 kB	5.6 kB/s	
http_req_blocked.....	avg=10.42ms	min=0s	
http_req_connecting.....	avg=2.4ms	min=0s	
http_req_duration.....	avg=464.62ms	min=270.12ms	

# Cookie

You can send an HTTP request with a cookie to simulate a logged in state

```
import { check } from "k6";
import http from "k6/http";

export default function () {
  const res = http.get("https://docfunc.com/posts/create", {
    cookies: {
      docfunc_session: "...",
    },
  });

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

# Stage

We can set the scenario in more detail, such as ramp-up and ramp-down. You can change the VUs at certain times.

```
import http from "k6/http";
import { check } from "k6";

export const options = {
  stages: [
    // Ramp-up from 1 to 20 VUs in 10s
    { duration: "10s", target: 20 },
    // Ramp-down from 20 to 10 VUs for 10s
    { duration: "10s", target: 10 },
    // Ramp-down from 10 to 0 VUs for 10s
    { duration: "10s", target: 0 },
  ],
};

export default function () {
  const res = http.get("https://docfunc.com/");

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

# References

- [什麼是 Performance test / Loading test](#)
- [Software performance testing](#)
- [軟體測試的種類 - 番外篇](#)
- [Performance Testing | Software Testing](#)
- [快速上手 Grafana k6 壓力測試工具](#)