

Performance Testing with **K6**

Agenda

- What is k6?
- Core Concepts
- Anatomy of a Test
- Writing or recording tests
- Result Visualisation
- Scaling out your test
- Questions

What is k6?

Unit-testing for performance

What is k6?

- Modern
- Flexible
- User Friendly

Core Concepts

Core Concepts

- What's a test?
- What are Virtual Users (VUs)?
- Types of tests
- Metrics
- Checks and Thresholds
- Test runtime and functions

What's a test?

What's a test?

A test can be as simple as this:

```
// script.js

import { sleep } from "k6";
import http from "k6/http";

export default function() {
    http.get("https://test.loadimpact.com/");
    sleep(3);
}
```

Virtual Users (VUs)

- An entity that executes a test, and makes requests.
- Simulate an actual user session.
- They run concurrently, and **keep repeating the test** until the test is over.
- When testing anything that is “User journey” related, webapps, websites, or API endpoints in a specific order you should think in terms of Virtual Users.

How many VUs?

$$VUs = \frac{numHourlySessions * avgSessionDurationInSecs}{3600}$$

- Use data from different times to:
 - simulate regular traffic
 - simulate busiest/peak hour
 - stress test your system

Types of tests

Types of tests

Baseline

```
export let options = {  
  stages: [  
    { duration: "1m", target: 10 },  
    { duration: "10m", target: 10 },  
    { duration: "1m", target: 0 }  
],  
};
```

Types of tests

Load

```
export let options = {  
  stages: [  
    { duration: "5m", target: 2000 },  
    { duration: "15m", target: 2000 },  
    { duration: "5m", target: 0 }  
],  
};
```

Types of tests

Spike

```
export let options = {  
  stages: [  
    { duration: "1m", target: 2000 },  
    { duration: "9m", target: 2000 },  
    { duration: "3m", target: 10000 },  
    { duration: "7m", target: 10000 },  
    { duration: "10m", target: 0 }  
],  
};
```

Metrics

Metrics

Standard Metrics

```
2. robin@robin-mbp: ~ (zsh)
$ ./k6 run ~/script.js
      ^   |---| /--/ /-
     / \  |  | / / / /
    /   \ |  |  | / --\ 
   / _____\ |  |  | \_\ \_\_/.io

execution: local
output: -
script: /Users/robin/script.js

duration: -, iterations: 1
vus: 1, max: 1

done [=====] 1 / 1

data_received.....: 4.5 kB 1.3 kB/s
data_sent.....: 527 B 148 B/s
http_req_blocked....: avg=432.78ms min=432.78ms med=432.78ms max=432.78ms p(90)=432.78ms p(95)=432.78ms
http_req_connecting....: avg=116.6ms min=116.6ms med=116.6ms max=116.6ms p(90)=116.6ms p(95)=116.6ms
http_req_duration....: avg=115.44ms min=115.44ms med=115.44ms max=115.44ms p(90)=115.44ms p(95)=115.44ms
http_req_receiving....: avg=59.38µs min=59.38µs med=59.38µs max=59.38µs p(90)=59.38µs p(95)=59.38µs
http_req_sending....: avg=205µs min=205µs med=205µs max=205µs p(90)=205µs p(95)=205µs
http_req_tls_handshaking...: avg=314.67ms min=314.67ms med=314.67ms max=314.67ms p(90)=314.67ms p(95)=314.67ms
http_req_waiting....: avg=115.17ms min=115.17ms med=115.17ms max=115.17ms p(90)=115.17ms p(95)=115.17ms
http_reqs.....: 1 0.281749/s
iteration_duration....: avg=3.54s min=3.54s med=3.54s max=3.54s p(90)=3.54s p(95)=3.54s
iterations.....: 1 0.281749/s
vus.....: 1 min=1 max=1
vus_max.....: 1 min=1 max=1
```

Metrics

Standard Metrics

- These are well documented on the [k6 docs website](#).

HTTP-specific built-in metrics

There are also *built-in* metrics that will only be generated when/if HTTP requests are made:

| Metric name | Type | Description |
|--------------------------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| http_reqs | Counter | How many HTTP requests has k6 generated, in total. |
| http_req_blocked | Trend | Time spent blocked (waiting for a free TCP connection slot) before initiating request. <code>float</code> |
| http_req_looking_up | Trend | Time spent looking up remote host name in DNS. <code>float</code> |
| http_req_connecting | Trend | Time spent establishing TCP connection to remote host. <code>float</code> |
| http_req_tls_handshaking | Trend | Time spent handshaking TLS session with remote host. |
| http_req_sending | Trend | Time spent sending data to remote host. <code>float</code> |
| http_req_waiting | Trend | Time spent waiting for response from remote host (a.k.a. "time to first byte", or "TTFB"). <code>float</code> |
| http_req_receiving | Trend | Time spent receiving response data from remote host. <code>float</code> |
| http_req_duration | Trend | Total time for the request. It's equal to <code>http_req_sending + http_req_waiting + http_req_receiving</code> (i.e. how long did the remote server take to process the request and respond, without the initial DNS lookup/connection times). <code>float</code> |

Metrics

Custom Metrics

```
import http from "k6/http";
import { Counter } from "k6/metrics";

let CounterErrors = new Counter("Errors");

export default function() {
    let res = http.get("https://test.loadimpact.com");
    let contentOK = res.html("h2").text().includes("Welcome to the LoadImpact.com demo site!");

    if (!contentOK) {
        CounterErrors.add(1);
    }
};
```

Metrics

Custom Metrics

```
chrisjames@Chris-James-C02WW09QJG5M: ~/repos/performance-test-k6

→ performance-test-k6 git:(master) ✘ docker-compose run k6 run /scripts/01-simple/test.js

.io

execution: local
output: influxdb=http://influxdb:8086/k6 (http://influxdb:8086)
script: /scripts/01-simple/test.js

duration: -, iterations: 1
vus: 1, max: 1

done [=====] 1 / 1

Errors.....: 0      0/s
data_received.....: 6.8 kB 6.7 kB/s
data_sent.....: 576 B 570 B/s
http_req_blocked.....: avg=759.95ms min=759.95ms med=759.95ms max=759.95ms p(90)=759.95ms p(95)=759.95ms
http_req_connecting.....: avg=228.13ms min=228.13ms med=228.13ms max=228.13ms p(90)=228.13ms p(95)=228.13ms
http_req_duration.....: avg=248.75ms min=248.75ms med=248.75ms max=248.75ms p(90)=248.75ms p(95)=248.75ms
http_req_receiving.....: avg=185.4μs min=185.4μs med=185.4μs max=185.4μs p(90)=185.4μs p(95)=185.4μs
http_req_sending.....: avg=280.1μs min=280.1μs med=280.1μs max=280.1μs p(90)=280.1μs p(95)=280.1μs
http_req_tls_handshaking.....: avg=514.9ms min=514.9ms med=514.9ms max=514.9ms p(90)=514.9ms p(95)=514.9ms
http_req_waiting.....: avg=248.28ms min=248.28ms med=248.28ms max=248.28ms p(90)=248.28ms p(95)=248.28ms
http_reqs...: 1      0.990931/s
iteration_duration.....: avg=1s      min=1s      med=1s      max=1s      p(90)=1s      p(95)=1s
iterations.....: 1      0.990931/s
vus.....: 1      min=1 max=1
vus_max.....: 1      min=1 max=1

→ performance-test-k6 git:(master) ✘
```

Metrics

Custom Metrics

- Counter metrics
- Gauge metrics
- Rate metrics
- Trend metrics

Metrics

Counter Metrics

A metric that cumulatively sums added values.

```
import { Counter } from "k6/metrics";  
  
var myCounter = new Counter("my_counter");  
  
export default function() {  
    myCounter.add(1);  
    myCounter.add(2);  
};
```

Metrics

Gauge metrics

A metric that stores the last value added to it.

```
import { Gauge } from "k6/metrics";  
  
var myGauge = new Gauge("my_gauge");  
  
export default function() {  
    myGauge.add(3);  
    myGauge.add(1);  
    myGauge.add(2);  
};
```

Metrics

Rate Metrics

A metric that tracks the percentage of added values that are non-zero.

```
import { Rate } from "k6/metrics";

var myRate = new Rate("my_rate");

export default function() {
    myRate.add(true);
    myRate.add(false);
    myRate.add(1);
    myRate.add(0);
};
```



```
$ k6 run rate.js
Welcome to k6 v0.14.0!
execution: local
output: -
script: /Users/ragnarlonn/src/go/src/github.com/loadimpact/k6/rate.js (js)

duration: 0s, iterations: 1
vus: 1, max: 1

web ui: http://127.0.0.1:6565/

done [=====] 0s / 0s

iterations...: 1
my_rate.....: 50.00%
vus.........: 1
vus_max.....: 1
$
```

Metrics

Trend metrics

A metric that allows for calculating statistics on the added values (min, max, average and percentiles).

```
import { Trend } from "k6/metrics";  
  
var myTrend = new Trend("my_trend");  
  
export default function() {  
    myTrend.add(1);  
    myTrend.add(2);  
};
```

Checks and Thresholds

Checks are like asserts but differ
in that they don't halt execution.

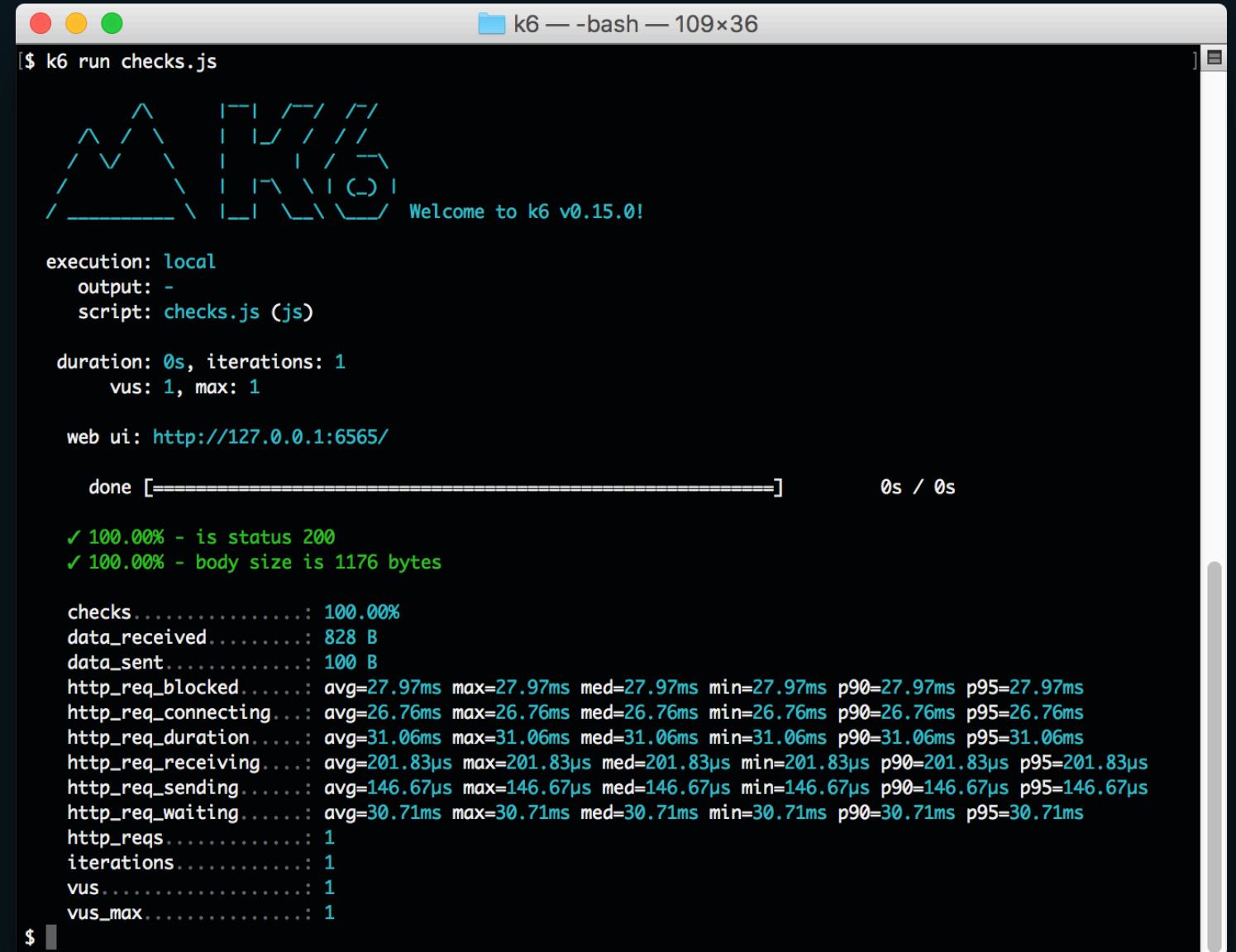
Thresholds are global pass/fail criteria that you can configure k6 to use.

Checks

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

export default function() {
  let res = http.get("http://test.loadimpact.com/");
  check(res, {
    "is status 200": (r) => r.status === 200,
    "body size is 1176 bytes": (r) => r.body.length == 1176
  });
}
```

- One important thing to understand regarding checks is that a failed check will not fail the whole load test.



The screenshot shows a terminal window titled "k6 — -bash — 109x36". The command "\$ k6 run checks.js" is run, followed by a welcome message with a stylized logo and "Welcome to k6 v0.15.0!". The configuration details are listed:

- execution: local
- output: -
- script: checks.js (js)
- duration: 0s, iterations: 1
- vus: 1, max: 1
- web ui: http://127.0.0.1:6565/

The test results show two successful checks:

- ✓ 100.00% - is status 200
- ✓ 100.00% - body size is 1176 bytes

Performance metrics are displayed at the bottom:

- checks.....: 100.00%
- data_received.: 828 B
- data_sent.....: 100 B
- http_req_blocked....: avg=27.97ms max=27.97ms med=27.97ms min=27.97ms p90=27.97ms p95=27.97ms
- http_req_connecting....: avg=26.76ms max=26.76ms med=26.76ms min=26.76ms p90=26.76ms p95=26.76ms
- http_req_duration....: avg=31.06ms max=31.06ms med=31.06ms min=31.06ms p90=31.06ms p95=31.06ms
- http_req_receiving....: avg=201.83µs max=201.83µs med=201.83µs min=201.83µs p90=201.83µs p95=201.83µs
- http_req_sending....: avg=146.67µs max=146.67µs med=146.67µs min=146.67µs p90=146.67µs p95=146.67µs
- http_req_waiting....: avg=30.71ms max=30.71ms med=30.71ms min=30.71ms p90=30.71ms p95=30.71ms
- http_reqs.....: 1
- iterations.....: 1
- vus.....: 1
- vus_max.....: 1

Thresholds

If you want to fail a load test,
use thresholds.

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

export let errorRate = new Rate("errors");

export let options = {
  thresholds: {
    "errors": ["rate<0.1"], // <10% errors
  }
};

export default function() {
  check(http.get("http://fakesite.com"), {
    "status is 200": (r) => r.status == 200
  }) || errorRate.add(1);
}
```

```
$ k6 run check_threshold.js ; echo $?
Welcome to k6 v0.15.0!

execution: local
output: -
script: check_threshold.js (js)

duration: 0s, iterations: 1
vus: 1, max: 1

web ui: http://127.0.0.1:6565/

done [=====] 300ms / 300ms

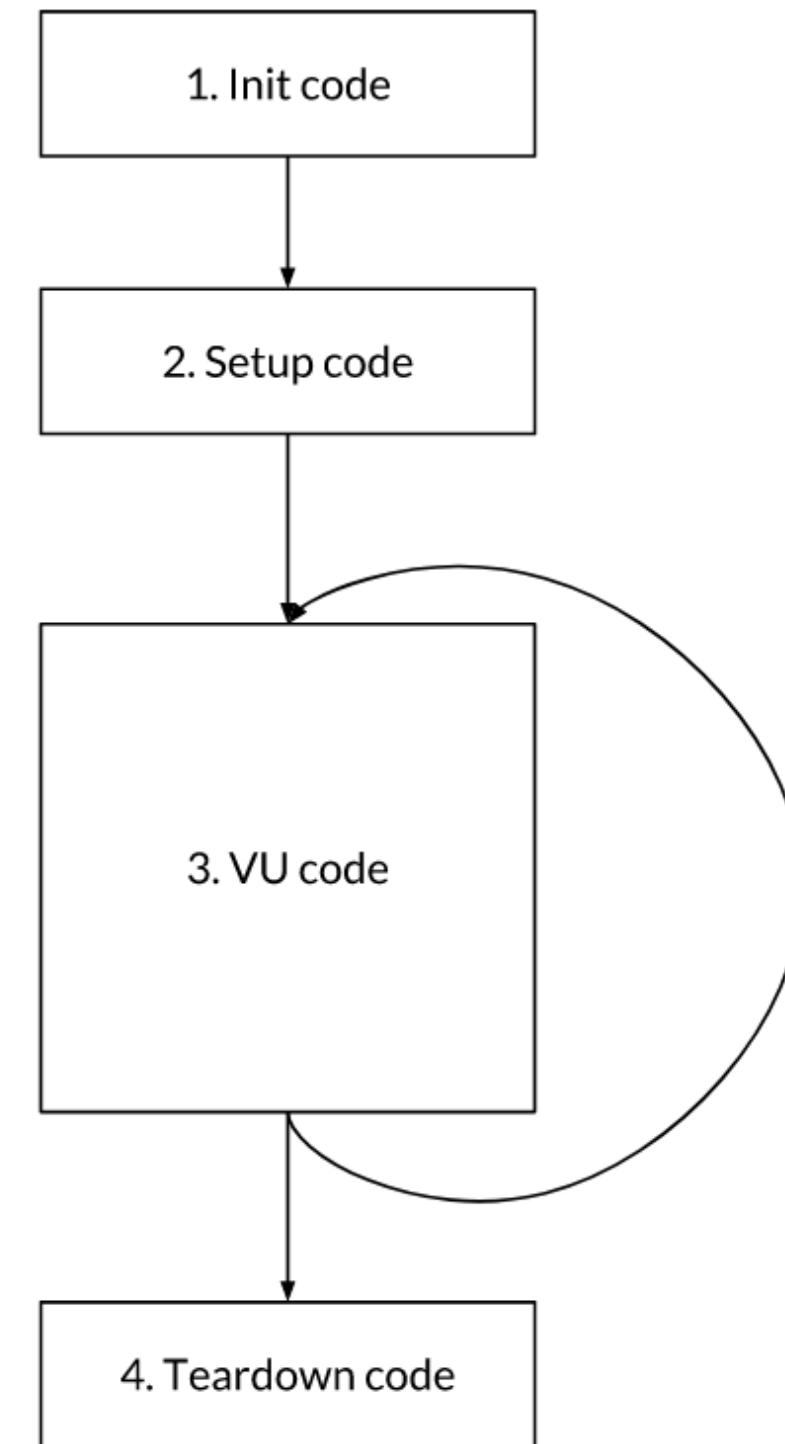
x 0.00% - status is 200

checks.....: 0.00%
data_received.....: 13 kB
data_sent.....: 92 B
x errors.....: 100.00%
http_req_blocked.....: avg=145.73ms max=145.73ms med=145.73ms min=145.73ms p90=145.73ms p95=145.73ms
http_req_connecting.....: avg=143.8ms max=143.8ms med=143.8ms min=143.8ms p90=143.8ms p95=143.8ms
http_req_duration.....: avg=163.54ms max=163.54ms med=163.54ms min=163.54ms p90=163.54ms p95=163.54ms
http_req_receiving.....: avg=10.09ms max=10.09ms med=10.09ms min=10.09ms p90=10.09ms p95=10.09ms
http_req_sending.....: avg=143.66μs max=143.66μs med=143.66μs min=143.66μs p90=143.66μs p95=143.66μs
http_req_waiting.....: avg=153.3ms max=153.3ms med=153.3ms min=153.3ms p90=153.3ms p95=153.3ms
http_reqs.....: 1
iterations.....: 1
vus.....: 1
vus_max.....: 1
```

Anatomy of a Test

Anatomy of a Test

```
// 1. init code, imports and setup counters etc.  
  
// OPTIONAL  
export const options = {  
    // ...  
}  
  
// OPTIONAL  
export function setup() {  
    // 2. setup code  
}  
  
export default function(data) {  
    // 3. vu code  
}  
  
// OPTIONAL  
export function teardown(data) {  
    // 4. teardown code  
}
```



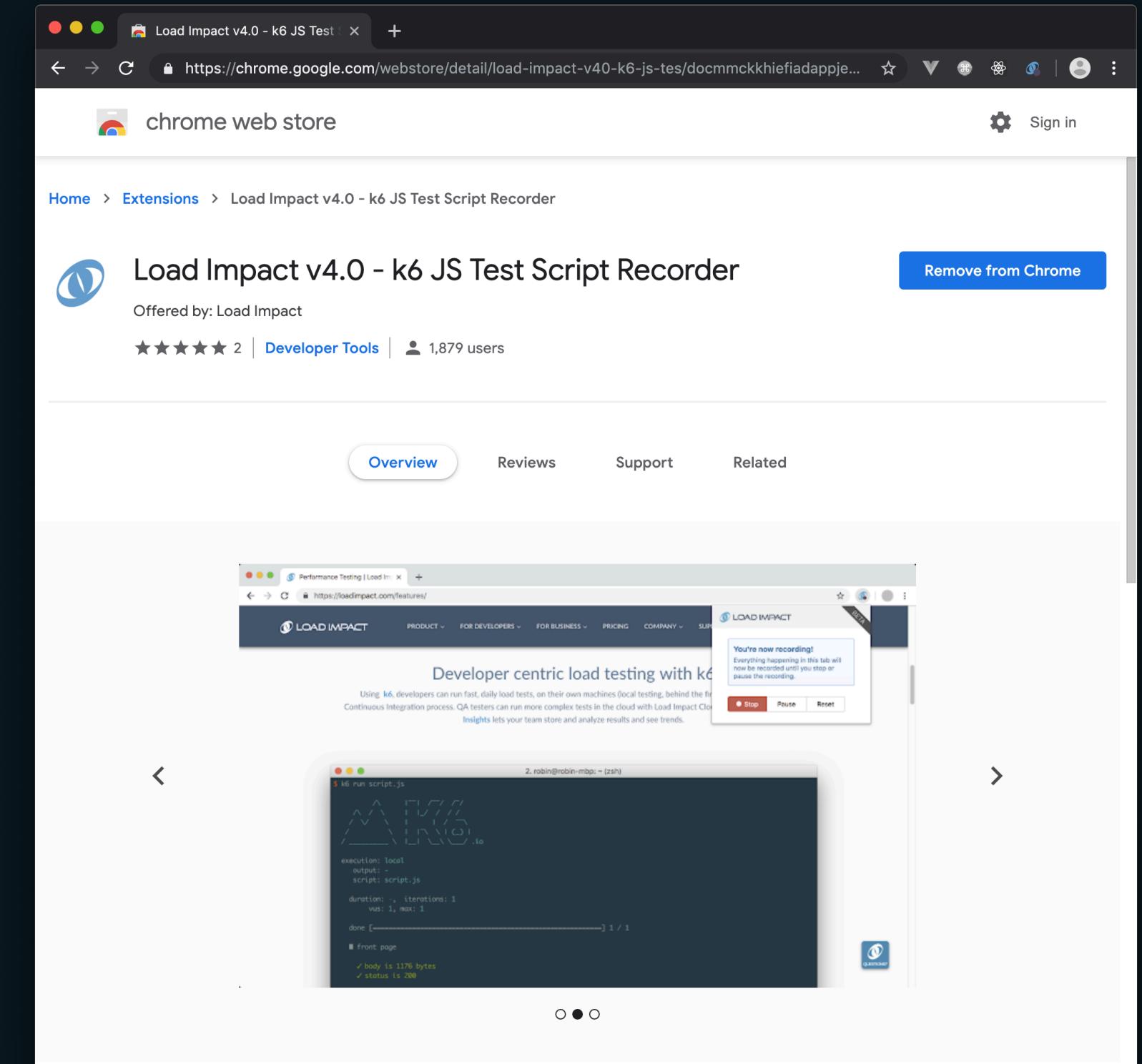
VU code loops over and over for the duration of the test

Writing and Running Tests

Demo

Recording Tests

- Writing these tests can get tedious, especially for websites.
- Use the test recorder extension to simulate load and generate a baseline test.



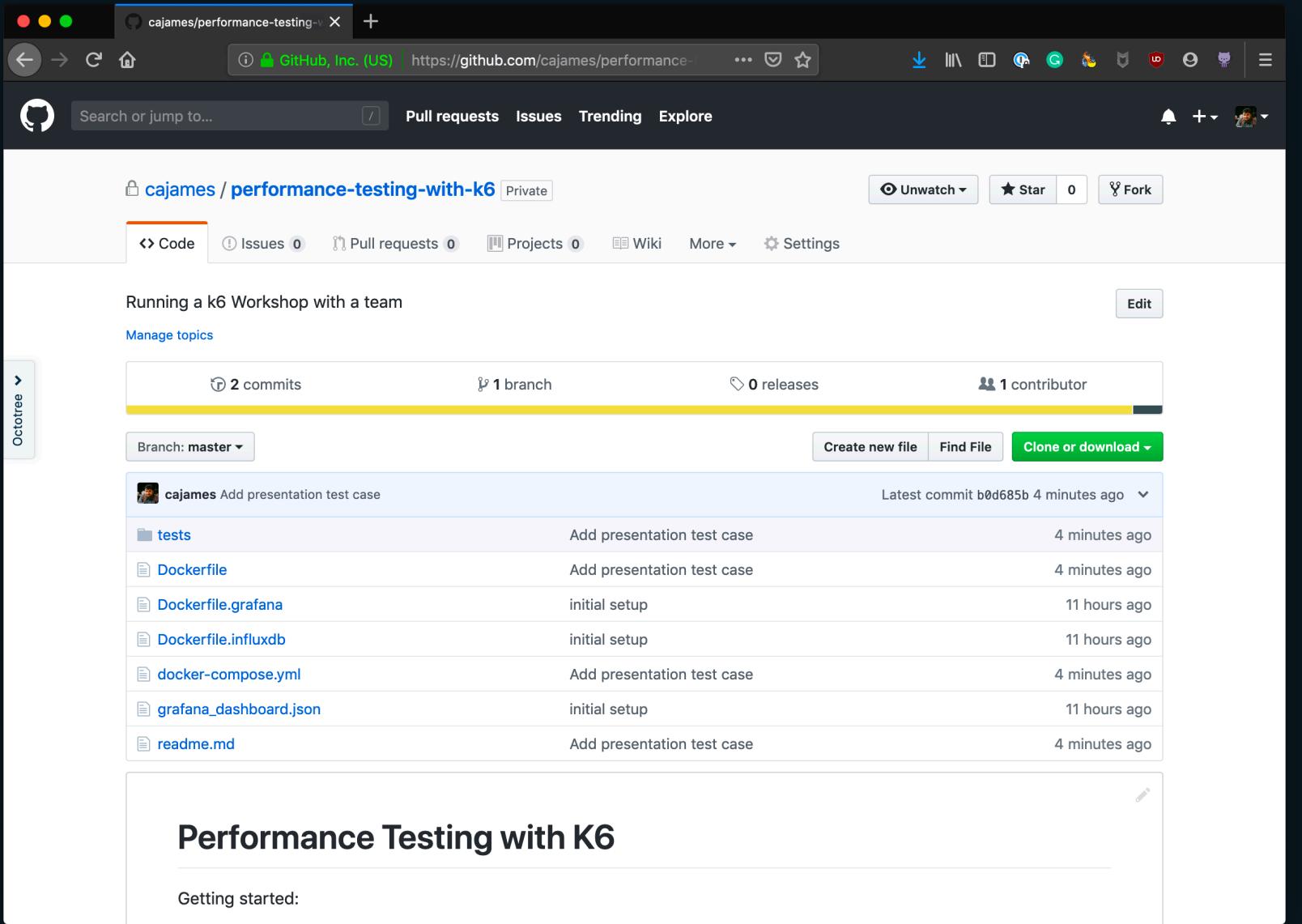
Demo

Result Visualisation

Demo

Scaling out your test

Demo



github.com/cajames/performance-testing-with-k6

Questions?

Thanks

