K6

Rex Hsieh

Goal

- Know what is K6.
- Know how to use K6.
- Know about test types.

What is K6

- k6 is a developer-centric, free and open-source load testing tool built for making performance testing a productive and enjoyable experience.
- An implementation of ES2015(ES6) JavaScript on pure Golang language.
- High-performance tool — The K6 engine is written in Go and it is one of the most efficient load testing tools.

Setup K6 + InfluxDB + Grafana

- Mac

  git clone https://github.com/rexhsieh888/K6.git
  cd K6
  docker-compose up -d \
    influxdb \
    grafana

- Windows

  git clone https://github.com/rexhsieh888/K6.git
  cd K6
  docker-compose up -d influxdb grafana

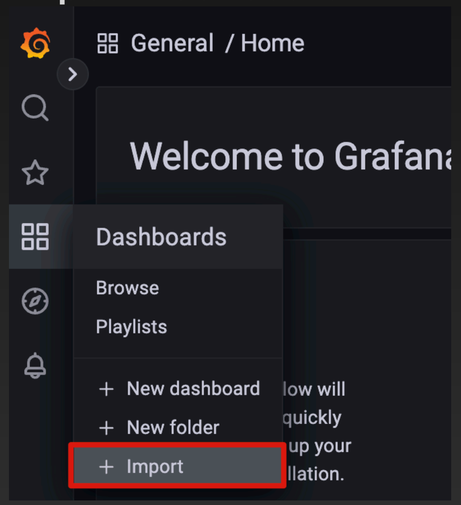
Let’s do a test

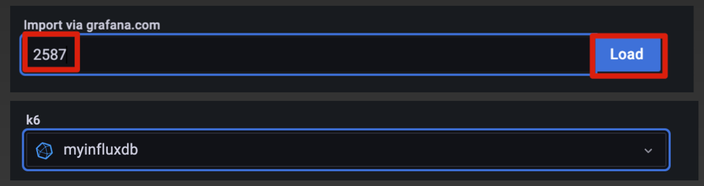
- Mac
  docker-compose run -v $PWD/samples:/scripts k6 run /scripts/class/ex_1.js
- Windows
  docker-compose run -v /$PWD/samples:/scripts k6 run /scripts/class/ex_1.js

Grafana

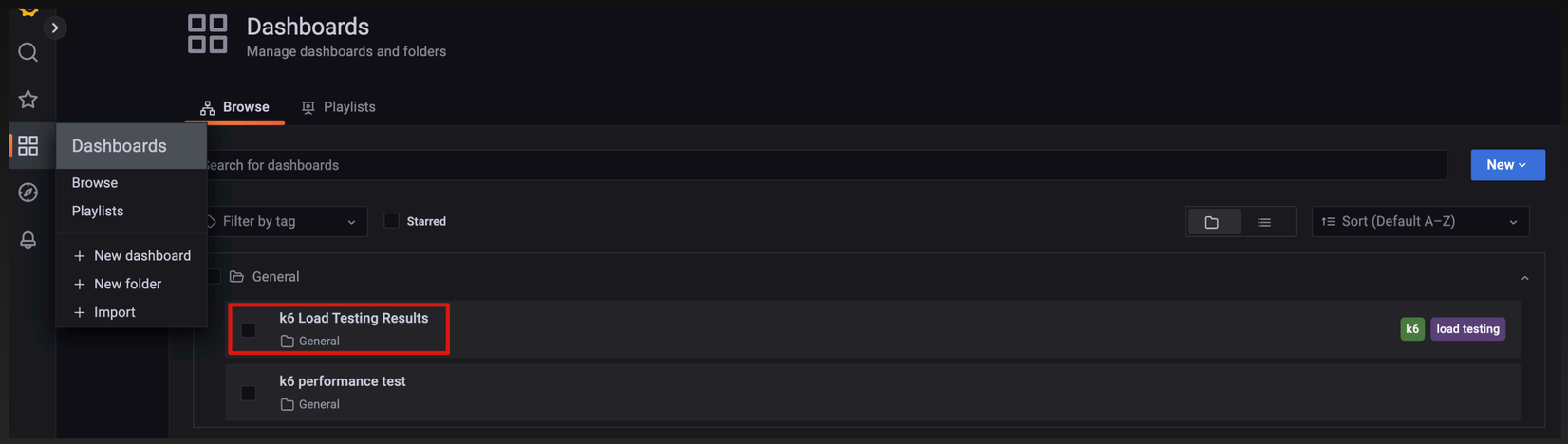
- Go to Grafana UI
  http://localhost:3000

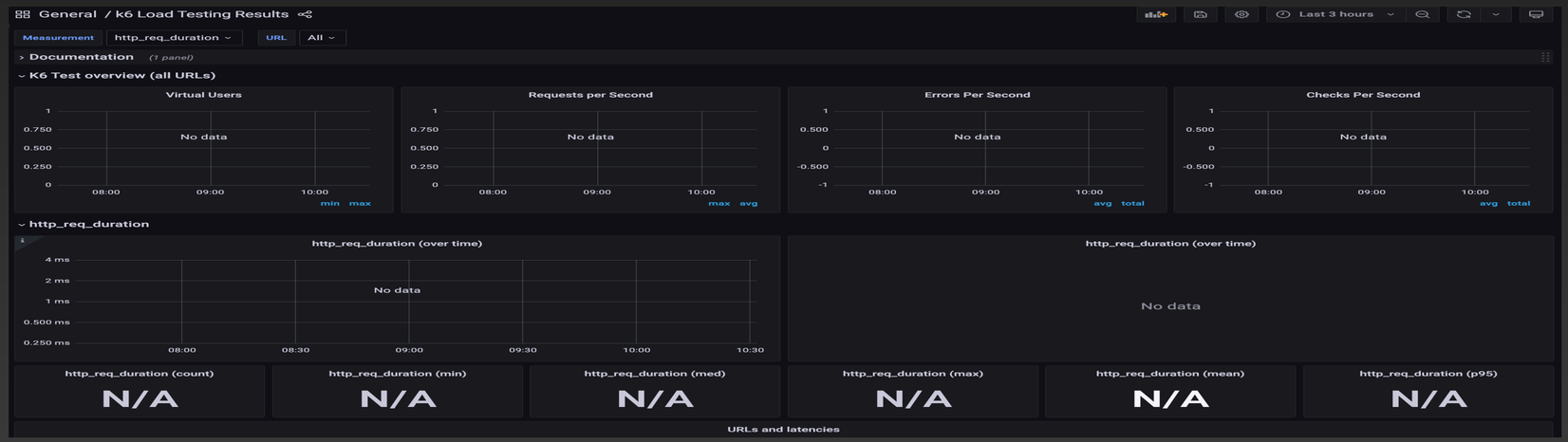
- Import Dashboard





Grafana


Dashboards
Manage dashboards and folders
Browse
Playlists
Dashboards
Browse
Playlists
New dashboard
New folder
Import
Search for dashboards
New
Filter by tag
Starred
Sort (Default A–Z)
General
k6 Load Testing Results
General
k6
load testing
k6 performance test
General


General / k6 Load Testing Results
Last 3 hours
Measurement
http_req_duration
URL
All
Documentation
(1 panel)
K6 Test overview (all URLs)
Virtual Users
Requests per Second
Errors Per Second
Checks Per Second
No data
http_req_duration
http_req_duration (over time)
http_req_duration (count)
http_req_duration (min)
http_req_duration (med)
http_req_duration (max)
http_req_duration (mean)
http_req_duration (p95)
N/A
URLs and latencies

Options

- Mac
  docker-compose run -v $PWD/samples:/scripts k6 run /scripts/class/detail/options/op_1.js

- Windows
  docker-compose run -v /$PWD/samples:/scripts k6 run /scripts/class/detail/options/op_1.js

Metrics

- Mac
  docker-compose run -v $PWD/samples:/scripts k6 run /scripts/class/detail/metrics/me_1.js

- Windows
  docker-compose run -v /$PWD/samples:/scripts k6 run /scripts/class/detail/metrics/me_1.js

Modules

- Mac
  docker-compose run -v $PWD/samples:/scripts k6 run /scripts/class/detail/modules/mo_1.js

- Windows
  docker-compose run -v /$PWD/samples:/scripts k6 run /scripts/class/detail/modules/mo_1.js

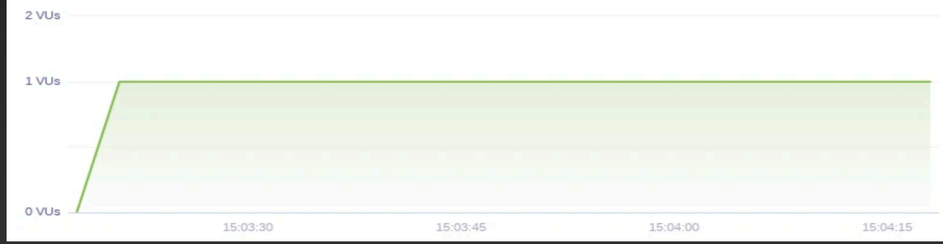
Smoke testing

- Verify that your test script doesn't have errors.
- Verify that your system doesn't throw any errors when under minimal load.

```
export const options = {
  vus: 1, // 1 user looping for 1 minute
  duration: '1m',

  thresholds: {
    http_req_duration: ['p(99)<1500'], // 99% of requests must complete below 1.5s
  },
};
```

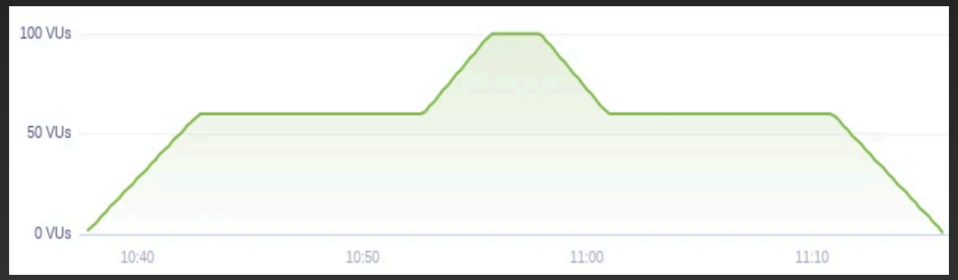
The VU chart of a smoke test should look similar to this. You want to use only 1 or 2 VUs.



Load Testing

- Note that the number of users starts at 0, and slowly ramps up to the nominal value, where it stays for an extended period of time. The ramp down stage is optional.
- Make sure you don't go over your normal number of VUs - that's not load testing, but stress testing.

```
export const options = {
  stages: [
    { duration: '5m', target: 60 }, // simulate ramp-up of traffic from 1 to 60 users over 5 m
    { duration: '10m', target: 60 }, // stay at 60 users for 10 minutes
    { duration: '3m', target: 100 }, // ramp-up to 100 users over 3 minutes (peak hour starts)
    { duration: '2m', target: 100 }, // stay at 100 users for short amount of time (peak hour)
    { duration: '3m', target: 60 }, // ramp-down to 60 users over 3 minutes (peak hour ends)
    { duration: '10m', target: 60 }, // continue at 60 for additional 10 minutes
    { duration: '5m', target: 0 }, // ramp-down to 0 users
  ],
  thresholds: {
    http_req_duration: ['p(99)<1500'], // 99% of requests must complete below 1.5s
  },
};
```



Stress Testing

- How your system will behave under extreme conditions.
- What the maximum capacity of your system is in terms of users or throughput.
- The breaking point of your system and its failure mode.
- If your system will recover without manual intervention after the stress test is over.
- Note that a stress test doesn't overwhelm the system immediately —
  that's a spike test, which we'll cover soon.

```
export const options = {
  stages: [
    { duration: '2m', target: 100 }, // below normal load
    { duration: '5m', target: 100 },
    { duration: '2m', target: 200 }, // normal load
    { duration: '5m', target: 200 },
    { duration: '2m', target: 300 }, // around the breaking point
    { duration: '5m', target: 300 },
    { duration: '2m', target: 400 }, // beyond the breaking point
    { duration: '5m', target: 400 },
    { duration: '10m', target: 0 }, // scale down. Recovery stage.
  ],
};
```

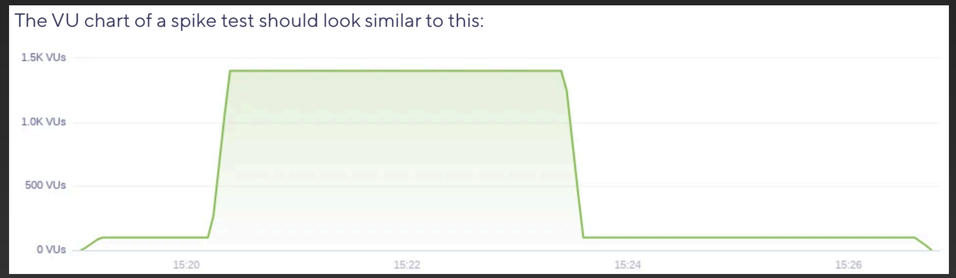
The VU chart of a stress test should look similar to this:

Spike Testing

- How your system will perform under a sudden surge of traffic.
- If your system will recover once the traffic has subsided.
- A classic need for a spike testing is if you've bought advertising on a big television event, such as the Super Bowl or a popular singing competition.
- Note, the test starts with a period of 1 minute of low load, a quick spike to very high load, followed by a recovery period of low load.

```
export const options = {
  stages: [
    { duration: '10s', target: 100 }, // below normal load
    { duration: '1m', target: 100 },
    { duration: '10s', target: 1400 }, // spike to 1400 users
    { duration: '3m', target: 1400 }, // stay at 1400 for 3 minutes
    { duration: '10s', target: 100 }, // scale down. Recovery stage.
    { duration: '3m', target: 100 },
    { duration: '10s', target: 0 },
  ],
};
```

The VU chart of a spike test should look similar to this:



Soak Testing

- Verify that your system doesn't suffer from bugs or memory leaks, which result in a crash or restart after several hours of operation.
- Verify that expected application restarts don't lose requests.
- Find bugs related to race-conditions that appear sporadically.
- Make sure your database doesn't exhaust the allotted storage space and stops.
- Make sure your logs don't exhaust the allotted disk storage.
- Make sure the external services you depend on don't stop working after a certain amount of requests are executed.
- We recommend you to configure your soak test at about 80% capacity of your system. If your system can handle a maximum of 500 simultaneous users, you should configure your soak test to 400 VUs.

```
export const options = {
  stages: [
    { duration: '2m', target: 400 }, // ramp up to 400 users
    { duration: '3h56m', target: 400 }, // stay at 400 for ~4 hours
    { duration: '2m', target: 0 }, // scale down. (optional)
  ],
};
```

The VU chart of a Soak Test should look similar to this:

Test api

https://test-api.k6.io/
https://test.k6.io/
http://ecommerce.test.k6.io/

Thank You!!