Lab 9 - Web deployment

Advanced Node.js process manager

pm2 เป็นโปรแกรมที่ใช้ในการสั่งให้ javascript (Node.js) ทำงานในลักษณะ service (daemon) ได้ หากโปรแกรมเกิดข้อผิดพลาด ก็จะให้เริ่มการทำงานใหม่ (restart) อัตโนมัติ รวมถึงสามารถให้ service นี้ ทำงานในลักษณะ cluster เพื่อรองรับการทำงานได้ดีขึ้น

installation

$ npm i -g pm2

รายละเอียดการทำงาน อ้างอิงจาก https://pm2.io/doc/en/runtime/quick-start

index.js

```
var express = require('express');
var app = express();
app.get('/', (req, res) => res.send('Hello world') );
app.listen(80);
```

pm2 script

สามารถกำหนด script ไว้ run pm2 และ กำหนด production environment ได้ดังนี้

process.yml

```
apps:
   - script: srv1.js
     name: mms
     env:
         NODE_ENV: production
```

$ pm2 start process.yml

pm2 cluster mode

$ pm2 start index -i 3

pm2 delete service

$ pm2 stop index

$ pm2 delete index

ให้ทดลองคำสั่งดังต่อไปนี้ พร้อมทั้งสังเกตผลลัพธ์

$ pm2 restart index

$ pm2 info index

$ pm2 list

$ pm2 monit

Nginx as reverse proxy

Nginx หนึ่งใน web server ที่กำลังได้รับความนิยมเพิ่มขึ้นเรื่อย ๆ เนื่องจากความเร็วในการทำงาน โดยเฉพาะในส่วนของ static files นอกจากนี้ ยังสามารถทำหน้าที่อื่น ๆ อย่างเช่น reverse proxy หรือ load balancer ได้อีกด้วย

Forward Proxy
สำหรับพวก Gamer, Hacker ที่ต้องการปกปิด หรือ เปลี่ยนแปลงที่อยู่ของตนเอง เช่นบางบริการอาจจะจำกัดสิทธิ์ในการเข้าถึงจากกลุ่ม IP ที่กำหนด (อาจจะตามพื้นที่) ทำให้ User ต้องเข้าถึงผ่าน Forward Proxy เพื่อไม่ให้ Web server รู้ที่อยู่ที่แท้จริง หรือเพื่อให้สามารถใช้บริการนั้นได้

```
          Client                          |     Server
User  <->  (Forward) Proxy <-> Internet <-|-> Web Server
```

Reverse Proxy
ใช้สำหรับ web server เพื่อให้บริการ High availability และ load balance ทำให้ User ไม่ทราบว่า backend ข้างหลังที่ให้บริการอยู่คือ service อะไร จากเครื่องไหน เลยเป็นที่มาของคำว่า Reverse

```
          Client          |              Server
User  <->  Internet <-  |  ->  Reverse Proxy <-> Web Server
```

Nginx Setup

Prerequisite: nodejs, npm, pm2, docker, docker-compose

สั่งให้ backend server ทำงานที่ path /api ที่ port: 4000 ดังนี้

index.js

```
var express = require('express');
var app = express();
app.get('/api', (req, res) => res.send('Hello world') );
app.listen(4000);
```

$ pm2 start index

ทดสอบการทำงาน โดยเรียกที่ http://localhost:4000/api จาก web browser ซึ่งต้องแสดงข้อความ Hello world

Nginx resource files

docker-compose.yml

```
version: '3'
services:
```

```
  nginx:
    image: nginx:latest
    container_name: nginx
    volumes:
      - ./default.conf:/etc/nginx/conf.d/default.conf:ro
    ports:
      - 80:80
      - 443:443
```

ข้อระวัง *ไฟล์ .yml ให้ความสำคัญกับ ช่องว่างของย่อหน้า ต้องเว้นให้เท่ากันตามตัวอย่างเสมอ*

จากนั้นเตรียมไฟล์ configuration ของ nginx ดังนี้

default.conf

```
server {
  listen       80;
  server_name  192.168.1.107;
  expires off;

  location / {
    root   /usr/share/nginx/html;
  index  index.html index.htm;
}

  location /api {
    proxy_pass http://192.168.1.107:4000/api;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
  }
}
```

และเปลี่ยน IP 192.168.1.107 เป็น IP ของเครื่อง Host machine ที่ทำงาน ตรวจสอบจาก
$ ifconfig | grep inet | grep 172

ให้ไฟล์ docker-compose.yml และ default.conf อยู่ใน folder เดียวกัน (ที่ path ไหนก็ได้)

Nginx run

สั่งให้ nginx ทำงาน โดยการสั่งครั้งแรก docker จะไป download image nginx มาก่อน แล้วก็ start service ตามที่กำหนดไว้ใน configuration file ด้วยคำสั่ง

$ docker-compose up -d

```
coc:nginx wwarodom$ docker-compose up -d
Creating network "nginx_default" with the default driver
Creating nginx ... done
coc:nginx wwarodom$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                                            NAMES
19283c94811c        nginx:latest        "nginx -g 'daemon of…"   6 seconds ago       Up 5 seconds        0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp   nginx
```

การเรียกใช้คำสั่งนี้ จะต้องอยู่ในโฟลเดอร์ที่มีไฟล์ docker-compose.yml

ทดสอบการทำงาน โดยเรียกที่ http://localhost/api จาก web browser ซึ่งต้องแสดงข้อความ Hello world โดยไม่ต้องระบุ port 4000 เพราะ nginx ที่เป็น Reverse proxy จะทำหน้าที่จับ request ที่ส่งเข้ามาผ่าน ผ่าน port 80 และ ส่งต่อไปยัง node service ที่ port 4000 และ ส่งค่าผลลัพธ์กลับมาให้ผู้ใช้ผ่าน web browser หากเห็นข้อความ Hello world แสดงว่า Reverse สามารถทำงานได้ถูกต้อง

คำสั่งอื่น ๆ ที่ใช้ตรวจสอบ การทำงาน

$ docker ps	จะต้องเห็น service ของ nginx ทำงานอยู่
$ docker-compose exec nginx sh	เข้าไปใน container (nginx container จะต้องทำงานได้ก่อน)
$ docker-compose logs -f nginx	ดู logs ของ nginx container
$ docker-compose down	หยุดการทำงานของ container
$ docker restart nginx	เริ่มการทำงานใหม่ของ nginx container

Nginx as HTTP Load Balancer: http://nginx.org/en/docs/http/load_balancing.html

อ้างอิง: Reverse proxy:
https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-18-04

9.1 จงใช้โปรแกรม pm2 ในการ start service (express) มา 1 service ที่ port 3000 จากนั้นให้ setup reverse proxy เพื่อให้ ผู้ใช้เข้าถึง service ตัวนี้ ผ่านทาง http://localhost/

K6 Load testing tool

Introduction

เป็นการทดสอบการทำงานของ web server และ services ที่ให้บริการว่ารองรับการรองข้อจาก client พร้อม ๆ กันได้มากเท่าไร

Installation

Download docker images โดยใช้คำสั่ง $ docker pull loadimpact/k6

สร้าง script.js ดังนี้

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

export default () => {
  let res=http.get('https://httpbin.org')
  check(res, { "is status 200": (r) => r.status === 200 })
  sleep(3)
}
```

Execute

จากนั้นสั่งให้ทำงานผ่าน docker โดยใช้คำสั่ง

$ docker run -i loadimpact/k6 run --vus 10 --duration 30s -< script.js

- vus คือ virtual users คือ ผู้ใช้เสมือน จำลองว่า มีผู้ใช้เข้ามาใช้งานระบบ
- duration คือ ช่วงเวลาที่ให้ทำ load test

Result

```
coc:k6 wwarodom$ docker run -i loadimpact/k6 run --vus 10 --duration 30s -< script.js

          /\      |‾‾|  /‾‾/  /‾/
     /\  /  \     |  |_/  /  / /
    /  \/    \    |      |  /  ‾‾\
   /          \   |  |‾\  \ | (_) |
  / __________ \  |__|  \__\ \___/ .io

  execution: local-----------------------------------------------------]   servertor
     output: -
     script: -

    duration: 30s, iterations: -
         vus: 10,  max: 10

time="2019-03-29T02:32:47Z" level=info msg=Running i=0 t=957.6248ms-] starting
time="2019-03-29T02:32:48Z" level=info msg=Running i=0 t=1.957466s
time="2019-03-29T02:32:49Z" level=info msg=Running i=0 t=2.957519s
time="2019-03-29T02:32:50Z" level=info msg=Running i=0 t=3.8475579s
time="2019-03-29T02:32:51Z" level=info msg=Running i=0 t=4.9574446s
```

...

```
time="2019-03-29T02:33:16Z" level=info msg=Running i=70 t=29.9575376s

    ✓ is status 200

time="2019-03-29T02:33:16Z" level=info msg="Test finished" i=70 t=30.0003175s
    checks.....................: 100.00% ✓ 80   × 0
    data_received..............: 743 kB  25 kB/s
    data_sent..................: 11 kB   366 B/s
    http_req_blocked...........: avg=260.34ms min=23.9µs   med=41.2µs   max=2.08s    p(90)=2.08s    p(95)=2.08s
    http_req_connecting........: avg=102.43ms min=0s       med=0s       max=820.8ms  p(90)=819.06ms p(95)=819.44ms
    http_req_duration..........: avg=579.51ms min=352.9ms  med=574.07ms max=794.27ms p(90)=782.86ms p(95)=785.22ms
    http_req_receiving.........: avg=96.77µs  min=27.4µs   med=61.3µs   max=962.6µs  p(90)=160.09µs p(95)=227.48µs
    http_req_sending...........: avg=78.19µs  min=24.4µs   med=47.6µs   max=324.3µs  p(90)=182.84µs p(95)=295.22µs
    http_req_tls_handshaking...: avg=128.28ms min=0s       med=0s       max=1.02s    p(90)=1.02s    p(95)=1.02s
    http_req_waiting...........: avg=579.34ms min=352.81ms med=573.87ms max=794.16ms p(90)=782.5ms  p(95)=785.14ms
    http_reqs..................: 80      2.666638/s
    iteration_duration.........: avg=3.89s    min=3.35s    med=3.66s    max=5.65s    p(90)=5.55s    p(95)=5.6s
    iterations.................: 70      2.333309/s
    vus........................: 10      min=10 max=10
    vus_max....................: 10      min=10 max=10
```

Metric name	Type	Description
vus	Gauge	Current number of active virtual users
vus_max	Gauge	Max possible number of virtual users (VU resources are preallocated, to ensure performance will not be affected when scaling up the load level)
iterations	Counter	The aggregate number of times the VUs in the test have executed the JS script (the `default` function). Or, in case the test is not using a JS script but accessing a single URL, the number of times the VUs have requested that URL.
data_received	Counter	The amount of received data.
data_sent	Counter	The amount of data sent.
checks	Rate	Number of failed checks.

ลองเปลี่ยนไปทดสอบกับ Host ของตัวเอง

$ docker run -i loadimpact/k6 run --vus 100 --duration 20s - < script.js

ทดลองทำ load test ของ server นี้

index.js (server script ไว้สำหรับเป็นหมายที่ถูกทดสอบ)

```
var express = require('express');
var app = express();

app.get('/', function(req, res){
   setTimeout(()=> {
       process.stdout.write('.')
       res.send('Hello world')
   }, 1000);

});

app.listen(80);
```

ลองเปลี่ยนค่าเวลาที่รอใน setTimeout(____ , เวลาที่รอ) และให้สังเกตค่าของจำนวนค่า http_reqs ที่ได้

9.2 จงใช้ K6 load test ในการทดสอบ service ที่ได้กำหนดให้ทำงานในข้อ 9.1

K6 with InfluxDB and Grafana

Installation

ติดตั้ง influxdb และ grafana ด้วย docker compose

```
git clone 'https://github.com/loadimpact/k6'
cd k6
git submodule update --init
docker-compose up -d influxdb grafana
```

docker-compose.yml

```
version: '3'
services:
  influxdb:
    build:
      context: .
dockerfile: Dockerfile.influxdb
ports:
  - "8086:8086"
grafana:
  build:
    context: .
dockerfile: Dockerfile.grafana
links:
  - influxdb
environment:
  - GF_AUTH_ANONYMOUS_ORG_ROLE=Admin
  - GF_AUTH_ANONYMOUS_ENABLED=true
  - GF_AUTH_BASIC_ENABLED=false
ports:
  - "3000:3000"
k6:
  build: .
ports:
  - "6565:6565"
environment:
  - K6_OUT=influxdb=http://influxdb:8086/k6
command: 'version'
```

Run K6 with template script

$ docker run -i loadimpact/k6 run - < samples/es6sample.js

Run K6 and Output to InfluxDB

ถ้าใช้ docker

$docker run --net="host" -i loadimpact/k6 run --out influxdb=http://$(HOST_IP):8086/k6 - < samples/es6sample.js

ถ้าใช้ docker-compose

$ docker-compose run -v $PWD/samples:/scripts k6 run /scripts/es6sample.js

InfluxDB Command Snippets

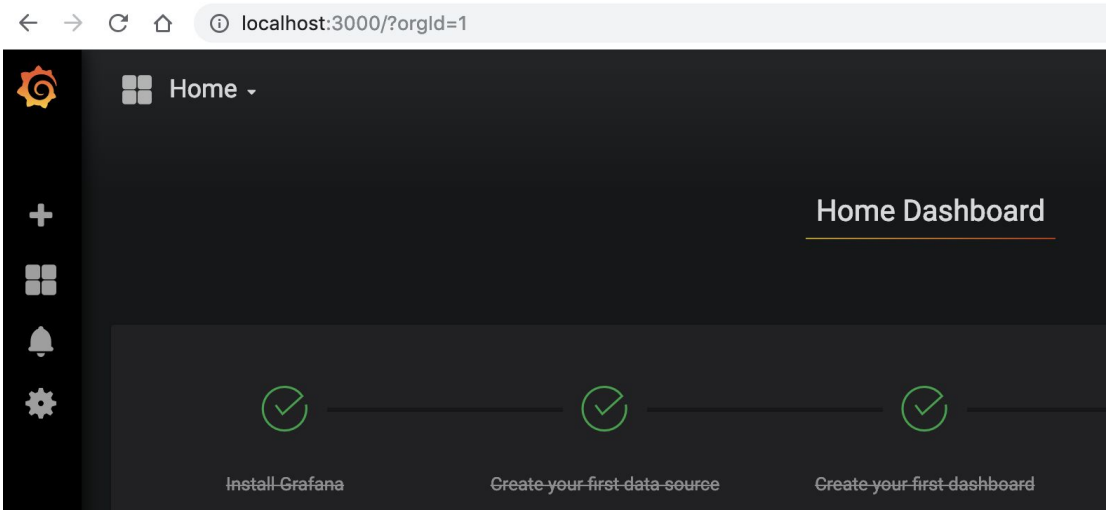
```
$ docker-compose exec influxdb sh
# influx -precision rfc3339
# show databases
# use k6
# select * from  /.*/
# show measurements
```

ทดสอบเป็นเวลา 3 นาที เพื่อมีข้อมูลไปแสดงใน dashboard

```
$ docker run --net="host" -i loadimpact/k6 run --out influxdb=http://192.168.1.107:8086/k10 --vus 10 --duration 180s  - < script.js
```

Grafana

ให้เข้าไปที่ http://localhost:3000/ เพื่อไปกำหนดการแสดงผลของ Grafana

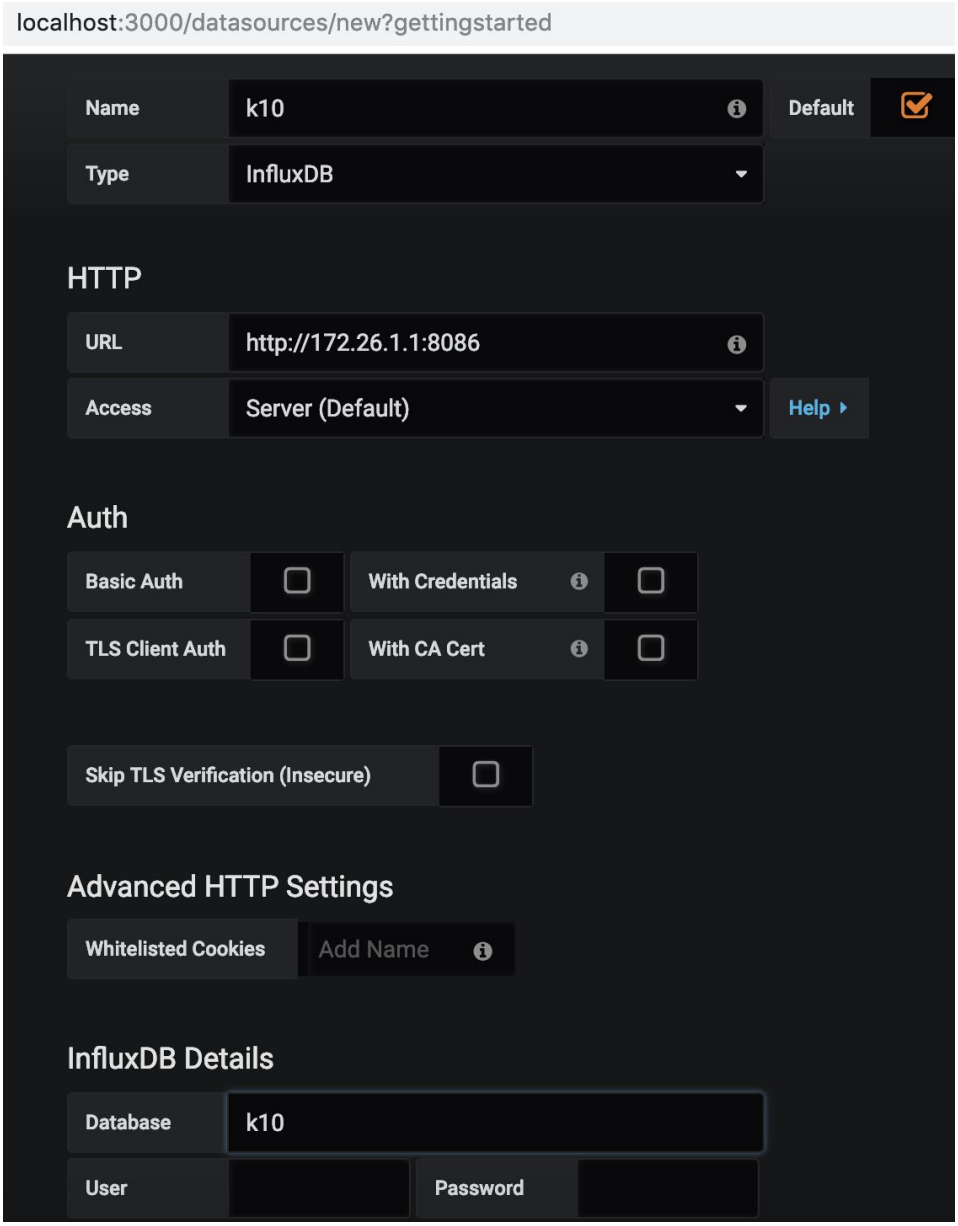


กำหนดการตั้งค่า Query ใน Grafana Dashboard ดังนี้

1. กำหนด datasource

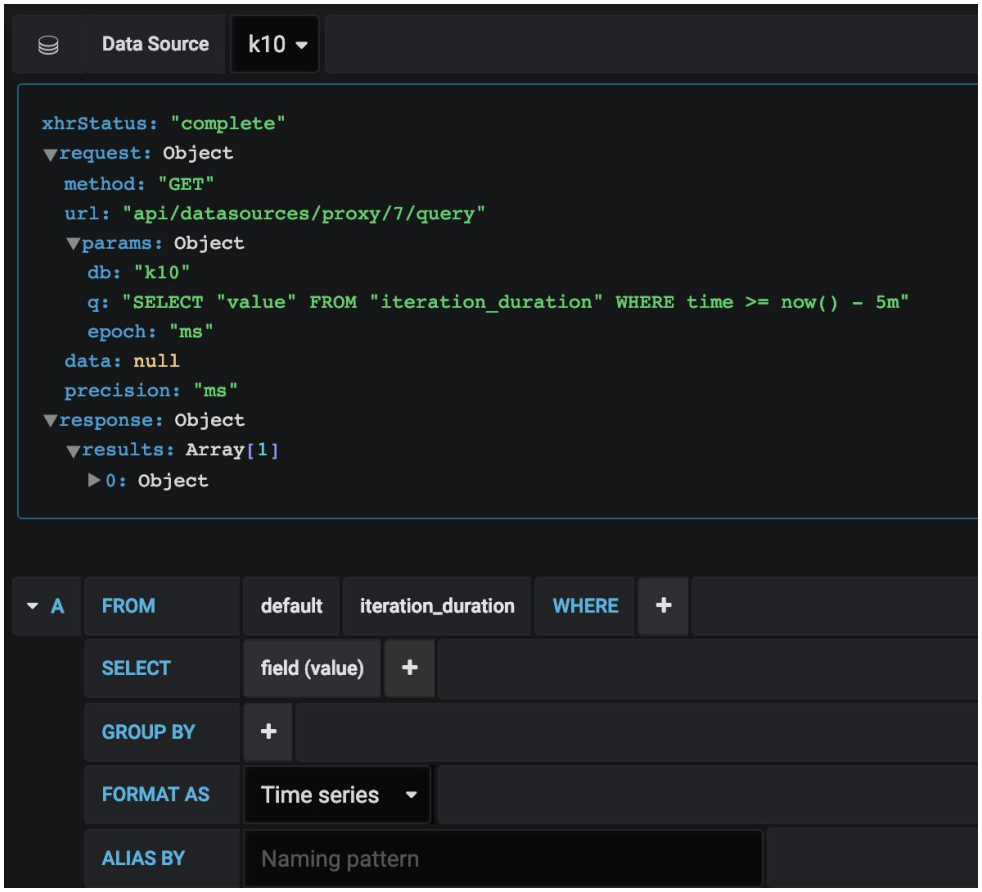
ไปที่ Home Dashboard เลือก create your first datasource

กำหนดชื่อ Name: k10 (ตั้งชื่ออะไรก็ได้), Type: influxDB, URL: http://YOUR_SERVER_IP:8086/ (not localhost เพราะอ้างอิงจาก docker container) และกำหนดชื่อฐานข้อมูล Database: k10 (ชื่อฐานข้อมูลนี้ จำเป็นต้องตรงกับ ชื่อฐานข้อมูล ตอนที่กำหนด option k6 (--out influxdb=http://192.168.1.107:8086/k10) ที่เหลือกำหนดเป็น default และ save datasource ไป

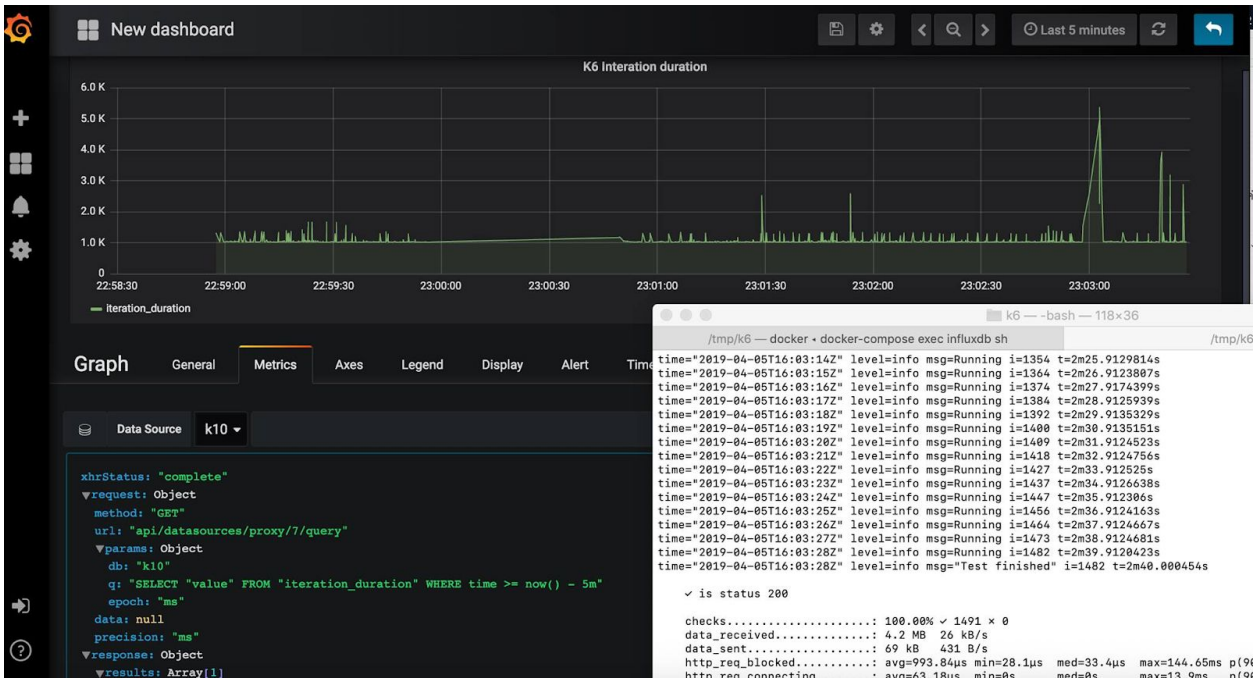


2. กำหนด Graph datasource

เลือก create your first dashboard -> Graph จากนั้น configure ดังนี้



เลือก datasource ตามที่กำหนดไว้ในข้อ 1 จากนั้น ในส่วนของ Query ที่เลือก Group By ให้ Remove ออก จากนั้นสังเกตผลการทำงาน



อ้างอิง: https://docs.k6.io/docs/influxdb-grafana