테스트 자동화

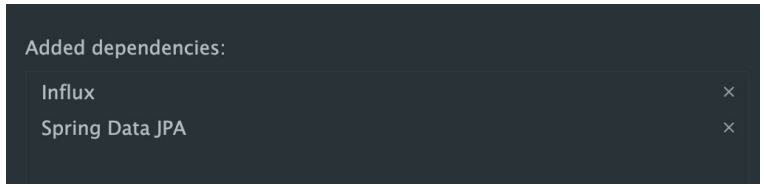
테스트 자동화 간단히 진행해보기

Kotest / Junit

- JVM 진영에서 테스트 코드 짤 때 가장 많이 씁니다

왜 자동화인지 ?

- 테스트코드 = 사람이 테스트 할 기능목록 중 코드에게 맡길 수 있는 부분은 맡깁시다!! 라는 논제에서 탄생



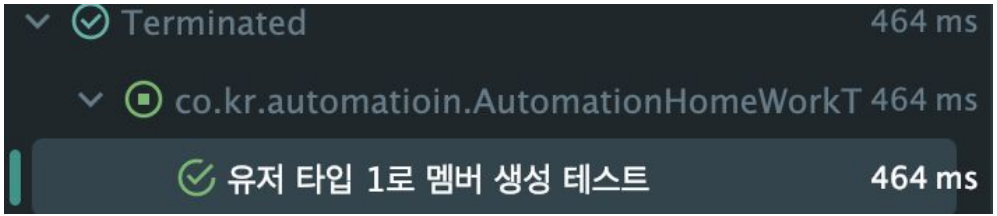
요정도만
넣어볼까요

로직은 [1이면 멤버, 2이면 운영진] 입니다

도커로 DB정도만 띄워주겠습니다
(메모리디비 사용하셔도 무방해요)

```
"유저 타입 1로 멤버 생성 테스트" {
    // given
    val userService = mockk<UserService>()
    val handler = AutomationHomeWorkTestHandler(userService)

    val name = "멤버유저"
    val email = "member@example.com"
    val userType = 1
    val expectedUser = User( id = 1L, name, email, UserType.MEMBER.value)

    every { userService.createUser(name, email, UserType.MEMBER.value) } returns expectedUser

    // when
    val result = handler.processUserCreation(name, email, userType)

    // then
    result shouldContain "멤버 유저가 생성되었습니다"
    result shouldContain "ID: 1"
    result shouldContain "이름: $name"
    verify(exactly = 1) { userService.createUser(name, email, UserType.MEMBER.value) }
}
```

테스트 시작! 옆에 화살표 누르면 실행입니다



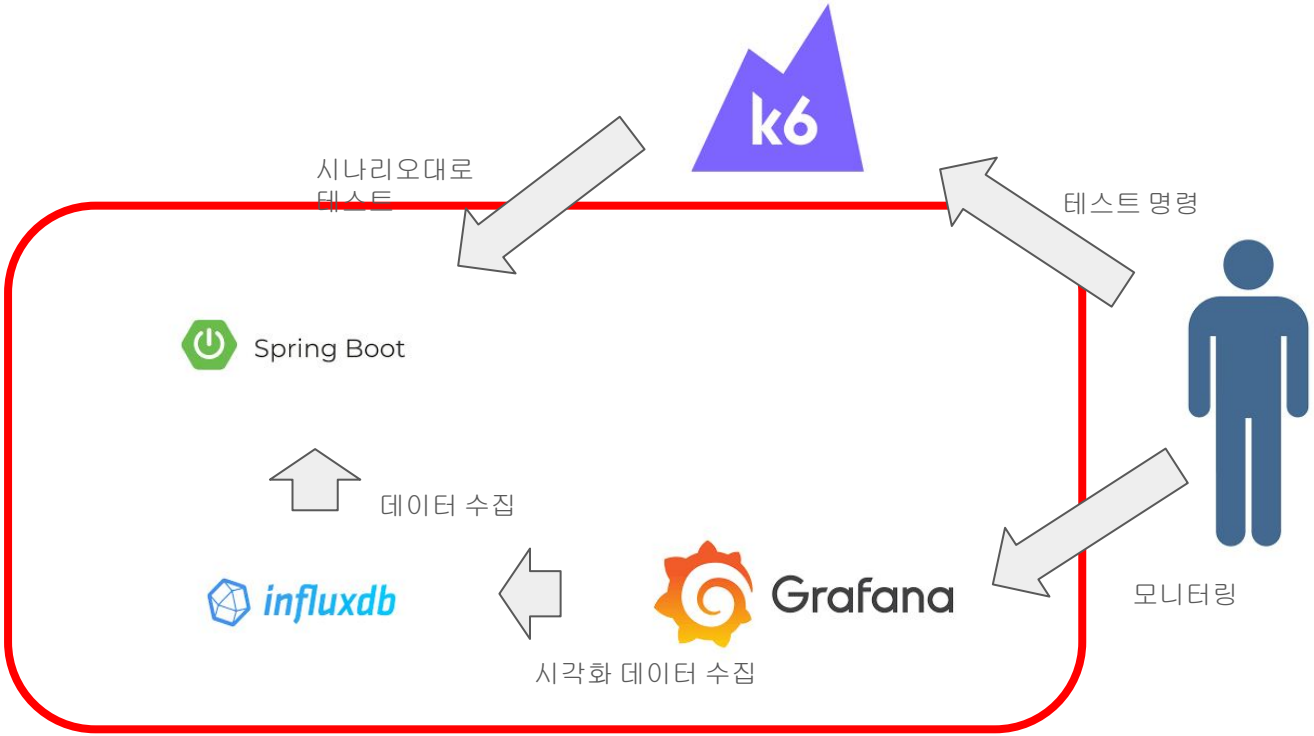
성공했답니다

테스트 자동화 툴 사용

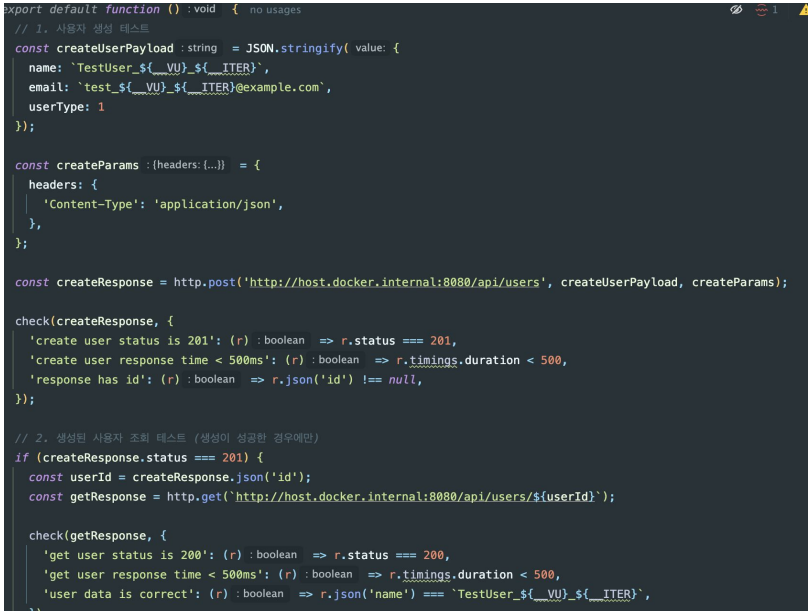
끝

이번엔 Stress Test라는걸 해보겠습니다

몇명까지 내 서버가 버티는지 알고싶고,
auto scale out도 기준을 걸어야 할 텐데,
해당 기준을 알기위해 테스트를 하자니
일일히 다 클릭할수는 없겠죠??
해당 툴이 대신해줄겁니다 ㅎㅎ

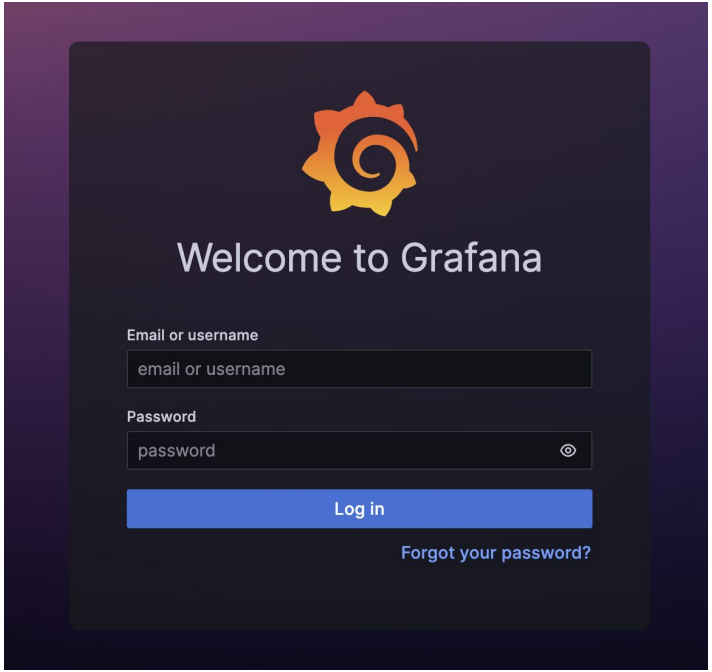

k6
시나리오대로
테스트
테스트 명령
Spring Boot
데이터 수집
influxdb
Grafana
시각화 데이터 수집
모니터링

이런 그림이
되겠네요

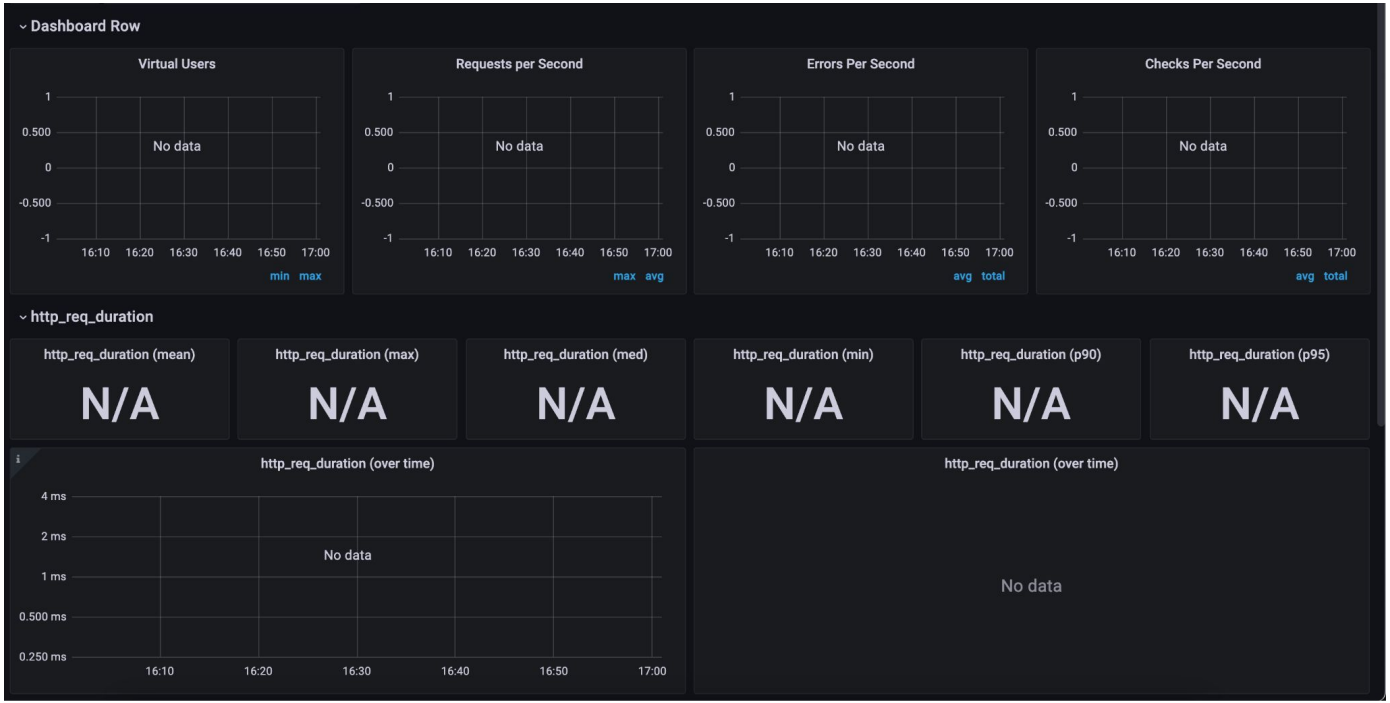


```
export default function () : void  {  no usages
  // 1. 사용자 생성 테스트
  const createUserPayload : string  = JSON.stringify( value: {
    name: `TestUser_${__VU}_${__ITER}`,
    email: `test_${__VU}_${__ITER}@example.com`,
    userType: 1
  });

  const createParams : {headers: {...}}  = {
    headers: {
      'Content-Type': 'application/json',
    },
  };

  const createResponse = http.post('http://host.docker.internal:8080/api/users', createUserPayload, createParams);

  check(createResponse, {
    'create user status is 201': (r) : boolean  => r.status === 201,
    'create user response time < 500ms': (r) : boolean  => r.timings.duration < 500,
    'response has id': (r) : boolean  => r.json('id') !== null,
  });

  // 2. 생성된 사용자 조회 테스트 (생성이 성공한 경우에만)
  if (createResponse.status === 201) {
    const userId = createResponse.json('id');
    const getResponse = http.get(`http://host.docker.internal:8080/api/users/${userId}`);

    check(getResponse, {
      'get user status is 200': (r) : boolean  => r.status === 200,
      'get user response time < 500ms': (r) : boolean  => r.timings.duration < 500,
      'user data is correct': (r) : boolean  => r.json('name') === `TestUser_${__VU}_${__ITER}`,
```

테스트 전에 요런식으로 시나리오 테스트 해줘! 하는 스크립트는 작성해주셔야 돼요..^^



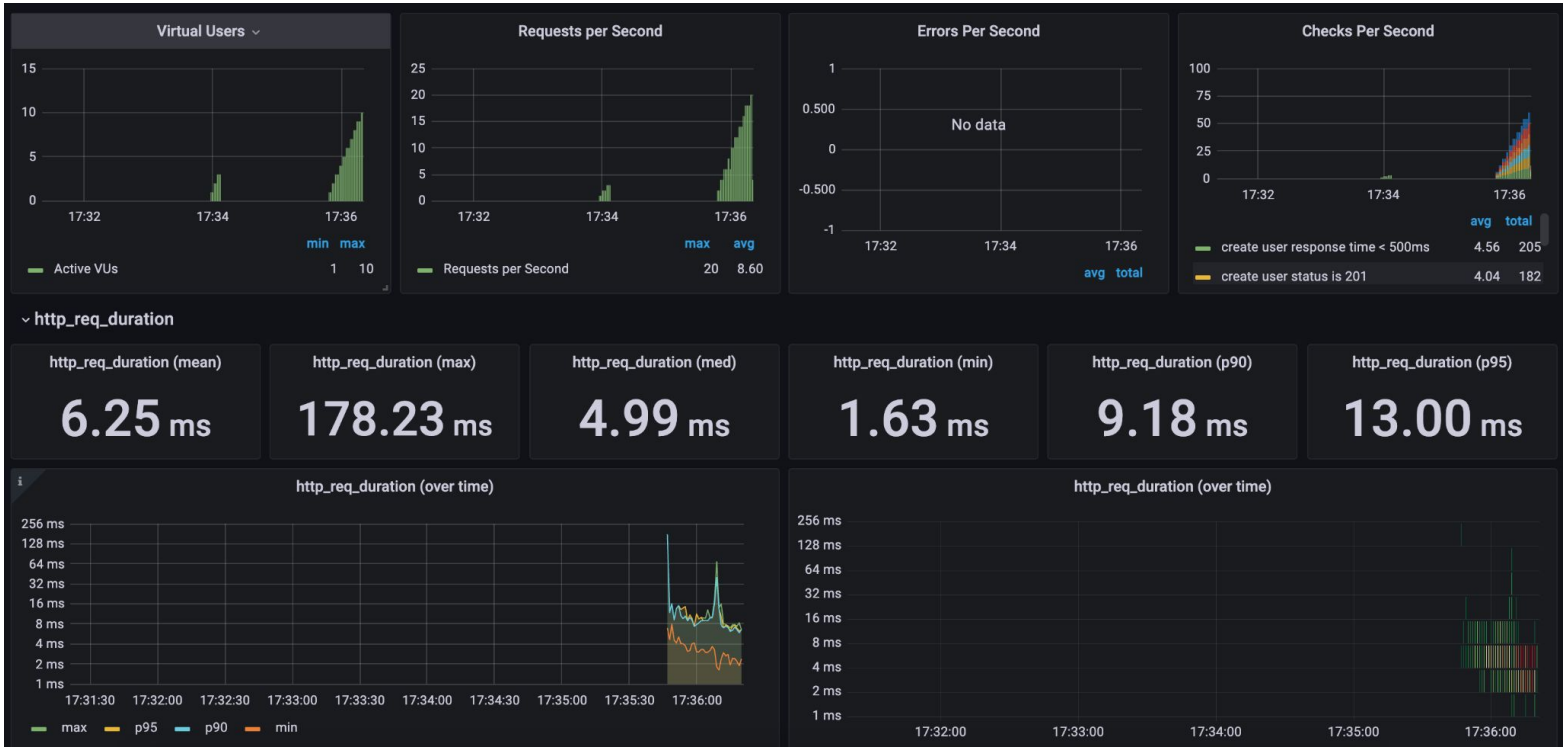
전부 도커로
올리겠습니다



대시보드는 아무거나 가져오시면 됩니다.
당연히 지금은 아무것도 없습니다.

```
dallo@ip-192-168-5-3 automation-test % docker compose up k6
WARN[0000] /Users/dallo/Desktop/projects/01-homework/automat
e it to avoid potential confusion
[+] Running 1/1
 ✔ Container automation-test-k6-1  Recreated
```

이제 k6를 작동시켜보겠습니다.


Virtual Users
15
10
5
0
17:32
17:34
17:36
min max
Active VUs 1 10
Requests per Second
25
20
15
10
5
0
17:32
17:34
17:36
max avg
Requests per Second 20 8.60
Errors Per Second
1
0.500
0
-0.500
-1
No data
17:32
17:34
17:36
avg total
Checks Per Second
100
75
50
25
0
17:32
17:34
17:36
avg total
create user response time < 500ms 4.56 205
create user status is 201 4.04 182
http_req_duration
http_req_duration (mean)
6.25 ms
http_req_duration (max)
178.23 ms
http_req_duration (med)
4.99 ms
http_req_duration (min)
1.63 ms
http_req_duration (p90)
9.18 ms
http_req_duration (p95)
13.00 ms
http_req_duration (over time)
256 ms
128 ms
64 ms
32 ms
16 ms
8 ms
4 ms
2 ms
1 ms
17:31:30
17:32:00
17:32:30
17:33:00
17:33:30
17:34:00
17:34:30
17:35:00
17:35:30
17:36:00
max p95 p90 min
http_req_duration (over time)
256 ms
128 ms
64 ms
32 ms
16 ms
8 ms
4 ms
2 ms
1 ms
17:32:00
17:33:00
17:34:00
17:35:00
17:36:00

뭐가 변하기 시작하네요

끝났을 때의 모습입니다.
수치 해석은 사람마다 환경마다 다르니 패스!

```
running (2m00.0s), 01/10 VUs, 908 complete and 0 interrupted iterations
default   [ 100% ] 01/10 VUs  2m00.0s/2m00.0s


  TOTAL RESULTS

    checks_total.......: 5454    45.20776/s
    checks_succeeded...: 100.00% 5454 out of 5454
    checks_failed......: 0.00%   0 out of 5454

    ✓ create user status is 201
    ✓ create user response time < 500ms
    ✓ response has id
    ✓ get user status is 200
    ✓ get user response time < 500ms
    ✓ user data is correct

    HTTP
    http_req_duration..............: avg=4.66ms min=541.25µs med=3.73ms max=184.84ms p(90)=7.2ms p(95)=9.05ms
      { expected_response:true }...: avg=4.66ms min=541.25µs med=3.73ms max=184.84ms p(90)=7.2ms p(95)=9.05ms
    http_req_failed................: 0.00%  0 out of 1818
    http_reqs......................: 1818   15.069253/s

    EXECUTION
    iteration_duration.............: avg=1.01s  min=1s       med=1.01s  max=1.23s    p(90)=1.01s p(95)=1.02s
    iterations.....................: 909    7.534627/s
    vus............................: 1      min=1         max=10
    vus_max........................: 10     min=10        max=10

    NETWORK
    data_received..................: 413 kB 3.4 kB/s
    data_sent......................: 278 kB 2.3 kB/s
```

k6는 요런식으로 테스트 요약본도 전달해준답니다! 끝!