Fullstack Development

API Architectures and Design #4

Content

- What is API?
- API Architecture Styles
- RESTful API design
- API Security
- **API Testing**

API Testing

Functionality <Cypress, Postman, Insomnia>

- Endpoints return the `correct data` and perform the `expected operations`

Reliability

- Ability to handle various scenarios (error conditions) `without crashing`

API Testing (2)

Security <OWASP Tools, Google Apigee>

- Identifying **vulnerabilities**
- `Unauthorized access`, `data breaches`, `injection flaw`

Performance <Apache JMeter, K6>

- Measuring `response time` and `throughtput` under different **load conditions**

API Performance Testing

Performance Testing Tool

K6

- An open-source **load testing tool** developed by **Grafana Labs**
- Write tests in `JavaScript` or `TypeScript`
  - Run locally (`Windows`, `macOS`, `Linux`, `Docker`)
  - Run on **Grafana Cloud**
- Supports `RESTful`, `GraphQL`, `WebSocket`, `gRPC`
- Supports different types of testing

Types of Testing

Load testing

Verify that applications can handle the expected traffic. Different goals require different tests: **stress tests**, **spike tests**, **soak tests**, **smoke tests**, etc.

End-to-end web testing

Mix browser and API testing—interact with real browsers and collect frontend metrics to get a holistic user view.

Synthetic monitoring

Traditional ping testing is not enough anymore. Reuse your k6 tests with **Synthetic Monitoring** to continuously verify production environments.

Fault injection testing

Inject faults in Kubernetes-based apps to recreate application errors. Test resilience patterns and tolerance of internal errors to improve reliability.

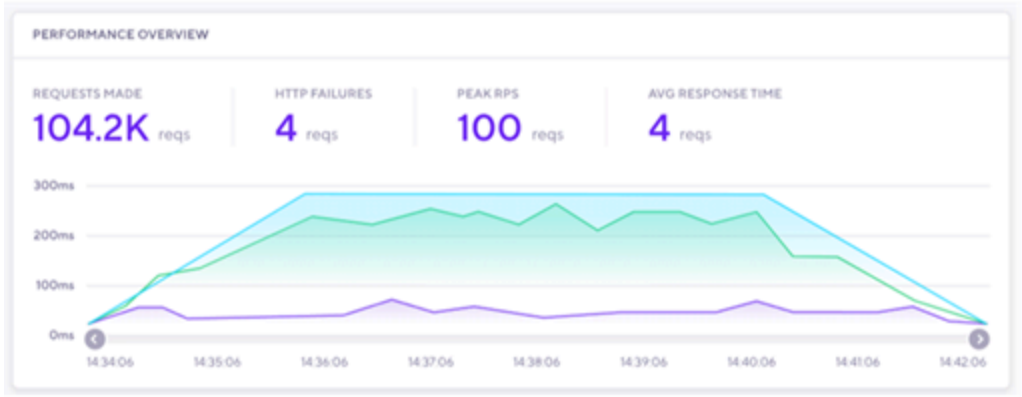
Infrastructure testing

Test how cloud-native systems scale. Isolate bottlenecks. Plan and provision infrastructure capacity.

Regression testing

Test continuously to track changes in performance and reliability. Prevent software regressions from reaching production.

K6 Results




```
duration: 3s, iterations: -
     vus: 50, max: 50

done [==========================================================] 3s / 3s

✓ is status 200

checks.....................: 100.00% ✓ 1137 ✗ 0
data_received..............: 1.8 MB  599 kB/s
data_sent..................: 122 kB  41 kB/s
http_req_blocked...........: avg=24.26ms  min=0s      med=0s       max=569.88ms
http_req_connecting........: avg=4.89ms   min=0s      med=0s       max=126.58ms
http_req_duration..........: avg=105.77ms min=96.76ms med=103.42ms max=156.32ms
http_req_receiving.........: avg=441.85µs min=43µs    med=97µs     max=14.77ms
http_req_sending...........: avg=43.6µs   min=15µs    med=29µs     max=443µs
http_req_tls_handshaking...: avg=16.72ms  min=0s      med=0s       max=389.87ms
http_req_waiting...........: avg=105.29ms min=96.61ms med=103.04ms max=156.21ms
http_reqs..................: 1137    378.974978/s
iteration_duration.........: avg=130.16ms min=96.89ms med=103.51ms max=686.94ms
iterations.................: 1137    378.974978/s
vus........................: 50      min=50 max=50
vus_max....................: 50      min=50 max=50
```

What does performance mean?

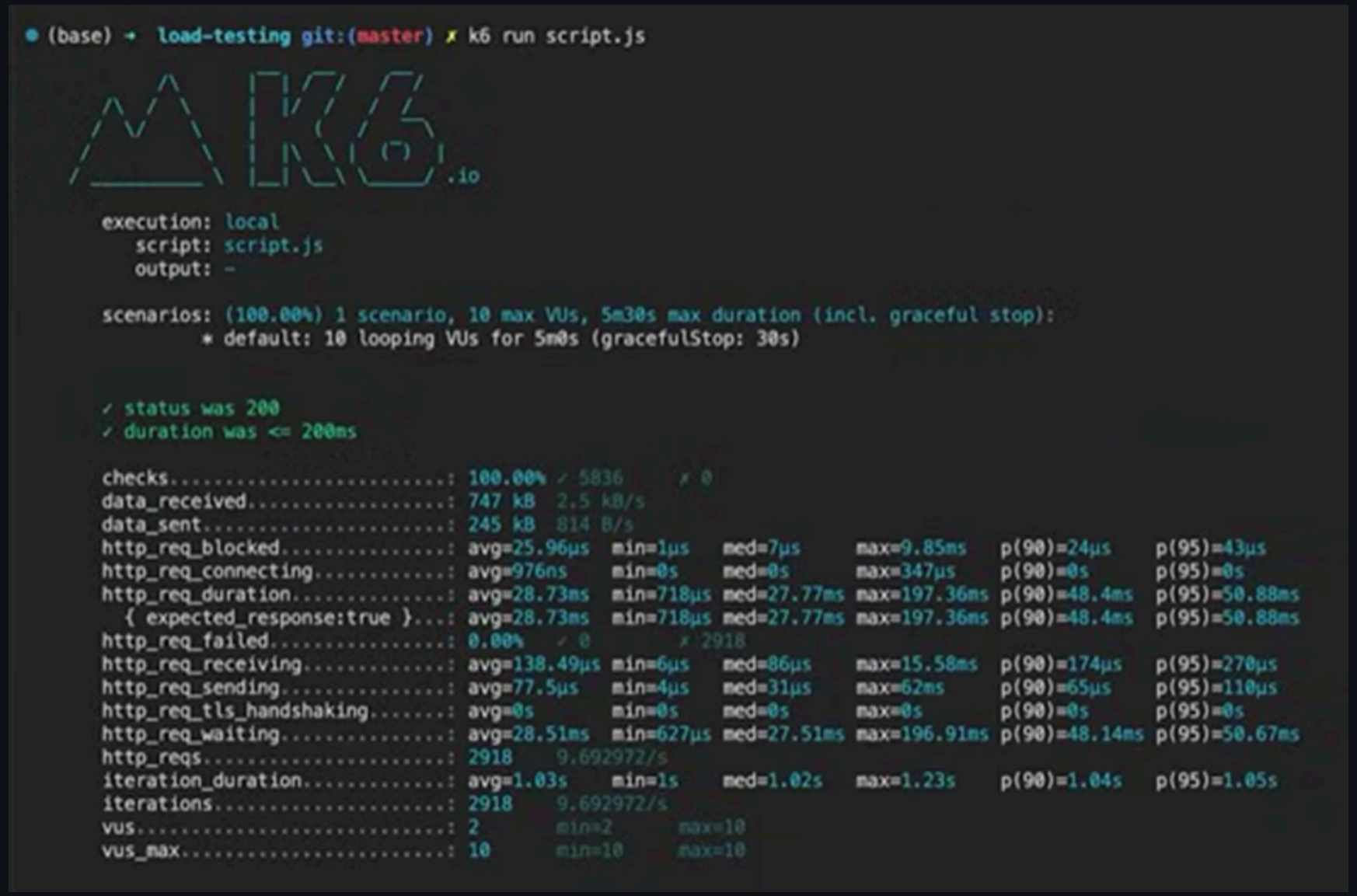
- Influences the **type of tests** you should perform
- Define normal API traffic and **acceptable response time**

```
// script.js
import http from 'k6/http';
import { sleep, check } from 'k6';

export const options = {
  vus: 10,
  duration: '5m',
};

export default function () {
  const res = http.get('http://localhost:3000/book');
  check(res, {
    'status was 200': (r) => r.status == 200,
    'duration was <= 200ms': (r) => r.timings.duration <= 200,
  });
  sleep(1);
}
```

Load testing:
Request from 10 virtual users
Response time < 200ms?
Run test for 5 minutes


● (base) → load-testing git:(master) ✗ k6 run script.js
.io
execution: local
script: script.js
output: -
scenarios: (100.00%) 1 scenario, 10 max VUs, 5m30s max duration (incl. graceful stop):
* default: 10 looping VUs for 5m0s (gracefulStop: 30s)
✓ status was 200
✓ duration was <= 200ms
checks.....: 100.00% ✓ 5836 ✗ 0
data_received.....: 747 kB 2.5 kB/s
data_sent.....: 245 kB 814 B/s
http_req_blocked.....: avg=25.96µs min=1µs med=7µs max=9.85ms p(90)=24µs p(95)=43µs
http_req_connecting.....: avg=976ns min=0s med=0s max=347µs p(90)=0s p(95)=0s
http_req_duration.....: avg=28.73ms min=718µs med=27.77ms max=197.36ms p(90)=48.4ms p(95)=50.88ms
{ expected_response:true }...: avg=28.73ms min=718µs med=27.77ms max=197.36ms p(90)=48.4ms p(95)=50.88ms
http_req_failed.....: 0.00% ✓ 0 ✗ 2918
http_req_receiving.....: avg=138.49µs min=6µs med=86µs max=15.58ms p(90)=174µs p(95)=270µs
http_req_sending.....: avg=77.5µs min=4µs med=31µs max=62ms p(90)=65µs p(95)=110µs
http_req_tls_handshaking.....: avg=0s min=0s med=0s max=0s p(90)=0s p(95)=0s
http_req_waiting.....: avg=28.51ms min=627µs med=27.51ms max=196.91ms p(90)=48.14ms p(95)=50.67ms
http_reqs.....: 2918 9.692972/s
iteration_duration.....: avg=1.03s min=1s med=1.02s max=1.23s p(90)=1.04s p(95)=1.05s
iterations.....: 2918 9.692972/s
vus.....: 2 min=2 max=10
vus_max.....: 10 min=10 max=10

Specify Thresholds

```
// script.js
import http from 'k6/http';
import { sleep, check } from 'k6';

export const options = {
  vus: 10,
  duration: '5m',
  thresholds: {
    http_req_failed: ['rate<0.01'], // http errors should be less than 1%
    http_req_duration: ['p(99)<200'], // 95% of requests should be below 200ms
  },
};

export default function () {
  const res = http.get('http://localhost:3000/book');
  check(res, {
    'status was 200': (r) => r.status == 200,
    'duration was <= 200ms': (r) => r.timings.duration <= 200,
  });
  sleep(1);
}
```

Is it good enough to deploy in production?

Stress Testing

```
export const options = {
  vus: 10,
  duration: '5m',
  thresholds: {
    http_req_failed: ['rate<0.01'], // http errors should be less than 1%
    http_req_duration: ['p(99)<200'], // 95% of requests should be below 200ms
  },
  stages: [
    // level 1
    { duration: '1m', target: 100 },
    { duration: '2m', target: 100 },
    // level 2
    { duration: '1m', target: 200 },
    { duration: '2m', target: 200 },
    // level 3
    { duration: '1m', target: 500 },
    { duration: '2m', target: 500 },
    // coll down
    { duration: '1m', target: 0 },
  ],
```

- Multiple stages of testing
- Shows how system behave in different situations
- Gradually increases #users
  - **10 > 100 > 200 >500 >0**
- Performance will be degraded
- Still acceptable?

Spike Testing

```
export const options = {
  vus: 10,
  duration: '5m',
  thresholds: {
    http_req_failed: ['rate<0.01'], // http errors should be less than 1%
    http_req_duration: ['p(99)<200'], // 95% of requests should be below 200ms
  },
  stages: [
    // warm up
    { duration: '30s', target: 100 },

    // spike
    { duration: '1m', target: 2_000 },
    { duration: '10s', target: 2_000 },
    { duration: '1m', target: 100 },

    // cool down
    { duration: '30s', target: 0 },
  ],
};
```

- High volumes of requests during a short period of time
- e.g., ticket selling
- No concept of normal traffic
- Starts with small #requests
- Drastically **increase** #request and sustain that high load
- Simulate the end of the spike

Soak Testing

```
export const options = {
  vus: 10,
  duration: '5m',
  thresholds: {
    http_req_failed: ['rate<0.01'], // http errors should be less than 1%
    http_req_duration: ['p(99)<200'], // 95% of requests should be below 200ms
  },
  stages: [
    // warm up
    { duration: '1m', target: 200 },

    // sustained load over a long time
    { duration: '4h', target: 200 },

    // cool down
    { duration: '1m', target: 0 },
  ],
};
```

- Resource usage testing
- Identify "memory leak" problem can not be found in short testing
- Takes a few hours
- Not only the "**success**" rate is important

References

- API Testing: A Guide for Beginners and Expert
- Is Your API actually ready for user traffic?
- Git - Template to use TypeScript with k6
- Getting Started with Performance Testing in Typescript Using K6