Performance tests with K6

/thoughtworks

Performance tests with K6

Agenda

What are performance tests?

Definitions

*"**Non-functional testing** is testing software for its **non-functional requirements: the way a system operates**, rather than specific behaviors of that system. This is in contrast to functional testing, which tests against functional requirements that describe the functions of a system and its components."*

*"In software quality assurance, **performance testing is in general a testing practice performed to determine how a system performs in terms of responsiveness and stability under a particular workload**."*

Wikipedia >

Why do we need performance tests?

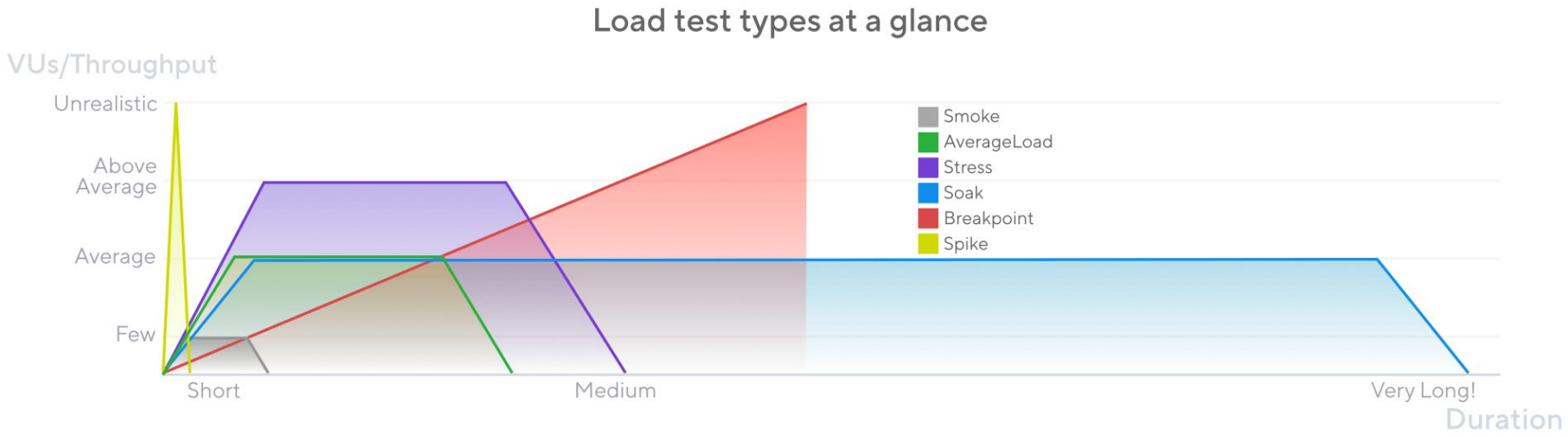
A performance test typically aims to do one of two things:

- Validate reliability under expected conditions (traffic, workload...)
  - Is the application meeting our SLOs?
- Discover problems and system limits under unusual conditions
  - Is the new environment ready for production workload?

Types of performance tests

Before you configure performance tests, you should know what traffic patterns you want to test the API for.

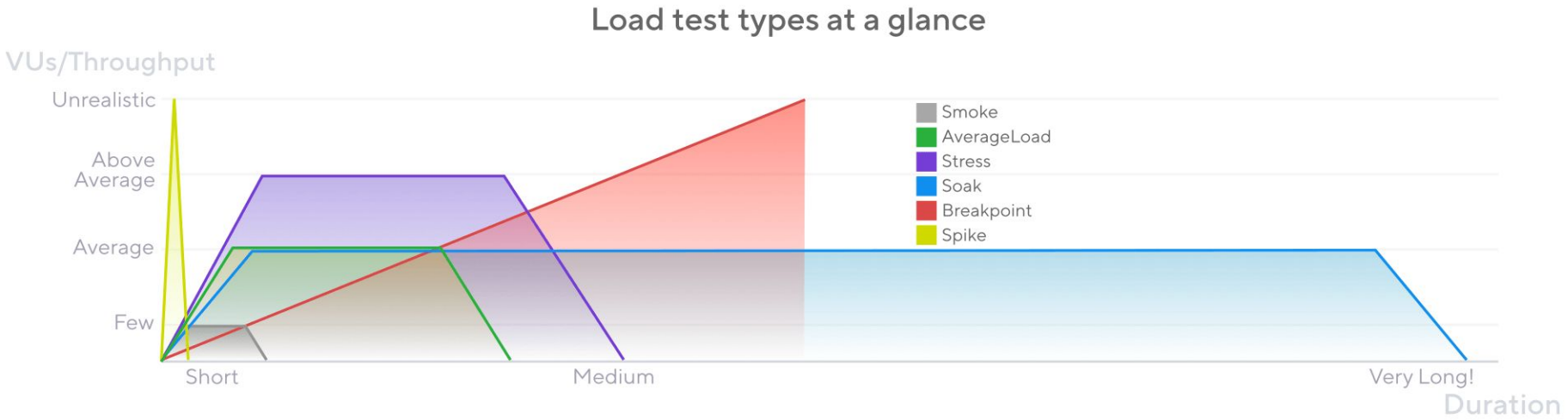
There is no consensus about the names: capacity, endurance, rush-hour, stamina...



Types of performance tests

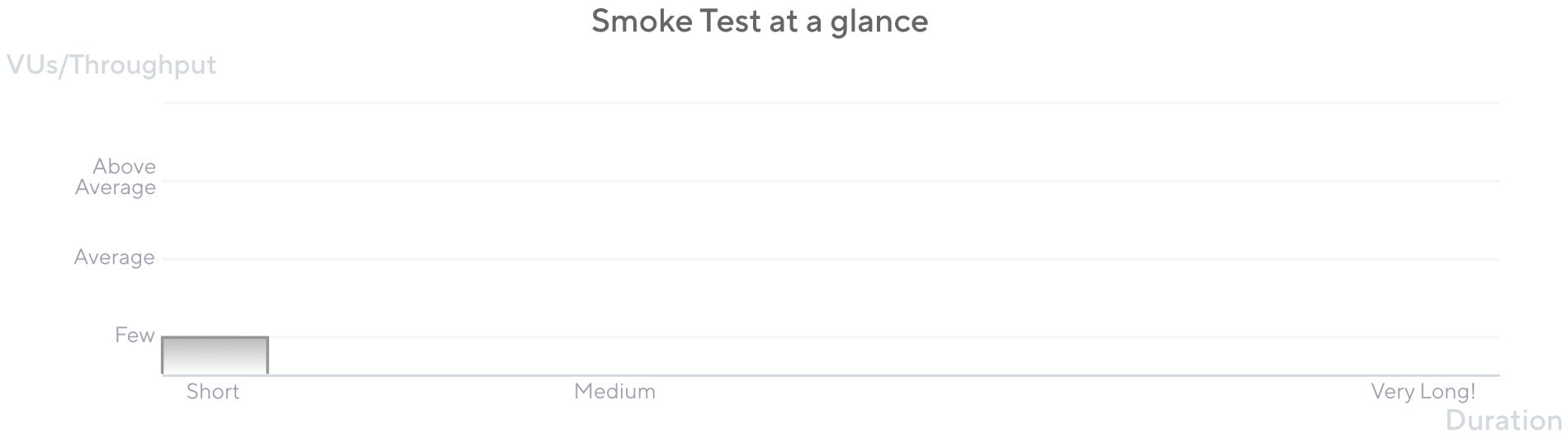
The order matters!

- Smoke: Test for script errors and verify SUT with minimal traffic.
- Average-Load: Test for regular/normal traffic.
- Stress: Test for maximum expected traffic.
- Soak: Test for a prolonged period of traffic.
- Spike: Test for a surge of traffic.
- Breakpoint tests gradually increase load to identify the capacity limits of the system



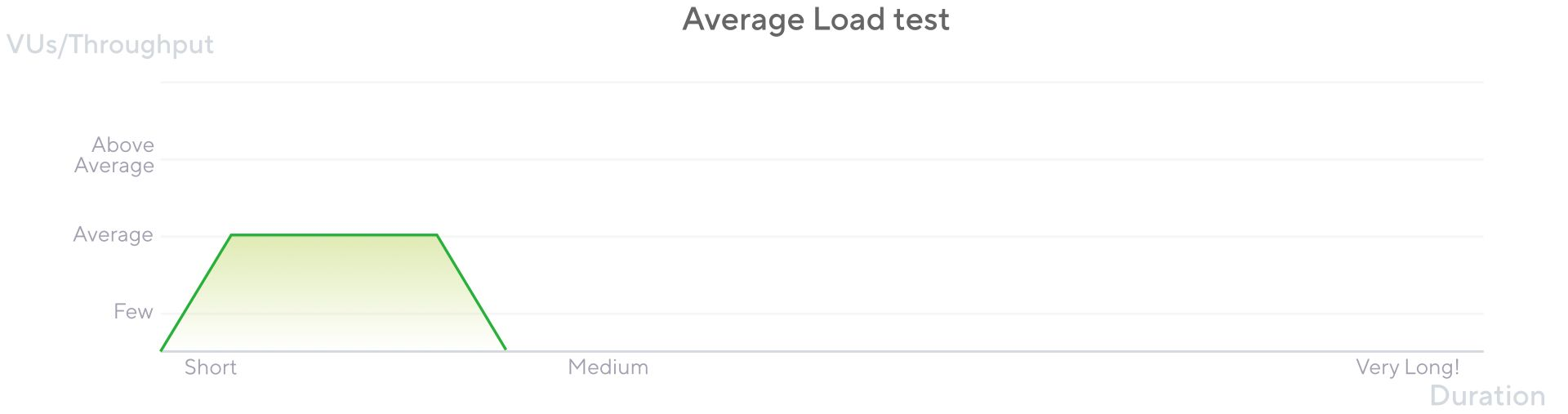
Smoke tests

- Smoke testing should also be done whenever the relevant application code is updated.
- Run them to verify that the system works well under minimal load and to gather baseline performance values.
- This test type consists of running tests with a few VUs — more than 5 VUs could be considered a mini-load test
- Goal: Test basic business functionality that must work all the time:
    - Can I get flight offers?
    - Can I buy a flight ticket?



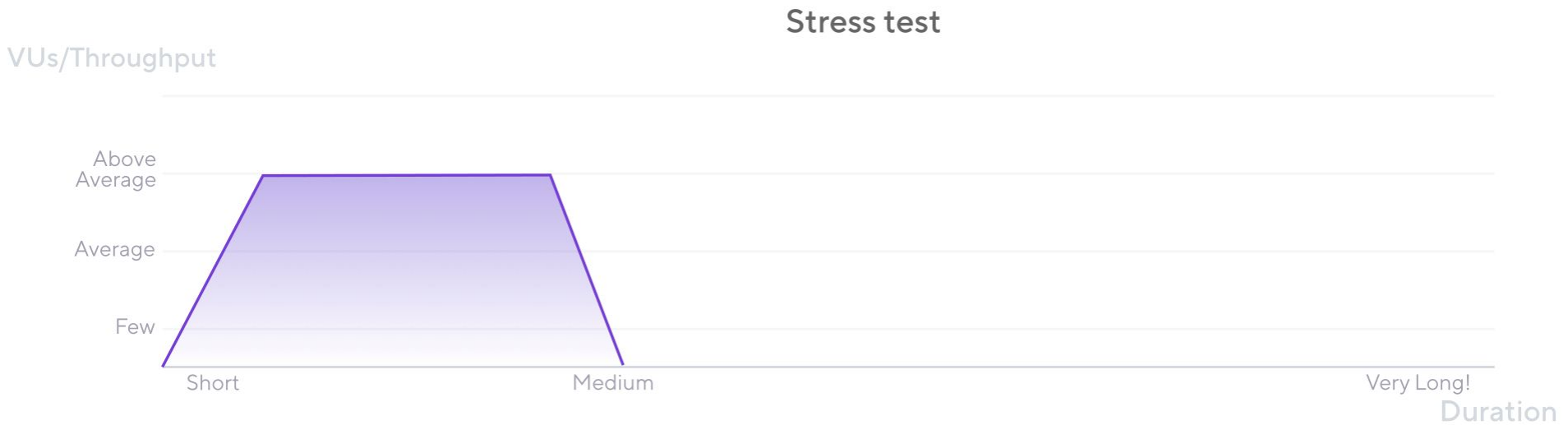
Average load tests

- An average-load test assesses how the system performs under typical load. Typical load might be a regular day in production
- This type of test typically increases the throughput or VUs gradually and keeps that average load for some time
- Assure that the system still meets the performance standards after system changes (code and infrastructure)
- Goal: validate it is ready for production under normal operating conditions
- First time running load tests, **we recommend starting small or configuring the ramp-up to be slow**



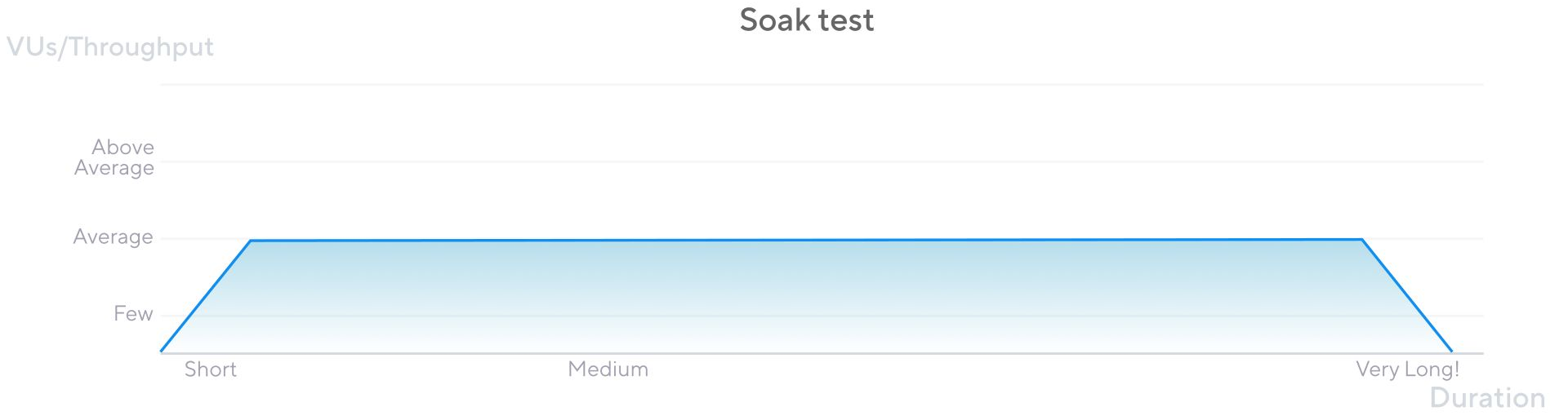
Stress

- Stress testing assesses how the system performs when loads are heavier than usual.
- Similar pattern to average load test, but the load is higher than usual (50% - 100%...)
- This test determines how much the performance degrades with the extra load and whether the system survives it
- Goal: verify the stability, scalability and reliability of the system under conditions of heavy use
    - Black Friday (hours-days), Christmas days (weeks), Marketing campaigns (weeks-months)...
    - Intervals and patterns: day/night, traffic from regions with different time zones…



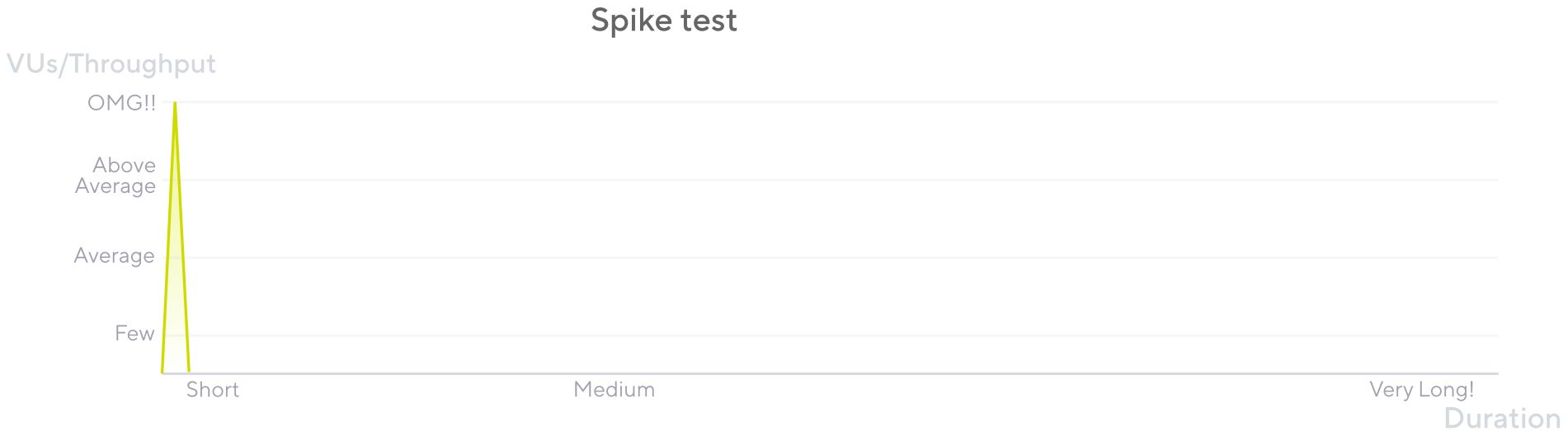
Soak tests

- Now that you know that your system can handle outstanding load events, check if the system performs well over extended periods
- This test type checks for common performance defects that show only after extended use.
- Goal:
  - The system's degradation of performance and resource consumption over extended periods: memory leaks, CPU usage, network…
  - The system's availability and stability during extended periods: functional degradation, loading/response times…



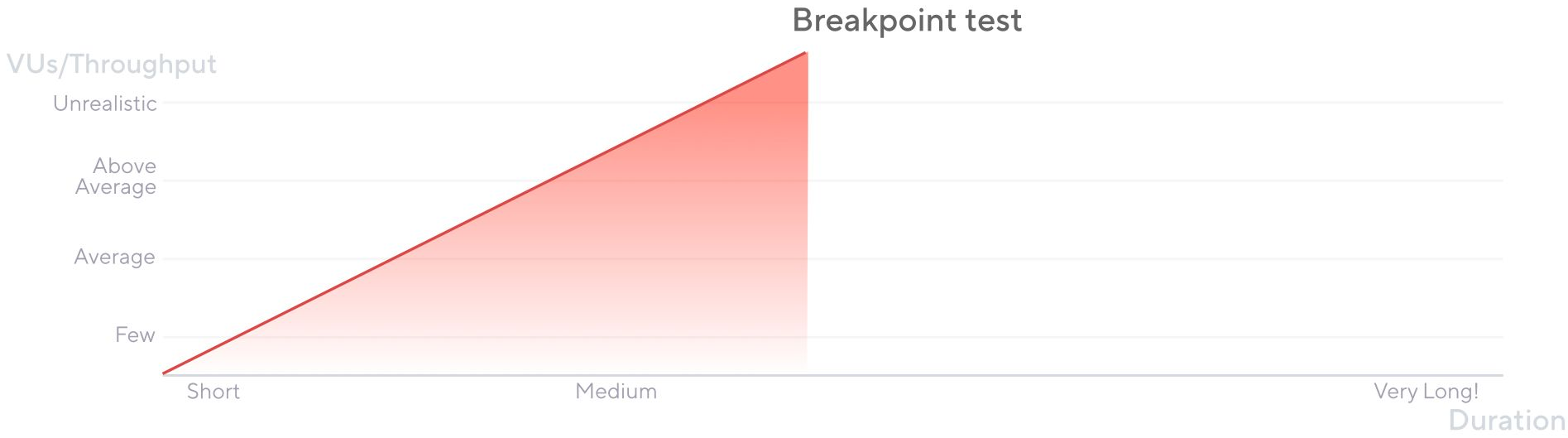
Spike tests

- Verifies whether the system survives and performs under sudden and massive rushes of utilization:
  - Tickets for a concert, aggressive marketing campaigns…
  - Very time dependant
- Spike testing increases to extremely high loads in a very short or non-existent ramp-up time
- Backend monitoring is a must for successful outcomes of this test.
- Check how infrastructure scales and recover after the load rush
- Run, tune, repeat…



Breakpoint tests

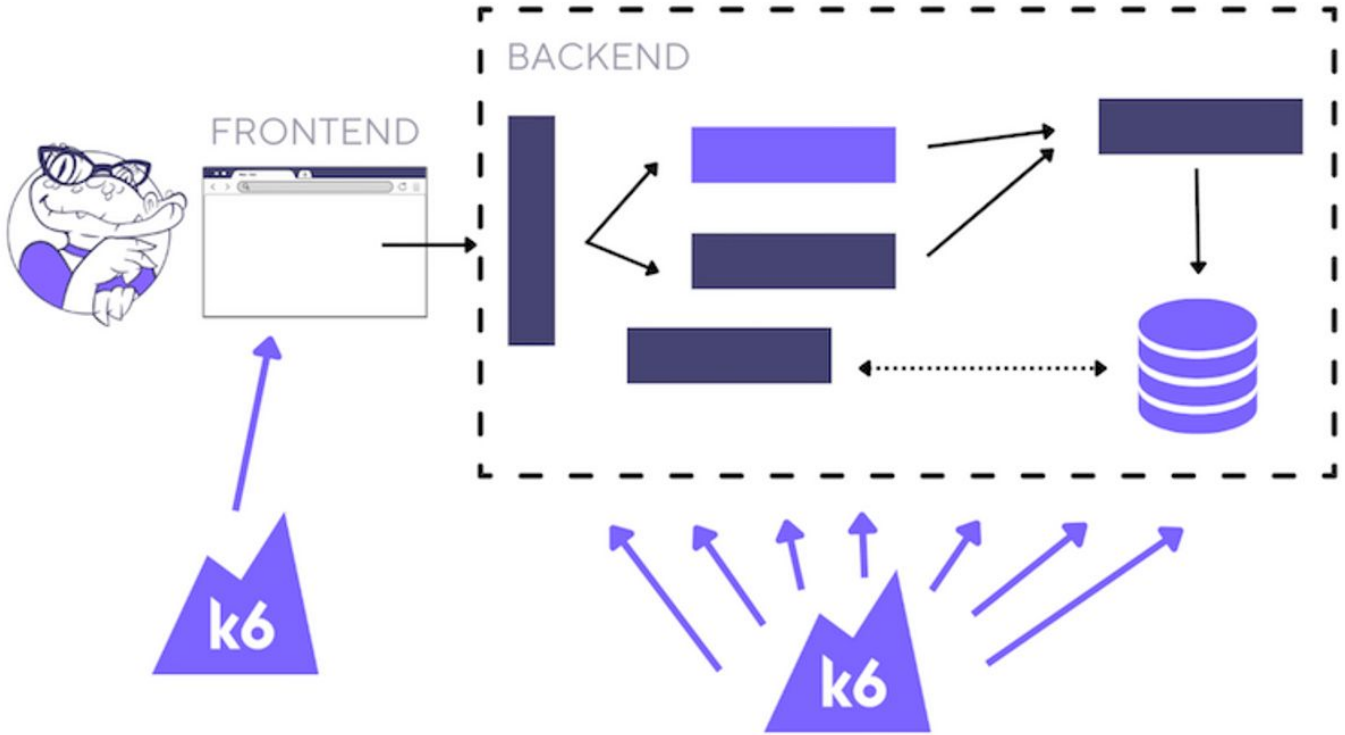
- Breakpoint testing aims to find system limits
- Goal: knowing where and how a system starts to fail helps prepare for such limits
- A breakpoint ramps to unrealistically high numbers
- Watch out in elastic cloud environments…! => **Costs++**
- Run after significant changes to the code-base or infrastructure
- After results, analyse and iterate:
  - accept and prevent
  - tune the system



Performance tests cheat sheet

Type	VUs/Throughput	Duration	When?
Smoke	Low	Short (seconds or minutes)	When the relevant system or application code changes. It checks functional logic, baseline metrics, and deviations
Load	Average production	Mid (5-60 minutes)	Often to check system maintains performance with average use
Stress	High (above average)	Mid (5-60 minutes)	When system may receive above-average loads to check how it manages
Soak	Average	Long (hours)	After changes to check system under prolonged continuous use
Spike	Very high	Short (a few minutes)	When the system prepares for seasonal events or receives frequent traffic peaks
Breakpoint	Increases until break	As long as necessary	A few times to find the upper limits of the system

K6



What is K6?

k6 is an open-source, developer-friendly, and extensible load testing tool.

k6 Open Source is designed for load testing. Extensible to support other 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 test

JavaScript Copy

```javascript
import http from 'k6/http';
import { sleep } from 'k6';

export const options = {
  iterations: 10,
};

// The default exported function is gonna be picked up by k6 as the entry point for
export default function () {
  // Make a GET request to the target URL
  http.get('https://test-api.k6.io');

  // Sleep for 1 second to simulate real-world usage
  sleep(1);
}
```

K6 test lifecycle

Test stage	Purpose	Example	Called
1. init	Load local files, import modules, declare lifecycle functions	Open JSON file, Import module	Once per VU*
2. Setup	Set up data for processing, share data among VUs	Call API to start test environment	Once
3. VU code	Run the test function, usually `default`	Make https requests, validate responses	Once per iteration, as many times as the test options require
4. Teardown	Process result of setup code, stop test environment	Validate that setup had a certain result, send webhook notifying that test has finished	Once **

JavaScript

```
// 1. init code

export function setup() {
  // 2. setup code
}

export default function (data) {
  // 3. VU code
}

export function teardown(data) {
  // 4. teardown code
}
```

K6 options

How to configure test load:

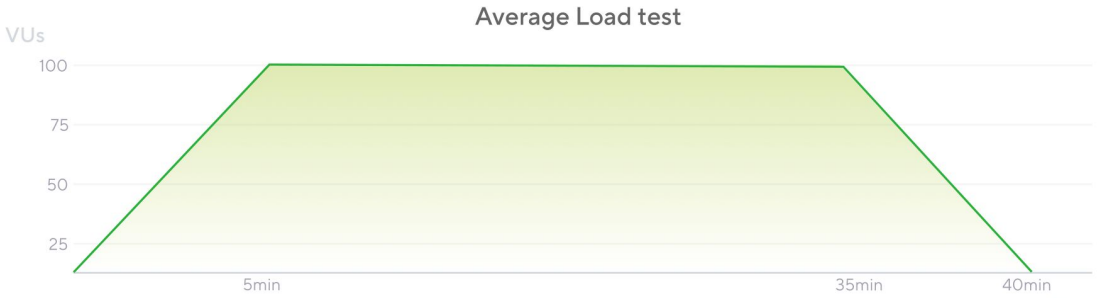
K6 options

- **VUs:** an integer value specifying the number of Virtual users to run concurrently
  - used together with the **iterations or duration** options (stages / scenarios)
- **Duration:** A string specifying the total duration a test run should be run for. During this time each VU will execute the script in a loop
- **Iterations:** An integer value, specifying the total number of iterations of the default function to execute in the test run
- **Stages:** A list of VU `{ target: ..., duration: ... }` objects that specify the target number of VUs to ramp up or down to for a specific period
- **VUs + Duration:** VUs are essentially parallel `while(true)` loops
- **VUs + Iterations:** 10 VUs to run 10 times each
  - iterations aren't fairly distributed and a VUs compete each other

K6 options

How to configure test load:

- Duration + Target (VUs): ramp up from X to Y VUs in a duration of D




```
export const options = {
  stages: [
    { duration: '5m', target: 100 }, // traffic ramp-up from 1 to 100 users over 5 minutes.
    { duration: '30m', target: 100 }, // stay at 100 users for 30 minutes
    { duration: '5m', target: 0 }, // ramp-down to 0 users
  ],
};
```

K6 options

- **Thresholds:** Thresholds are the **pass/fail criteria** that you define for your test metrics. If the performance of the system under test (SUT) does not meet the conditions of your threshold **(SLO)**, the test finishes with a failed status.

JavaScript

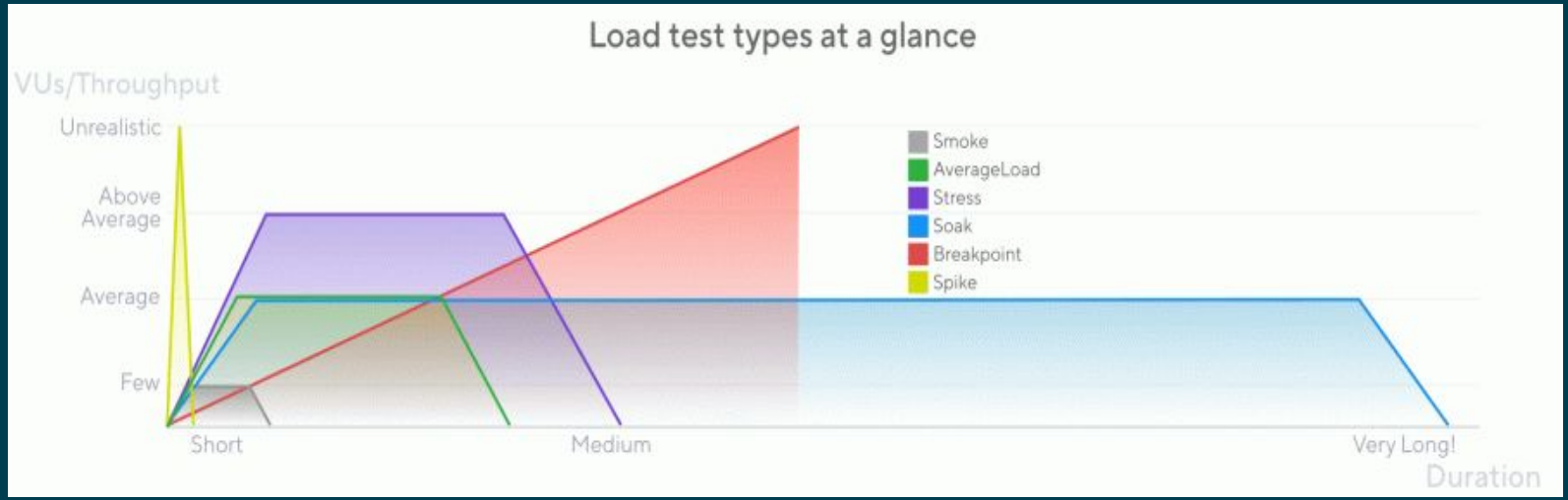
```
import http from 'k6/http';

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

export default function () {
  http.get('https://test-api.k6.io/public/crocodiles/1/');
}
```

K6 results visualization

Examples



/thoughtworks

Questions...?

Resources

- K6
  - https://k6.io
- Performance tests types
  - https://grafana.com/docs/k6/latest/testing-guides/test-types/
- Testing APIs
  - https://grafana.com/docs/k6/latest/testing-guides/api-load-testing/
- Thresholds
  - https://grafana.com/docs/k6/latest/using-k6/thresholds/
- Sandbox
  - https://test-api.k6.io/

Thanks!

Miguel Vela Romera
Lead developer
miguel.vela@thoughtworks.com

/thoughtworks