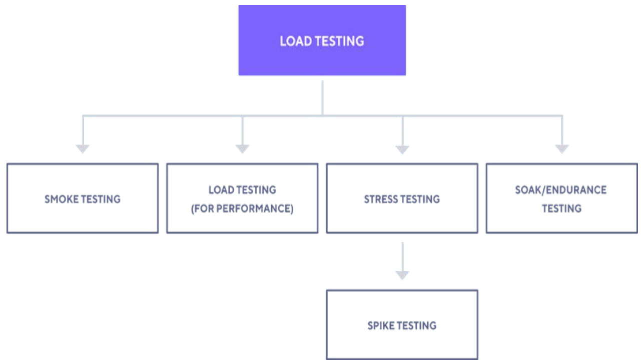
LOAD TESTING

Jozef Cipa, Backend Developer at STRV

STRV

WHAT IS LOAD TESTING?



- Generate artificial load against your backend / frontend
- Measure how the systems operates and reveal potential bottlenecks before releasing

- **Smoke testing**
  - Verify that your system can handle **minimal load**, without any problems

- **Load testing**
  - How the system behaves under high, **above-normal conditions**
  - In this case, it is important that the **system still responds** to all requests but the response time might no longer be the focus
  - *e.g. When 1000 users call the API within 30s seconds, the average response time should be below 1000 ms and no errors should occur*

- **Stress testing**
  - How the system behaves under **extreme conditions**, way above what should happen in normal scenarios
  - Essentially we are trying to find out the **breaking point**

- **Soak testing**
  - Assess reliability and performance of your system over an **extended period of time** (e.g. 2-3 hours)
  - Detect memory leaks or issues that appear after some time

TESTING TOOL

- Open-source tool **k6** (k6.io) - *"This is how load testing should look in the 21st century."*
- Written in Go, scripting in Javascript
  => No Node.js though, instead uses goja - JS interpreter written in Go

  => File imports don't work (need to use webpack or other bundler)

  => Node.js / Browser API not supported - e.g. `window` object, modules like `fs`, `os`, `crypto`, no EventLoop

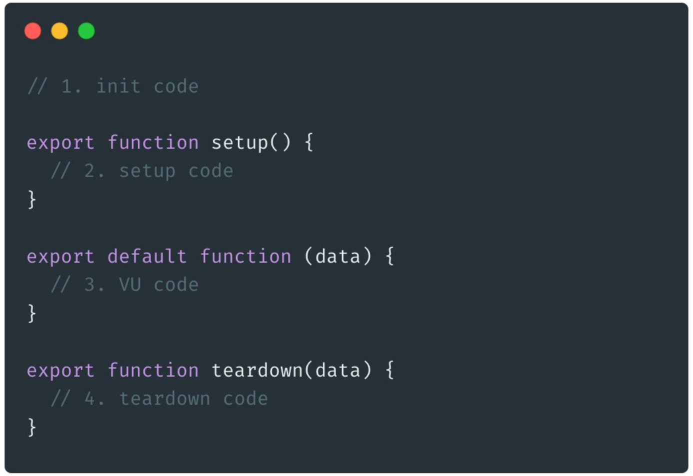
  => Provides custom utils like `open(file)`, `http.get(url)`

  => https://k6.io/docs/using-k6/javascript-compatibility-mode/

  => Very efficient and powerful ( )
- Many integrations (Grafana, InfluxDB, Cloudwatch, etc.)
- Easy to use
- Great documentation
- k6 cloud
- A lot of examples to get you started https://k6.io/docs/examples/

TESTING TOOL

- Virtual Users (VUs)
  - essentially parallel `while(true)` loops
  - Execute code repeatedly while the test is running
- Metrics and thresholds to define testing criteria



```
// 1. init code

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

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

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

STRV

METRICS & THRESHOLDS

- Measure how a system performs under test conditions
    - **Counters** - **Sum** values, e.g. number of requests
    - **Gauges** - Stores **min, max** and **latest** values, e.g. API response content size
    - **Rates** - Tracks **% of non-zero** values, e.g. % of failed requests
    - **Trends** - Calculates statistics (**min, max, average, percentiles**), e.g. API response time
- Built-in metrics
    - e.g. `http_req_duration`(trend), `iterations`(counter), `http_req_failed`(rate), …
- Possible to define and track custom metrics
- Thresholds allow to define pass/fail criteria for the metrics

TESTING SCENARIOS & MODELING THE WORKLOAD

- Scenarios allow to **model different traffic patterns**, thus simulate real traffic better
- Multiple scenarios can exist and may be executed in parallel or sequentially
- **Executors** are the workhorses of k6
  - Schedule VUs and iterations
  - Configured in the `options` object

Shared iterations - A fixed amount of iterations are "shared" between a number of VUs.

Per VU iterations - Each VU executes an exact number of iterations.

Constant VUs - A fixed number of VUs execute as many iterations as possible for a specified amount of time

Ramping VUs - A variable number of VUs execute as many iterations as possible for a specified amount of time

Constant Arrival Rate - A fixed number of iterations are executed in a specified period of time.

Ramping Arrival Rate - A variable number of iterations are executed in a specified period of time.

https://k6.io/docs/using-k6/scenarios/executors/

LARGE-SCALE TESTS

- k6 uses all CPU cores and manages memory very efficiently
- No need for distributed tests execution in most cases
- Single machine is often enough to generate 30-40k VUs (~300k requests per second)
- With some OS fine-tuning you can get even better results
- Don't forget to monitor the load generator server (memory, cpu, network)
- Simple tests will use ~1-5MB per VU
- `SharedArray`- share data between VUs (processes), otherwise each VU has its **own copy in memory**
- `discardResponseBodies`option, to avoid storing API responses in memory
- If distributed tests are needed, you can use execution segment, k6 cloud or the kubernetes operator

Be aware of data transfer costs in AWS!

AWS k6 Benchmark

https://k6.io/docs/testing-guides/running-large-tests/

https://k6.io/blog/comparing-best-open-source-load-testing-tools/

API PREREQUISITES

- Improve application logging
  - Generate **`requestId`** and attach it to all logs (using async hooks)
  - Return `requestId` (`correlationId`) in error responses
  - **Log request bodies** (only for failed requests to avoid bloating log stream, don't forget about redacting sensitive data)
- Enable Performance Insights on RDS
- Configure **reporting** (e.g. Cloudwatch alarms)
  - API & Lambda error logs
  - API CPU & memory utilization
  - SQS messages age
  - Redis memory utilization
  - ELB slow requests
- Install some **tracing** software (e.g. Sentry)
- Fix existing reported (known) issues first

https://k6.io/docs/testing-guides/api-load-testing/

RUNNER EC2

- Run tests against an environment that is the most similar to production (usually **staging,** or create production replica for testing)
- Prepare an EC2 instance in AWS
    - Install k6
    - Configure AWS Cloudwatch Agent
        - Send k6 logs and metrics (CPU, RAM, k6 metrics - VUs, delays, failures ...) to Cloudwatch
        - Beware of AWS custom metrics pricing - **0.30$ / metric / month** (k6 generated almost 90 metrics just in few seconds) *eventually we decided not to use it*
    - *Run tests*
- **Note:** k6 might generate a lot of traffic which may **increase the bill for data transfer !!!**

TEST RESULTS & LESSONS LEARNED

- **Load balancers are not magical**, they need to scale out as well (warm up)
- **OpenSearch** might be a bottleneck - do your **research and configure it properly** (scaling might be necessary too)
- **Fargate scaling** takes time (scaling events in AWS) - set scaling thresholds appropriately
- Use **caching** (don’t forget to invalidate, set proper keys to not return incorrect data)
- Look out for **inefficient database queries** (e.g. N+1 problem) - **Sentry** can help here
- Learned how to run load tests using k6

QUESTIONS?

STRV

THANK YOU!

Jozef Cipa / jozef.cipa@strv.com

STRV