Load testing

Stavanger Software Developers Meetup - 26. April 2022

CHIPMUNK
TECHNOLOGY

www.chipmunk.no

What goes through your mind when your boss, customer or client asks:

“You are absolutely confident that this can scale, right?”

Agenda

- Why load testing?
- Tools for load testing
- Common gotchas when load testing
- Demo-taim
    - Some preparations
    - Simple web server
    - Advanced web server with stable load
    - Advanced web server with exponential load
- Summary & questions

Why load testing?

A few questions that load testing can help us answer:

- What is the capacity of my system?
- Does my auto-scaling work as expected?
- How does the system behave when it’s overloaded?

A load test by itself cannot give you where the bottlenecks are, but it’s essential to be able to generate loads to introspect the systems to locate the bottlenecks and improve them!

Tools for load testing

There are (probably) hundreds of tools for load testing. Don’t overanalyze it until you are more experienced!

Here are some of the more popular ones:

- k6.io (Go)
- **Gatling (Java/Kotlin/Scala)**
- **Netling (.net)**
- **Locust (Python)**
- **jMeter (Java)**

Comparing features, pros and cons of each one is easily a two-day course.

Common gotcha when load testing:

Testing the load tester!

Are you load testing your target?

Or are you load testing your load testing tools and infrastructure?

Demo locust vs k6

Common gotcha when load testing:

Testing the load tester!

Are you load testing your target?

Or are you load testing your load testing tools and infrastructure?

Locust:
5700 requests/sec while pinning 1 CPU core to 100%
Web server: 83% CPU

K6:
32.000 requests/sec at 500% CPU
Web server: 230% CPU

No more talking.
Time for action.

Preparations

Installing tools

We're gonna use my favorite tool, k6.

It does not store metrics or visualize them. We'll use Grafana and InfluxDB for that.

```
git clone git@github.com:StianOvrevage/ssdm-april22-loadtesting.git

bash installtools.sh

bash starttools.sh
```

Open grafana at http://localhost:3000 and log in with admin/admin. Import the dashboard with ID 4411

Simple web server

This is a very simple web server that answers requests as fast as possible.

Console window 1 - resource monitoring

```
top
```

Console window 2 - grafana & influxdb

```
bash starttools.sh
```

Console window 3 - load target

```
cd webserver
go run main-simple.go
```

Console window 4 - load test

```
k6 run --out influxdb=http://localhost:8086/myk6db --vus
60 --duration 20s k6/script-advanced.js
```

Advanced web server

Can only handle 200 requests at a time.

Each request takes a random time within a normal distribution.

If server is busy requests have a 10% chance of failing and 90% chance of waiting.

60 concurrent users

Console window 3 - load target

```
go run main-advanced.go
```

Console window 4 - load test

```
k6 run --out influxdb=http://localhost:8086/myk6db --vus
60 --duration 20s k6/script-advanced.js
```

Advanced web server

Can only handle 200 requests at a time.

Each request takes a random time within a normal distribution.

If server is busy requests have a 10% chance of failing and 90% chance of waiting.

Increasing load exponentially from 20 to 1280 concurrent users over 60 seconds.

Console window 4 - load test

```
k6 run --out influxdb=http://localhost:8086/myk6db
k6/script-advanced.js
```

When a system is congested there are only two outcomes:

- Increased latency
- Increased failure rate

Questions?

Summary:

- Load testing is very easy to get started with
- Ensure you are testing what you think you are testing
- Distributed systems might not behave like you expect

Thanks for listening!

I'm Stian Øvrevåge and I'm a consultant at Chipmunk Technology working passionately on all things Cloud, Kubernetes, DevOps, SRE.

Reach me at stian@chipmunk.no or by the bar ;)

Presentation and code:
github.com/StianOvrevage/ssdm-april22-loadtesting