EN
English

Universidad de Oviedo

School of Computer Science

SOFTWARE ARCHITECTURE

Software Architecture

Lab. 11

Load testing

Other tests…

2018-19

Jose Emilio Labra Gayo + Víctor Álvarez
victoralvarez@uniovi.es

What are load tests?

Measure performance under normal or anticipated peak load conditions

- Example: Several concurrent users

Goal: Anticipate possible failures

- verify work load of some system

What can we test

Web applications (Http/https)
SOAP/REST Web Services
FTP
Databases (JDBC)
LDAP
Mail (SMTP, POP3, IMAP)
Java Objects
Etc.

Why should we do load tests?

Anticipate performance problems

Detect bottlenecks

Prove quality attributes

Load testing tools

Gatling

Apache Jmeter ()

Locust.io (http://locust.io/)

Artillery.io ()

goReplay

Loader.io

BlazeMeter

Blitz …

Step by step guide:

https://github.com/pglez82/docker_solid_example/tree/pglez82-gattling-load-tests#load-tests-gatling

Gatling

Written in Scala

JVM compatible

Embedded DSL for testing

Easy to use

Light

Download & installation

http://gatling.io

- It needs Java 8 installed

2 scripts:

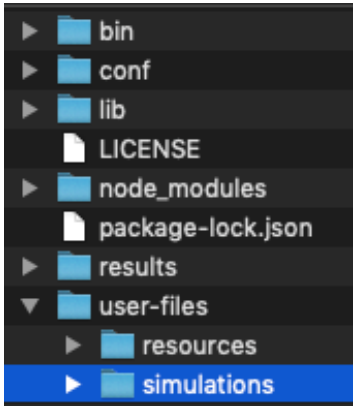
- Recorder.sh/Recorder.bat
- Gatling.sh/Gatling.bat

Gatling

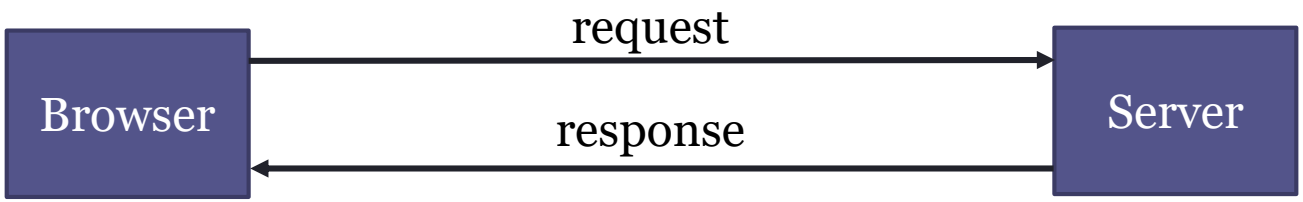
Install

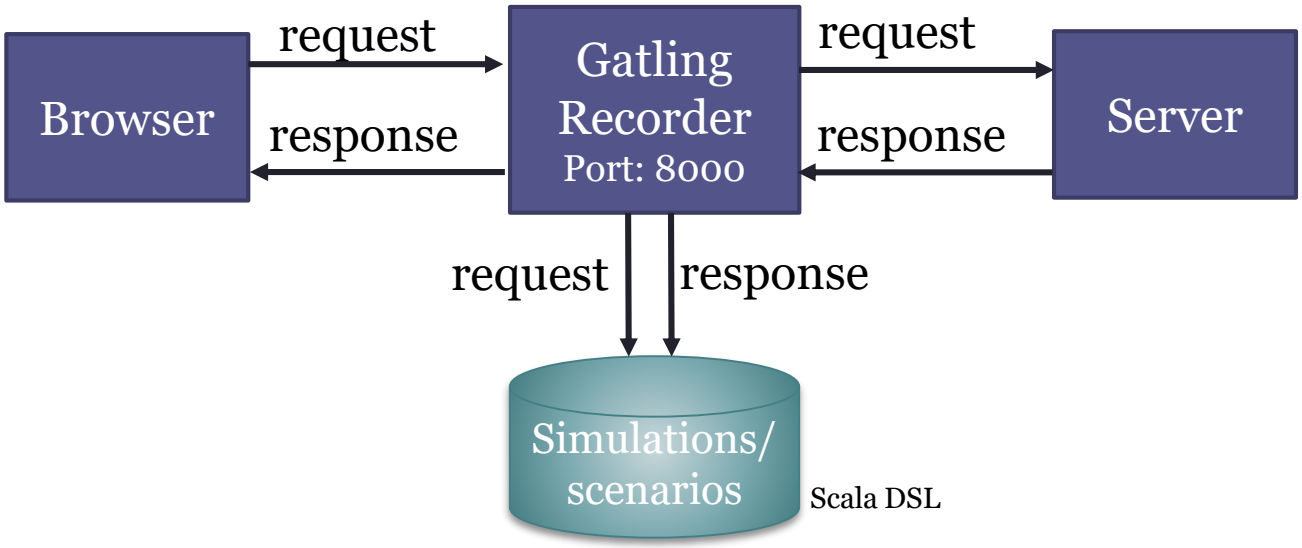
- Scala (programming language)
  - Plugin for IntelliJ IDEA
- Gatling
  - Download from https://gatling.io/download/
    - /usr/*local*/*Cellar*/gatling/*gatling-charts-highcharts-bundle-X.X.X*

  - **gatlingjs:** npm library to run gatling from node.js
    - Directory Structure

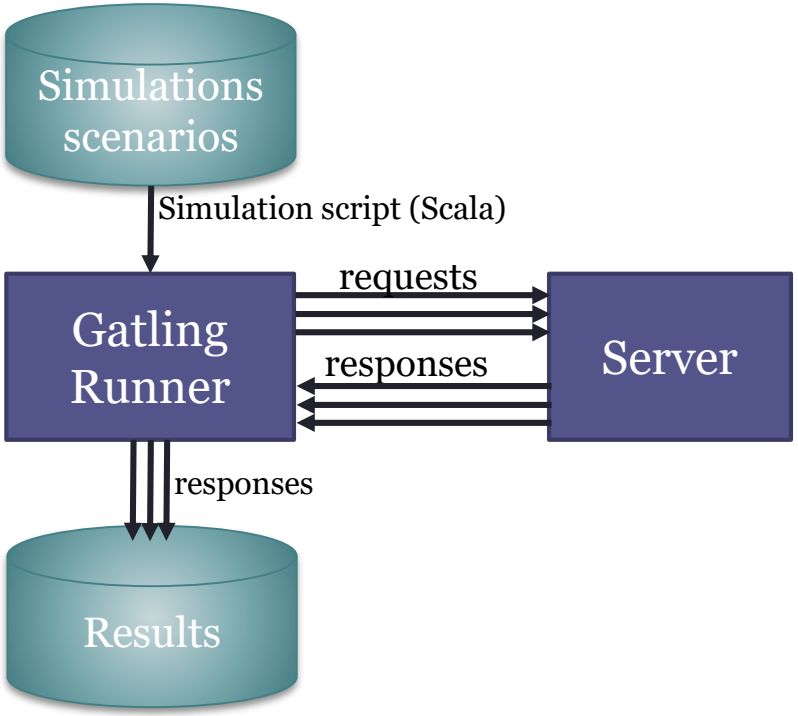


Gatling recorder

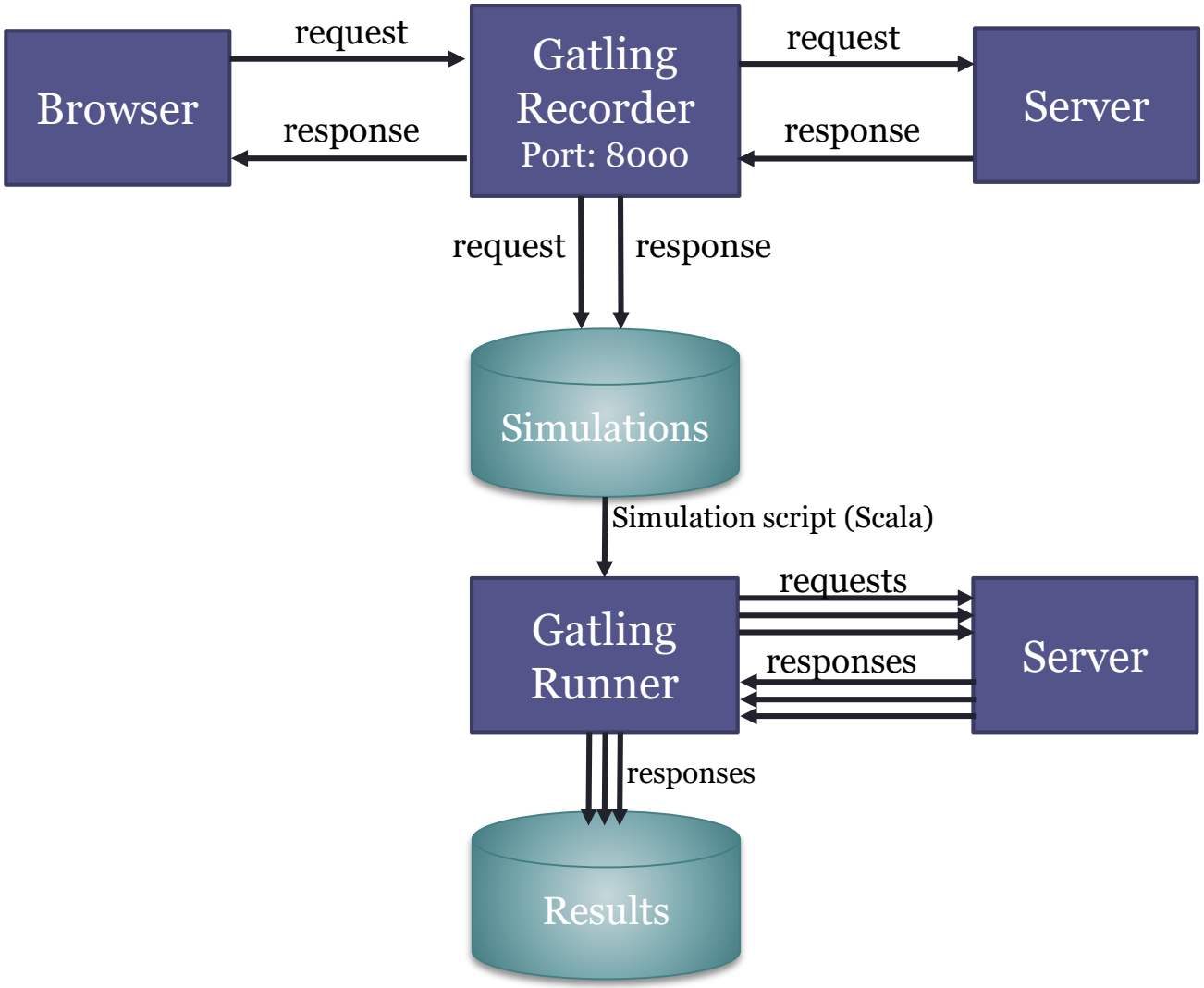




Gatling runner



Workflow


Browser
request
response
Gatling
Recorder
Port: 8000
request
response
Server
request
response
Simulations
Simulation script (Scala)
Gatling
Runner
requests
responses
Server
responses
Results

Recorder

Gatling Recorder - Configuration

Recorder mode: HTTP Proxy

Network

Listening port*: localhost HTTP/HTTPS 8888 HTTPS mode: Self-signed Certificate

Outgoing proxy: host: HTTP HTTPS Username Password

Simulation Information

Package: es.uniovi Class Name*: VotingSystem

Follow Redirects? Infer html resources? Automatic Referers?

Remove cache headers? Save & check response bodies?

Output

Output folder*: /Users/herminio/Downloads/gatling-charts-highcharts-bundle-2.2.4/user-files/simulations Browse

Encoding: Unicode (UTF-8)

Filters

Java regular expressions that matches the entire URI Strategy Disabled

Whitelist: + - Clear

Blacklist: + - Clear No static resources

Gatling: Recorder

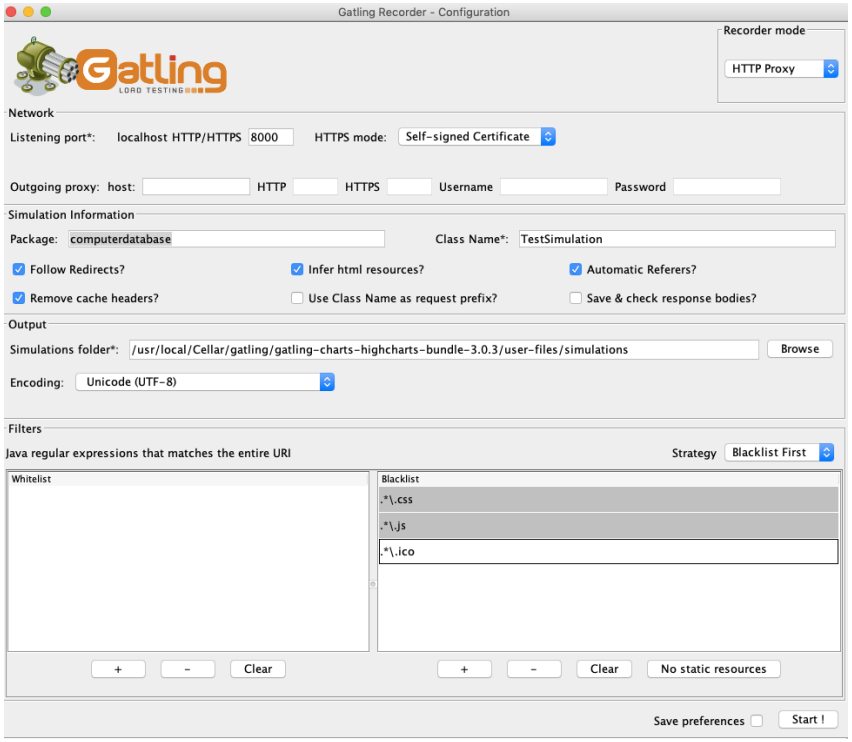
Test case: http://computer-database.gatling.io/computers

Launch recorder

```
✔ /usr/local/Cellar/gatling/gatling-charts-highcharts-bundle-3.0.3/bin [master L|•1]
11:57 $ ./recorder.sh
GATLING_HOME is set to /usr/local/Cellar/gatling/gatling-charts-highcharts-bundle-3.0.3
```

Recorder setup

1. Package: computerdatabase
2. Name: TestSimulation
3. Follow Redirects ✓
4. Automatic Referers ✓
5. Strategy: Black list first
6. Blacklist: .*\.css, .*\.js and .*\.ico



Configure Proxy

`localhost:8000`

- For all addresses, inclluded localhost
  - In case of HTTPS, the certificate must be configured
- Start the proxy

For localhost in firefox, set:
`network.proxy.allow_hijacking_localhost` to true in `about:config`

Gatling: Recorder

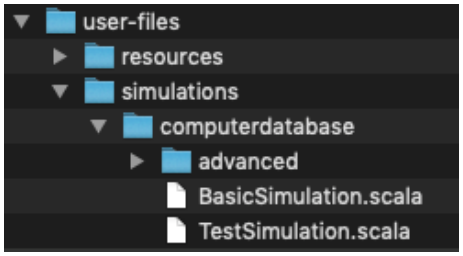
Browser > Web Proxy > localhost:8000

Recorder: Start

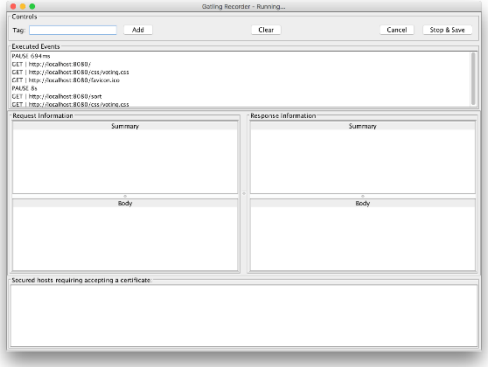
Sample scenario:

1. The user arrives at the application. Opens http://computer-database.gatling.io/computers
2. The user searches for ‘macbook’.
3. The user opens one of the related models.
4. The user goes back to home page.
5. The user browsers through pages.
6. The user creates a new computer model.

Recorder: Stop



New Scala script



Escenario

Definition & headers:

- Package
- Imports
- Class [extends Simulation]
- HTTP & headers

Scenario:

- Definition
- Requests
- Pauses
- **Inject**

TestSimulation.scala

```
package computerdatabase

import scala.concurrent.duration._

import io.gatling.core.Predef._
import io.gatling.http.Predef._
import io.gatling.jdbc.Predef._

class TestSimulation extends Simulation {

        val httpProtocol = http
                .baseUrl("http://computer-database.gatling.io")
                .inferHtmlResources(BlackList("""".*\.css""", """.*\.js""", """.*\.ico"""), WhiteList())
                .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8")
                .acceptEncodingHeader("gzip, deflate")
                .acceptLanguageHeader("en,en-US;q=0.9,es;q=0.8,pt;q=0.7,de;q=0.6")
                .upgradeInsecureRequestsHeader("1")
                .userAgentHeader("Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/7

        val headers_0 = Map("Proxy-Connection" -> "keep-alive")

        val headers_7 = Map(
                "Origin" -> "http://computer-database.gatling.io",
                "Proxy-Connection" -> "keep-alive")


        val scn = scenario("TestSimulation")
                .exec(http("request_0")
                        .get("/computers")
                        .headers(headers_0))
                .pause(35)
                .exec(http("request_1")
                        .get("/computers?f=macbook")
                        .headers(headers_0))
                .pause(15)
                .exec(http("request_2")
                        .get("/computers/517")
                        .headers(headers_0))
                .pause(18)
                .exec(http("request_3")
                        .get("/")
                        .headers(headers_0))
                .pause(6)
                .exec(http("request_4")
                        .get("/computers?p=1")
                        .headers(headers_0))
                .pause(2)
                .exec(http("request_5")
                        .get("/computers?p=2")
                        .headers(headers_0))
                .pause(4)
                .exec(http("request_6")
                        .get("/computers/new")
                        .headers(headers_0))
                .pause(40)
                .exec(http("request_7")
                        .post("/computers")
                        .headers(headers_7)
                        .formParam("name", "Atari (test)")
                        .formParam("introduced", "2002-05-28")
                        .formParam("discontinued", "2004-08-13")
                        .formParam("company", "2"))

        setUp(scn.inject(atOnceUsers(1))).protocols(httpProtocol)
}
```

How-to configure the number of users…

Injection profile

Control how users are injected in your scenario

Injection steps

- nothingFor
- atOnceUsers
- rampUsers
- constantUsersPerSec
- rampUsersPerSec
- splitUsers
- heavisideUsers

50 users during 60 seconds

50 simultaneous users

A new user enters every 1.2 seconds

They execute a given script

```
...
setUp(scn.inject(rampUsers(50) during(60 seconds))).
    protocols(httpProtocol)

}
```

Triggering Gatling

- Run script: `gatling.sh/.bat`
  - choose the class with the previous script
  - Configure ID and description
- In the execution we can see the textual progress
- At the end, an HTML file is generated
  - It contains graphical load test analysis

Triggering Gatling

Run Gatling (/bin/gatling.sh) and choose the scenario

```
20:39 $ ./gatling.sh
GATLING_HOME is set to /usr/local/Cellar/gatling/gatling-charts-highcharts-bundle-3.0.3
Choose a simulation number:
     [0] computerdatabase.BasicSimulation
     [1] computerdatabase.TestSimulation
     [2] computerdatabase.advanced.AdvancedSimulationStep01
     [3] computerdatabase.advanced.AdvancedSimulationStep02
     [4] computerdatabase.advanced.AdvancedSimulationStep03
     [5] computerdatabase.advanced.AdvancedSimulationStep04
     [6] computerdatabase.advanced.AdvancedSimulationStep05
1
Select run description (optional)
my test
Simulation computerdatabase.TestSimulation started...
```

Simulation output

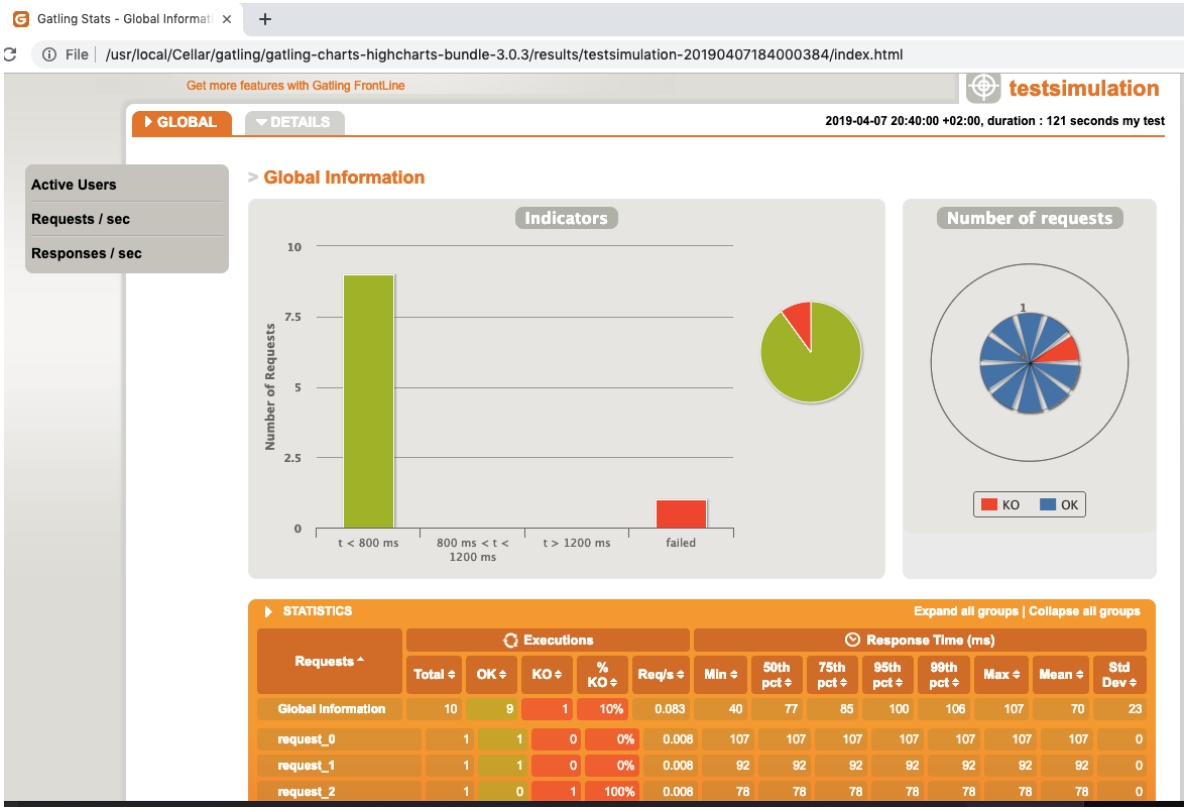
```
---- Requests ------------------------------------------------------------------
> Global                                                   (OK=1      KO=0     )
> request_0                                                (OK=1      KO=0     )

---- TestSimulation ------------------------------------------------------------
[--------------------------------------------------------------------------------]  0%
          waiting: 0      / active: 1      / done: 0
================================================================================
```

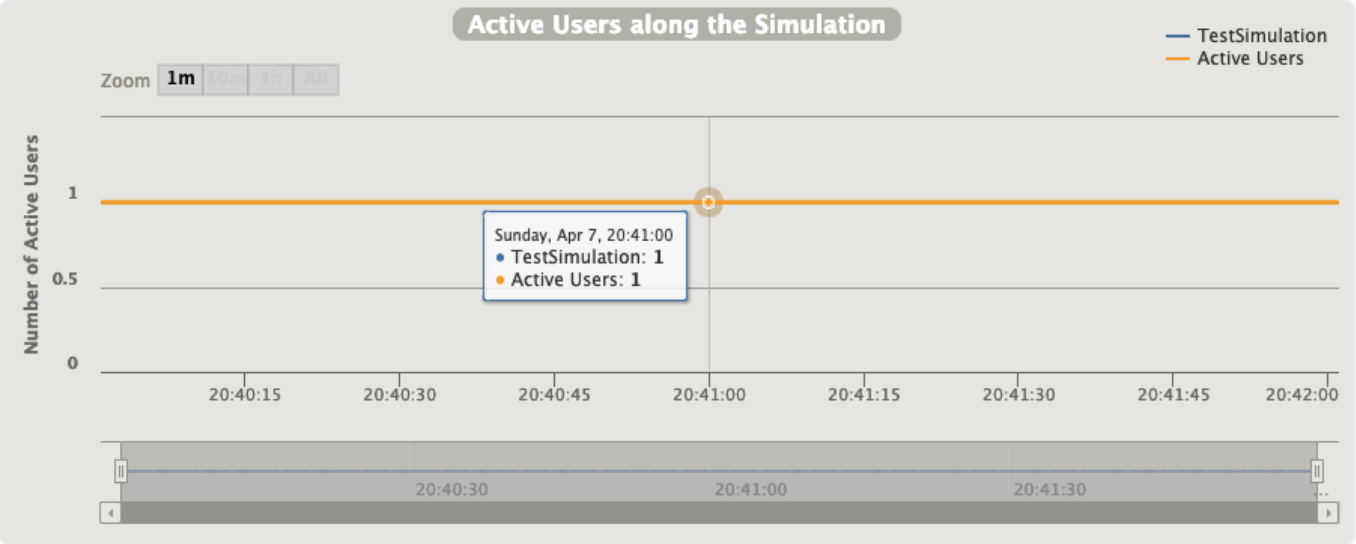
Result

```
Reports generated in 0s.
Please open the following file: /usr/local/Cellar/gatling/gatling-charts-highcharts-bundle-3.0.3/results/testsimulation-20190
407184000384/index.html
```

Gatling: HTML Report

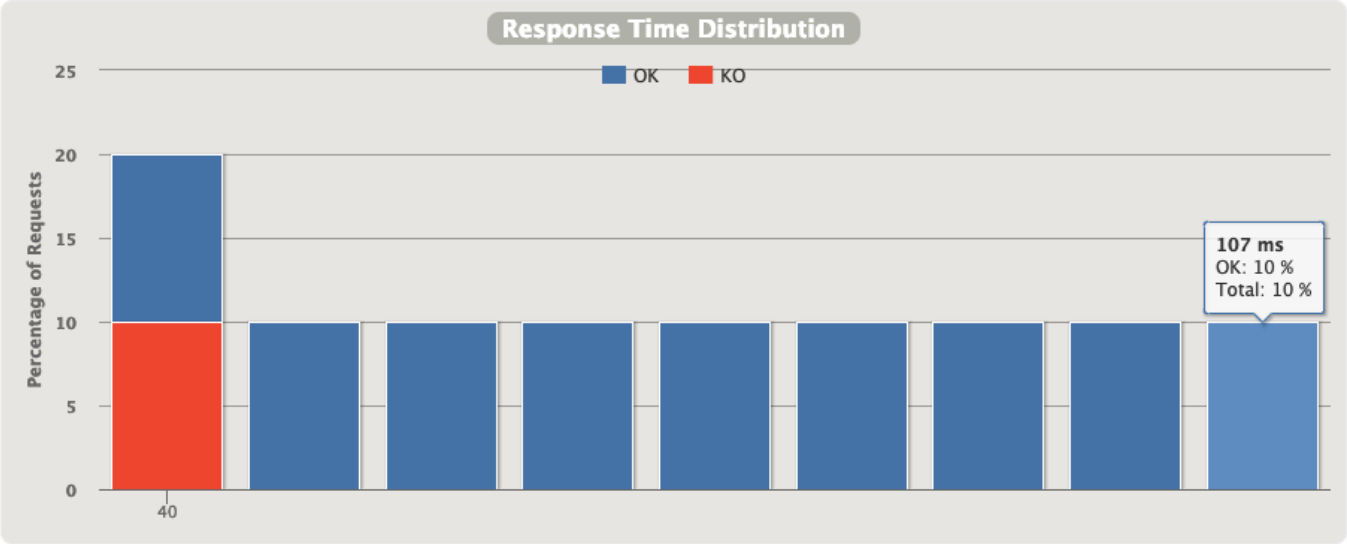


Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	10	9	1	10%	0.083	40	77	85	100	106	107	70	23
request_0	1	1	0	0%	0.008	107	107	107	107	107	107	107	0
request_1	1	1	0	0%	0.008	92	92	92	92	92	92	92	0
request_2	1	0	1	100%	0.008	78	78	78	78	78	78	78	0



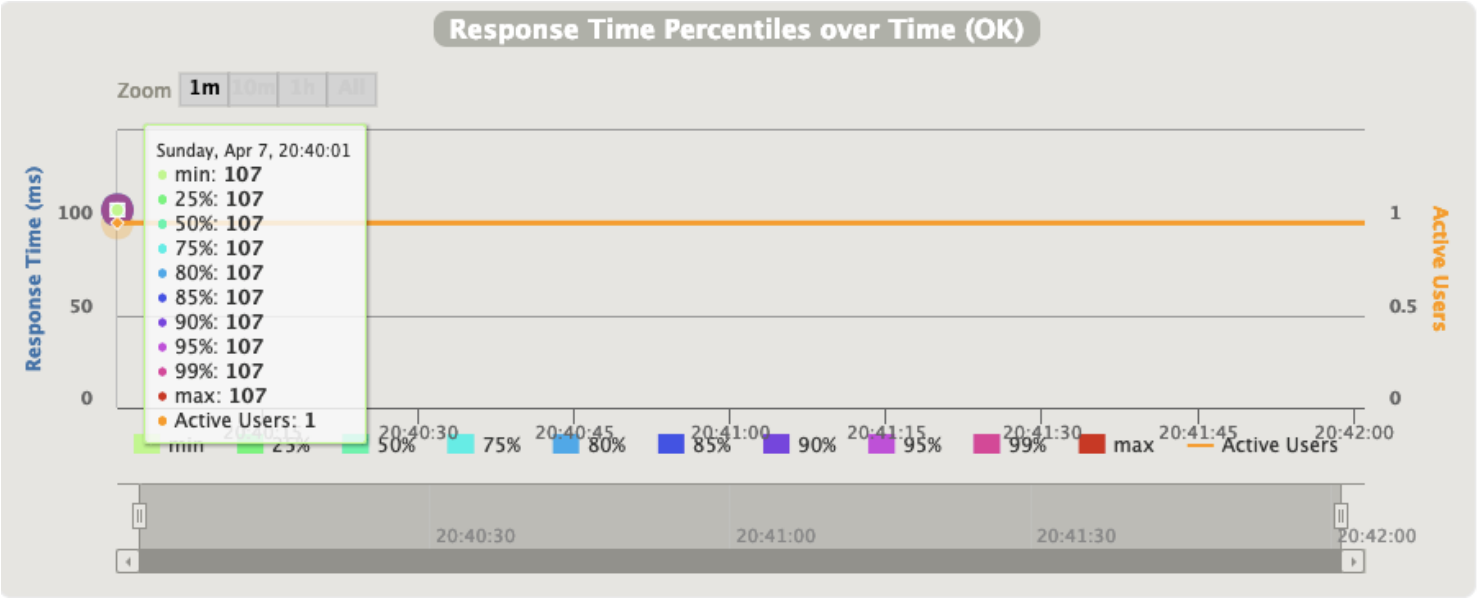
Active Users along the Simulation

It displays the number of active users (sending requests and receiving responses) along the simulation time. This measure can be related to others such as response times and number of requests-Se puede relaccionar con otras medidas como los tiempos de respuesta y el número de peticiones/respuestas por segundo.



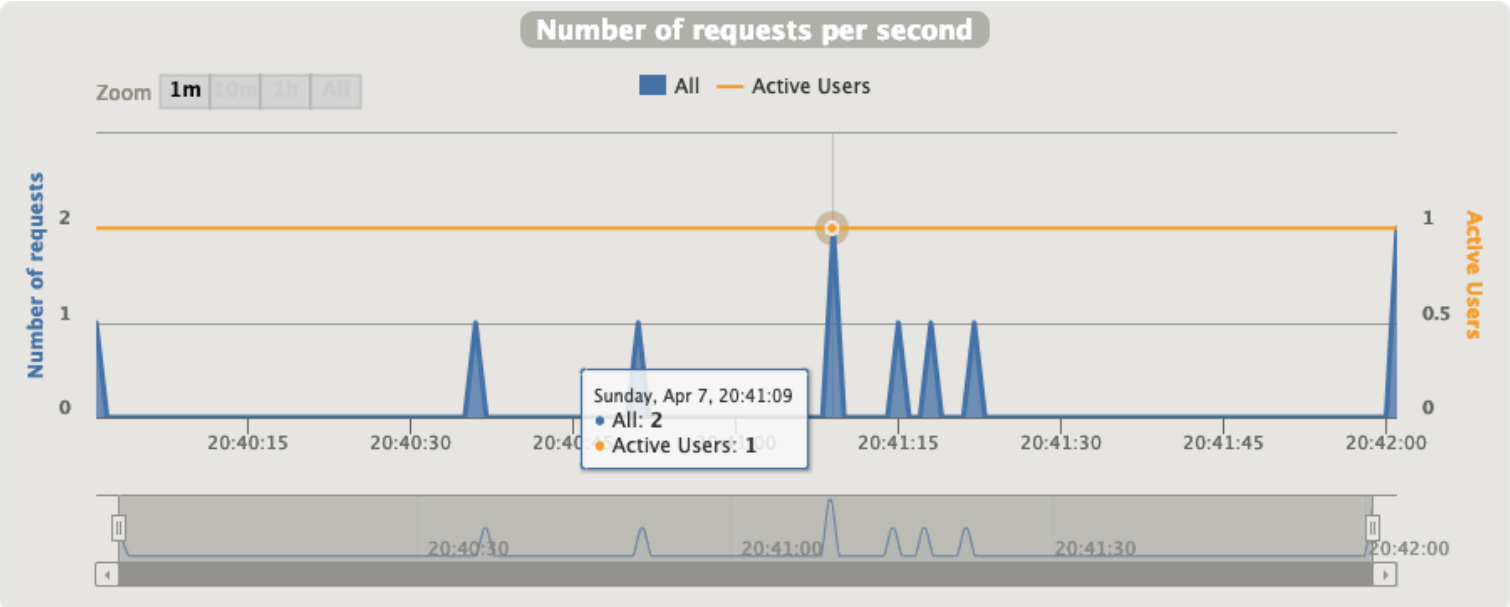
Response Time Distribution

This chart shows you the percentage of all requests made during your test run on the Y axis. It will include both successes and failures. All of the Y values should add up to 100%. The response time (the time it takes to request the page and send data back to the server to acknowledge you received it) is on the x axis. As you increase load on the server, you should see the data on this chart move farther to the right (response times will get slower).



Response Time Percentiles over Time

This is similar to Response Time Distribution, but it shows you the data over a longer period of time to assess how your system behaves when under a sustained load. For example, 200 users accessing various web pages over the course of 5 minutes.



Requests/responses per second

The number of times you make a request for a resource from the server per second. For example, if you simulate 200 users accessing one file on a server all at the same time once a second, you'll have 200 requests/responses per second.

Repository for tests

http://www.github.com/arquisoft/bddExample

Start application (`mvn spring-boot:run`)

Pages:

- Landing page
- Search a given name
  - It executes some random computation for any name
  - If name = "long" the computation is very long
  - If name = "error" throws an exception

DSL

Gatling uses a script written in an Scala embedded DSL

The scripts are located at:

user-files/simulations/...

It is possible to edit it for configuration

Documentation about DSL at:

http://gatling.io/docs/current/

Note.
It is possible to open it in IntelliJ/Scala IDE for code-completion
BUT...it may require some Scala knowledge

Example

```
package es.uniovi.asw
import scala.concurrent.duration._
import io.gatling.core.Predef._
import io.gatling.http.Predef._

class Bddxample extends Simulation {

 val httpConf = http.baseURL("http://localhost:8080")

 val scn = scenario("BddExample").
     exec(http("Root").get("/")).
     pause(3).
     exec(http("Search pepe").get("/search?name=pepe")).
     pause(3).
     exec(http("Search long").get("/search?name=long")).
     pause(3).
     exec(http("Search error").get("/search?name=error"))

 setUp(scn.inject(rampUsers(50) over(60 seconds))).
 protocols(httpConf)
}
```

Gatling concepts & DSL

- Simulation: Description of a load test
  - Defines method `setUp`
  - Scenario: Represents users' behaviours
    - It is possible to inject users to scenarios
      - Several possibilities:
        - nothingFor
        - atOnceUsers
        - rampUsers
        - constantUsersPerSec
        - ...
  - Protocols: set protocol definitions (usually http)
  - Assertions: Verify some statistics
    - Can be used for continuous integration

Other tests

Usability

Allow to determine if a given application is easy to use. They assess users´ experience before (formative) and after (summative) the release of a given software.

Among the measures they can provide:

- Ease of learning and memorising
- Precision and completeness
- Efficiency and productivity (time spent to perform a task)
- Errors
- Satisfation
- Accesibility

Testing techniques include observation, benchmarking, surveys, interviews, questionnaires, eye-tracking..

Otras pruebas

Security

Allow measuring the level of security.
Ethical Hacking
Vulnerability reports and possible solutions

Open source: Wapiti, Zed Attack Proxy, Vega, W3af, Skipfish, Ratproxy, SQLMap, Wfuzz, Grendel-Scan, Arachni, Grabber.

Scalability, maintainability, portability..

Links

Gatling https://gatling.io/

The Art of Destroying Your Web App With Gatling
https://gatling.io/2018/03/07/the-art-of-destroying-your-web-app/

The Scala Programming Language
https://www.scala-lang.org/

Refactoring (Advanced Gatling-Scala)
https://gatling.io/docs/2.3/advanced_tutorial#advanced-tutorial
https://github.com/gatling/gatling/tree/master/gatling-bundle/src/main/scala/computerdatabase

Testing Node.Js Application with Gatling
https://blog.knoldus.com/testing-node-js-application-with-gatling/

Other tests

Types of software testing
https://www.softwaretestinghelp.com/types-of-software-testing/

Qué son: Pruebas de usabilidad (Andrea Cantú)
https://blog.acantu.com/que-son-pruebas-usabilidad/

An overview on usability testing & 6 tools to automate it
https://www.cubettech.com/blog/an-overview-on-usability-testing-6-tools-to-automate-it/

¨Solución automatizada de pruebas de penetración y auditoría de seguridad para entornos de prestación de servicios empresariales en Cloud¨ David Lorenzo González, Trabajo fin de Grado (Universidad de Oviedo)