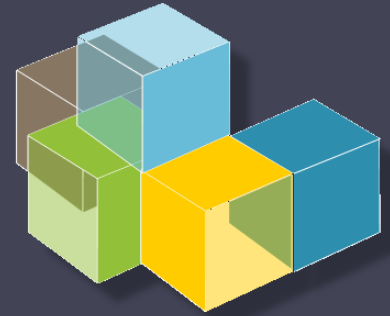


Universidad de Oviedo



Escuela de
Ingeniería
Informática



ARQUITECTURA
DEL SOFTWARE

Arquitectura del Software

Lab. 11

Pruebas de carga

Otras pruebas

2019-2020

Jose Emilio Labra Gayo
Pablo González
Irene Cid
Hugo Lebrede

¿Qué son?

- Son pruebas que permiten probar con cargas de usuarios concurrentes
- Permiten saber que carga de trabajo soporta una aplicación y arquitectura determinada
- Nos ayudan a dimensionar la aplicación y poder anticiparnos antes de su caída



¿Qué permiten probar?

- Aplicaciones web (HTTP/HTTPS)
- SOAP/REST Web Services
- FTP
- Databases (JDBC)
- LDAP
- Mail (SMTP, POP3, IMAP)
- Java Objects
- Etc.

¿Por qué hacer estos test?

- Permiten anticiparnos a problemas de rendimiento en la aplicación, arquitectura o infraestructura
- Permiten detectar cuellos de botella
- Permiten demostrar numéricamente los escenarios de calidad pactados en el contrato

Variables externas

- Red
- Servicios externos
- Sistemas de seguridad
- Eventos del sistema

¿Probamos en producción?

- Recomendación : Nuestras medidas deben ser para entornos de producción en vacío.

Planes de carga

- Dependiendo si se conoce o no el número de usuarios esperados.
- Variables que hay que llegar a conocer :Límite en el que nuestra aplicación deja de funcionar.
- Pruebas de picos: nos permiten ver la recuperación de nuestro sistema ante estados próximos a la saturación
- Pruebas de resistencia: Se prueba con una carga de 50-70% durante un tiempo largo.

Formas de hacer estos test

- Scripts propios
 - Mucho trabajo
 - Poco flexible
 - Analíticas pobres
 - Curva de aprendizaje corta
- Herramientas de terceros
 - Menos trabajo
 - Más flexibles
 - Mejores analíticas
 - Curva de aprendizaje elevada (según herramienta)

Herramientas

- Apache JMeter
- **Gatling**
- Loader.io
- BlazeMeter
- Blitz
- Etc.

!!!Vamos a aprender a usar el Gatling!!!

Gatling

- Escrita en Scala
- Compatible con la JVM
- Uso de un DSL propio
- Fácil de usar
- Ligera

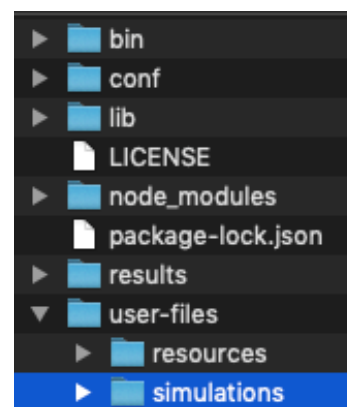


Descarga e instalación

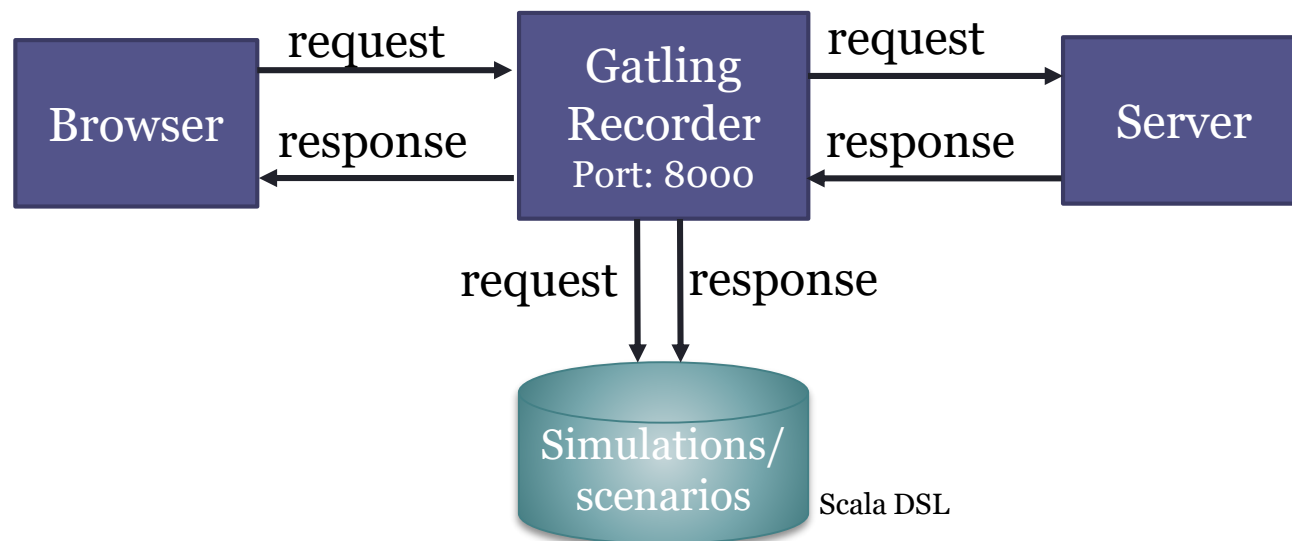
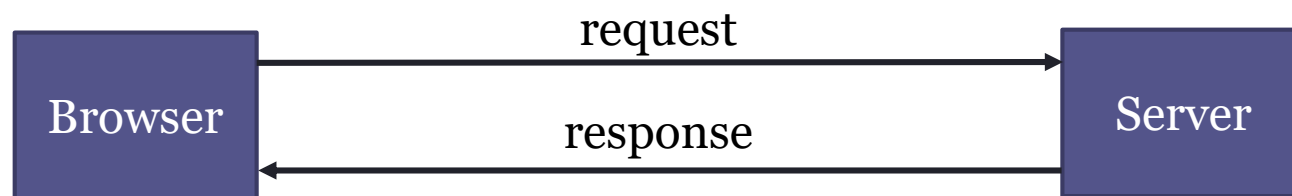
- <http://gatling.io>
- Necesita tener Java instalado (hecho en Scala)
- Listo para funcionar
- Dos scripts:
 - Recorder.sh/Recorder.bat
 - Gatling.sh/Gatling.bat

Gatling

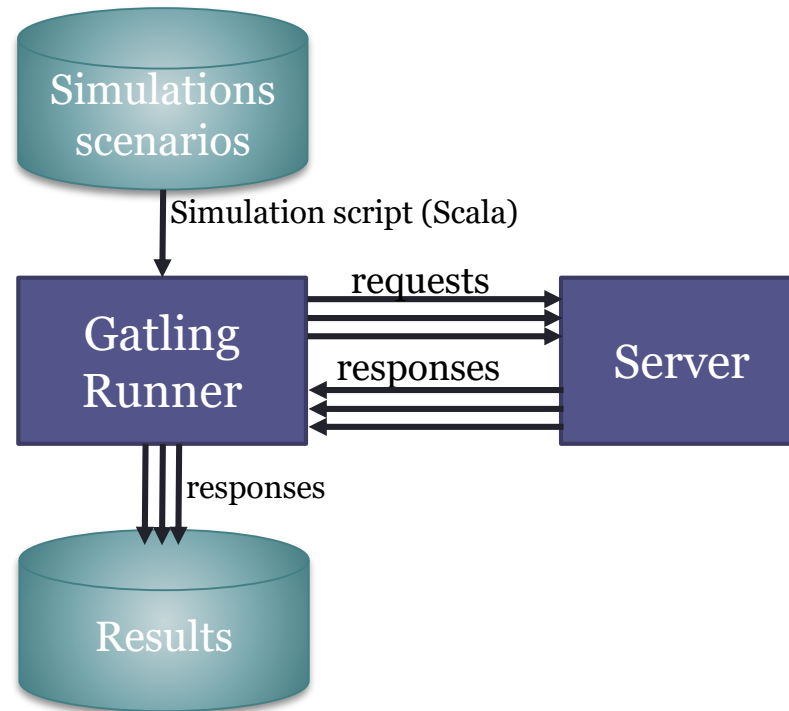
- Instalación
 - Scala (lenguaje de programación)
 - Plugin para IntelliJ IDEA
 - Gatling
 - Descargar desde <https://gatling.io/download/>
 - `/usr/local/Cellar/gatling/gatling-charts-highcharts-bundle-X.X.X`
 - **gatlingjs**: npm library to run gatling from node.js
 - Estructura de directorios



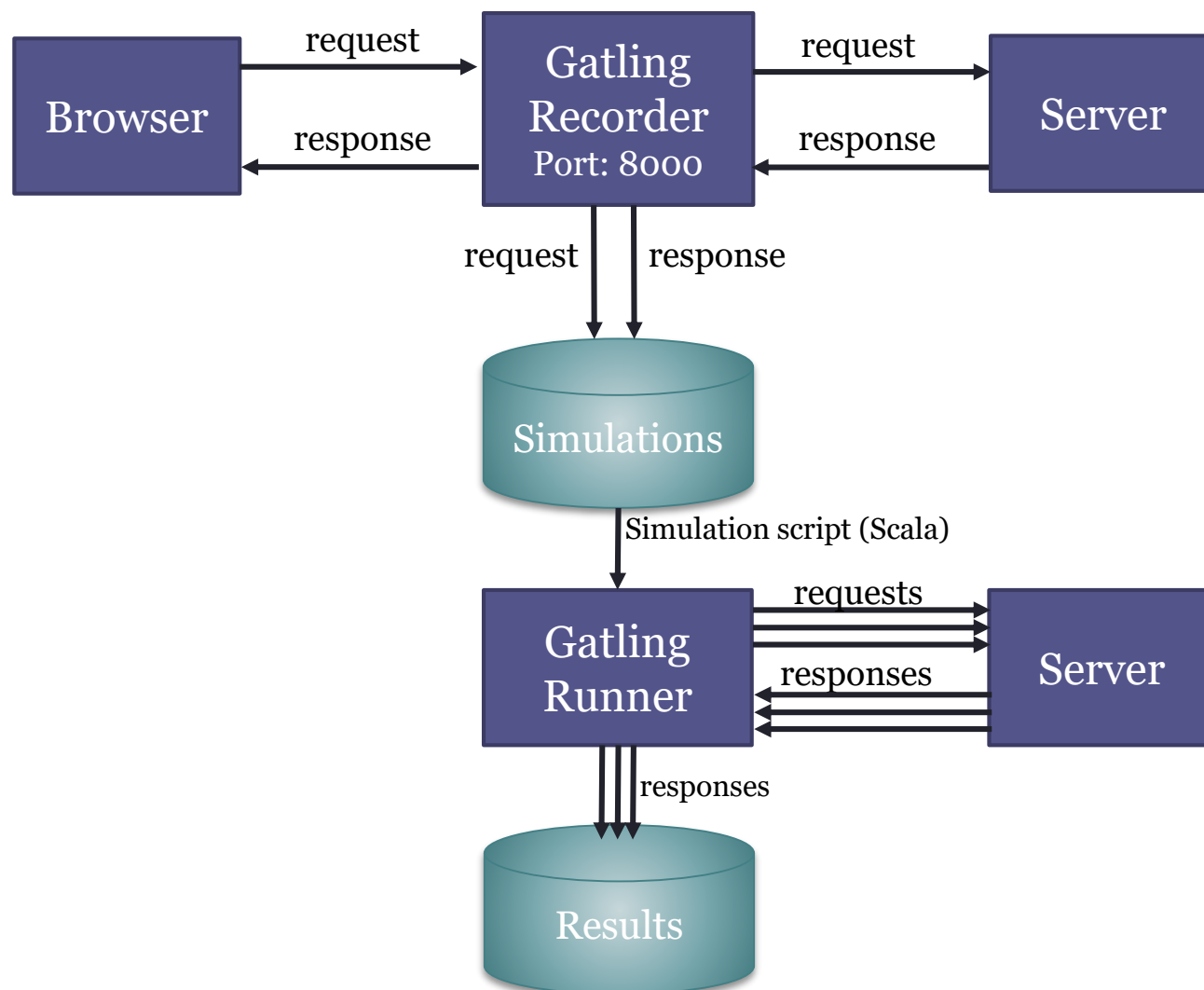
Gatling recorder




Gatling runner



Workflow



Recorder



Recorder mode

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

Blacklist

+

-

Clear

+

-

Clear

No static resources

Save preferences ☒

Start !

Gatling: Recorder

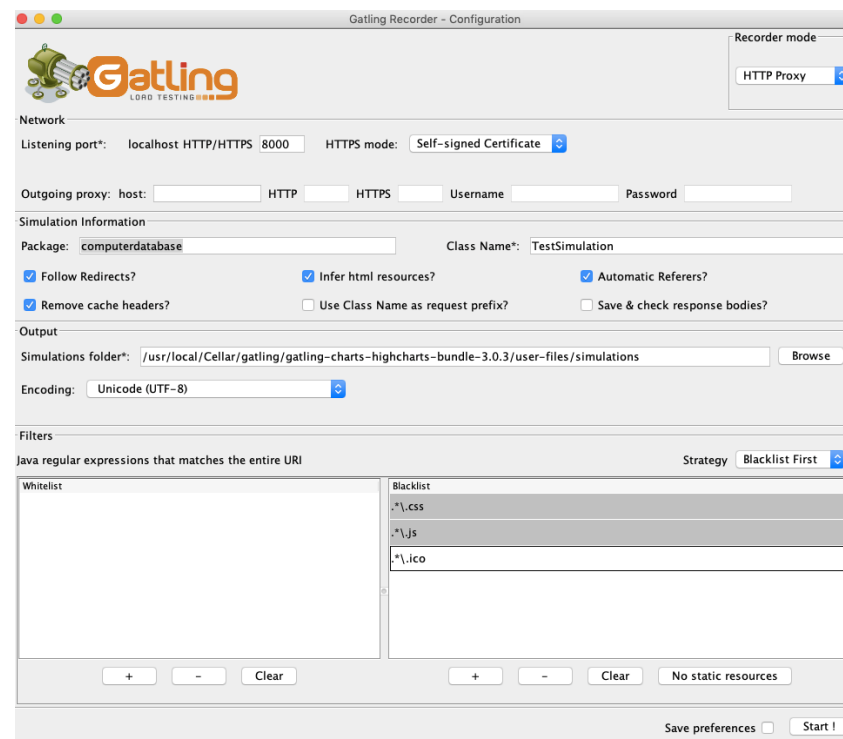
Test case: <http://computer-database.gatling.io/computers>

- Abrir el 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
```

- Configurar el recorder

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

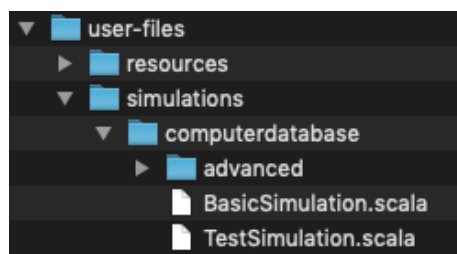


Configurar proxy

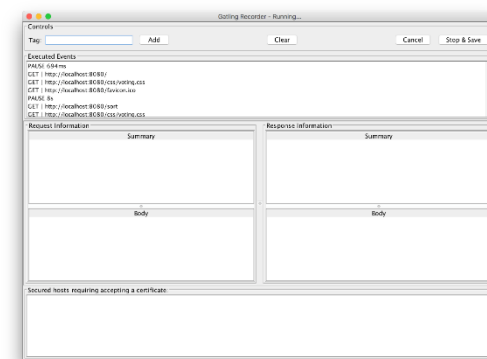
- localhost:8000
- Para todas las direcciones incluida localhost
- Si se usa HTTPS hay que configurar el certificado
- Arrancar el proxy
- Navegar tal como lo haría un usuario

Gatling: Recorder

- Navegador > Web Proxy > localhost:8000
- Recorder: Start
- Escenario de ejemplo:
 1. Abrir <http://computer-database.gatling.io/computers>
 2. Buscar 'macbook'
 3. Abrir uno de los modelos
 4. Volver a la página principal
 5. Navegar por los contenidos
 6. Añadir un nuevo modelo
- Recorder: Stop



Nuevo script en Scala



Escenario

Definición y cabeceras:

- Paquete
- Imports
- Clase [extends Simulation]
- Configuración HTTP y headers

Escenario:

- Definición
- Requests
- Pauses
- **Inject**

```

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/71.0.3538.102 Safari/537.36")

    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("/")
            .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))
}

```

TestSimulation.scala

Configurando el número de usuarios

Injection profile

Control how users are injected in your scenario

Injection steps

nothingFor

atOnceUsers

rampUsers

constantUsersPerSec

rampUsersPerSec

splitUsers

heavisideUsers

https://gatling.io/docs/current/general/simulation_setup

Configurando la conexión

- Caching
- HTTP Headers
- Authentication
- Protocols
- Max connection per host

https://gatling.io/docs/current/http/http_protocol/#max-connection-per-host

50 usuarios progresivos en 60 segundos

- 50 usuarios concurrentes
- Entra un usuario nuevo cada 1,2 segundos
- Desarrollan todo el script grabado anteriormente

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

Disparando el Gatling

- Script: `gatling.sh/.bat`
- Escogemos la clase con el script grabado previamente
- Podemos configurar el ID y la descripción
- En la ejecución vamos viendo un progreso textual de la prueba
- Al finalizar genera un informe con analíticas y gráficas en un fichero HTML

Disparando el Gatling

- Ejecutar Gatling (/bin/gatling.sh) y escoger el escenario

```
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...
```

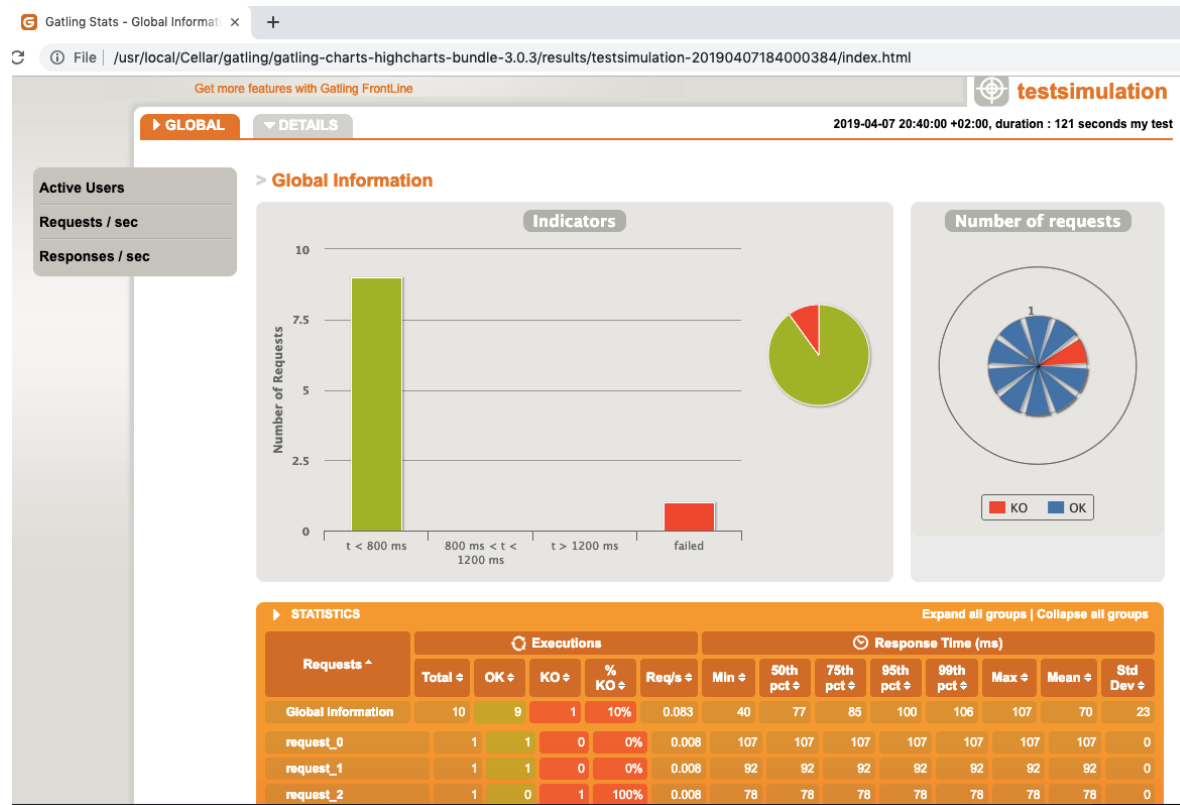
- Simulación

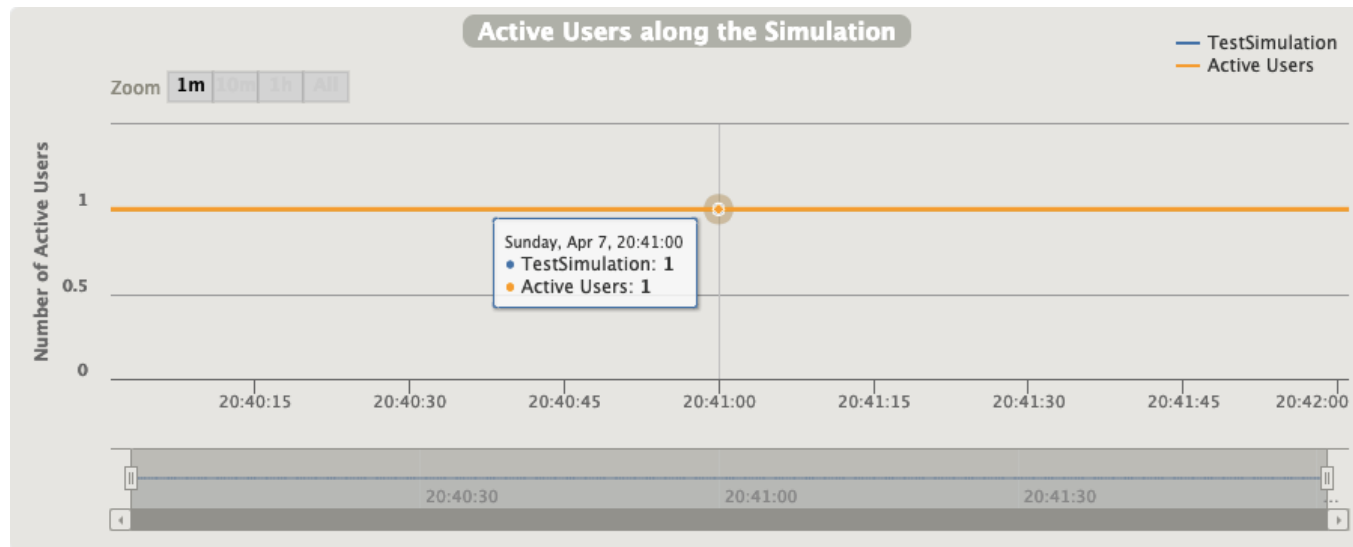
```
----- Requests -----
> Global                      (OK=1    KO=0    )
> request_0                   (OK=1    KO=0    )

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

- Resultado

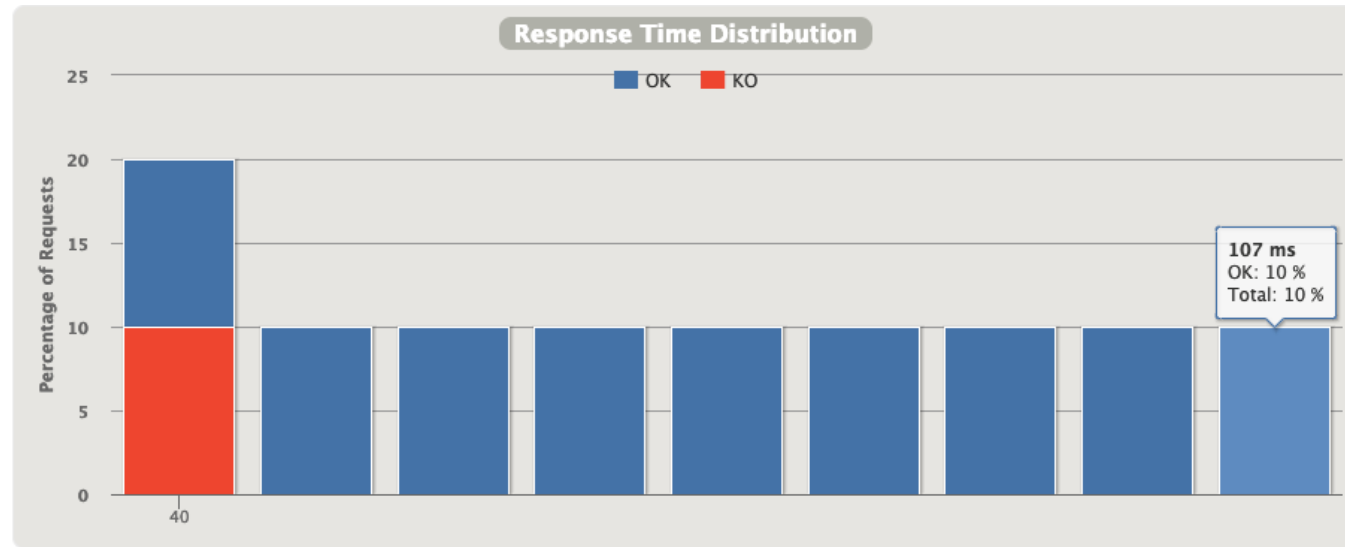
```
Reports generated in 0s.
Please open the following file: /usr/local/Cellar/gatling/gatling-charts-highcharts-bundle-3.0.3/results/testsimulation-20190407184000384/index.html
```

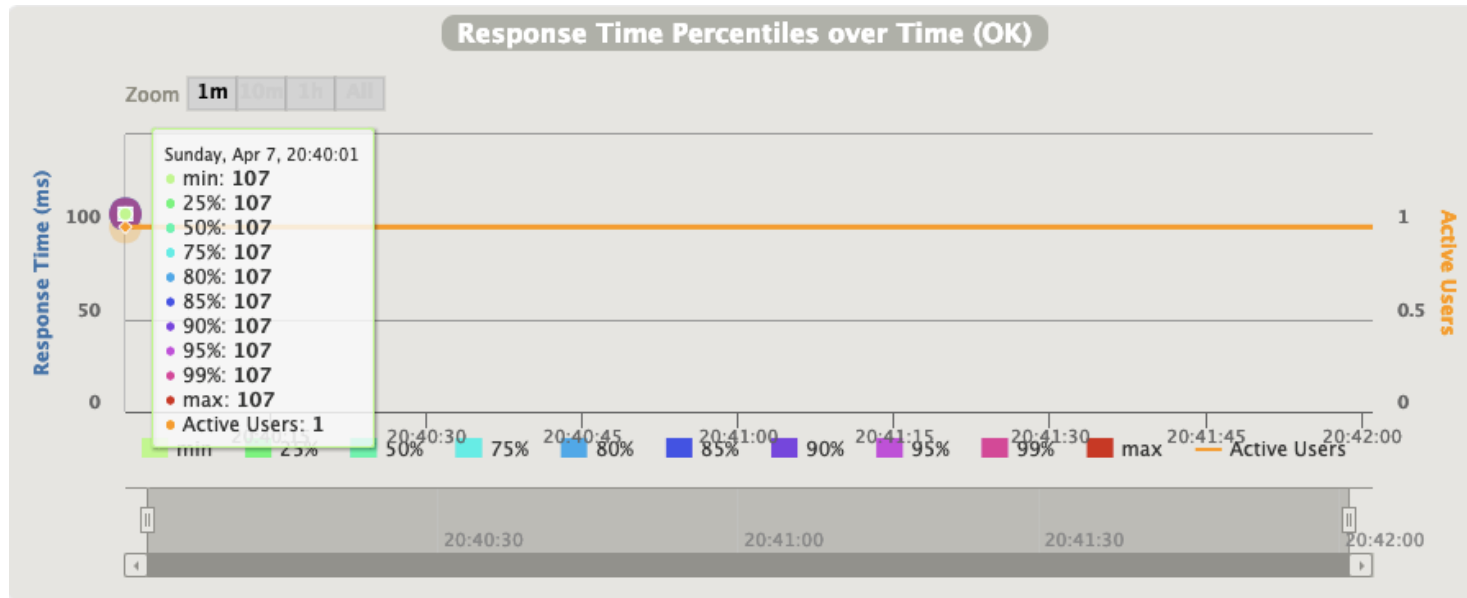





- **Usuarios activos durante la simulación**

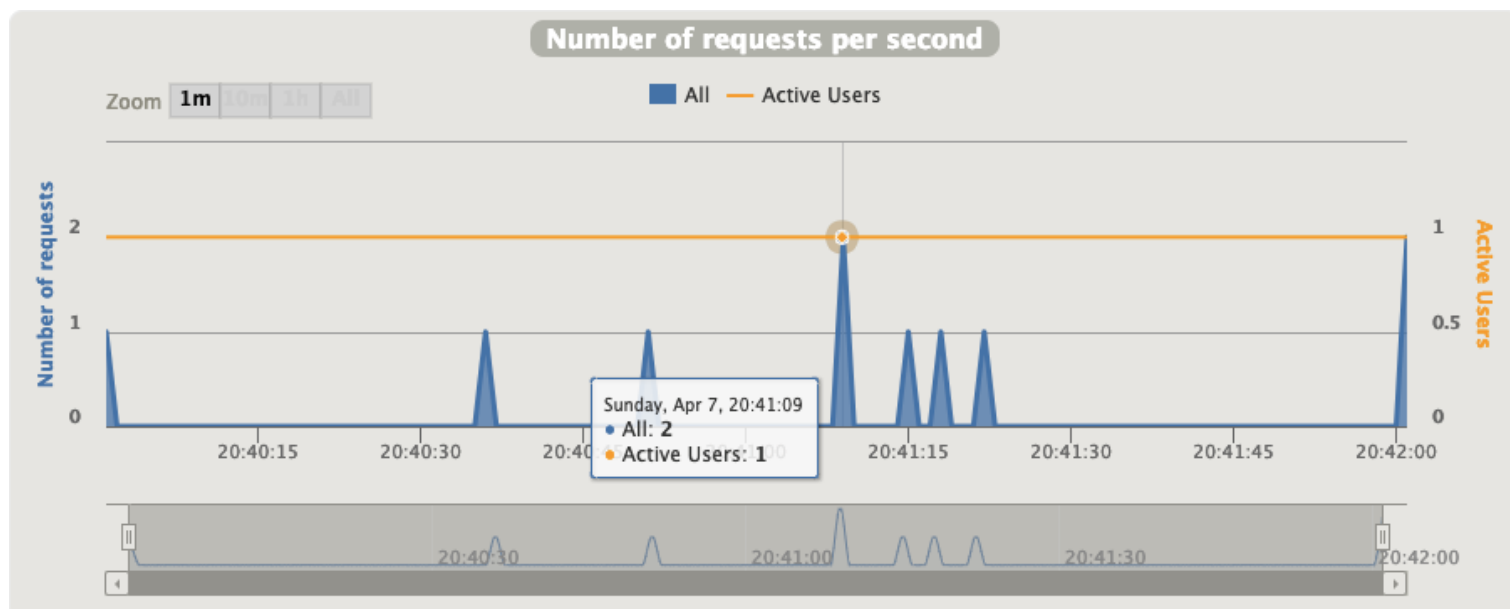
Muestra el número de usuarios activos (que realizan y reciben respuesta a solicitudes) a lo largo del tiempo de la simulación. Se puede relacionar con otras medidas como los tiempos de respuesta y el número de peticiones/respuestas por segundo.





- ## Percentiles de tiempos de respuesta en el tiempo

Similar a la distribución el tiempo de respuesta, pero muestra los datos durante un período de tiempo más largo para evaluar cómo se comporta el sistema bajo una carga sostenida. P.ejem 200 usuarios accediendo durante un período de 5 min.



- **Peticiones/respuestas por segundo**

Mide el número de veces que se realiza (o se recibe) una solicitud de un recurso desde el servidor por segundo. Se puede simular, por ejemplo, que 200 usuarios están haciendo peticiones simultáneas (200 solicitudes por segundo).

Otras pruebas

- Usabilidad

Permiten determinar si una aplicación es fácil de usar.

Evalúan la experiencia del usuario antes (formativas) y después (sumativas) de la puesta en producción.

Entre las características que se pueden medir están:

- Facilidad de aprendizaje y memorización
- Precisión y completitud de las tareas
- Eficiencia y productividad (tiempo en realizar la tarea)
- Errores
- Satisfacción
- Accesibilidad

Las técnicas de pruebas incluyen observación, benchmarking, encuestas, entrevistas, cuestionarios, eye-tracking..

Otras pruebas

- Seguridad

Permiten determinar las características de seguridad del sistema.
Se realizan auditorías de seguridad y hacking 'ético'.

Informe de vulnerabilidades y posibles soluciones.

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

- Escalabilidad, mantenibilidad, portabilidad.. 😊

Enlaces de interés

- **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/>
- **Otras Pruebas**
 - **Tipos de pruebas de software**
<http://ing-sw.blogspot.com/2005/04/tipos-de-pruebas-de-software.html>
 - **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)