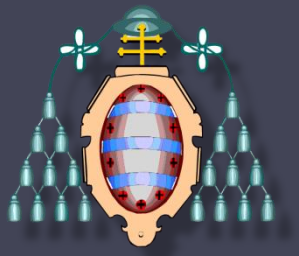
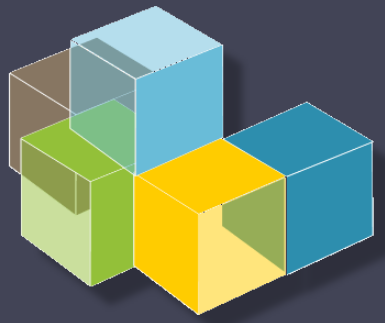
ES
Español



Universidad de Oviedo

Escuela de Ingeniería Informática



ARQUITECTURA DEL SOFTWARE

Arquitectura del Software

Lab.
Pruebas de carga y estrés

2016-17

Herminio García González

Escuela de Ingeniería Informática de Oviedo

¿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

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!!!

Repositorio de pruebas

- http://www.github.com/arquisoft/VotingSystem0
- Rama: loadTests (git checkout –b loadTests)
- Arrancamos (mvn spring-boot:run)
- Tres páginas:
  - Landing page
  - Sort page (repetición de algoritmo de ordenación para simular computación pesada)
  - Search page
    - Long – computación larga
    - Error – excepción
    - Otro – computación normal

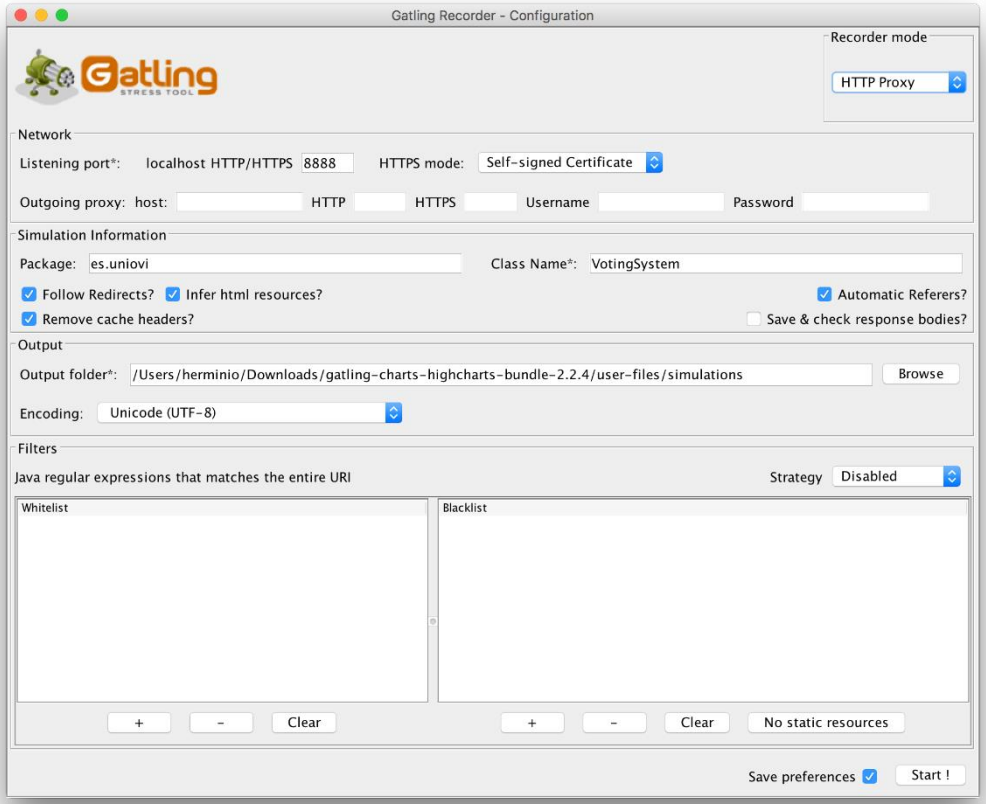
Gatling

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

Descarga e instalación

- http://gatling.io
- Java instalado?
- Listo para funcionar
- Dos scripts:
  - Recorder.sh/Recorder.bat
  - Gatling.sh/Gatling.bat

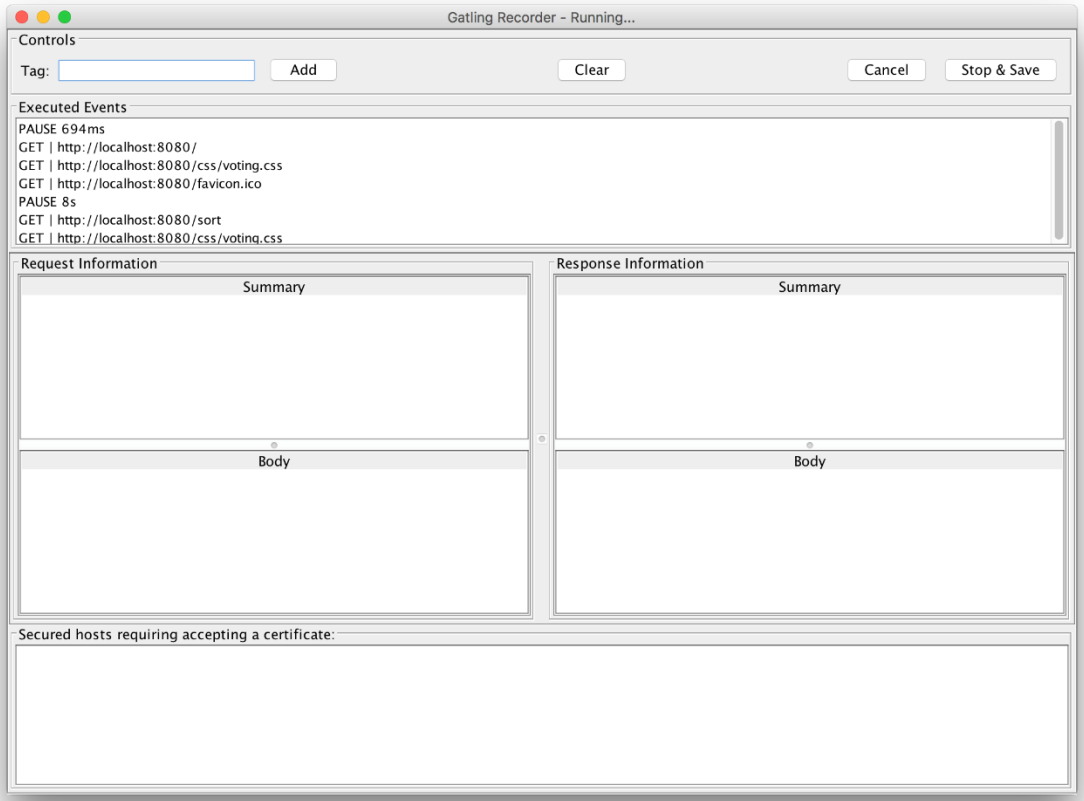
Recorder


Gatling Recorder - Configuration
Gatling
STRESS TOOL
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
Blacklist
+ - Clear
+ - Clear No static resources
Save preferences Start !

Configurar proxy

- localhost:8888
- 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

Resultado


Gatling Recorder - Running...
Controls
Tag:
Add
Clear
Cancel
Stop & Save
Executed Events
PAUSE 694ms
GET | http://localhost:8080/
GET | http://localhost:8080/css/voting.css
GET | http://localhost:8080/favicon.ico
PAUSE 8s
GET | http://localhost:8080/sort
GET | http://localhost:8080/css/voting.css
Request Information
Summary
Body
Response Information
Summary
Body
Secured hosts requiring accepting a certificate:

Script

- Este proceso genera un script en Scala usando el DSL

`user-files/simulations/es/uniovi/VotingSystem.scala`

- Posibilidad de configuración

Ejemplo

```
class VotingSystem0 extends Simulation {

val httpProtocol = http
.baseURL("http://localhost:8080")
.inferHtmlResources()
.acceptHeader("text/html...")
.acceptEncodingHeader("gzip, deflate")
.acceptLanguageHeader("en-US,es-ES;q=0.8,es;q=0.5,en;q=0.3")
.userAgentHeader("...")

val headers_0 = Map("Upgrade-Insecure-Requests" -> "1")
val headers_1 = Map("Accept" -> "text/css,*/*;q=0.1")

val scn = scenario("VotingSystem0")
 .exec(http("request_0").get("/").headers(headers_0).resources(http("request_1")
 .get("/css/voting.css").headers(headers_1)))
 .pause(4)
 .exec(http("request_2").get("/sort").headers(headers_0).resources(http("request_3")
 .get("/css/voting.css").headers(headers_1)))

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

Configurando el número de usuarios

Injection profile

Control how users are injected in your scenario

Injection steps

nothingFor	constantUsersPerSec
atOnceUsers	rampUsersPerSec
rampUsers	splitUsers
	heavisideUsers

50 usuarios progresivos en 60 segundos

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

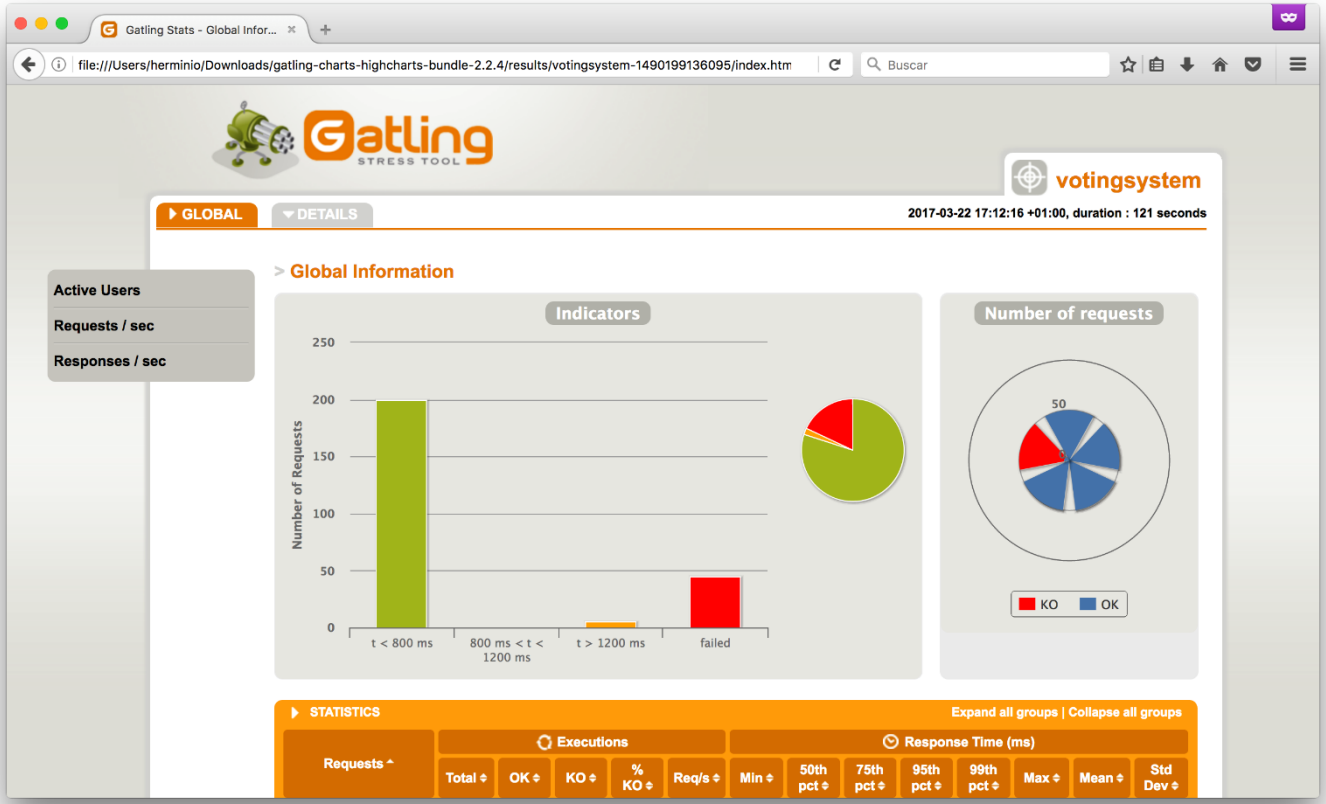
```
...
setUp(scn.inject(rampUsers(50) over(60 seconds))).
    protocols(httpProtocol)

}
```

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

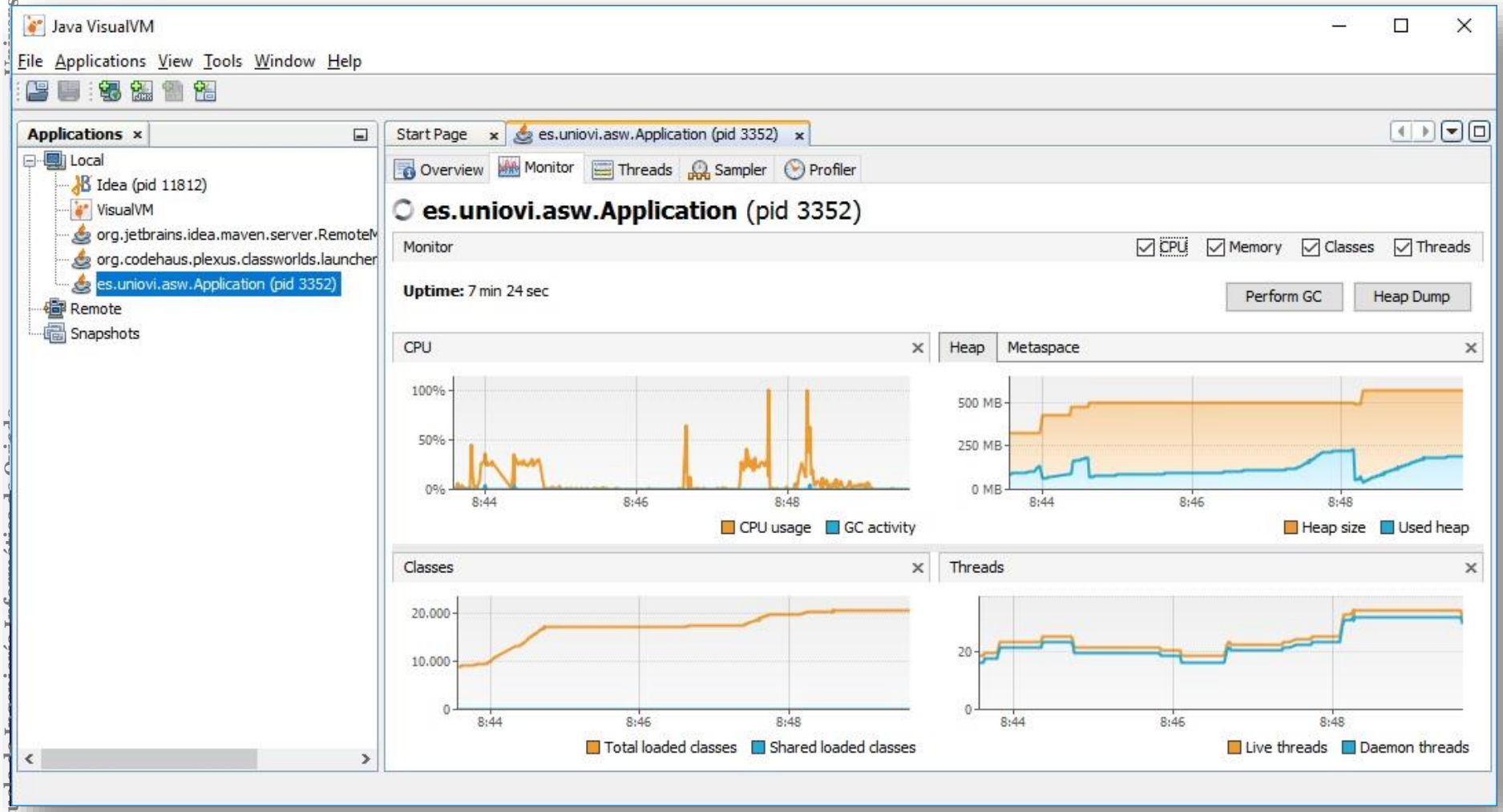
Aspecto final



Monitorización & profiling

- Monitorizar una aplicación mientras se ejecuta
- Registrar uso de CPU, memoria, hilos, etc.
- Algunas herramientas:
  - VisualVM, JProfiler, YourKit, etc.
- VisualVM
  - https://visualvm.java.net/
  - Ya está instalada con el JDK: `jvisualvm`

Ejemplo con VisualVM



Requisitos entregables

- Tercer entregable
  - Pruebas de carga opcionales
- Cuarto entregable
  - Pruebas de carga obligatorias
  - Cobertura de los escenarios de calidad