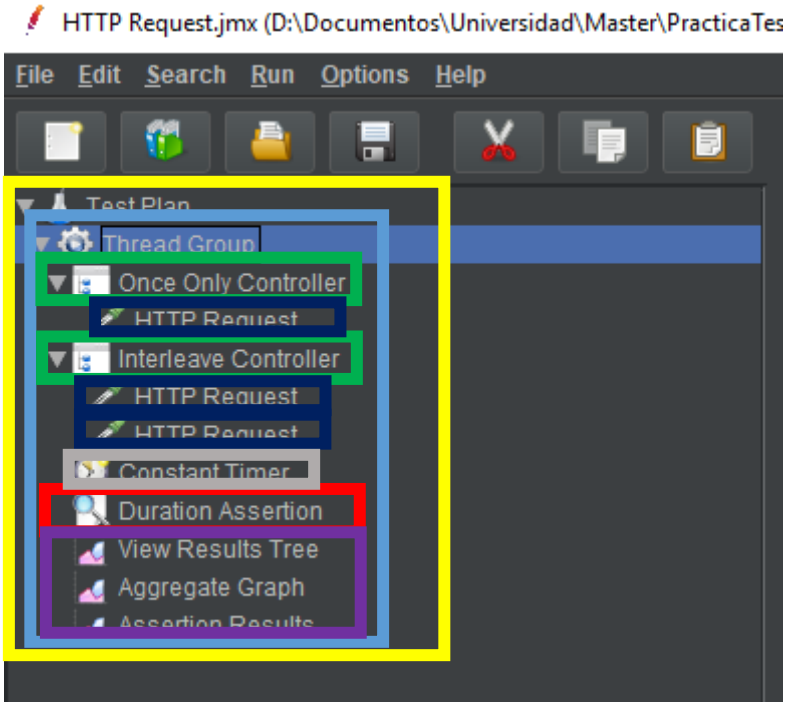
Tutoriales para Frameworks de Load Testing

Héctor Hernández Marqués

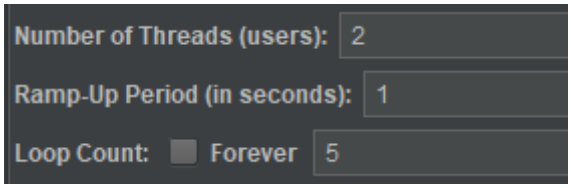
https://github.com/HectorHernandezMarques/LoadTesting

JMeter.

- Download http://apache.uvigo.es//jmeter/binaries/apache-jmeter-4.0.tgz and extract it.
- Execute bin/ApacheJMeter.jar



Aquí se puede ver un **TestPlan** que sigue una estructura jerárquica dónde irá toda la lógica del test. Dentro de él vemos el **ThreadGroup** que determina cuántos hilos simultáneos y durante cuánto tiempo se ejecutarán por Loop.

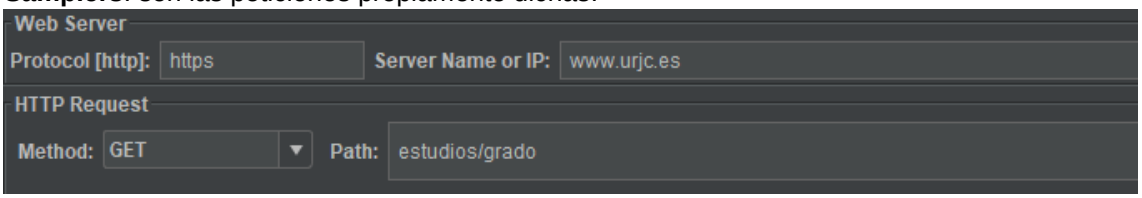


Dentro de este grupo, se declararán las acciones que queramos que ejecute cada hilo del TreadGroup.

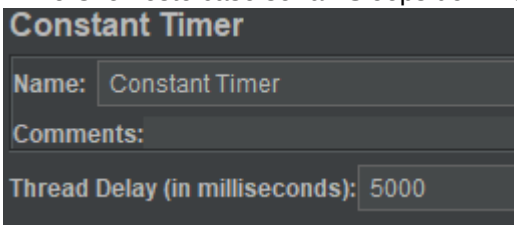
Estas acciones pueden ser, entre otras, **LogicControls**, **Samplers**, **Timers**, **Assertions** o **Listeners**. Que se ejecutarán para cada hilo de forma concurrente.

- **LogicControls**: suponen condiciones que varían la ejecución de cada hilo. En este caso OnceOnlyController solo ejecutará su Sampler en los hilos del primer Loop. Y InterleaveController alternará sus samplers en cada Loop.

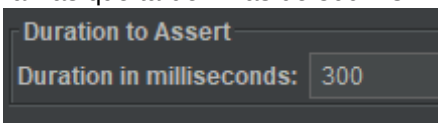
- **Samplers**: son las peticiones propiamente dichas.



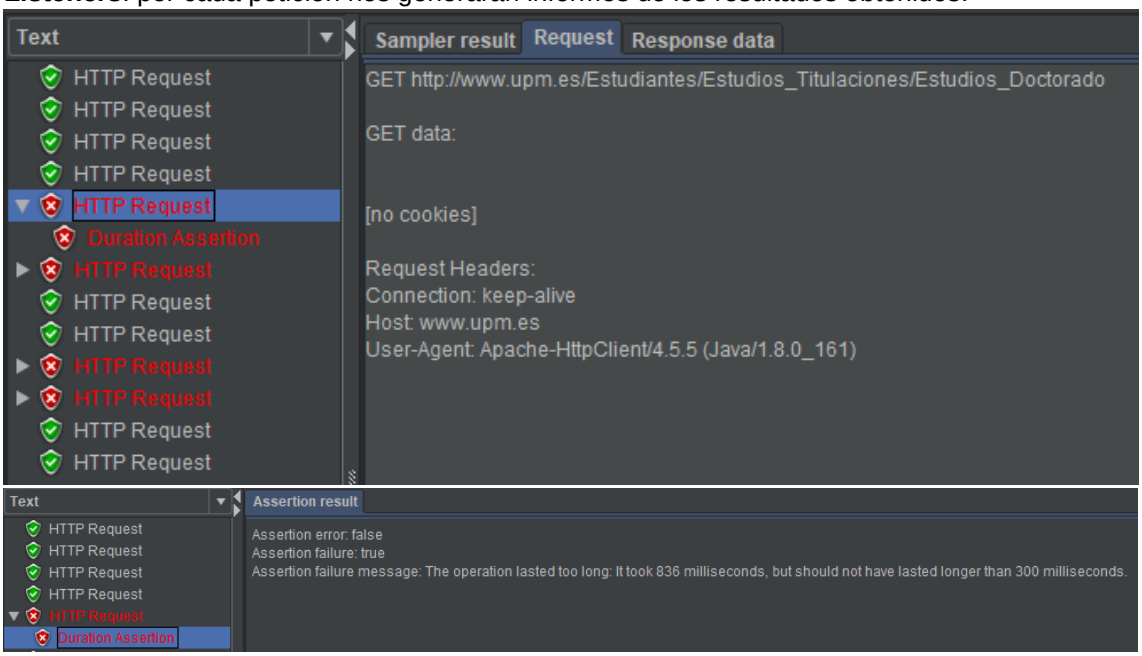
- **Timers**: en este caso serían Sleeps de milisegundos entre petición y petición.



- **Assertions**: condiciones que validan las peticiones. En este caso nos marcará como fail las que tarden más de 300 ms.



- **Listeners**: por cada petición nos generarán informes de los resultados obtenidos.

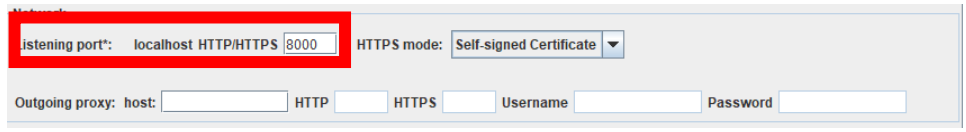


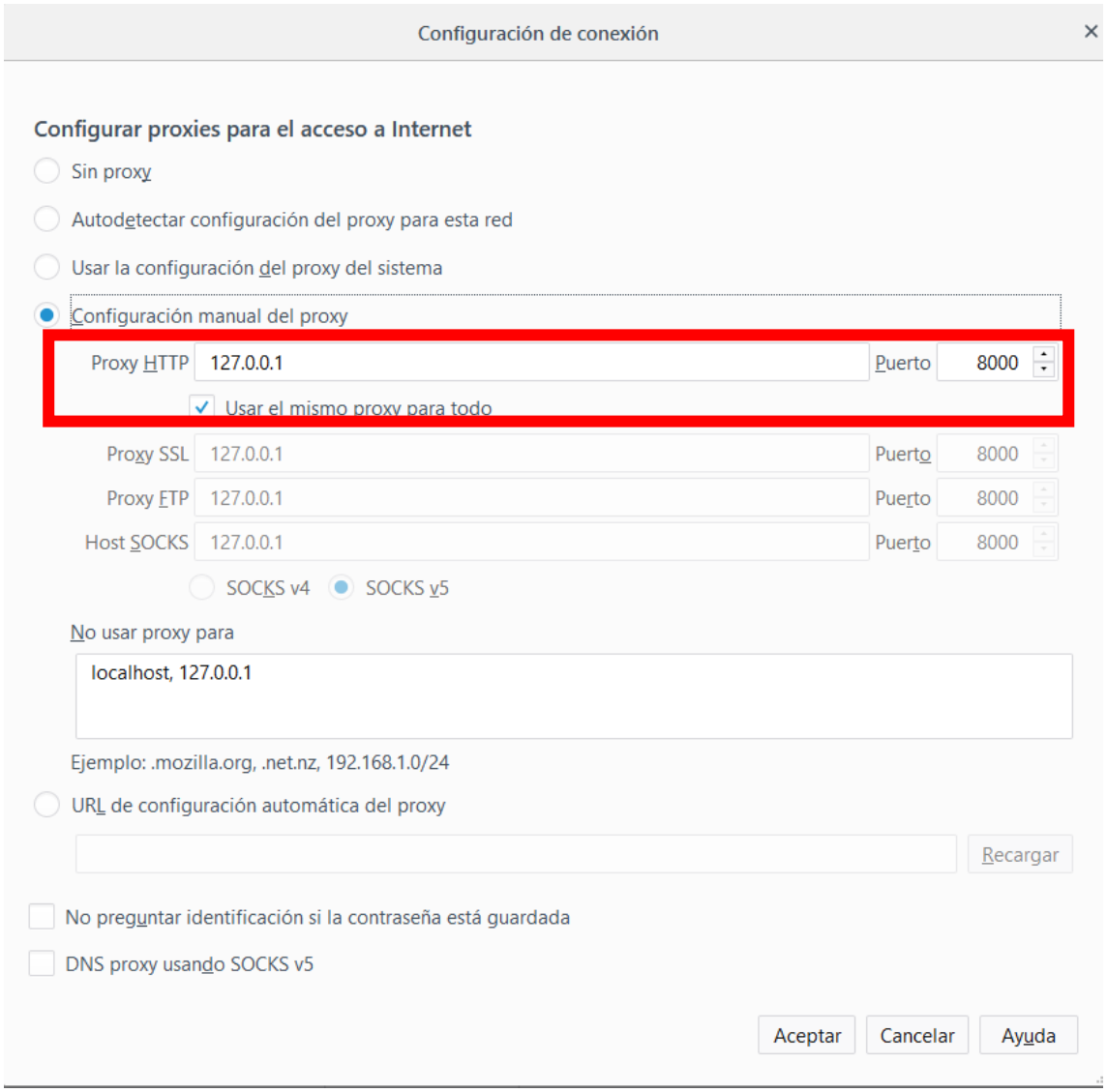
Gatling

- Download https://repo1.maven.org/maven2/io/gatling/highcharts/gatling-charts-highcharts-bundle/2.3.1/gatling-charts-highcharts-bundle-2.3.1-bundle.zip and extract it.
- Execute bin/recorder

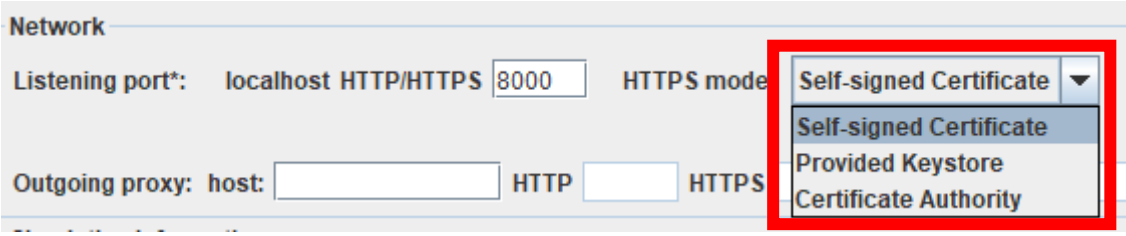
Gatling executa código Scala en sus experimentos. Pero añade un programa recorder que actuando como un man-in-the-middle capturando tu sesión en el navegador y reproduciéndola posteriormente.

Para capturar estas sesiones hay que configurar en nuestro navegador que acceda a través del proxy que Gatling levanta.





Hay que tener en mente que Gatling intercepta nuestra conexión, así que el navegador se quejará por tema de certificados. Hay varias formas de solucionar este problema. La que he usado yo es aceptar el certificado auto-generado por Gatling como trusted-certificate. También puedes generarte tus propios certificados y que Gatling los use.

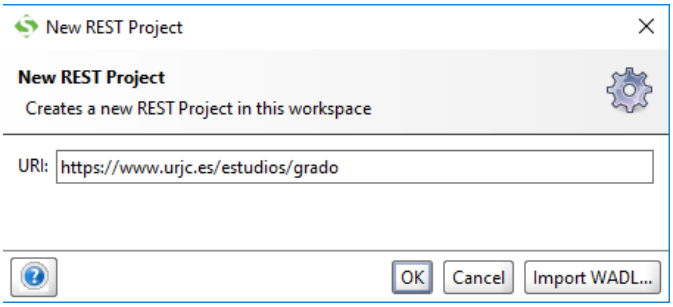


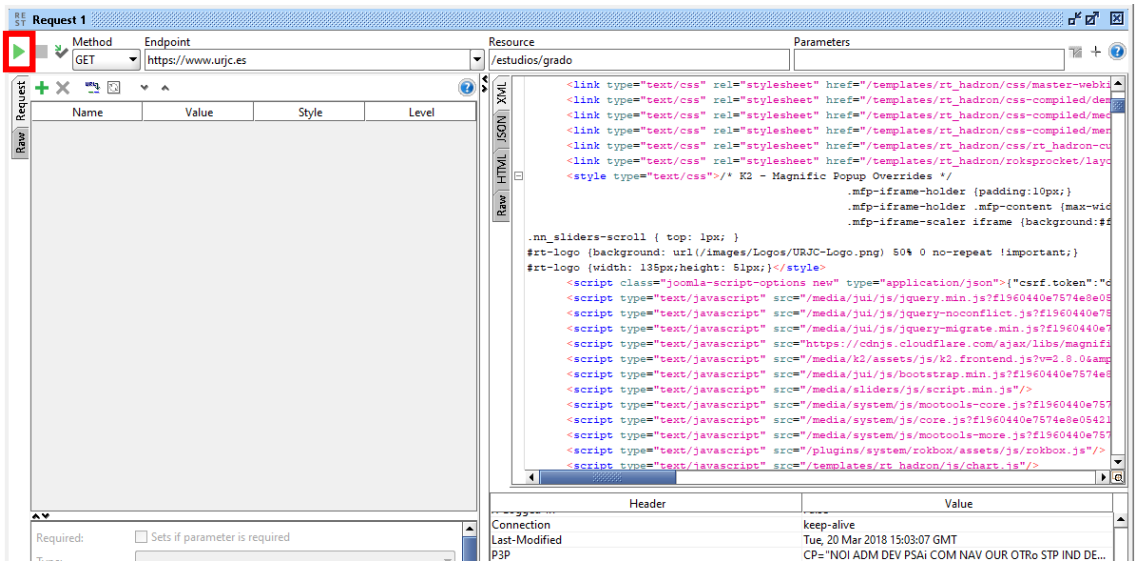
Empezamos a capturar con **START** y terminamos con **Stop&Save** generando un test case en Scala en la carpeta seleccionada.

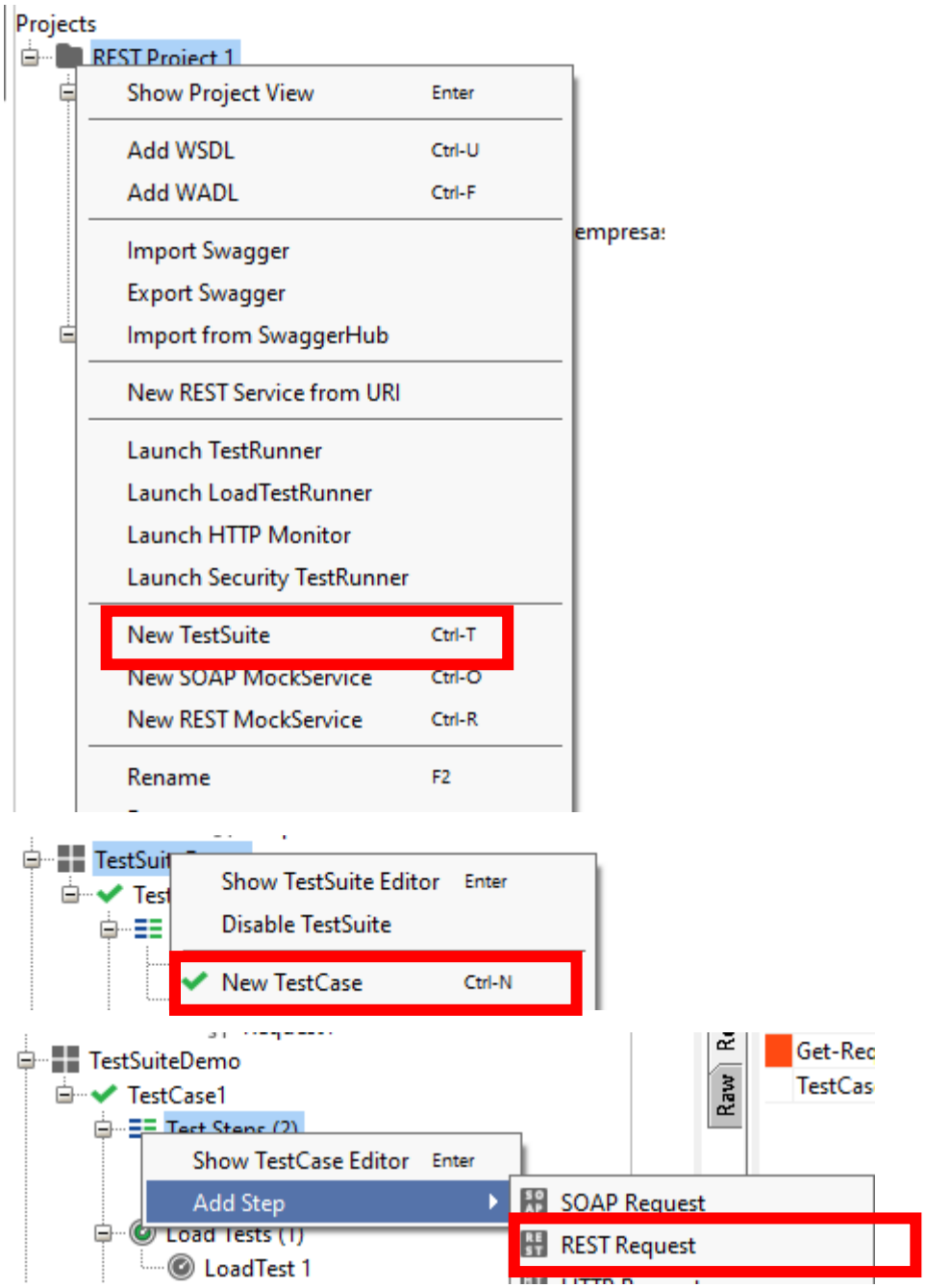
Ese test puede ser modificado si conocemos Scala por encima usando su DSL y ejecutado con bin/gatling

SoapUI

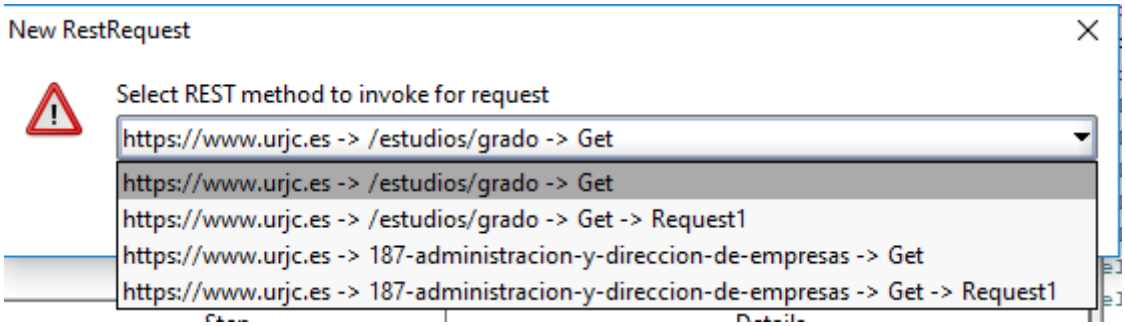
- Download https://www.soapui.org/downloads/soapui.html and install it.



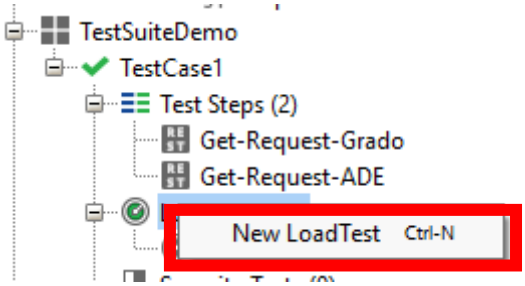


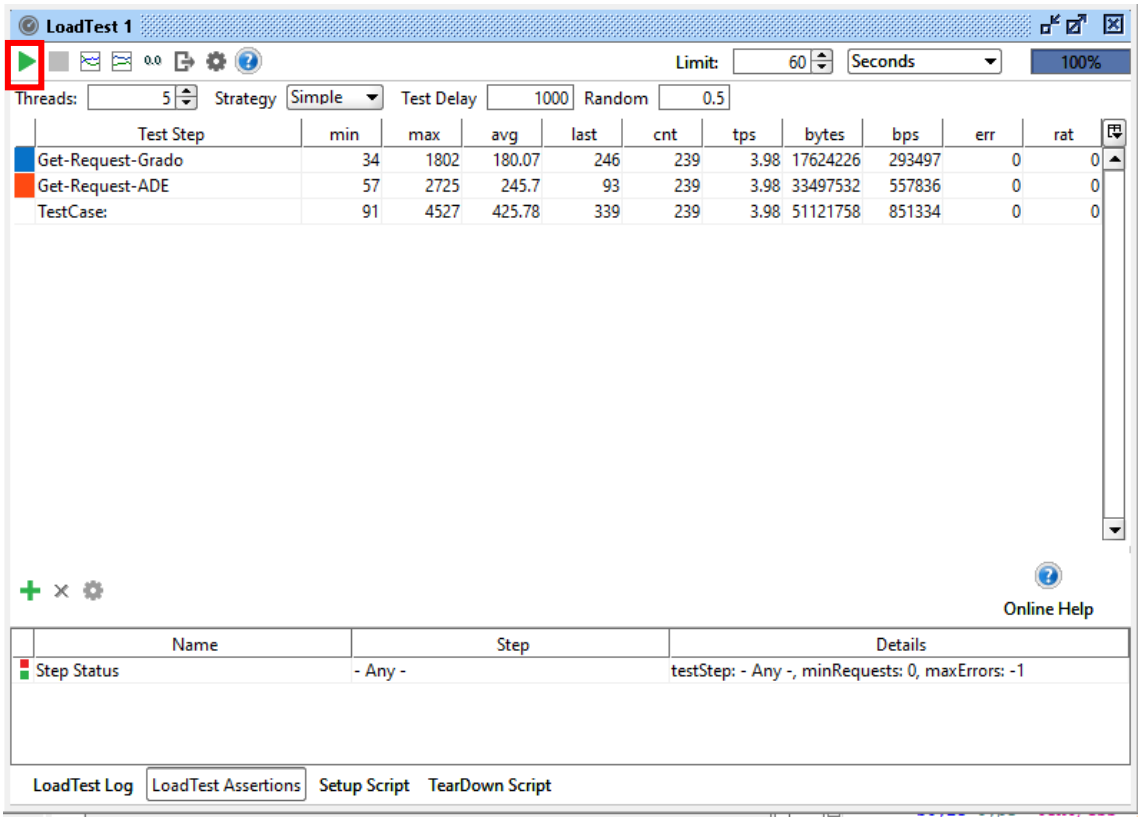


Seleccionamos un nombre para este Step del test y el recurso asociado.



Una vez hemos creado un TestCase simple lo usaremos en nuestro test de carga.





Artillery

- Instala Artillery. Una vez has instalado NodeJS:

```
$ npm install -g artillery
$ artillery –V
```

- Ejecuta un Test rápido

```
$ artillery quick --count 10 -n 20 https://artillery.io/
```

Este comando creará 10 usuarios que enviarán 20 periciones HTTP GET a https://artillery.io/.

MyTest.yml

```
config:
  target: 'https://www.urjc.es'
  phases:
    - duration: 60
      arrivalRate: 20
scenarios:
  - flow:
    - get:
        url: "/estudios/grado"
```

```
$ artillery run MyTest.yml
```

Definimos una fase de carga de 60 segundo con 20 usuarios virtuales llegando cada segundo (de media).