Práctica 4: Benchmarking y Ajuste del Sistema"
Ingeniería de Servidores

Raúl Durán Racero

12 de diciembre de 2021

ETSIIT
Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación

Índice

1. Ejercicio 1 **3**
1.1. Phoronix en UbuntuServer . . . 3
1.2. Phoronix en CentOS . . . 3
1.3. Comparación de Resultados . . . 3

2. Ejercicio 2 **6**
2.1. Test debe tener parametrizados el Host y el Puerto en el Test Plan . . . 8
2.2. Hacer 2 grupos de hebras distintos para simular al acceso de los alumnos y los administradores . . . 8
2.3. Añadimos esperas aleatorias (Gaussian Random Timer) . . . 13
2.4. Recuperar datos alumno . . . 13
2.5. Muestreo recogido por apiAlumnos.log . . . 13
2.6. Usar expresión regular para extrar token JWT . . . 14

3. Ejercicio Opcional **16**

1. Ejercicio 1

Una vez que haya indagado sobre los *benchmarks* disponibles, seleccione como mínimo dos de ellos y proceda a ejecutarlos en **Ubuntu** y **CentOS**. Comente las diferencias.

1.1. Phoronix en UbuntuServer

Lo primero que haremos será obtener el paquete de la página de Phoronix e instalarlo:

```
durar@durar:~$ sudo wget http://phoronix-test-suite.com/releases/repo/pts.
    debian/files/phoronix-test-suite_10.6.1_all.deb
durar@durar:~$ sudo dpkg -i phoronix-test-suite_10.6.1_all.deb
```

Una vez instalado, podemos ver los diferentes tests y suites que dispone Phoronix:

```
durar@durar:~$ phoronix-test-suite list-available-tests
durar@durar:~$ phoronix-test-suite list-available-suites
```

De la lista de tests, escogeremos dos. En mi caso, he escogido los siguientes:

pts/sudokut:

Sudokut es un test que mide cuánto tiempo tarda el sistema en resolver 100 puzzles *Sudoku* escritos en TCL (*Tool Command Language*):

```
durar@durar:~$ phoronix-test-suite benchmark pts/sudokut
```

pts/ramspeed:

RAMspeed SMP comprueba como actúa la RAM de nuestro sistema:

```
durar@durar:~$ phoronix-test-suite benchmark pts/ramspeed
```

Nos pedirá que escojamos distintas ocpiones. Escogemos las que queramos (en mi caso, solo compararé la función *Add*):

Si hay errores a la hora de instalar las dependencias necesarias para los tests, intenta actualizar:

```
durar@durar:~$ sudo apt-get update
```

1.2. Phoronix en CentOS

Repetimos los pasos de instalación que hicimos en UbuntuServer (si tienes instalados en CentOS wget y dpkg):

```
[durar@localhost ~]$ sudo wget http://phoronix-test-suite.com/releases/repo/pts
    .debian/files/phoronix-test-suite_10.6.1_all.deb
[durar@localhost ~]$ sudo dpkg -i phoronix-test-suite_10.6.1_all.deb
```

Y volvemos a comrpobar los tests y suites:

```
[durar@localhost ~]$ phoronix-test-suite list-available-tests
[durar@localhost ~]$ phoronix-test-suite list-available-suites
```

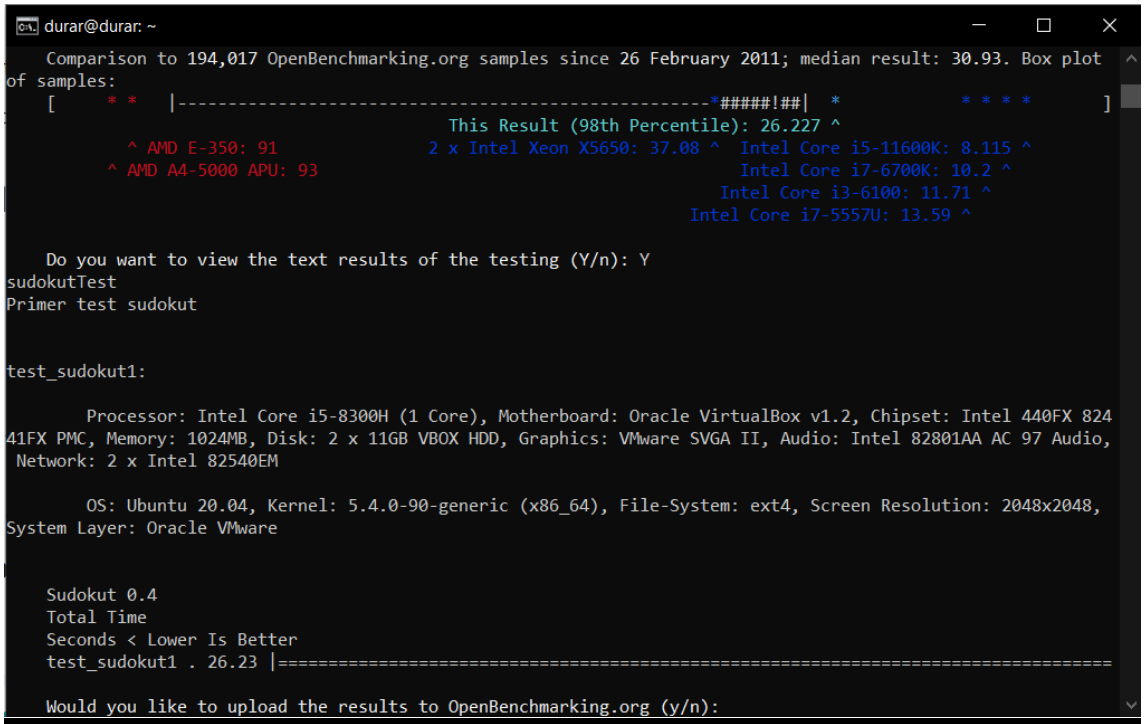
Ejecutaremos los mismos tests que en UbuntuServer para comparar resultados:

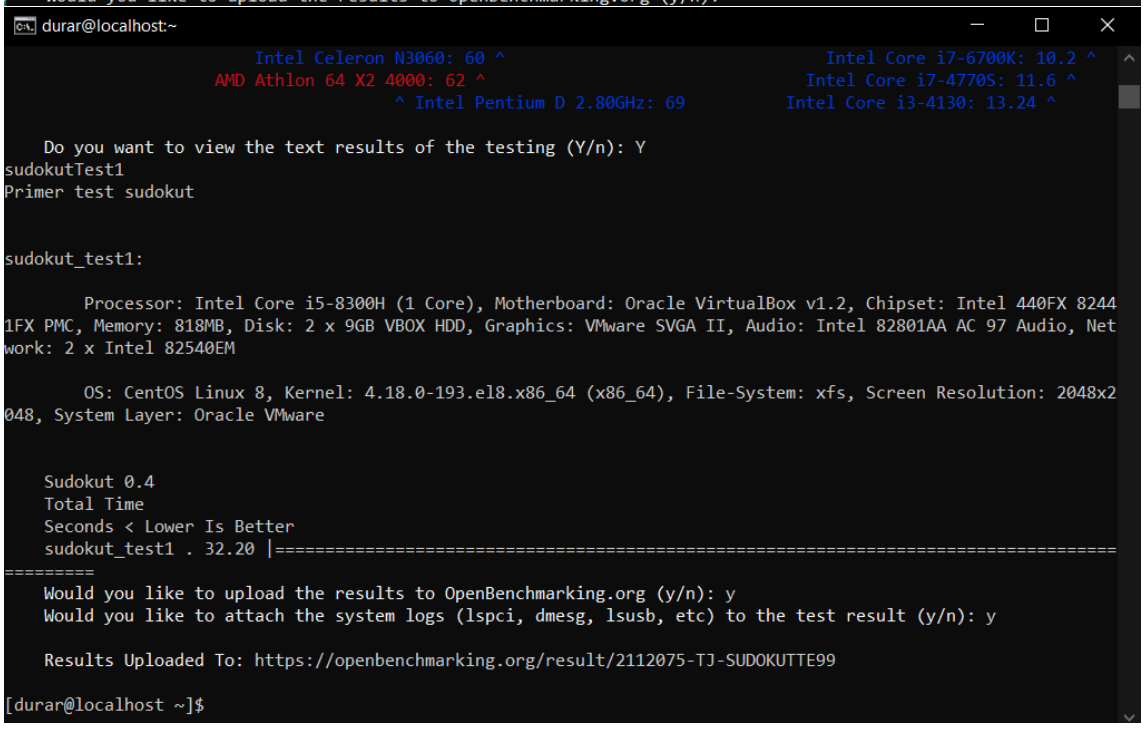
```
[durar@localhost ~]$ phoronix-test-suite benchmark pts/sudokut
[durar@localhost ~]$ phoronix-test-suite benchmark pts/ramspeed
```

1.3. Comparación de Resultados

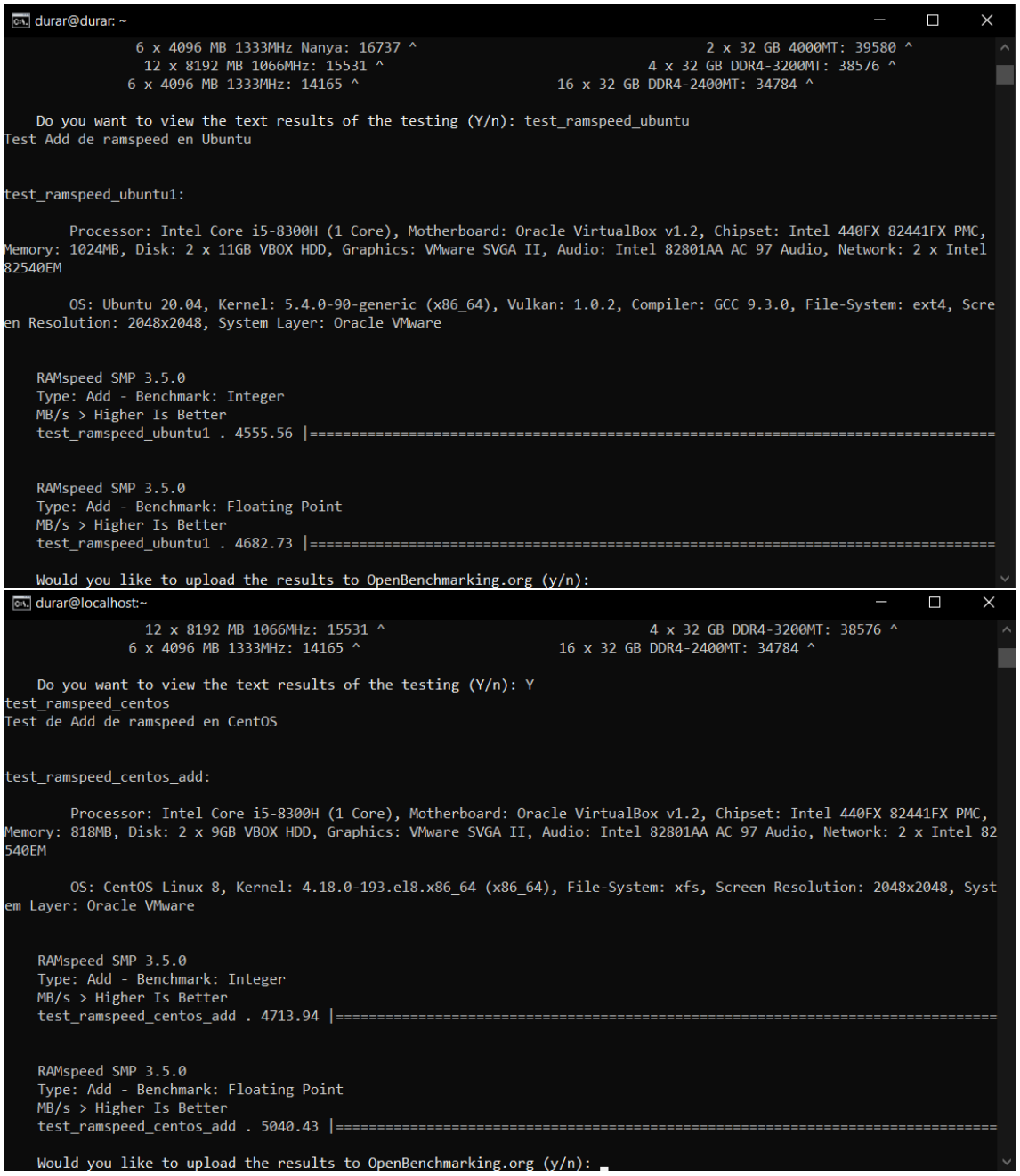
Sudokut:

Podemos ver como en la primera imagen (UbuntuServer), el test Sudokut ha tardado menos en ejecutarse (26.23s) que en CentOS (32.29s).


durar@durar: ~
Comparison to 194,017 OpenBenchmarking.org samples since 26 February 2011; median result: 30.93. Box plot of samples:
This Result (98th Percentile): 26.227 ^
^ AMD E-350: 91
2 x Intel Xeon X5650: 37.08 ^
Intel Core i5-11600K: 8.115 ^
^ AMD A4-5000 APU: 93
Intel Core i7-6700K: 10.2 ^
Intel Core i3-6100: 11.71 ^
Intel Core i7-5557U: 13.59 ^
Do you want to view the text results of the testing (Y/n): Y
sudokutTest
Primer test sudokut
test_sudokut1:
Processor: Intel Core i5-8300H (1 Core), Motherboard: Oracle VirtualBox v1.2, Chipset: Intel 440FX 82441FX PMC, Memory: 1024MB, Disk: 2 x 11GB VBOX HDD, Graphics: VMware SVGA II, Audio: Intel 82801AA AC 97 Audio, Network: 2 x Intel 82540EM
OS: Ubuntu 20.04, Kernel: 5.4.0-90-generic (x86_64), File-System: ext4, Screen Resolution: 2048x2048, System Layer: Oracle VMware
Sudokut 0.4
Total Time
Seconds < Lower Is Better
test_sudokut1 . 26.23
Would you like to upload the results to OpenBenchmarking.org (y/n):


durar@localhost:~
Intel Celeron N3060: 60 ^
Intel Core i7-6700K: 10.2 ^
AMD Athlon 64 X2 4000: 62 ^
Intel Core i7-4770S: 11.6 ^
^ Intel Pentium D 2.80GHz: 69
Intel Core i3-4130: 13.24 ^
Do you want to view the text results of the testing (Y/n): Y
sudokutTest1
Primer test sudokut
sudokut_test1:
Processor: Intel Core i5-8300H (1 Core), Motherboard: Oracle VirtualBox v1.2, Chipset: Intel 440FX 82441FX PMC, Memory: 818MB, Disk: 2 x 9GB VBOX HDD, Graphics: VMware SVGA II, Audio: Intel 82801AA AC 97 Audio, Network: 2 x Intel 82540EM
OS: CentOS Linux 8, Kernel: 4.18.0-193.el8.x86_64 (x86_64), File-System: xfs, Screen Resolution: 2048x2048, System Layer: Oracle VMware
Sudokut 0.4
Total Time
Seconds < Lower Is Better
sudokut_test1 . 32.20
Would you like to upload the results to OpenBenchmarking.org (y/n): y
Would you like to attach the system logs (lspci, dmesg, lsusb, etc) to the test result (y/n): y
Results Uploaded To: https://openbenchmarking.org/result/2112075-TJ-SUDOKUTTE99
[durar@localhost ~]$

RAMspeed:



Vemos que los resultados obtenidos en CentOS son mejores que en Ubuntu. Además, en ambos sistemas el resultado de Add con variables enteras son peores que con coma flotante.

2. Ejercicio 2

Tras probar un test básico para una web, utilizaremos Jmeter para hacer un test sobre una aplicación que ejecuta sobre dos contenedores (uno para la BD y otro para la aplicación en sí). El código está disponible en https://github.com/davidPalomar-ugr/iseP4JMeter donde se dan detalles sobre cómo ejecutar la aplicación en una de nuestras máquinas virtuales.

El servidor se distribuye en forma de una aplicación de contenedores Docker sobre Compose, por lo que necesitaremos ambas aplicaciones:

```
durar@durar:~$ sudo apt-get install docker docker-compose
```

Ahora clonaremos el repositorio de https://github.com/davidPalomar-ugr/iseP4JMeter:

```
durar@durar:~$ git clone https://github.com/davidPalomar-ugr/iseP4JMeter
```

Una vez lo hemos descargado, nos situamos al nivel del archivo docker-compose.yml y ejecutamos:

```
durar@durar:~/iseP4JMeter$ sudo docker-compose up
durar@durar:~/iseP4JMeter$ sudo docker-compose down
```

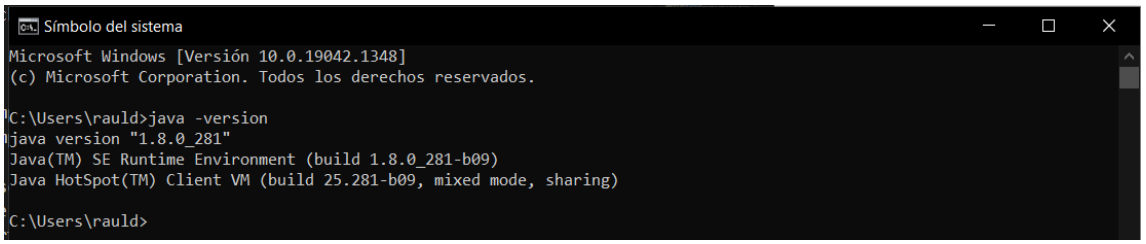
Con esto, Docker descargará las imágenes base y construirá las nuevas imágenes para la aplicación. Podemos comprobar que funciona el servicio correctamente buscando en nuestro navegador: **IP:3000**. Primero deberemos de abrir el puerto 3000 e instalar en UbuntuServer mongodb, ya que este sistema de base de datos es necesario para que funcione nuestro servicio. Tendremos esta interfaz:

ETSII Alumnos API

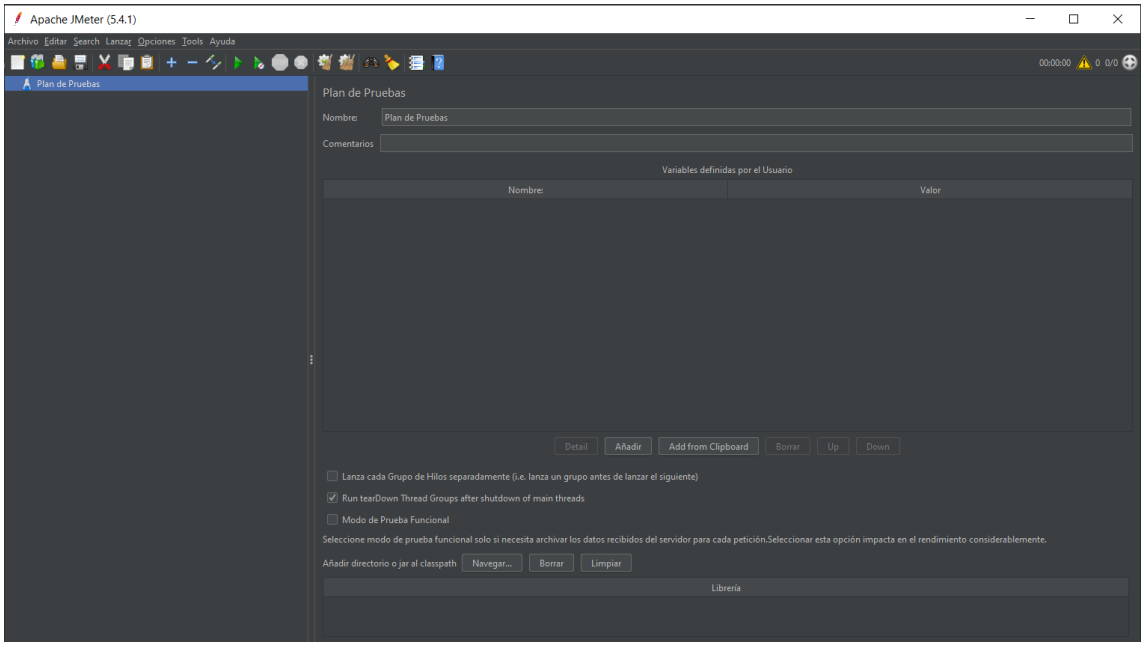
```
Descripción de la API Restful:
  POST /api/v1/auth/login
    Parametros:
      login:<emailUsuario>
      password:<secreto>
    Seguridad:
      Acceso protegido con BasicAuth (etsiiApi:laApiDeLaETSIIDaLache)
    Retorna:
      JWT Token

  GET /api/v1/alumnos/alumno/<email>
    Seguridad:
        Token JWT valido en cabecera estandar authorization: Bearer <token>
        Alumnos solo pueden solicitar sus datos. Administradores pueden solicitar cualquier alumno válido
    Retorna:
        Objeto Json con perfil de alumno
```

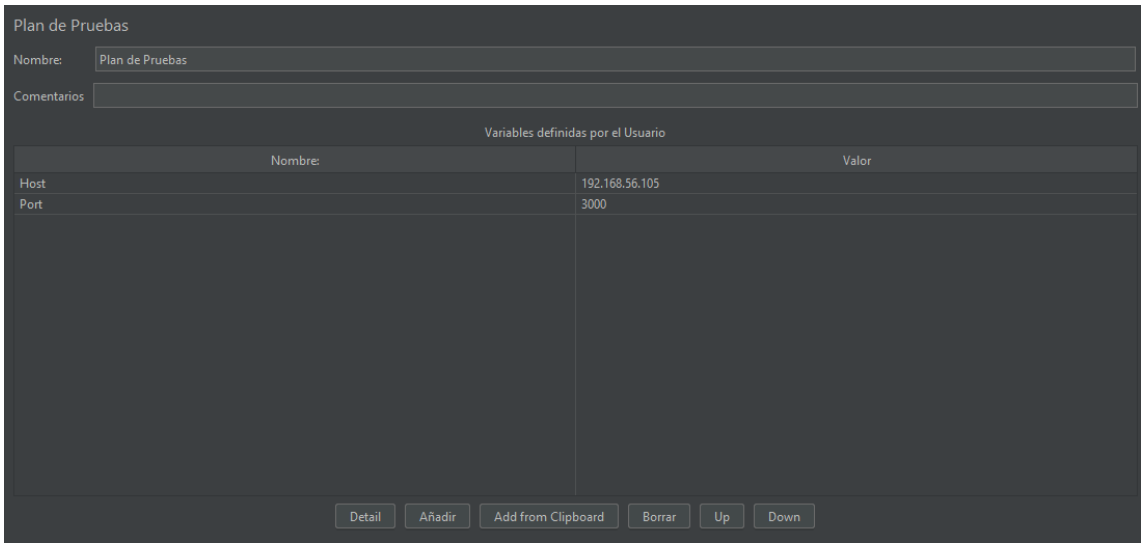
Ahora, tenemos que instalar **JMeter** en nuestro sistema host. Descargamos el paquete de JMeter y abrimos el ejecutable *ApacheJMeter* de la carpeta *bin* del archivo comprimido. Es necesario tener una versión de **Java 8** o superior.



Tendremos una interfaz como esta:



2.1. Test debe tener parametrizados el Host y el Puerto en el Test Plan



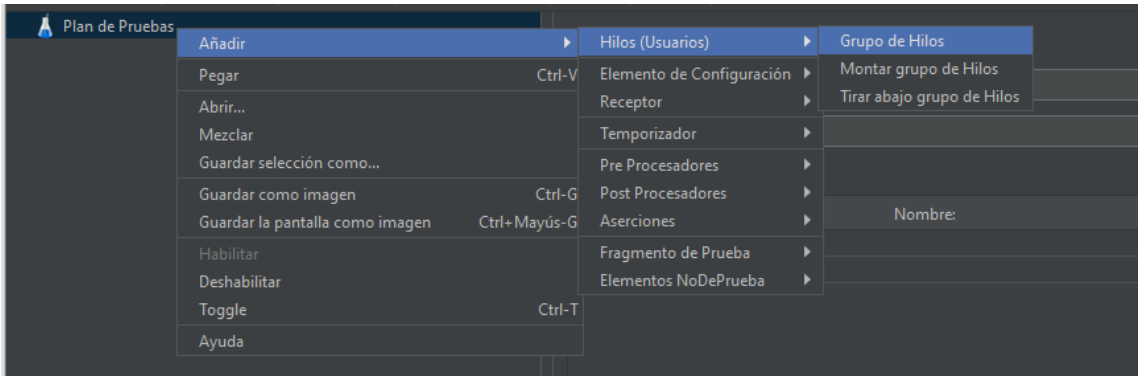
2.2. Hacer 2 grupos de hebras distintos para simular al acceso de los alumnos y los administradores

Se nos dice que los credenciales de alumno y administrador se cogen de los archivos *alumnos.csv* y *administrador.csv*, respectivamente, que se encuentran en el directorio iseP4JMeter/jMeter. Antes de hacer esto, escogeremos un administrador y un alumno (hacemos cat de los archivos .csv), para simplificar la comprobación del resto de pasos. Cogeremos los credenciales desde estos archivos como último paso. Yo he escogido:

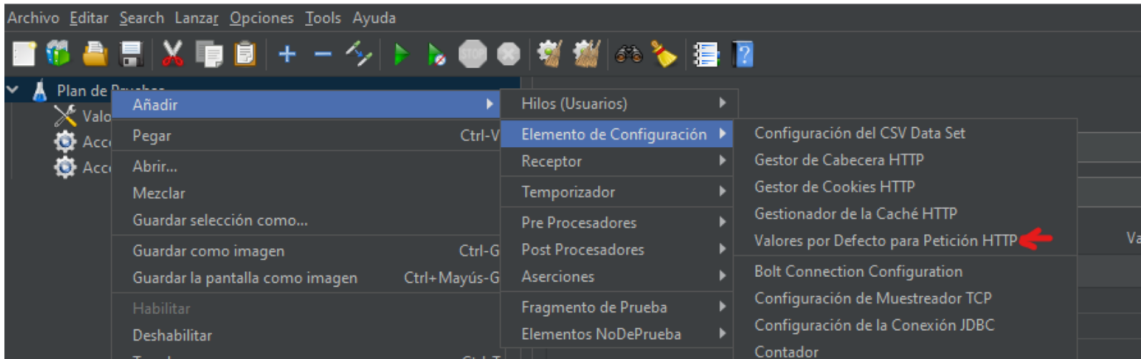
Administrador, contraseña: suarezgraves@etsii.ugr.es, laborum

Alumno, constraseña: margretdonovan@tropoli.com, non

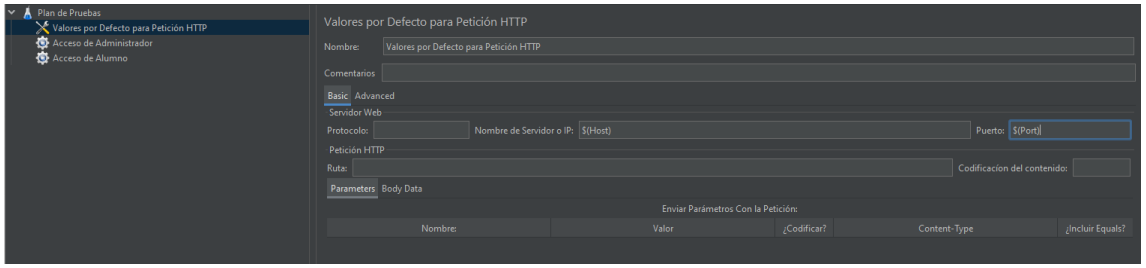
Ya escogidos nuestro administradory nuestro alumno, creamos 2 grupos de hebras en JMeter, uno para el administrador y el otro para el alumno. Para ello: Dejamos la configuración por defecto.



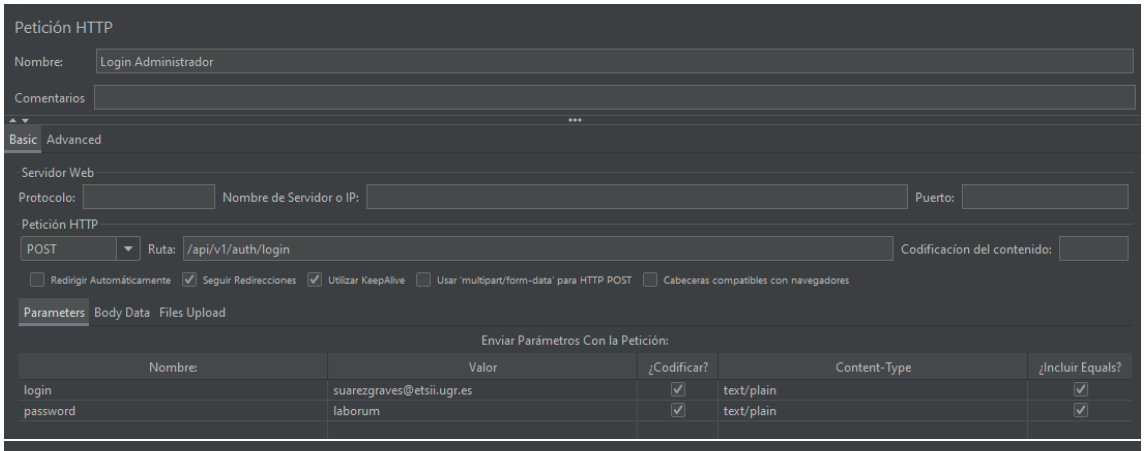
Ahora, añadimos una petición por defecto HTTP:

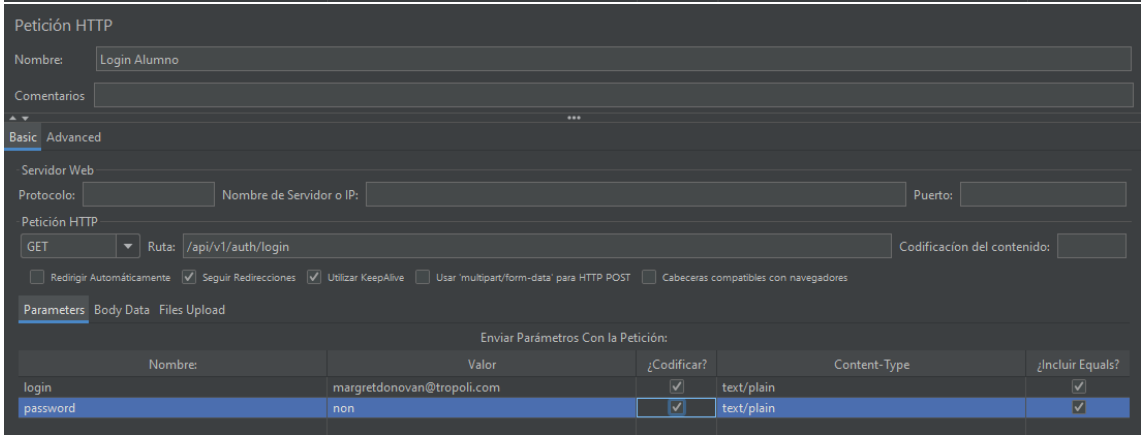


Y metemos nuestros parámetros Host y Port (se escribe de la forma ${param}):

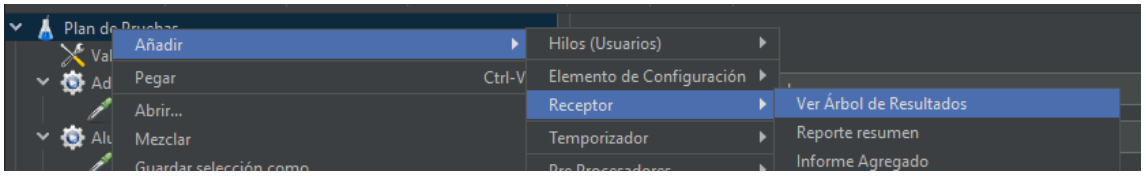


Esto lo hemos hecho para no tener que repetir Host y Port en cada petición. Para simular el acceso de un administrador y un alumno, creamos una petición HTTP para cada grupo de hebra, donde tendremos que introducir la ruta, usuario, constraseña, y tipo de peticion:

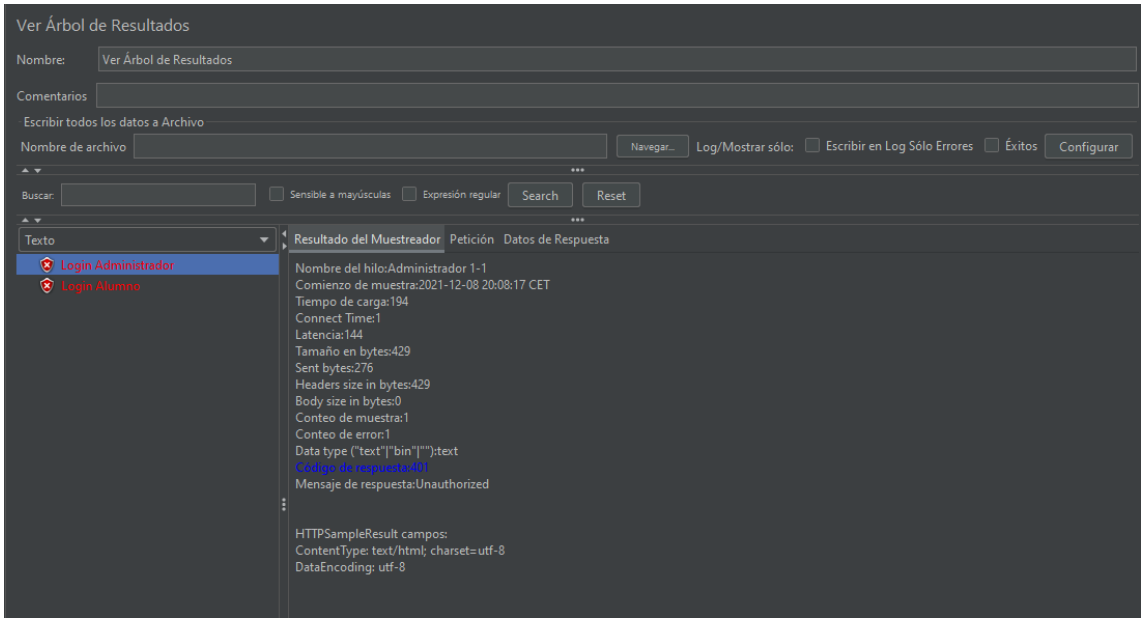




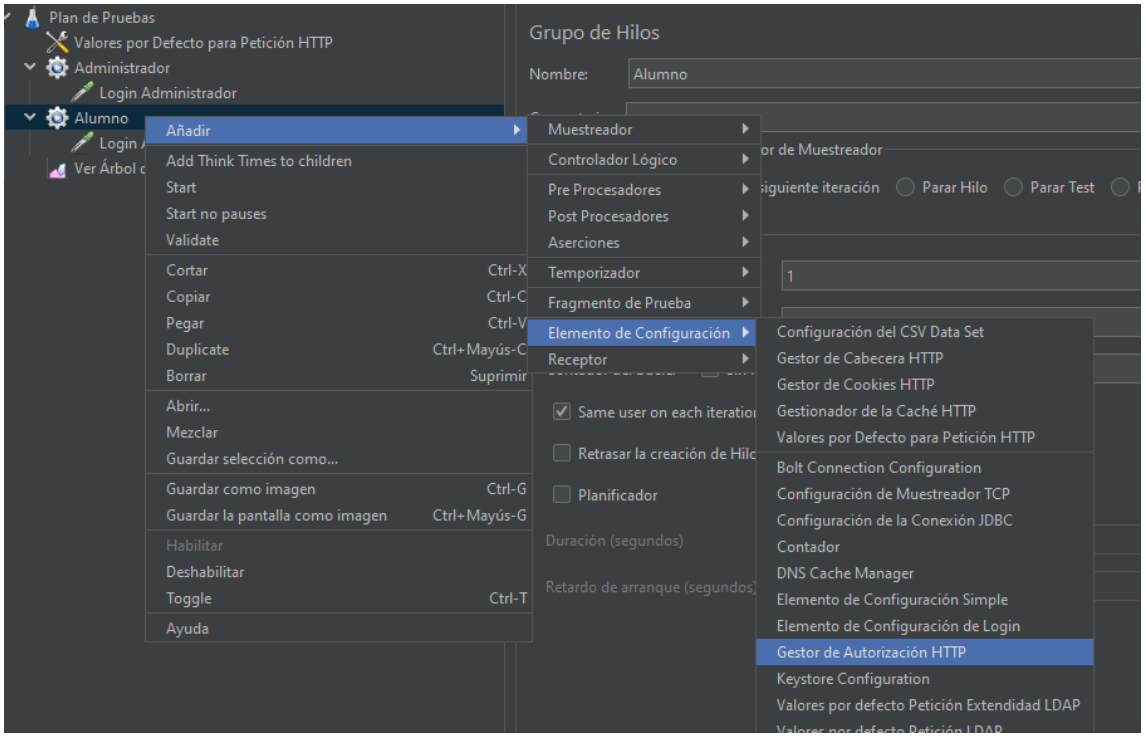
La ruta la podemos encontrar en el script pruebaEntorno.sh. Para ver el resultado de los accesos, podemos usar un árbol de resultados para ambos grupos de hilos:



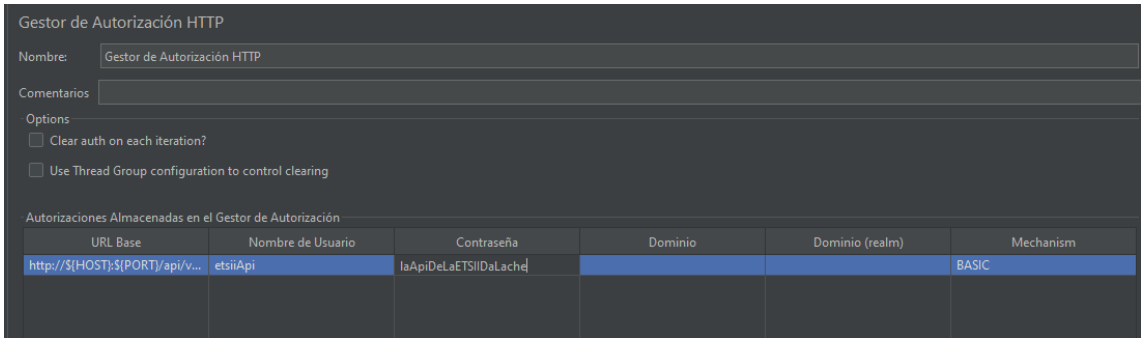
Al ejecutarlo, veremos que nos da el mismo error para ambos accesos, indicando que no está autorizado:



Esto se debe a que nos falta el gestor de autorización HTTP para cada grupo de hebras, ya que el acceso al servicio está protegido por HTTP BasicAuth:

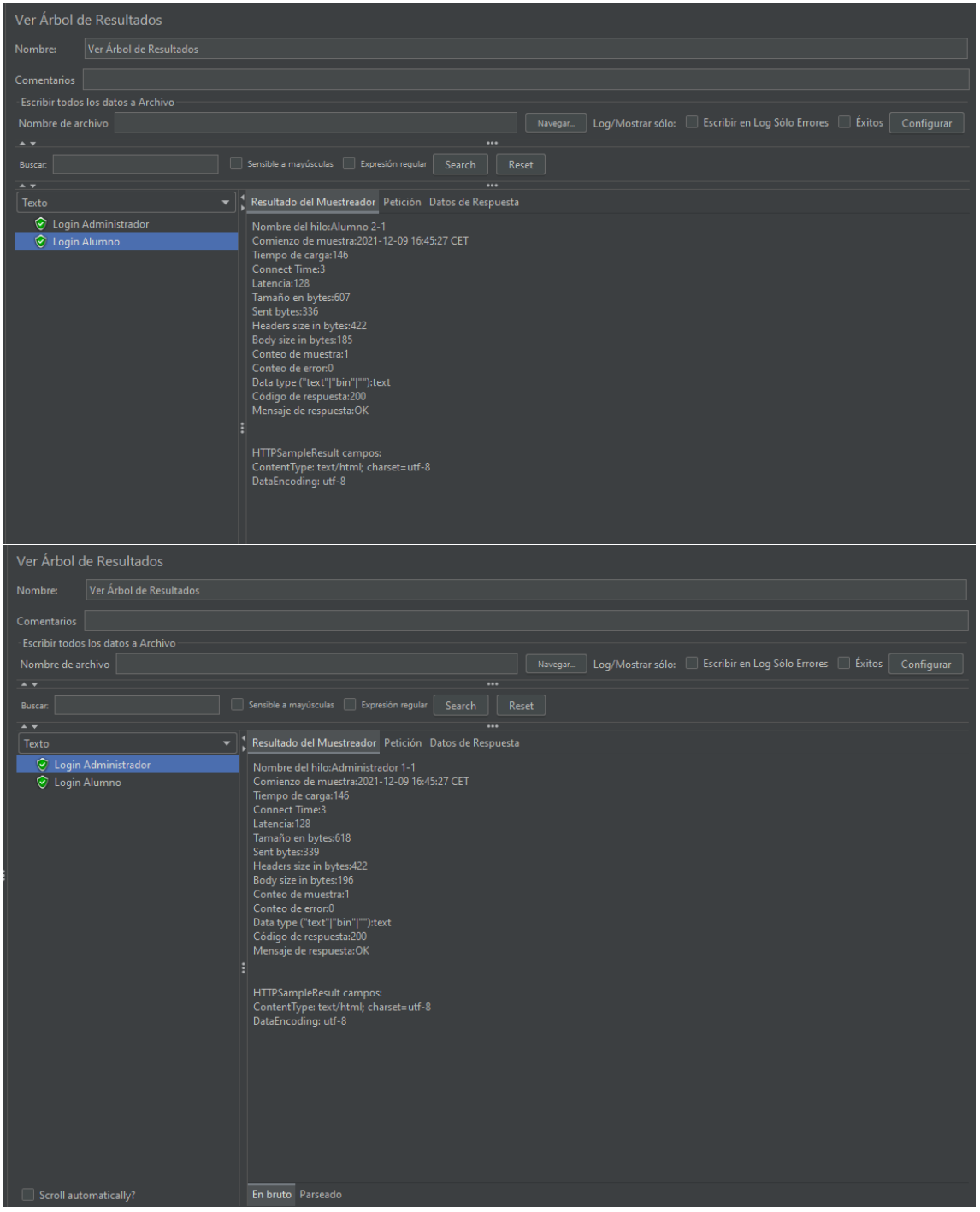


Lo configuramos:

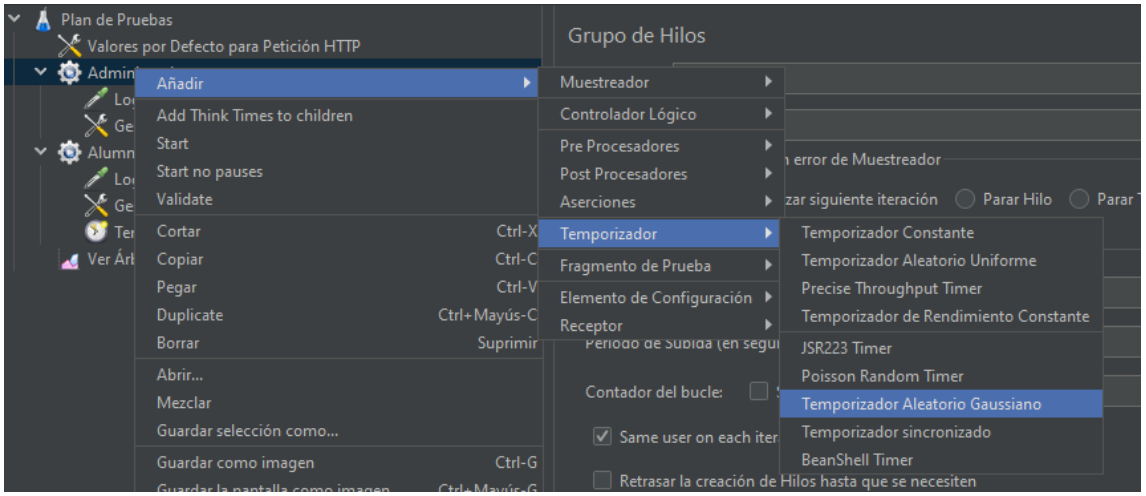


Podemos ver el nombre de usuario y la contraseña en el archivo mencionado anteriormente pruebaEntorno.sh.

Volvemos a arrancar JMeter, y esta vez habrán accedido correctamente:

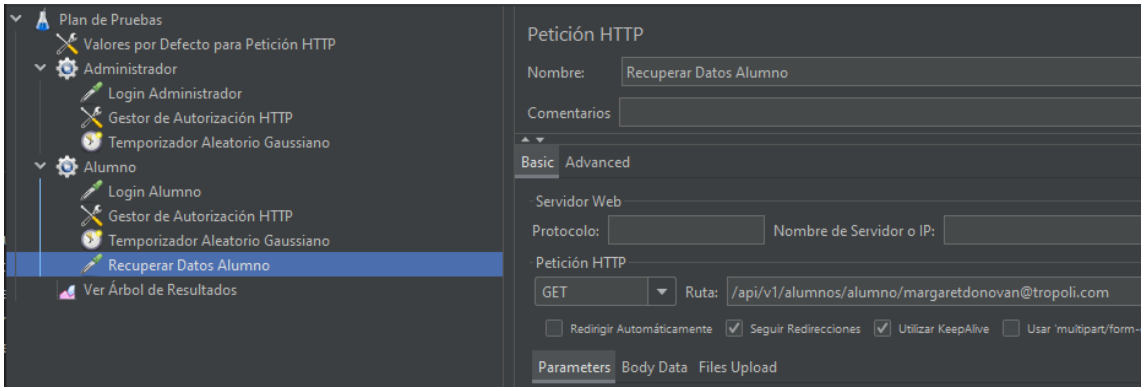


2.3. Añadimos esperas aleatorias (Gaussian Random Timer)



2.4. Recuperar datos alumno

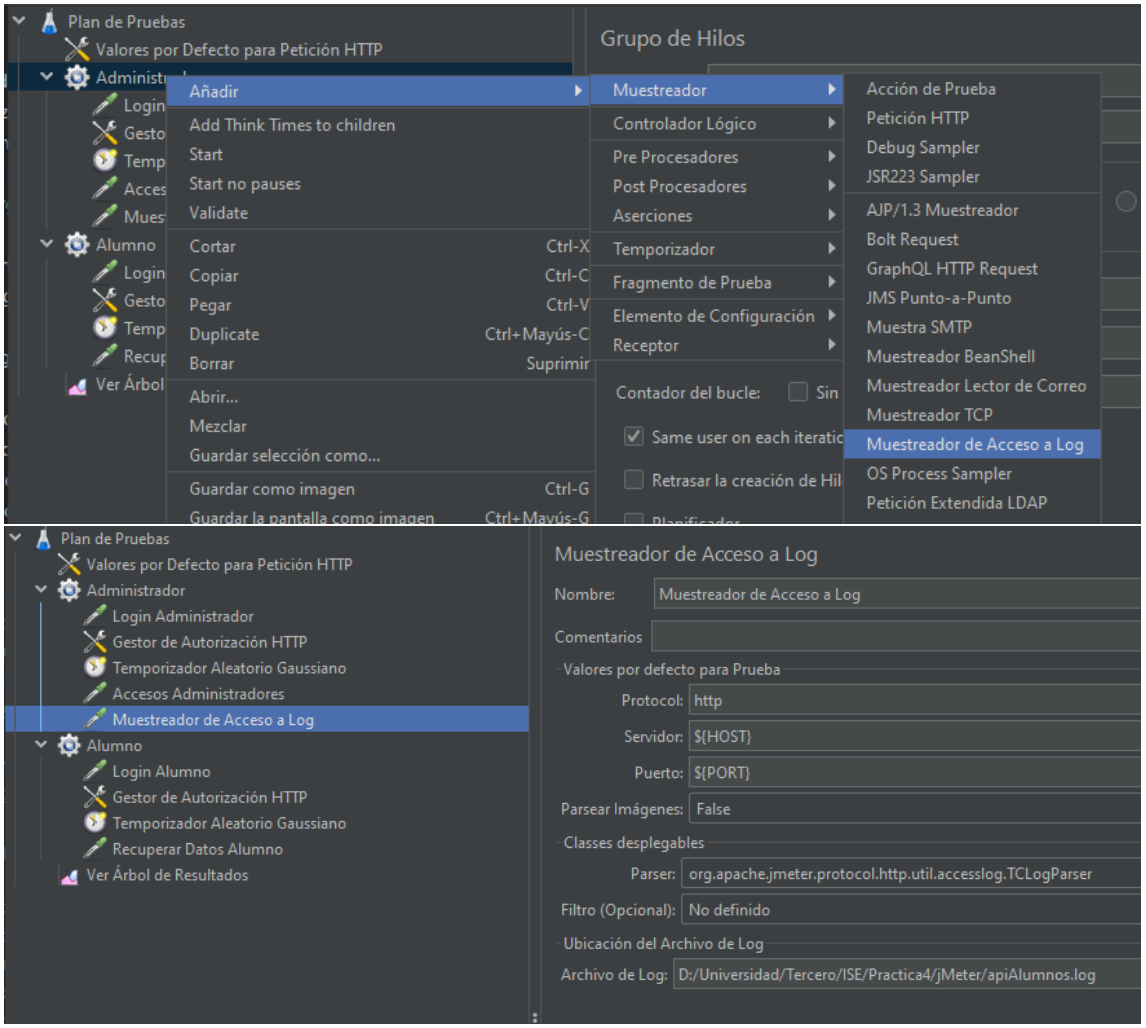
Tenemos que hacer una petición HTTP para recuperar los datos del alumno:



La ruta la encontraremos, una vez más, en el pruebaEntorno.sh (al final del mismo). Repetimos para el administrador (Acceso Administrador). En la ruta, deberemos poner el login de un administrador, ya que si usamos el de un alumno, nos dará error, ya que un alumno no puede consultar los datos de otro alumno.

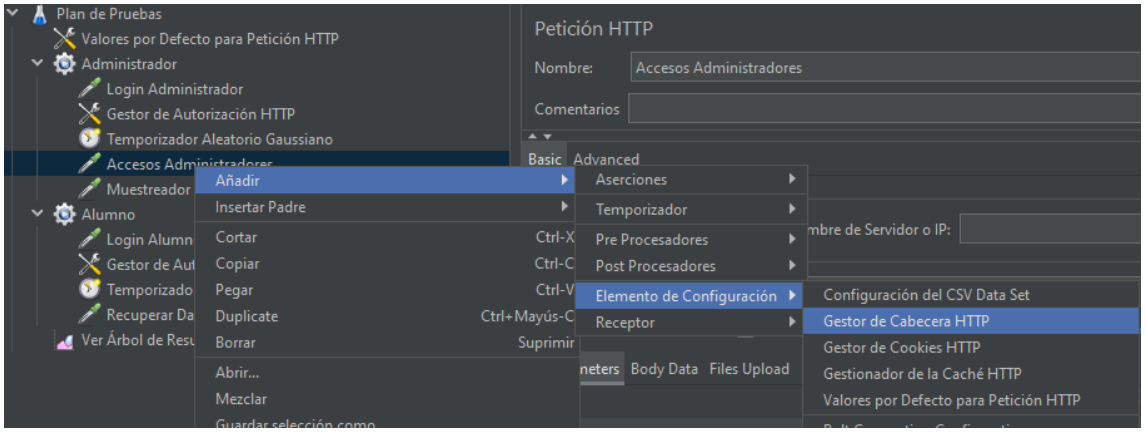
2.5. Muestreo recogido por apiAlumnos.log

Se nos pide que el muestreo para simular el acceso de los administradores lo debe coger el archivo apiAlumnos.log, usando un Muestreador de Acceso a Log:

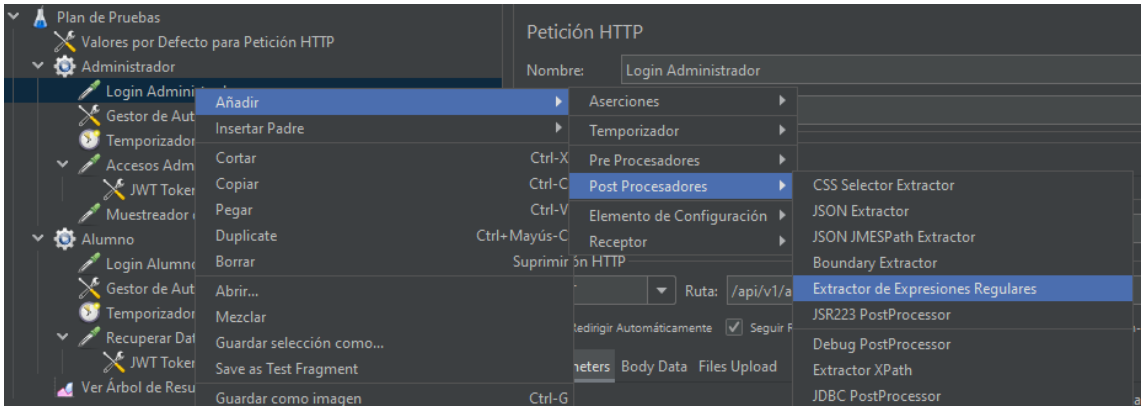


2.6. Usar expresión regular para extrar token JWT

Tenemos que utilizar una expresión regular para poder extraer el token JWT que hay que añadir a la cabecera de las peticiones. Usaremos el Gestor de Cabecera HTTP:



También tendremos que añadir un extractor de expresiones regulares para obtener el token:



Tendremos varios apartados que debemos configurar:

Nombre de referencia: Nombre de la variable en la que se guardará el texto extraído.

Expresión Regular: El patrón con el que el texto hará 'match'. Usaremos los siguientes carácteres especiales:

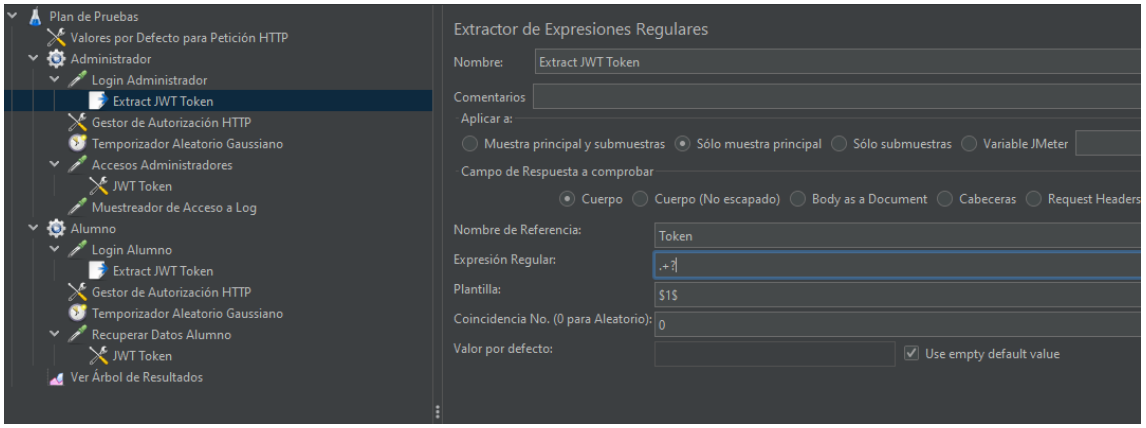
- **.** empareja cualquier caracter
- **+** uno o más veces
- **?** para cuando encuentra el primer 'match' exitoso

Plantilla: Agrupa los strings entre paréntesis. Con 1 selecciona el primero.

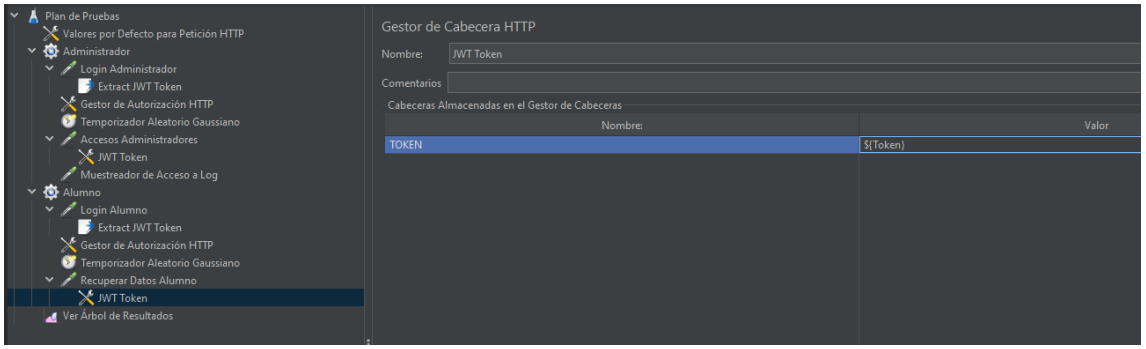
Coincidencia No.: Dice que coincidencia debe ser escogida. 0 para una aleatoria.

Valor por defecto: En caso de que no haya coincidencia, éste será el valor escogido.

Nos quedaría tal que así:



Y ahora configuramos los gestores de cabecera HTTP:



3. Ejercicio Opcional

Con esta información usted podría modificar los parámetros de configuración de Apache, PHP o MariaDB para observar un cambio en el comportamiento del servidor (CentOS o Ubuntu server) mediante la aplicación de un benchmark y analizando el cambio en las prestaciones o mediante el análisis de datos de monitorización ante una carga aplicada.

Referencias

[Descargar Phoronix] https://www.phoronix-test-suite.com/

[Instalar Phoronix Ubuntu] https://wiki.ubuntu.com/PhoronixTestSuite#installing

[OpenBenchmarking] https://openbenchmarking.org/s

[Instalar Docker] https://docs.docker.com/engine/install/

[Repositorio GitHub] https://github.com/davidPalomar-ugr/iseP4JMeter

[Apache JMeter] https://jmeter.apache.org/

[Usar JMeter] https://jmeter.apache.org/usermanual/index.html