Ingeniería de Servidores (2019-20)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 4: *Benchmarking* y ajuste del sistema (*tuning*)

Alonso Bueno Herrero

8 de agosto de 2020

Índice

Referencias

[1] `https://www.cedric-dumont.com/2017/02/01/install-apache-benchmarking-tool-ab-on-windows/`

[2] Página oficial del manual de *ab*: `http://httpd.apache.org/docs/current/programs/ab.html`

[3] `http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/388`

1. Introducción

Esta práctica está centrada en dos tareas de servidores importantes: benchmarking y ajuste del sistema. La primera parte es la que se solicitaba al alumno, y es precisamente dicho trabajo el que se expone, al completo, a continuación; por su parte, la sección de .ªjuste del sistema.º tuning, era opcional y puede consultarse en el guión oficial de la práctica.

Recordemos que un ***bechmark*** no es otra cosa que un test (un conjunto de programas de prueba) para medir y comparar el rendimiento de dos máquinas.

Existen benchmark para medir cosas muy diversas. En la web `https://sourceforge.net/directory/os:linux/?q=benchmark` se puede encontrar una lista muy amplia (en inglés) de *benchmark*s famosos.

2. *Benchmarks* con Phoronix y `ab`

2.1. *Benchmarks* con `phoronix-test-suite`

Este benchmark es realmente una suite que da acceso a varios conjuntos *benchmark* para testear varios índices del servidor (RAM, CPU, periféricos, etc.).

Se ha realizado la prueba con dos programas de los proporcionado por Phoronix: **gnupg** y **Dolfyn** (ambos dos miden el rendimiento de la CPU).

Primeramente, vamos a instalar, tal y como se ha indicado en clase, la suite de *benchmarks* **Phoronix**. Los pasos han sido los siguientes en centOS y luego en Ubuntu:

1. En primer lugar, vamos a la MV ***centOS***, y entramos en modo *root*.
2. Buscaremos el paquete de la suite Phoronix con el gestor de paquetes `yum` usando la opción:

   ```
   yum search phoronix
   ```

   y nos indica que el único paquete asociado a este nombre es `phoronix-test-suite` (el nombre parece tener sentido con lo que queremos).
3. Instalamos el paquete encontrado con:

   ```
   yum install phoronix-test-suite
   ```

4. Vamos al manual de phoronix con la orden

   ```
   man phoronix-test-suite  # entrada del manual en línea para phoronix
   ```

   para echar un ojo a algunas opciones.
5. Elegimos (por ejemplo) listar los programas de *benchmark* disponibles, para lo cual ejecutamos

   ```
   phoronix-test-suite list-available-test    # listar tests disponibles
   ```

   lo cual nos vuelca en pantalla todos los test disponibles en la *suite*.
6. Para instalar **Phoronix en Ubuntu** hay que ejecutar el comando

   ```
   sudo apt install phoronix-test-suite
   ```

Para ejecutar un *benchmark* cualquiera de la suite que nos ocupa necesitamos previamente instalarlo, pues en definitiva se trata del código de un programa que no está nativamente en nuestro ordenador.

Con el comando

```
phoronix-test-suite install <nombre-benchmark>
```

se puede instalar el programa `nombre-benchmark`.

Para ejecutarlo lanzaremos el comando

```
phoronix-test-suite benchmark <nombre-benchmark>
```

Prueba 1 Ahora vamos a ejecutar un programa benchmark de la suite y ver los resultados obtenidos.

En primer lugar, ejecutamos[1]

```
phoronix-test-suite benchmark pts/gnupg
```

y al acabar nos ofrece, si todo ha ido bien, una información parecida a la de la Figure 1, si estamos en centOS y como la de la Figure 2 si lo hemos ejecutado en Ubuntu.

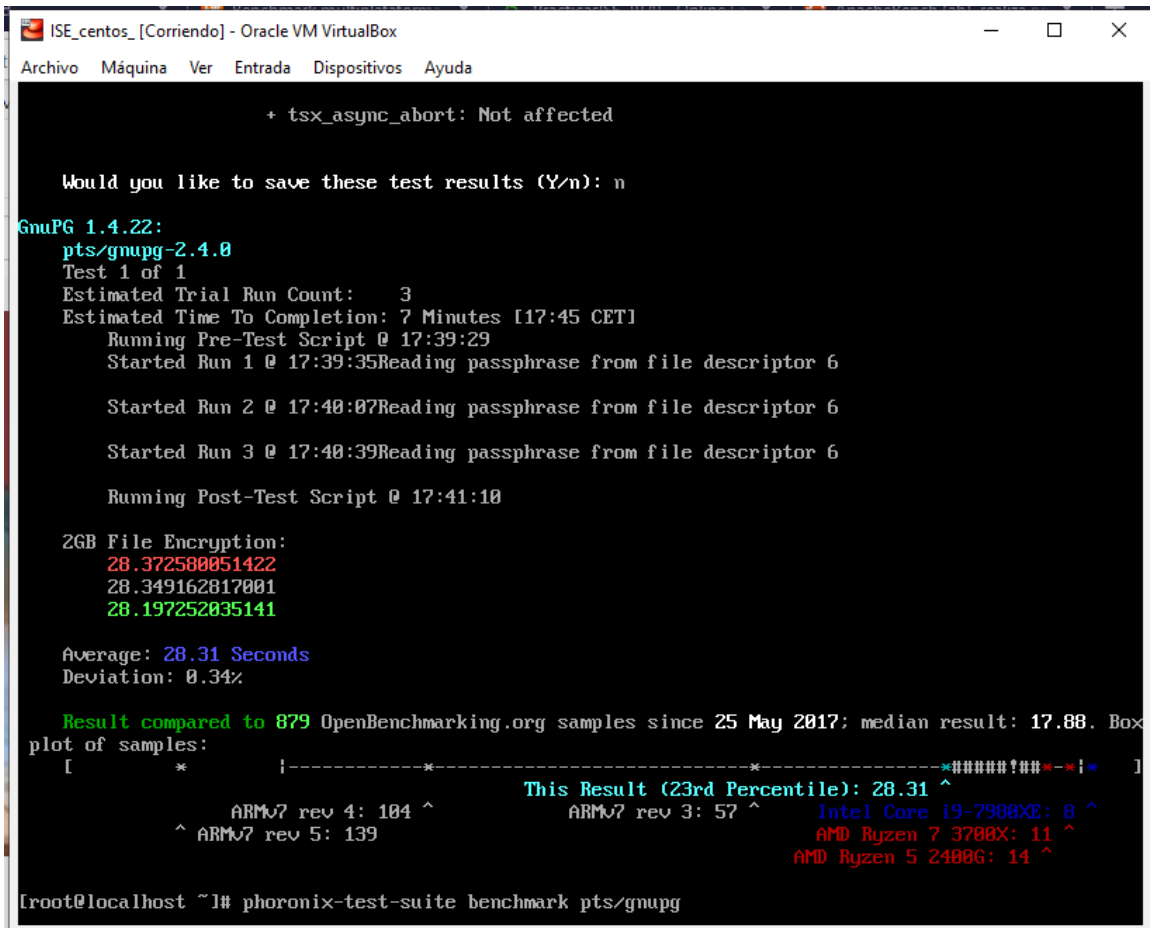

Figura 1: Resultado de una prueba con *gnupg* (Phoronix) en centOS.

A partir de los resultados que nos ofrecen la Figure 1 y la Figure 2, podemos hacer una serie de comparaciones sobre la carga (`Average`) a la que se ha sometido a ambas máquinas y ver el porcentaje de CPU que ha consumido la prueba en cada una. En este caso podemos ver que centOS ha sido más rápido (`Average = 28,31 Seconds`) frente a Ubuntu (`Average = 55,93 Seconds`).

2.2. La suite *ab*

En este caso vamos a utilizar la utilidad de *benchmarks* de Apache (**ab**) para las tareas de monitorización en los servidores. **ab** es una suite básica que nos permite hacer pruebas sobre un servidor HTTP especificándole cosas como el número de peticiones que se hacen, si en cada «iteración» se hacen n de manera concurrente, etc... y obviamente la IP del servidor.

[1]Podemos encontrar más información sobre este benchmark en la web de referencia `https://openbenchmarking.org/`.

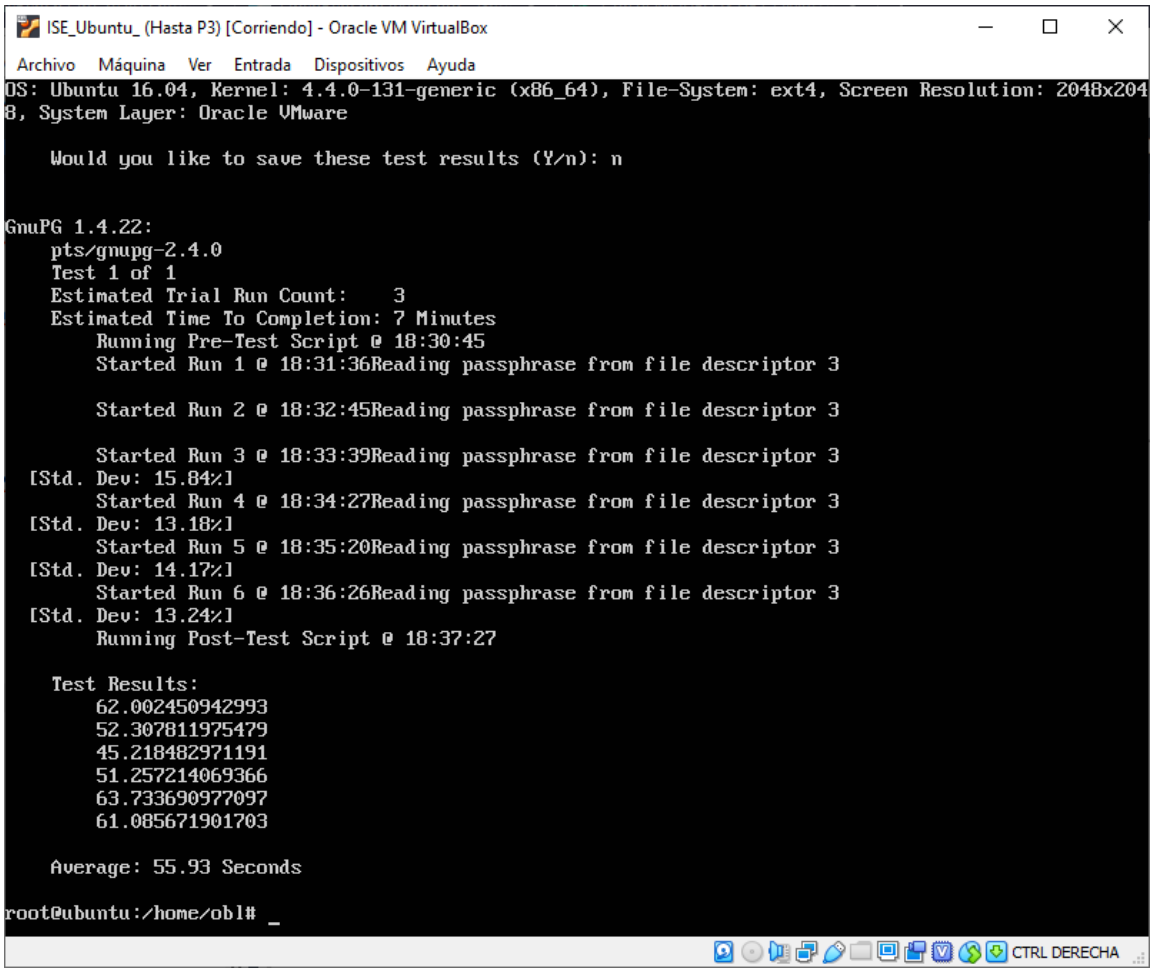

Figura 2: Resultado de una prueba con *gnupg* (Phoronix) en Ubuntu.

Algunas de las cuestiones que hay que tener en cuenta es el porqué de instalarlo en Windows. Esto se debe esencialmente a que necesitamos un *host* (tercera máquina) que nos haga las veces de servidor de benchmarks, tal que sea éste el que lance las pruebas sobre nuestros servidores (Ubuntu y centOS).

2.2.1. Pruebas con *ab* sobre las máquinas virtuales

Para lanzar un benchmark con **ab** ejecutaremos una orden del siguiente tipo (comandos básicos):

```
ab -n 5 -c 4 http://192.168.56.XXX/  # XXX = 105 para Ubuntu, 110 para centOS
```

donde:

`-n` indica el número de peticiones (*requests*) HTTP a realizar;

`-c` indica el grado de concurrencia de la prueba, es decir, el número de peticiones que se van a lanzar a la vez,

`http://192.168.56.XXX/` es la IP del servidor al que queremos que se ejecuten las pruebas.

Podemos encontrar el resto de opciones en la referencia (2)

Prueba 2 (Ejecución de `ab` sobre Ubuntu Server) *Tras ejecutar el comando anterior sobre Ubuntu Server, la salida por pantalla, que ha sido inmediata, es:*

```
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

```
Benchmarking 192.168.56.105 (be patient).....done


Server Software:        Apache/2.4.18
Server Hostname:        192.168.56.105
Server Port:            80

Document Path:          /
Document Length:        11321 bytes

Concurrency Level:      4
Time taken for tests:   0.009 seconds
Complete requests:      5
Failed requests:        0
Total transferred:      57975 bytes
HTML transferred:       56605 bytes
Requests per second:    569.28 [#/sec] (mean)
Time per request:       7.026 [ms] (mean)
Time per request:       1.757 [ms] (mean, across all concurrent requests)
Transfer rate:          6446.11 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   0.9      0       2
Processing:     2    4   1.8      4       7
Waiting:        1    3   2.2      3       7
Total:          3    4   1.5      4       7
WARNING: The median and mean for the initial connection time are not within a normal deviation
        These results are probably not that reliable.

Percentage of the requests served within a certain time (ms)
  50%      4
  66%      5
  75%      5
  80%      7
  90%      7
  95%      7
  98%      7
  99%      7
 100%      7 (longest request)
```

Prueba 3 (Ejecución de ab sobre centOS) *Para centOS, la salida ha sido, tras ejecutar el comando con* $n = 10,\ c = 10$*:*

```
Concurrency Level:      10
Time taken for tests:   0.712 seconds
Complete requests:      10
Failed requests:        0
Non-2xx responses:      10
Total transferred:      51790 bytes
HTML transferred:       48970 bytes
Requests per second:    14.05 [#/sec] (mean)
```

```
Time per request:       711.703 [ms] (mean)
Time per request:       71.170 [ms] (mean, across all concurrent requests)
Transfer rate:          71.06 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1    2   1.9      1       7
Processing:    17   91 215.9     24     706
Waiting:       17   88 205.5     23     673
Total:         18   93 215.7     25     707

Percentage of the requests served within a certain time (ms)
  50%     25
  66%     25
  75%     25
  80%     36
  90%    707
  95%    707
  98%    707
  99%    707
 100%    707 (longest request)
```

2.2.2. Conclusiones sobre las pruebas con `ab`

A partir de los resultados obtenidos en los dos volcados de pantalla de `ab`, podremos ver qué prueba ha tardado menos, cuál ha sido el tiempo dedicado a cada petición, y los milisegundos dedicados a la conexión, al procesamiento, etc.

3. *Benchmarking* de una API Rest con jmeter

En este caso vamos a hacer pruebas sobre una API Rest basada en contenedores (*dockers*). Para ello:

- Se usará la herramienta `docker-compose` para las tareas de *orquestación* de los contenedores, que se ejecutan sobre nuestra máquina virtual-servidor Ubuntu.
- El *software* de monitorización será **jmeter**, el cual instalaremos en windows y programaremos desde ahí nuestro test.

3.1. Instalando `docker/docker-compose` en Ubuntu Server

Se trata de instalar en nuestro servidor Ubuntu todo el cuerpo de la API que será sujeto de nuestras pruebas Los pasos para esta tarea han sido muy sencillos:

1. Arrancamos la máquina virtual de Ubuntu y entramos en modo root.
2. Seguimos los pasos indicados en el guión de prácticas para instalar:
   a) docker
   b) docker-compose
   c) la API de la aplicación **iseP4JMeter**, desde *github*.

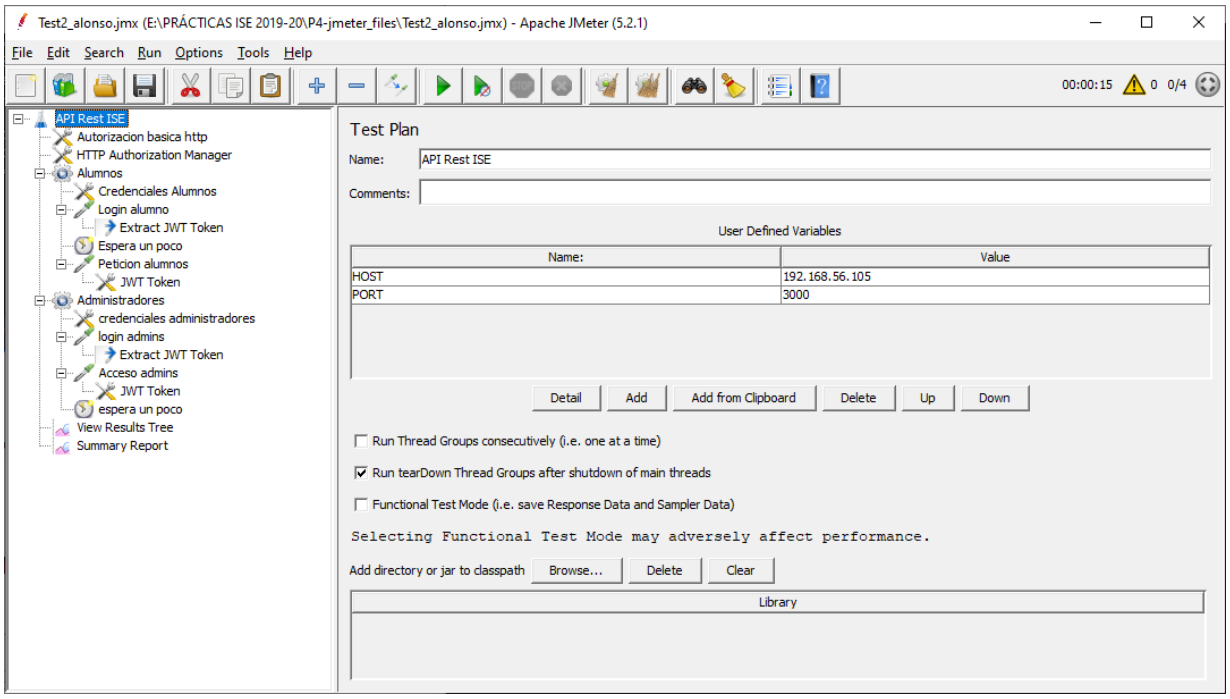

Figura 3: Captura de jmeter con el test completo que he realizado.

3.2. Programación del test con jmeter

El test con **jmeter** se ha programado siguiendo las indicaciones dadas en el fichero decriptivo de github de David Palomar. En concreto, se nos pedía que el Test Plan tuviese una estructura concreta, como la que he implementado en la Figura 3.

Para poder **lanzar el benchmark** sobre la API que queremos estudiar se han seguidos los siguientes pasos:

1. Tras haber importado el repositorio de iseP4JMeter desde github, ahora vamos a lanzar la API:
2. Nos vamos a la carpeta iseP4JMeter,
3. ejecutamos `$>docker-compose up`
4. Tras *levantarse* la aplicación y quedarse en escucha, damos a la opción *ejecutar* desde *jmeter*, teniendo nuestro plan cargado en dicho programa.

Tras ejecutarse al completo el test, podremos ir a los *listeners* que le hemos añadido para ver los resultados (ver figuras 17 y 18).

En las siguientes figuras (fig. 4 a la fig. 18) se pueden ir viendo capturas en jmeter de los distintos elementos del test. Sobre dichas figuras, cabe hacer algún breve comentario:

- La captura correspondiente al elemento tipo ***espera aleatoria*** (*timer*) sólo se ha mostrado una vez porque se aplica igual tanto a la hebra de los alumnos como a la de los administradores.
- Lo mismo que en el punto anterior sucede con lo extractores de expresiones regulares (**Extract JWT Token**), pues como el login de un alumno/administrador se realiza de la misma forma desde el benchmark, el extractor es igual.
- Lo mismo pasa para los **HTTP Header Manager** (JWT Token), que se encargar de añadir a la cabecera de los GET el token con la información del usuarios que hace la petición.
- **En las dos últimas figuras se incluyen algunos de los denominados *listeners* de jmeter, que contienen los resultados de una prueba correcta que realizado.**

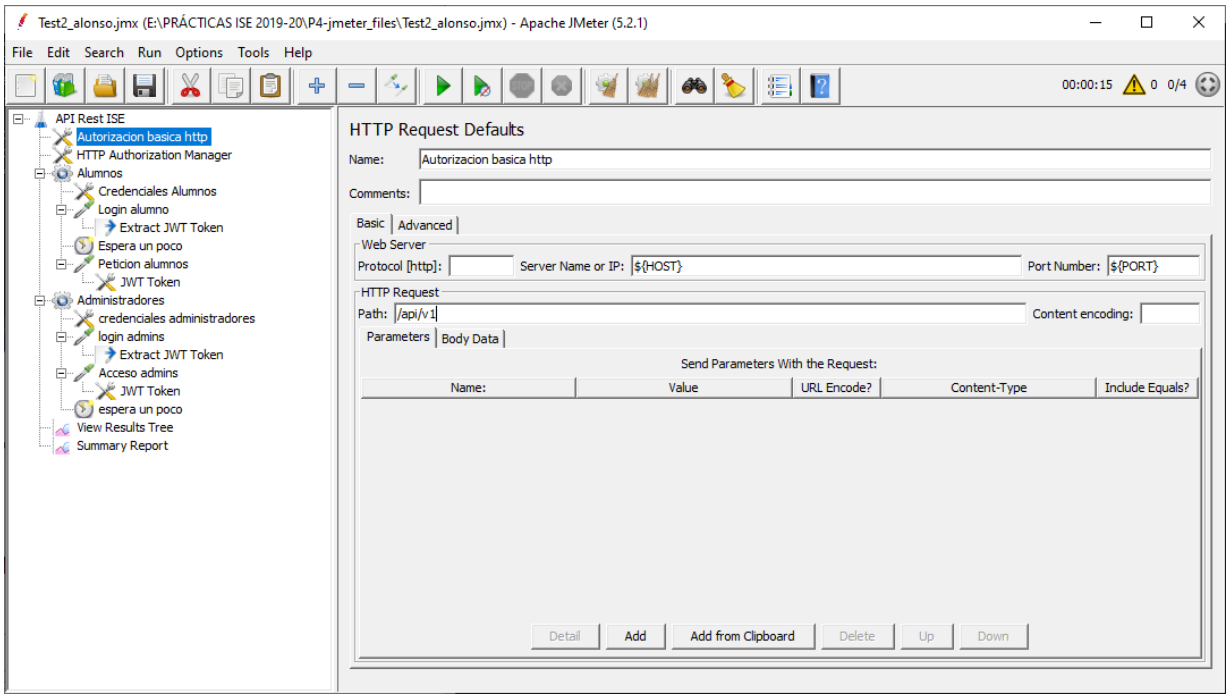

Figura 4: Elemento para establecer los valores por defecto de las conexiones.

Referencias

[1] `https://www.cedric-dumont.com/2017/02/01/install-apache-benchmarking-tool-ab-on-windows/`

[2] Página oficial del manual de *ab*: `http://httpd.apache.org/docs/current/programs/ab.html`

[3] `http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/388`

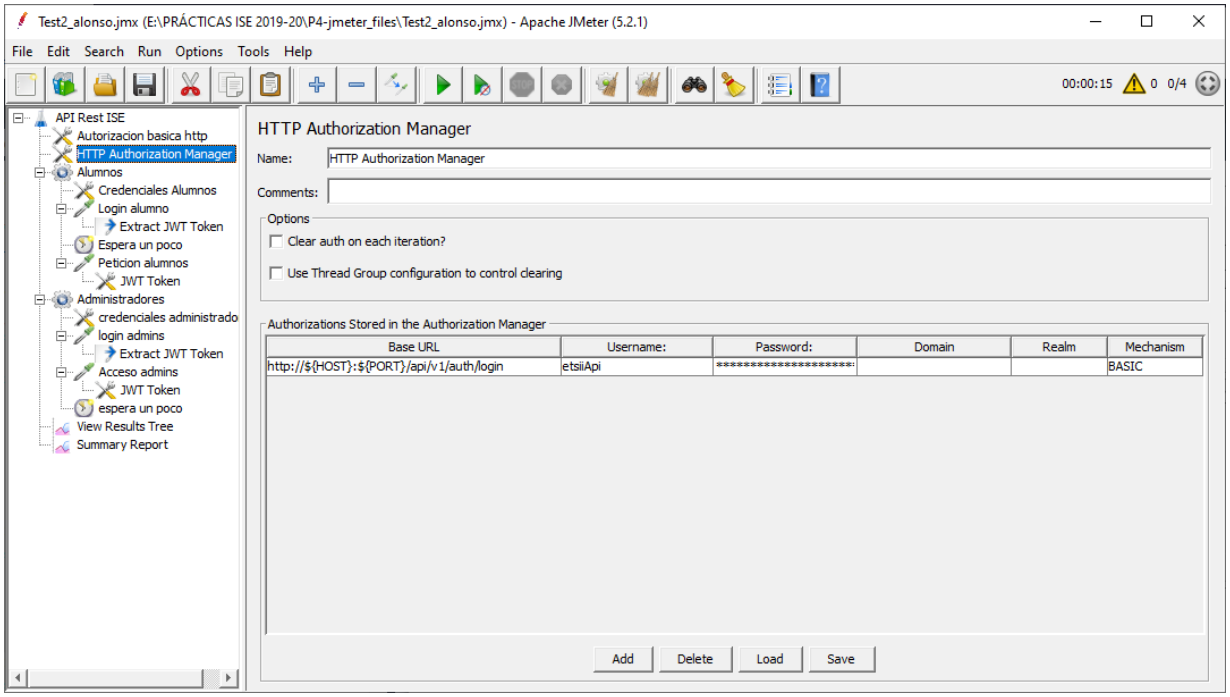

Figura 5: Para establecer la autorización básica.

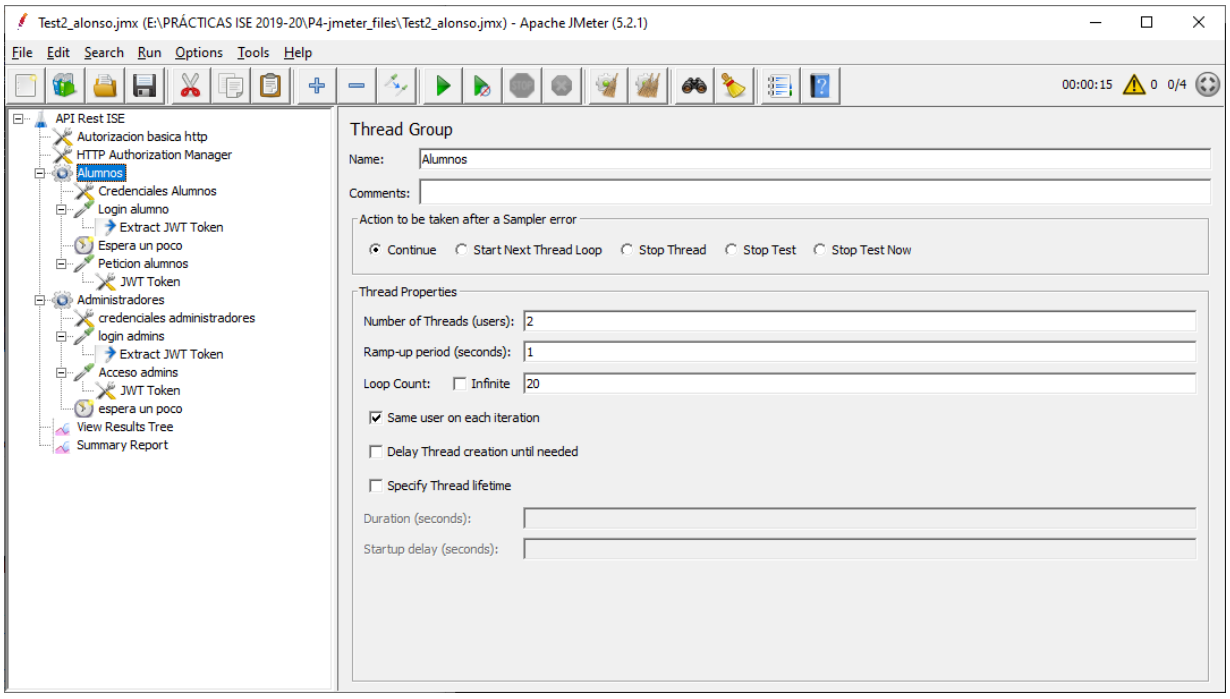

Figura 6: Configurando los parámetros básicos de la hebra de los alumnos.

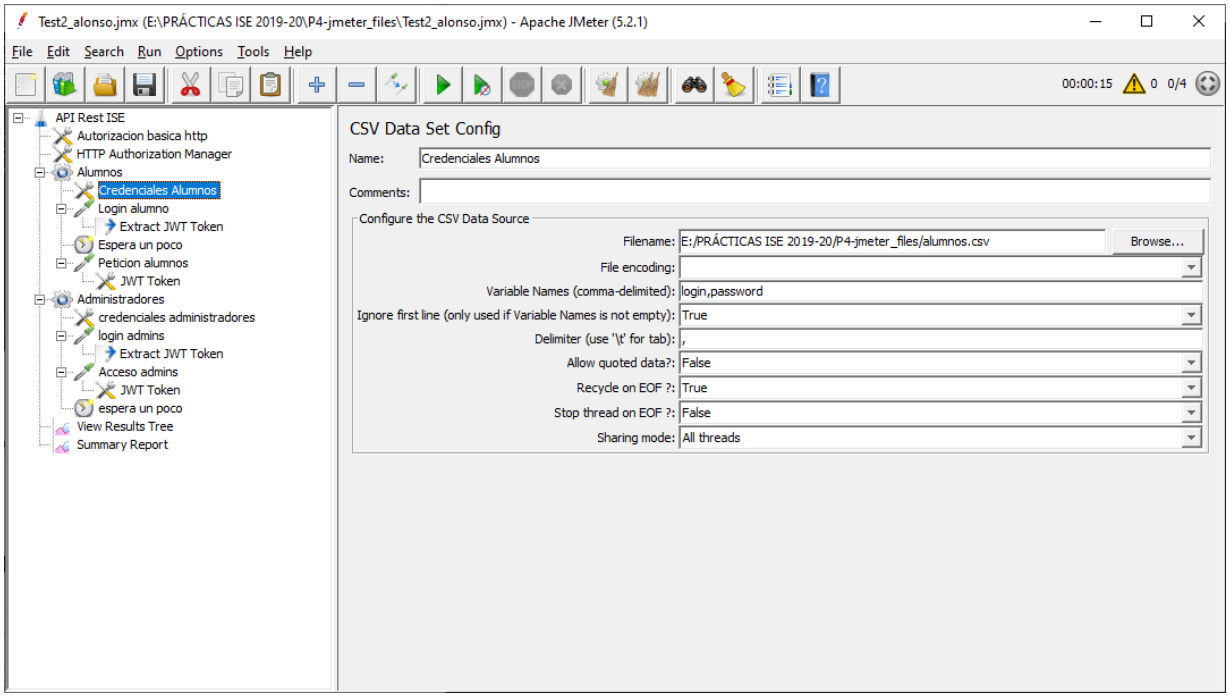

Figura 7: Estableciendo el archivo y la captura de credenciales de alumnos desde fichero con un objeto tipo CSV Data Config.

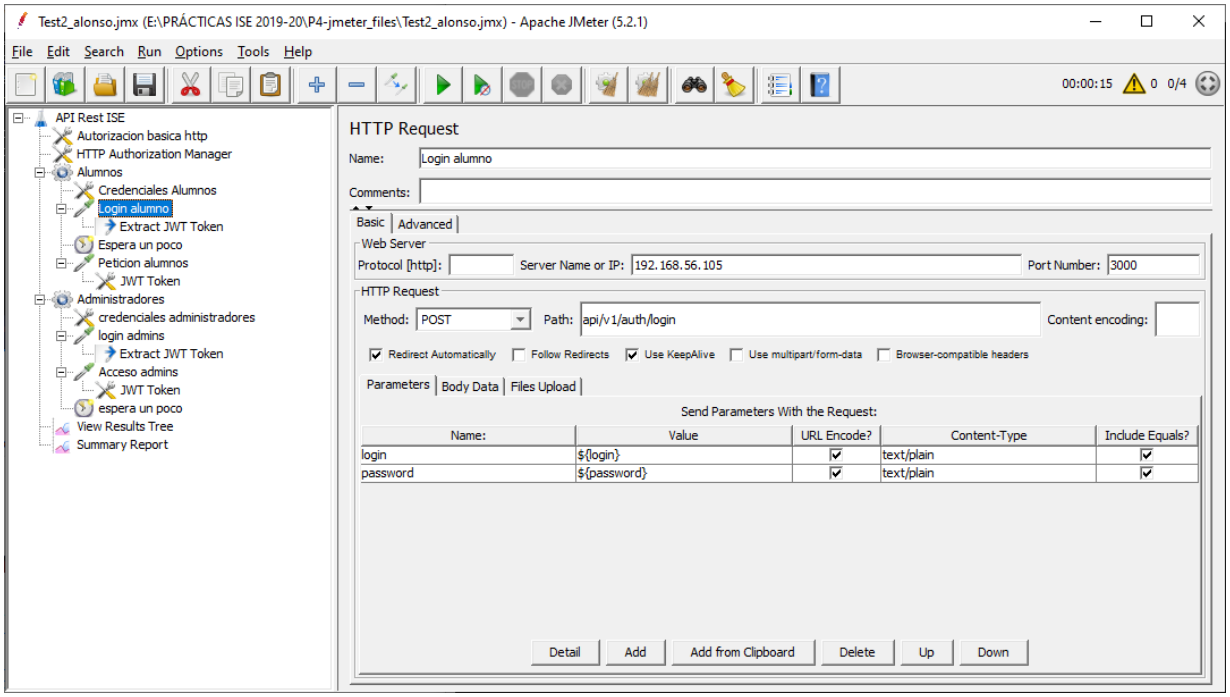

Figura 8: Configurando los *login* (HTTP Request tipo GET).

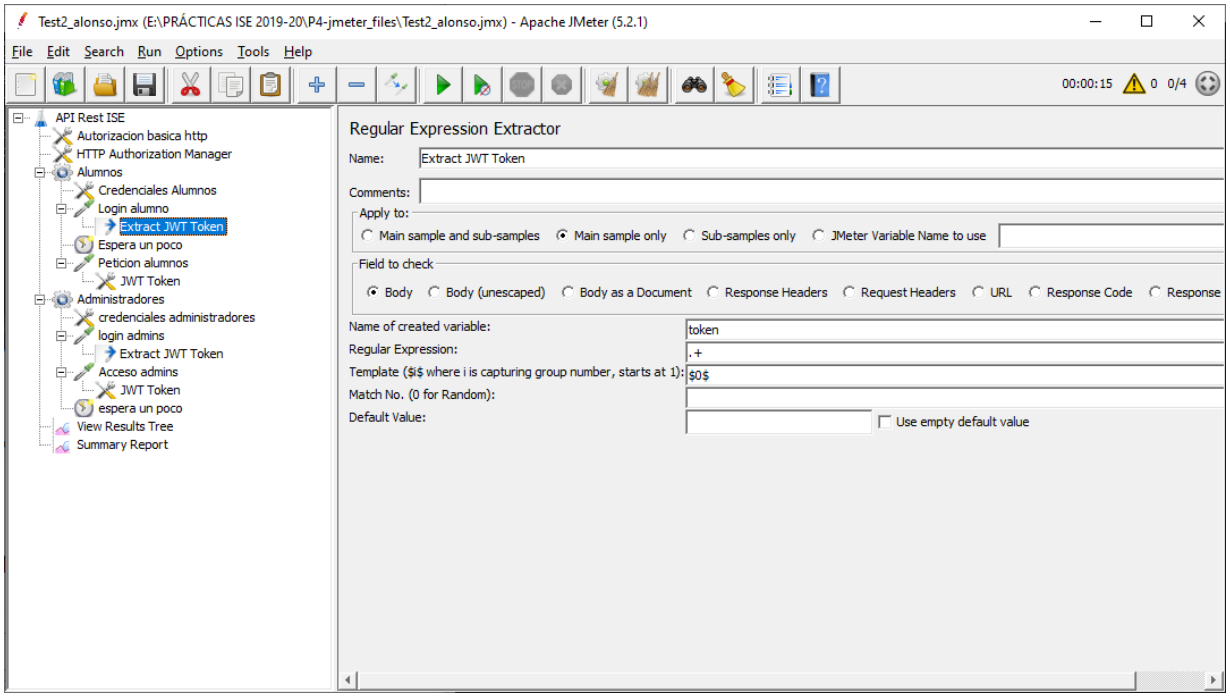

Figura 9: Extractor de expresiones regulares..

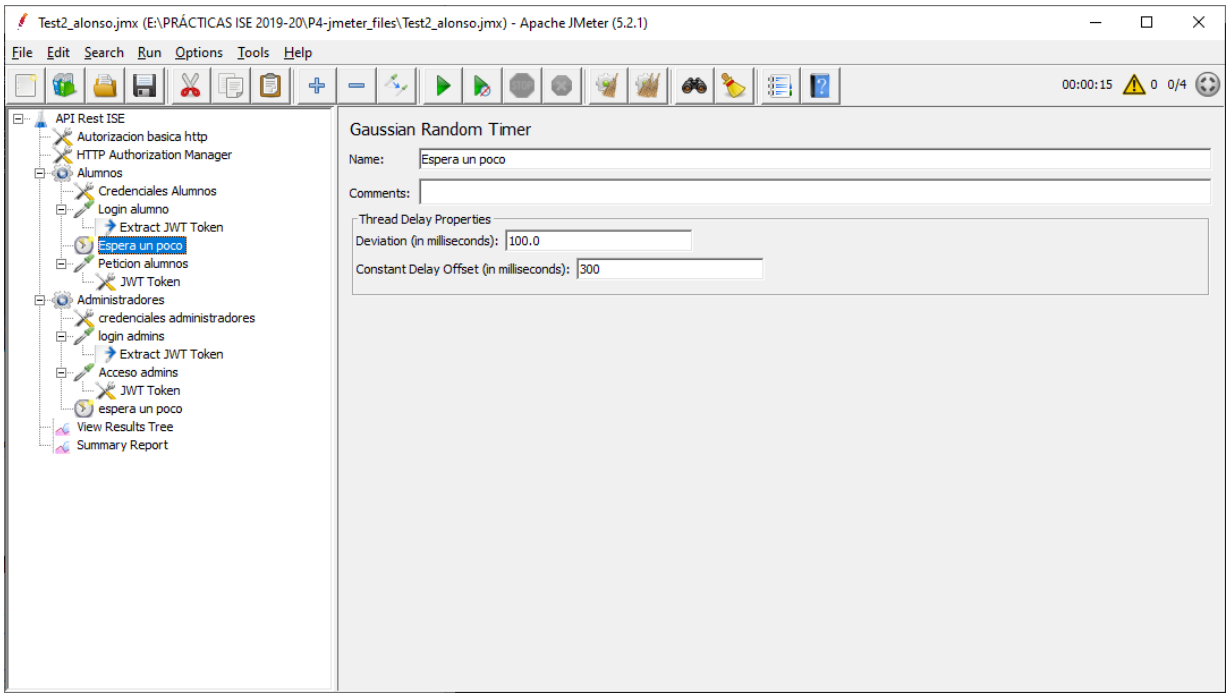

Figura 10: Esperas aleatorias tras el *login*

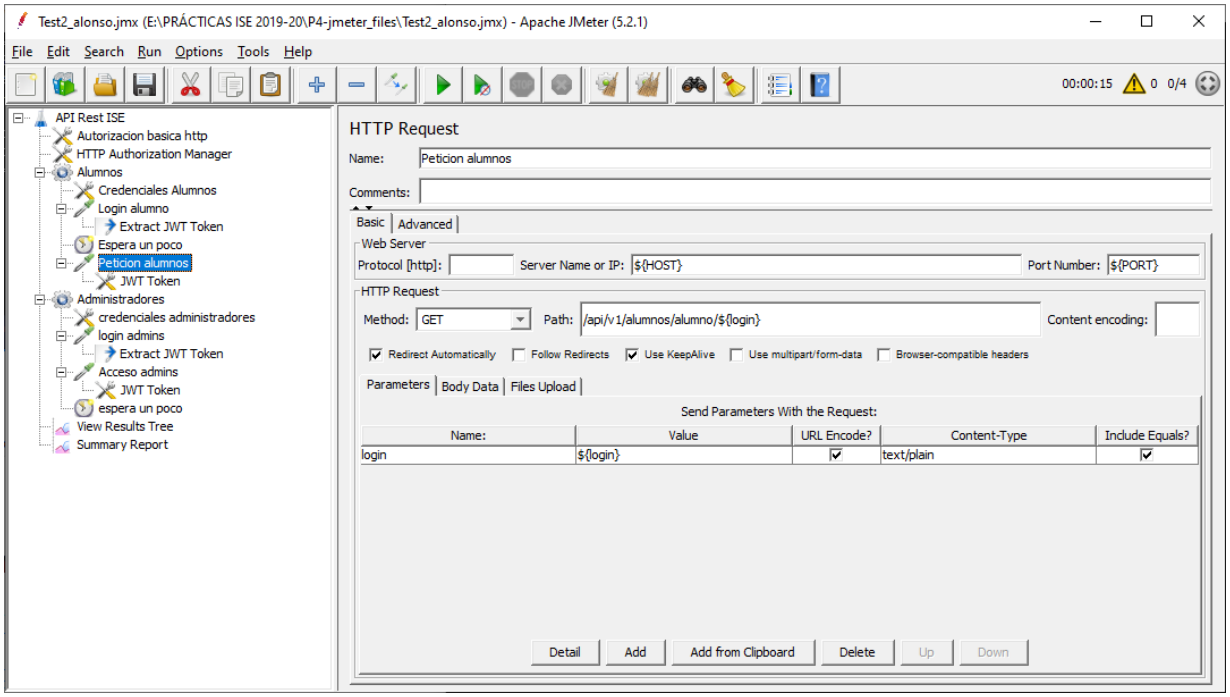

Figura 11: Configurando los GET de los alumnos.

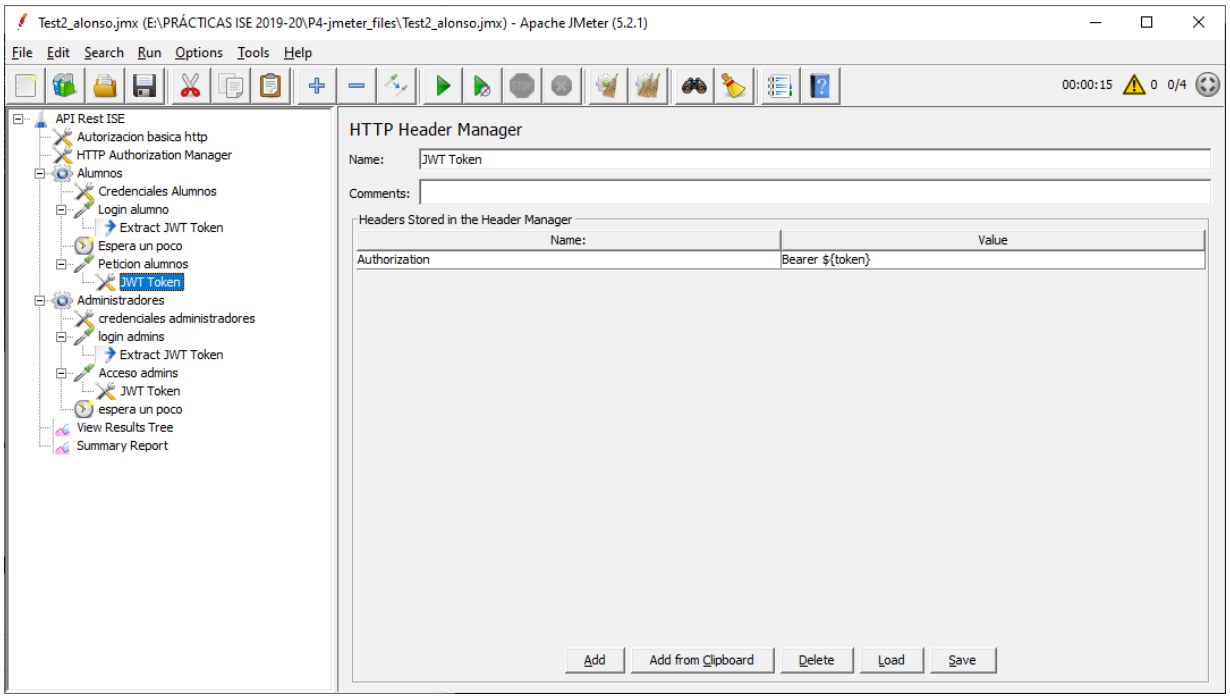

Figura 12: Añadiendo a los mensajes GET el token para acreditar la autenticación del que hace la petición.

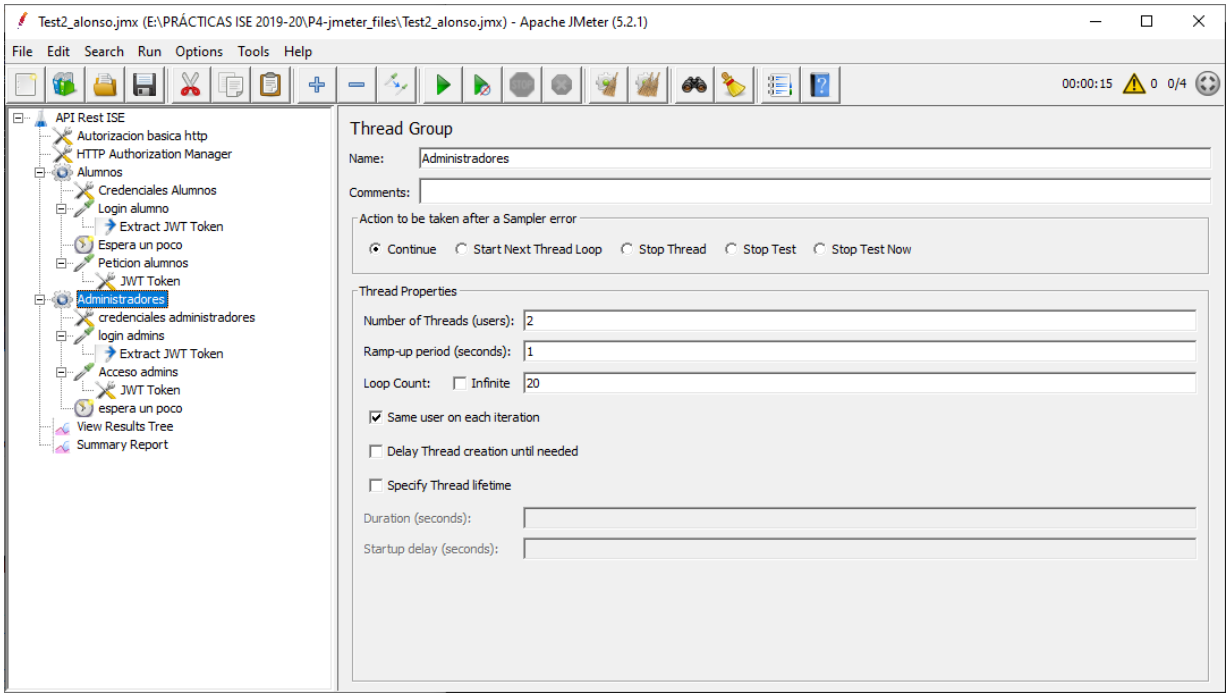

Figura 13: Configurando los aspectos básicos de la hebra Administradores.

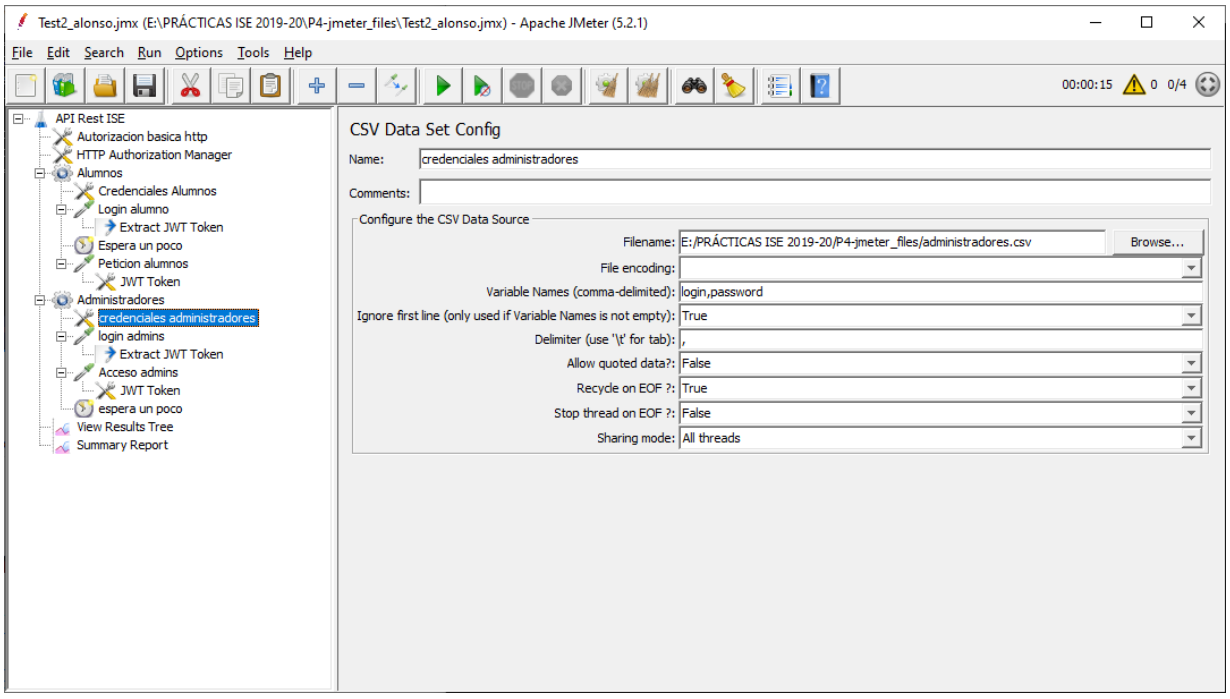

Figura 14: Credenciales de los administradores con un objeto tipo CSV Data Config

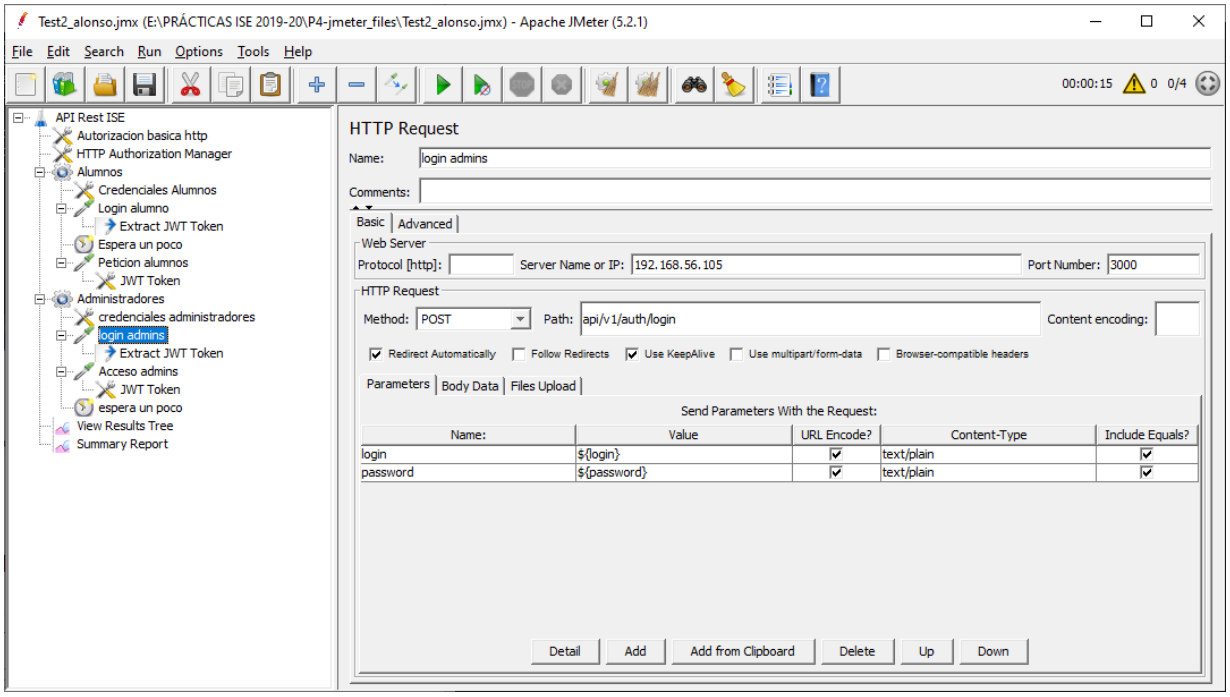

Figura 15: Login de los administradores (HTTP Request)

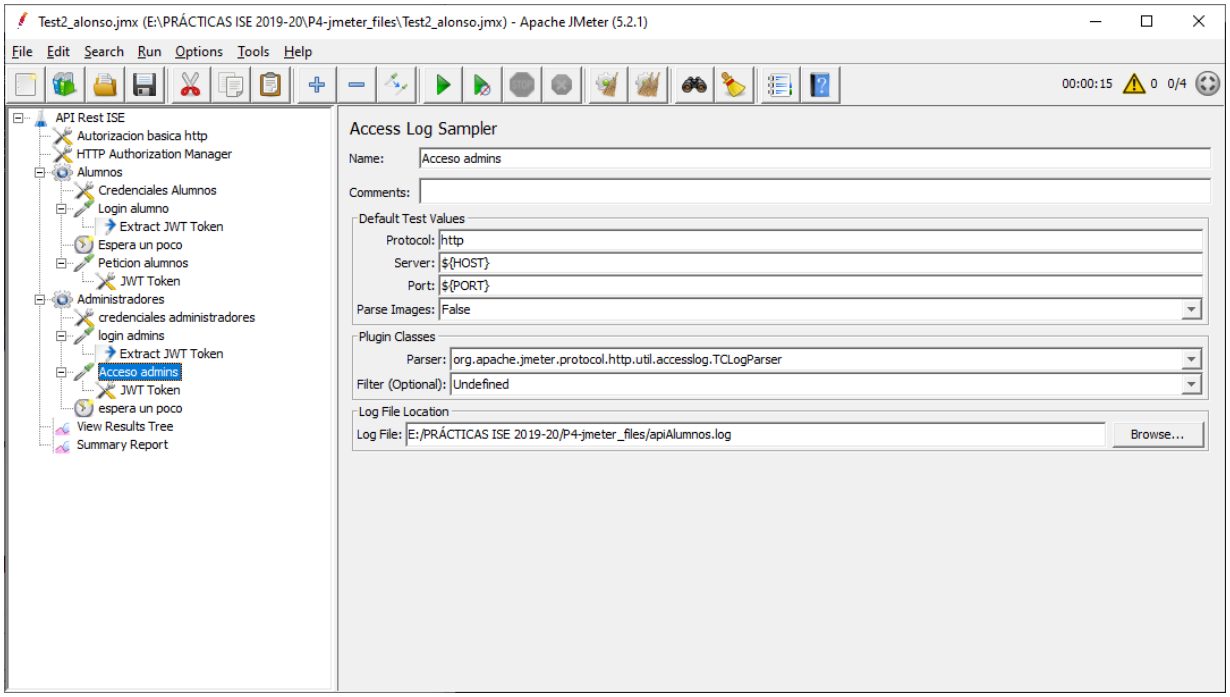

Figura 16: Configurando los GET de los administradores para que se vuelquen a un archivo de salida (log) con un elemento tipo Access Log Sampler

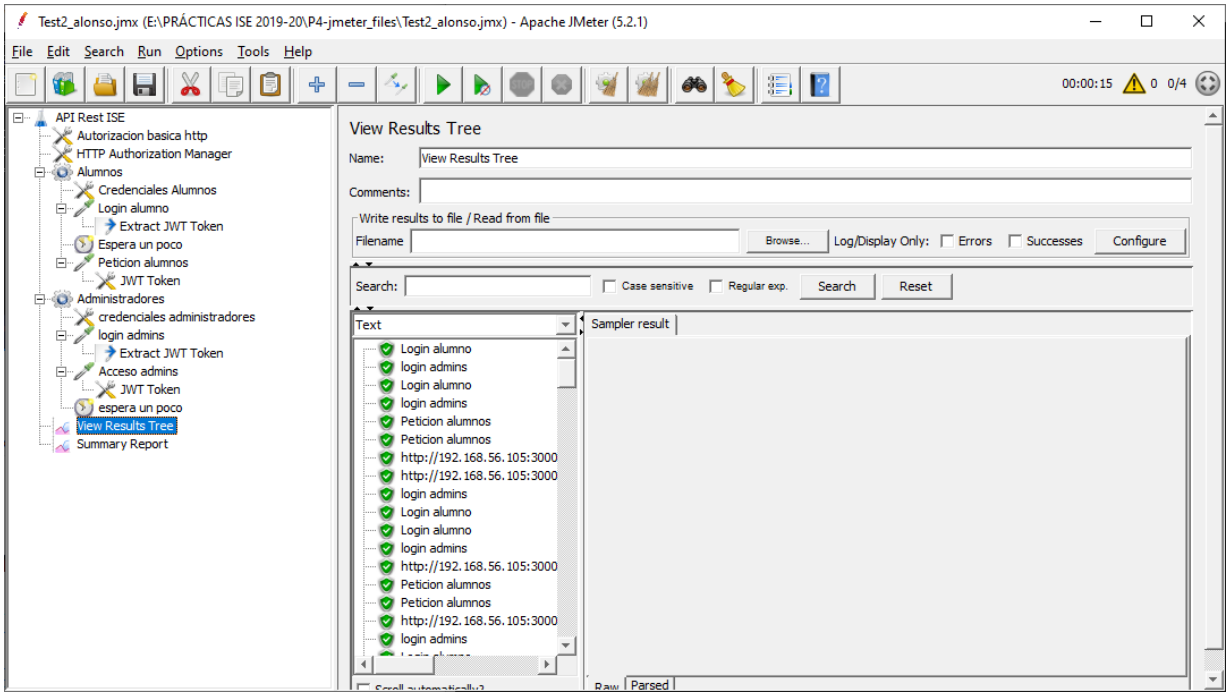

Figura 17: Árbol de resultados (primer *listener*).

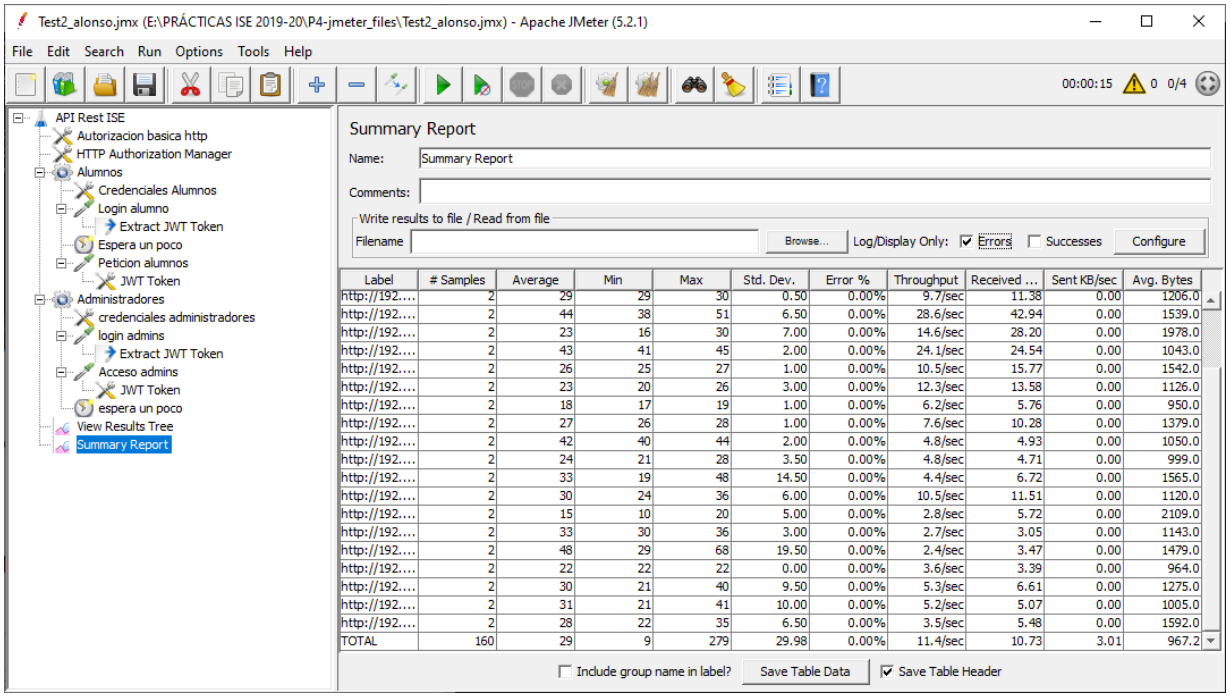

Figura 18: Tabla de resultados de las pruebas (segundo *listener*).