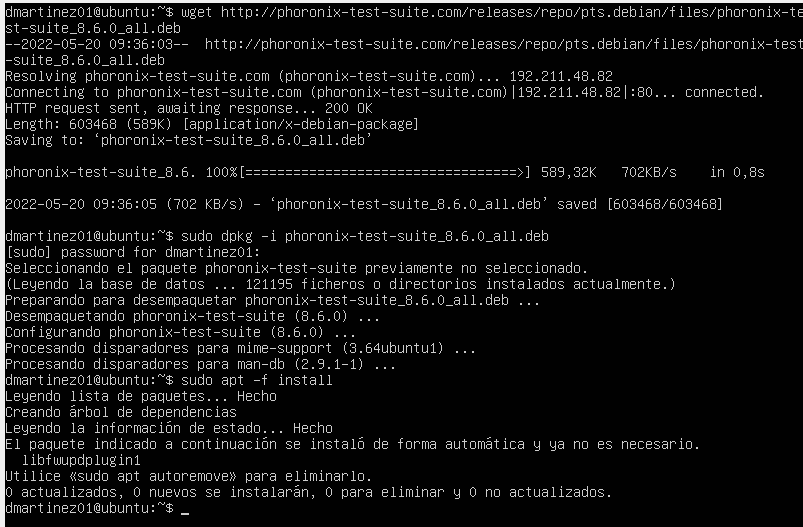
PRACTICA 4: MEMORIA

*David Martínez Diaz GII-ADE*

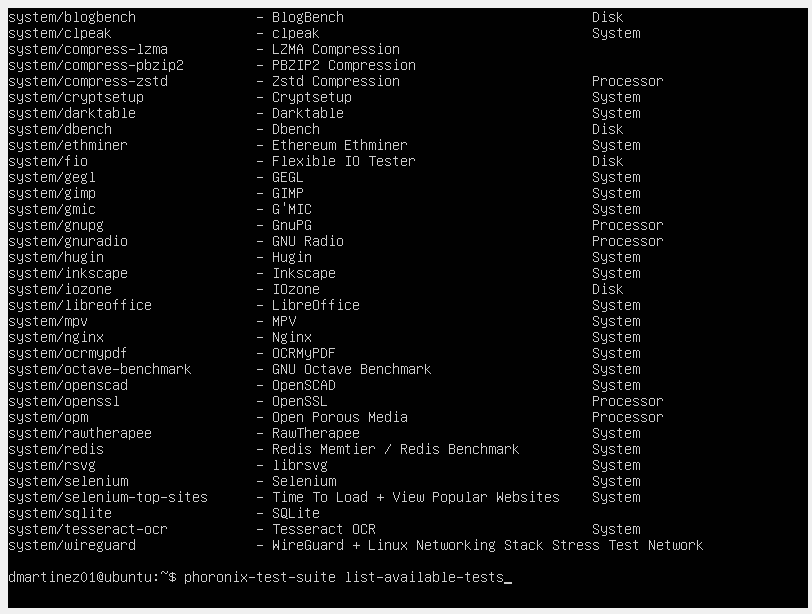
## EJERCICIO 1: Benchmarks

### Phoronix en Ubuntu:

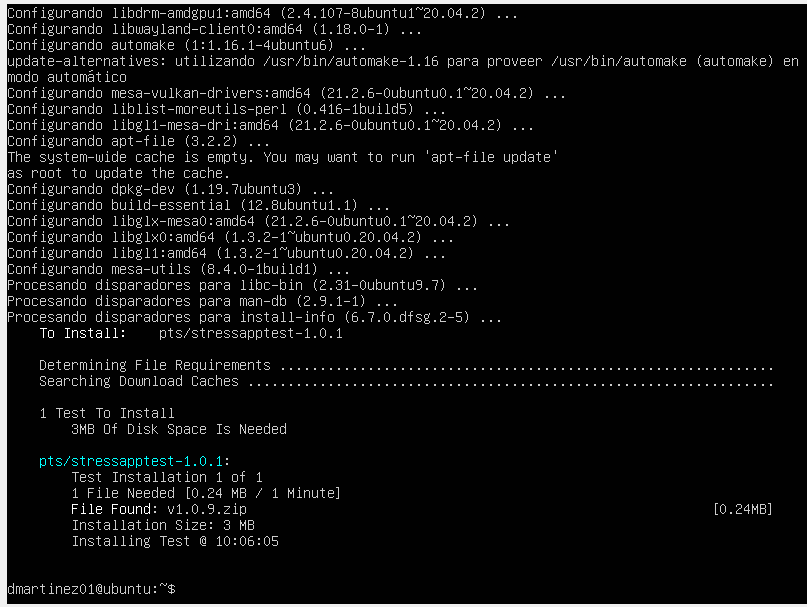
En primer lugar, instalamos phoronix en Ubuntu, siguiendo los comandos de la pagina web:



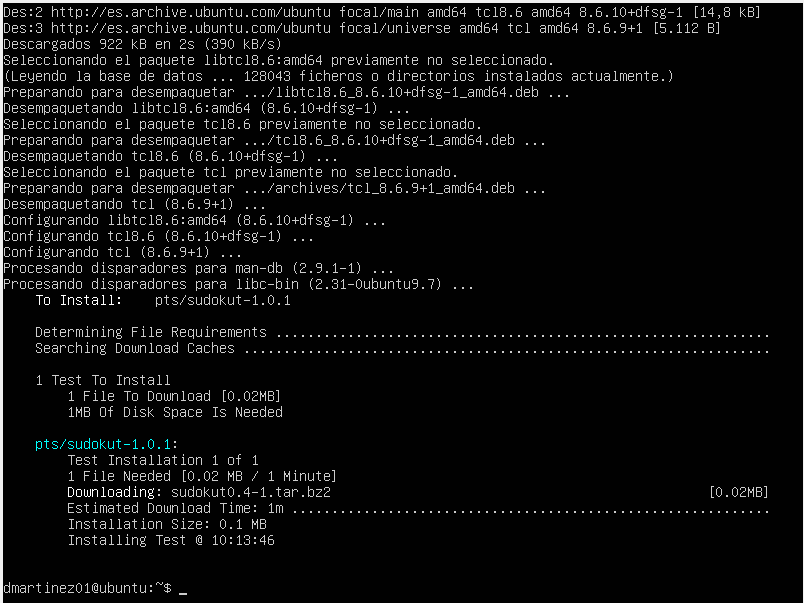
Ahora debemos de elegir los benchmarks que queramos ejecutar:



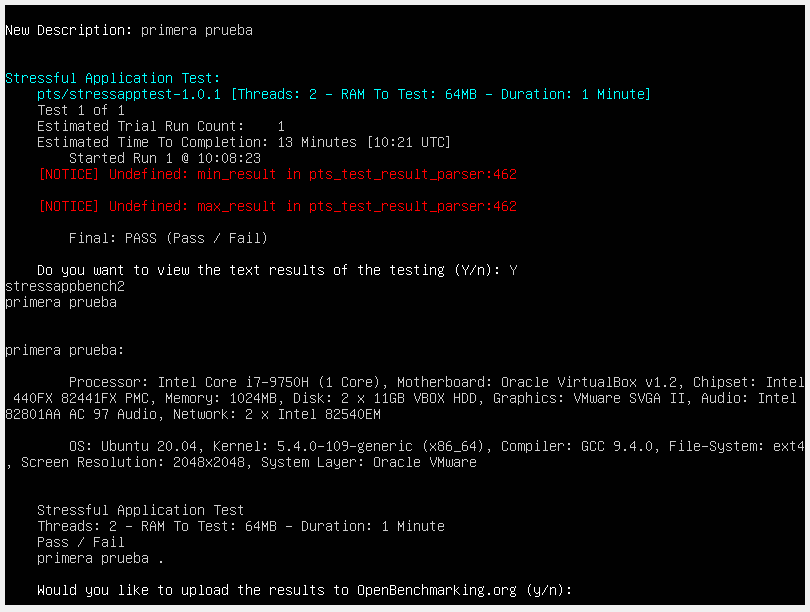
Por mi parte me interesa ejecutar benchmarks que pongan a prueba mi cpu y la memoria, por lo que voy a escoger los dos siguientes:



Este segundo benchmark me intereso porque se trata de ver cuantos sudokus es capaz de realizar:

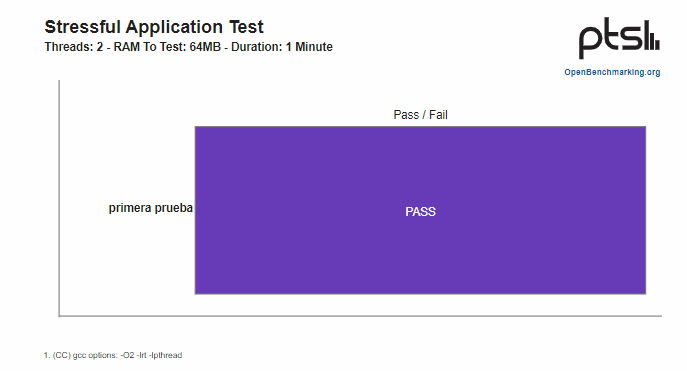


Y ejecutamos los benchmarks instalados:



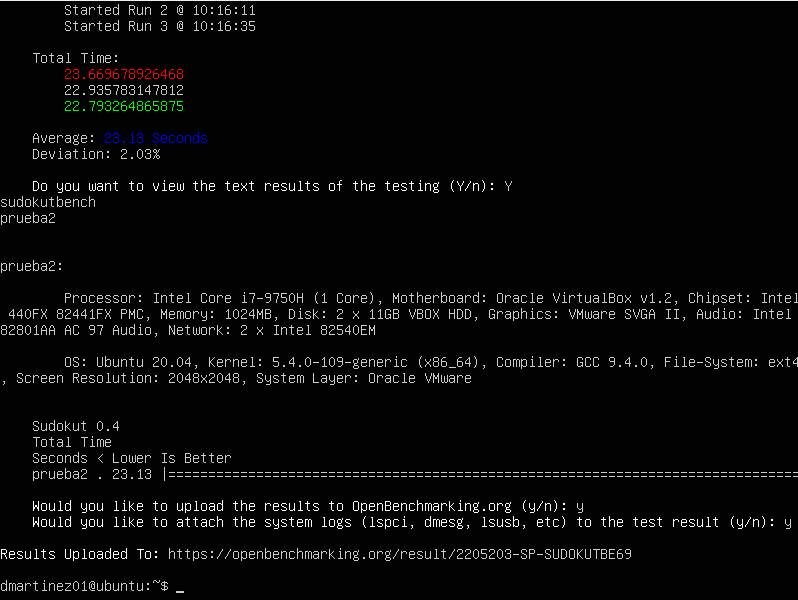
Y si nos metemos en la pagina web de openbenchmarking después de haber aceptado subir los resultados podemos comprobar nuestro test:

<https://openbenchmarking.org/result/2205207-SP-STRESSAPP72>



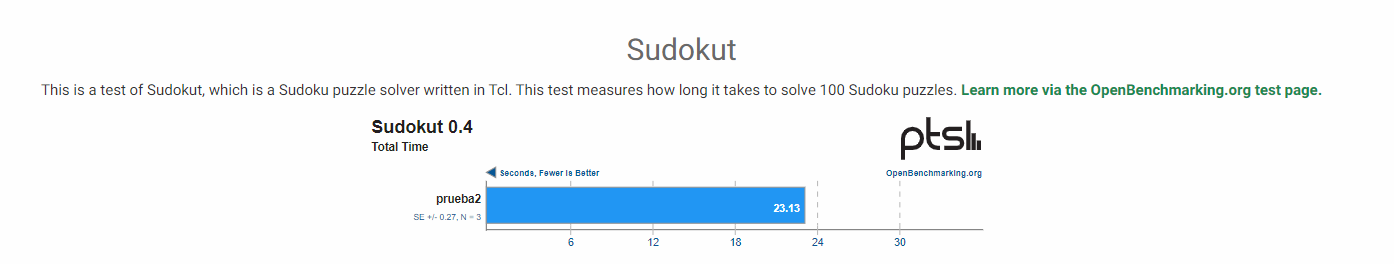
Y vemos que pasamos el test exitosamente.

Ejecutamos el segundo benchmark:



Y podemos comprobarlo también en la página web:

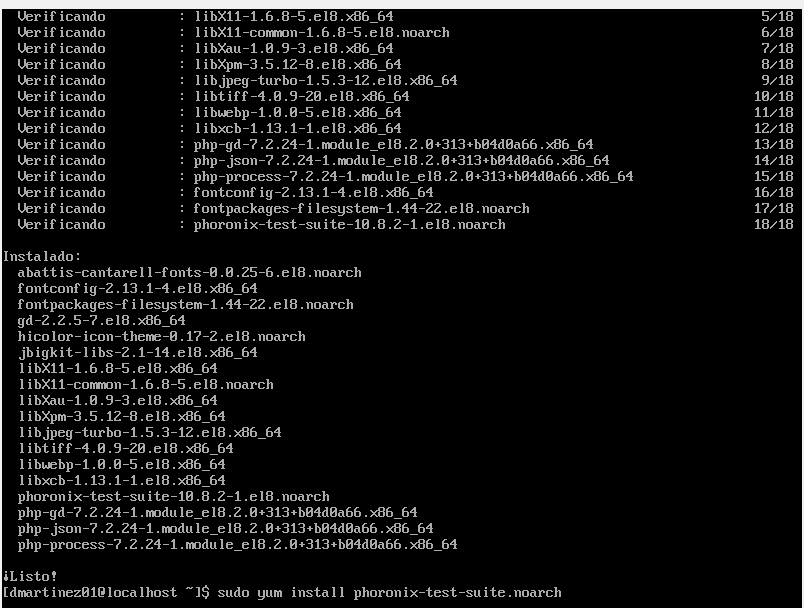
<https://openbenchmarking.org/result/2205203-SP-SUDOKUTBE69>



Podemos comprobar que ha tardado en realizar 100 sudokus en 23.13 segundos.

### Phoronix en CentOS:

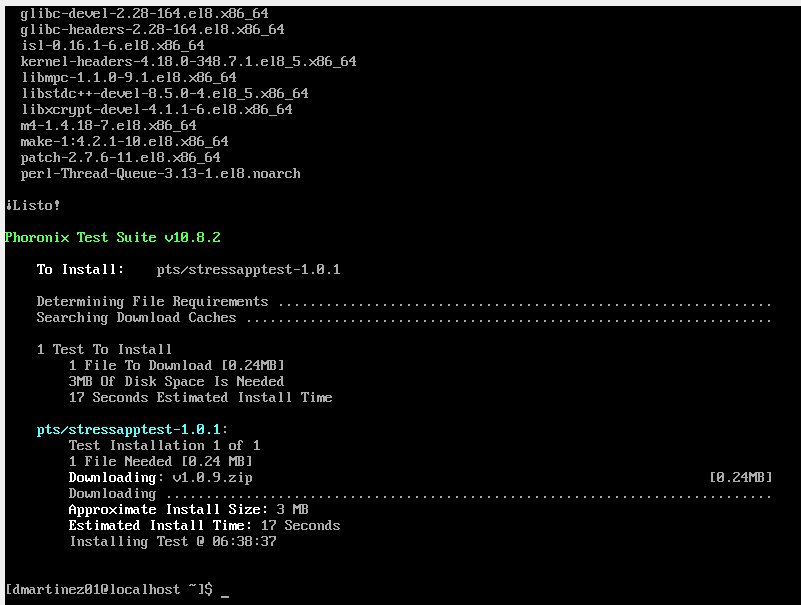
Instalamos Phoronix en CentOS, con yum install phoronix-test-suite:



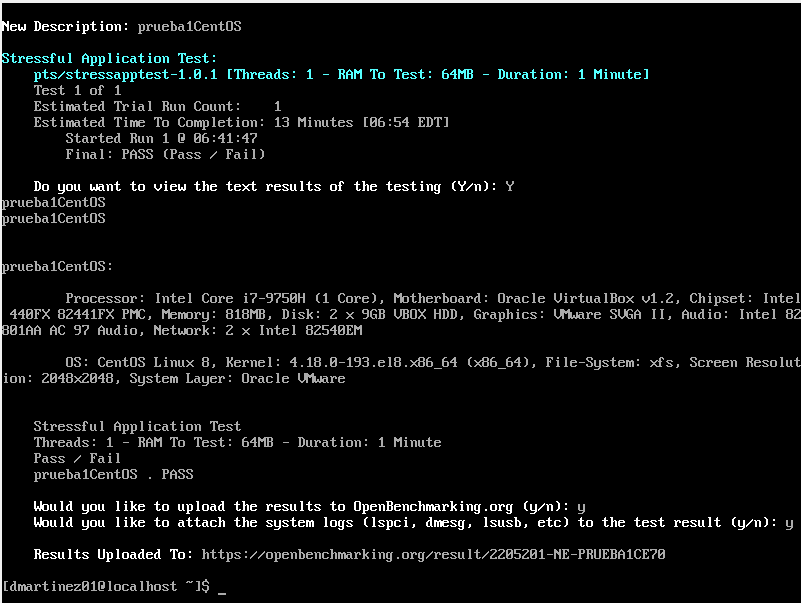
Procedemos a ver la lista de benchmarks a ver si se encuentran los mismos utilizados para Ubuntu:



Procedemos a instalarlos uno a uno, primero el de stressapptest:

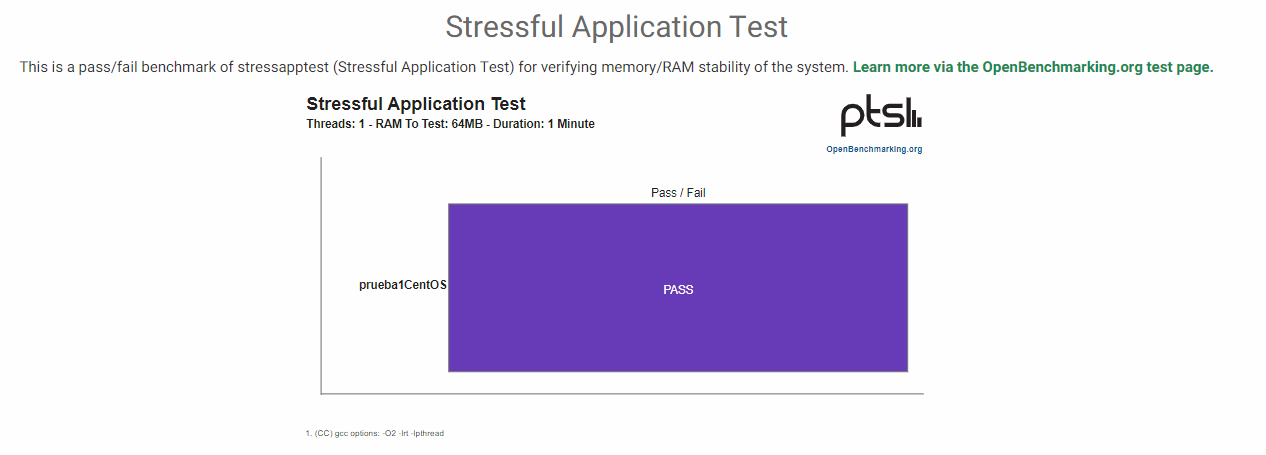


Y lo ejecutamos con phoronix-test-suite run:

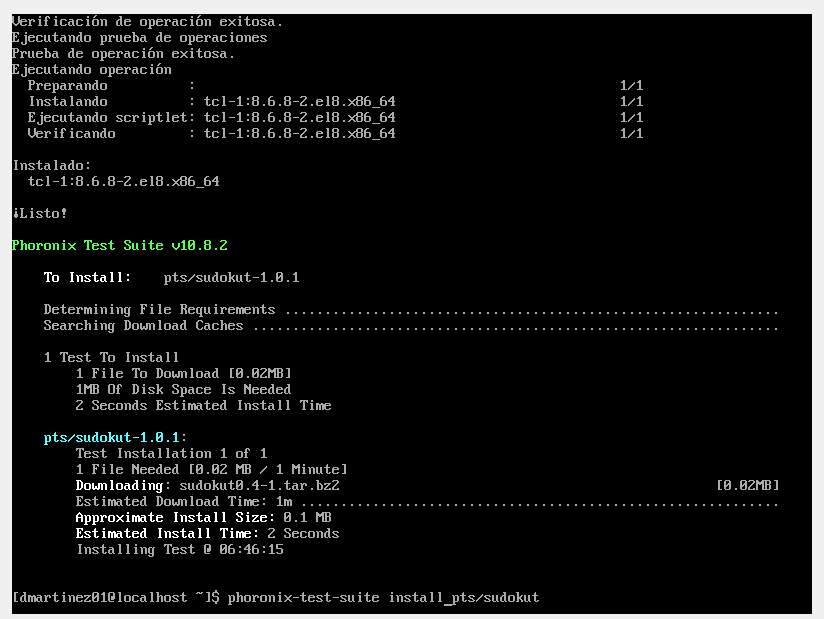


Y comprobamos de manera mas visual en la pagina web:

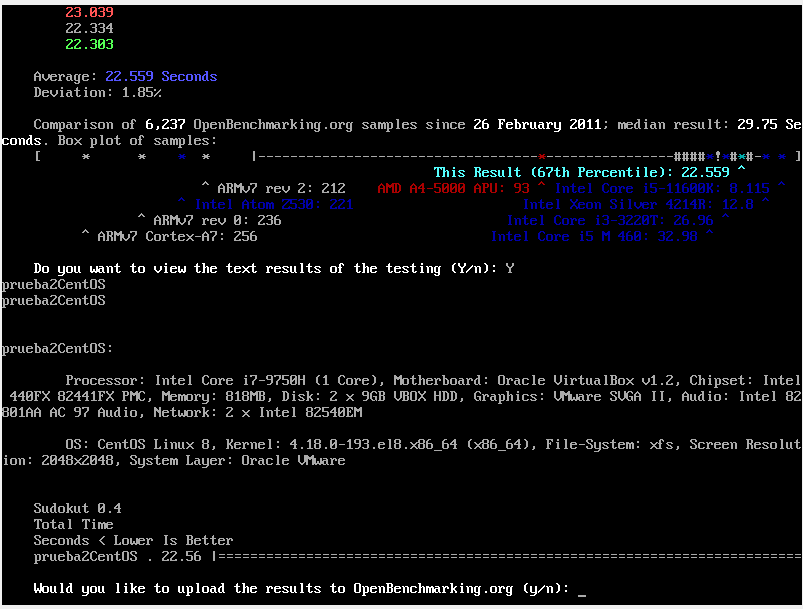
<https://openbenchmarking.org/result/2205201-NE-PRUEBA1CE70>



Hacemos el mismo proceso, pero con el benchmark de sudokus, empezamos intalandolo:

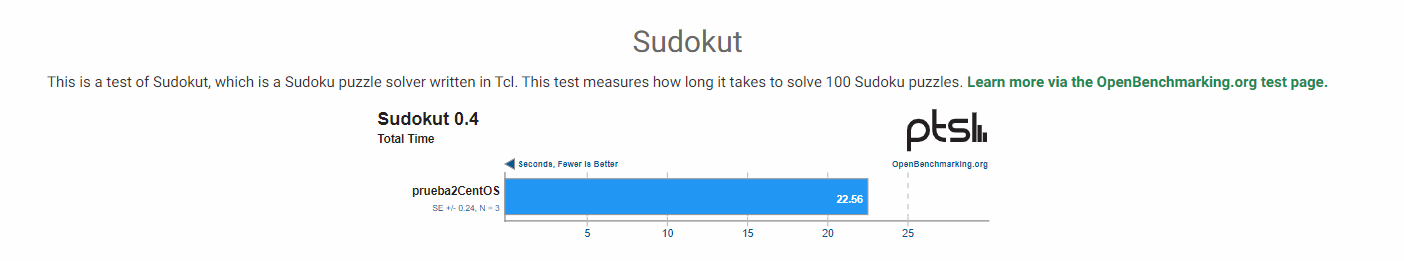


Y posteriormente lo ejecutamos:



Y comprobamos también en la pagina web:

<https://openbenchmarking.org/result/2205203-NE-PRUEBA2CE34>



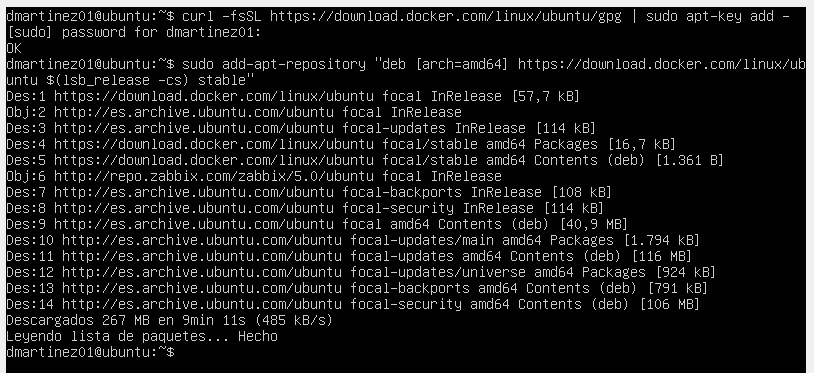
Como comentarios finales podemos destacar, que en Ubuntu era necesario la instalación de la librería de compilación estática, aunque finalmente los dos pasaron la prueba de estabilidad de la memoria RAM.

En cuanto al test de los sudokus, vemos que tienen diferentes formatos para mostrar los resultados y en cuanto a la desviación, vemos que en Ubuntu era del 2.03% mientras que en CentOS es de 1.85%.

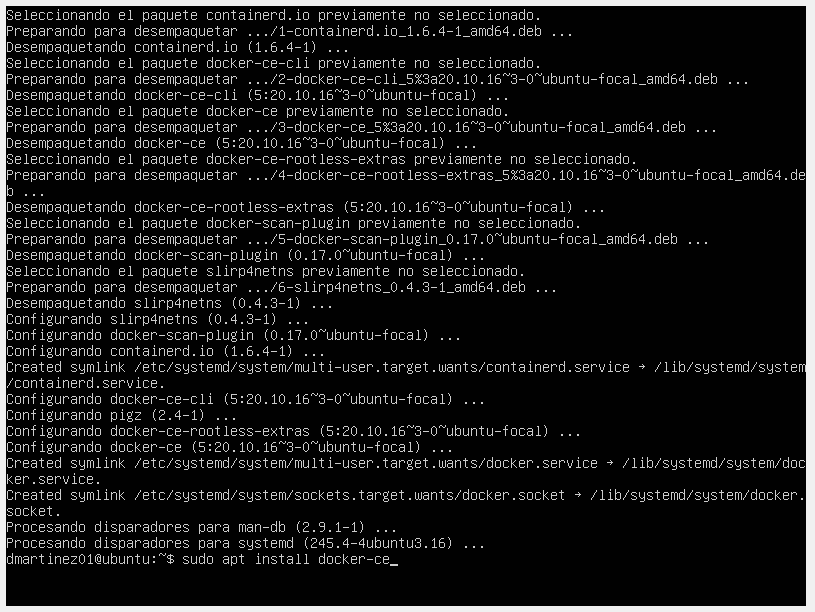
## EJERCICIO 2: Test de Jmeter sobre dos contenedores.

### Instalamos Docker en Ubuntu:

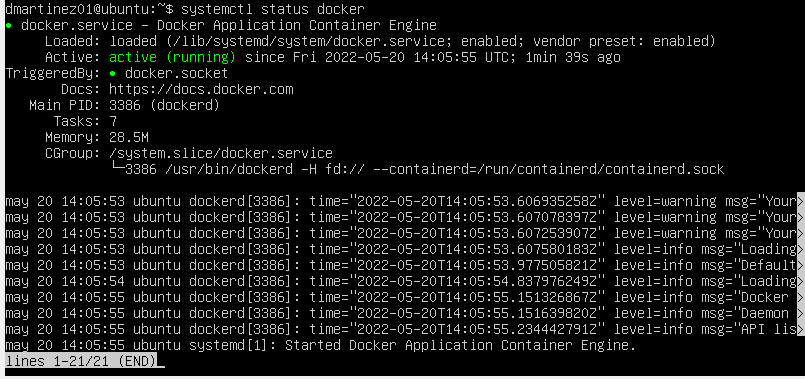
Seguimos los pasos para la instalación que viene en el pdf:



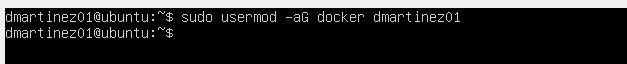
Luego actualizamos la lista de repositorios, buscamos los repositorios de Docker y procedemos con su instalación:



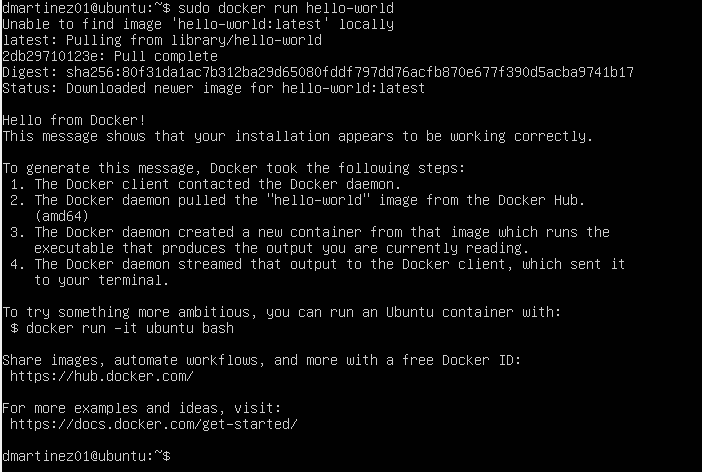
Podemos comprobar que el servicio esta en funcionamiento:



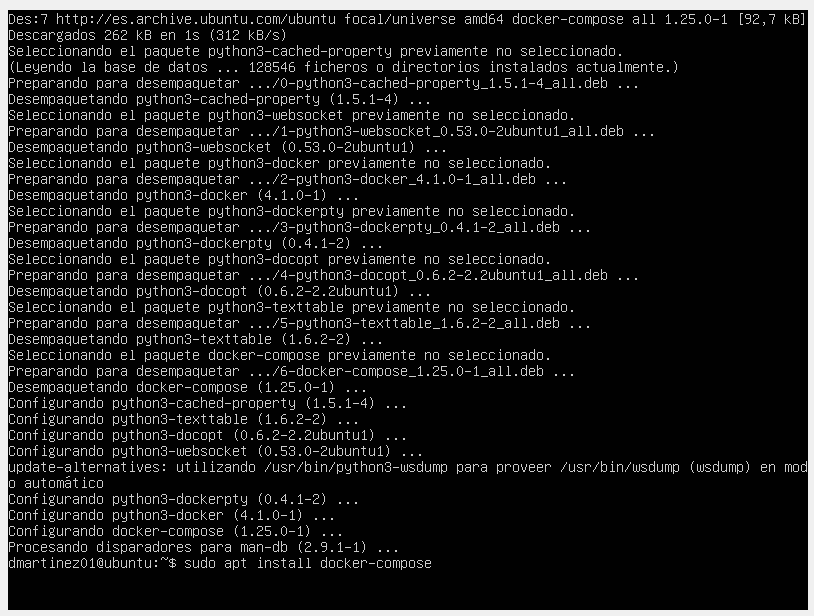
Y podemos añadir nuestro usuario a Docker:



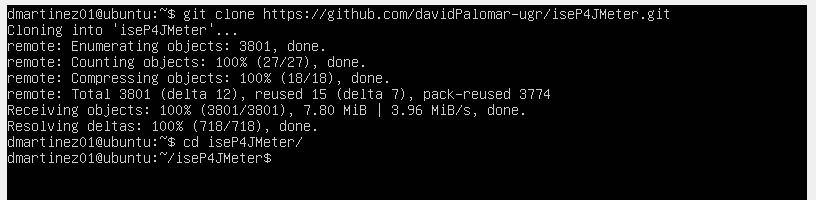
Probamos a ejecutar un comando de prueba:



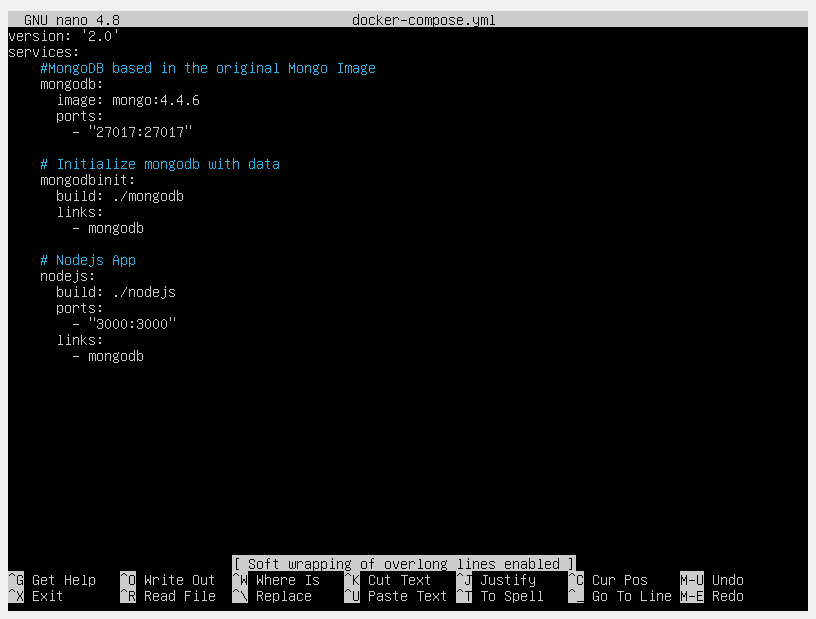
Instalamos Docker-compose:



Por último, añadimos la aplicación a través del GitHub:



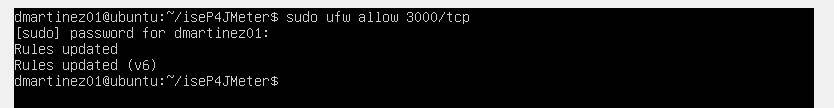
Cuando hacemos Docker-compose up, nos da un problema de instrucciones, para arreglarlo debemos cambiar el archivo Docker-compose.yml a la versión 4.4.6:



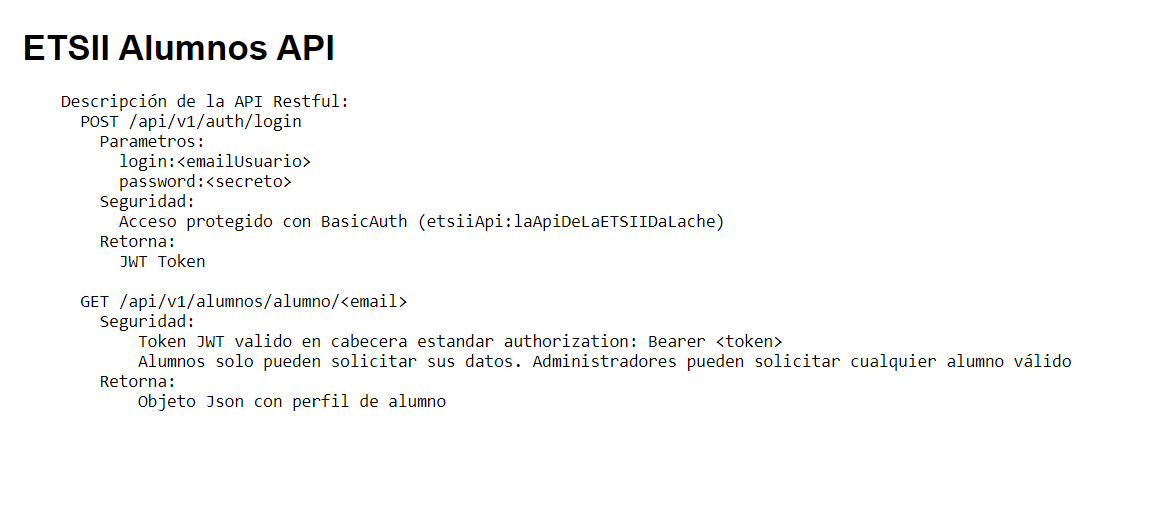
Y podemos ejecutar perfectamente el comando:



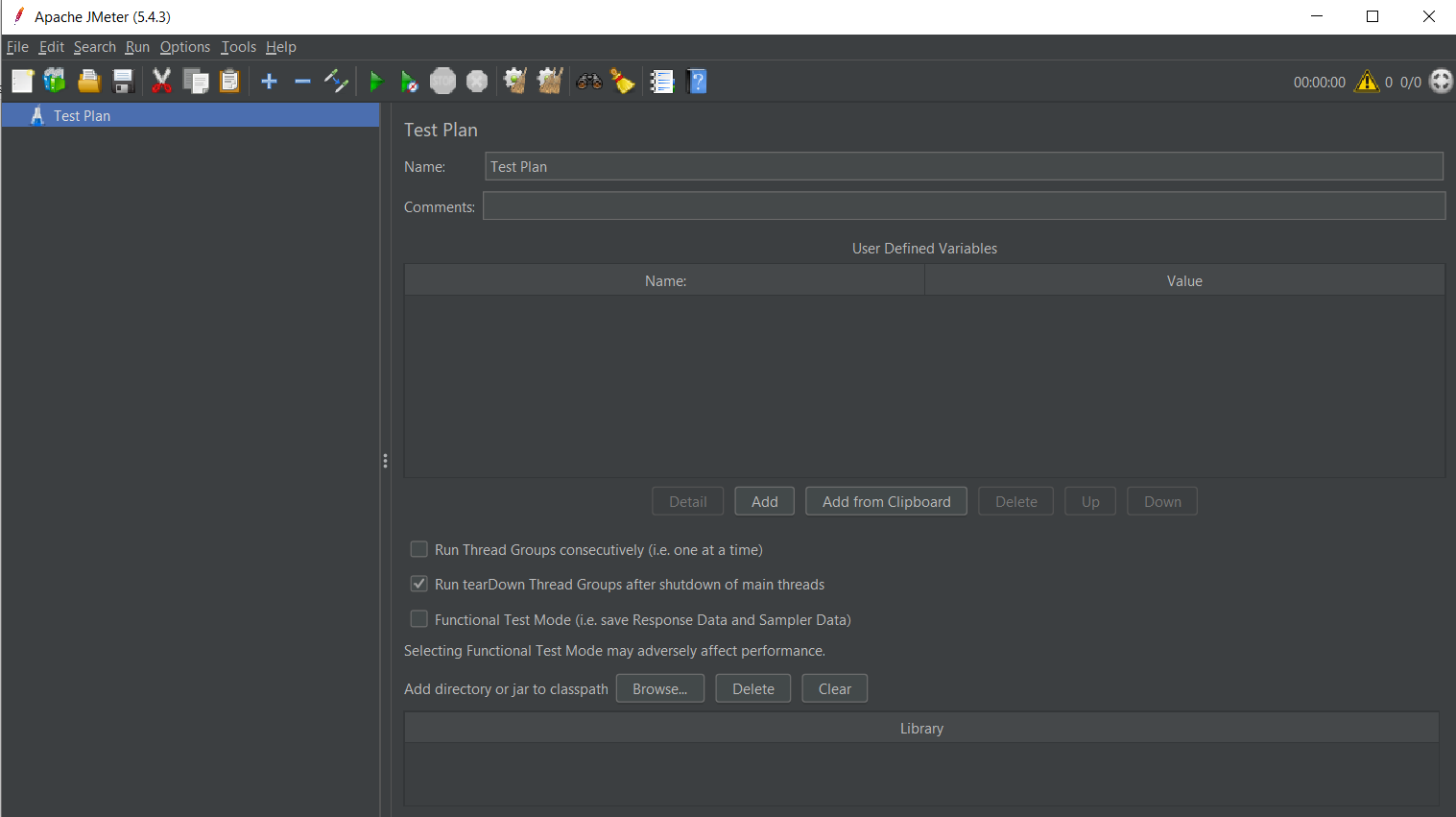
Posteriormente podemos abrir el puerto 3000 como dice el repositorio de github:



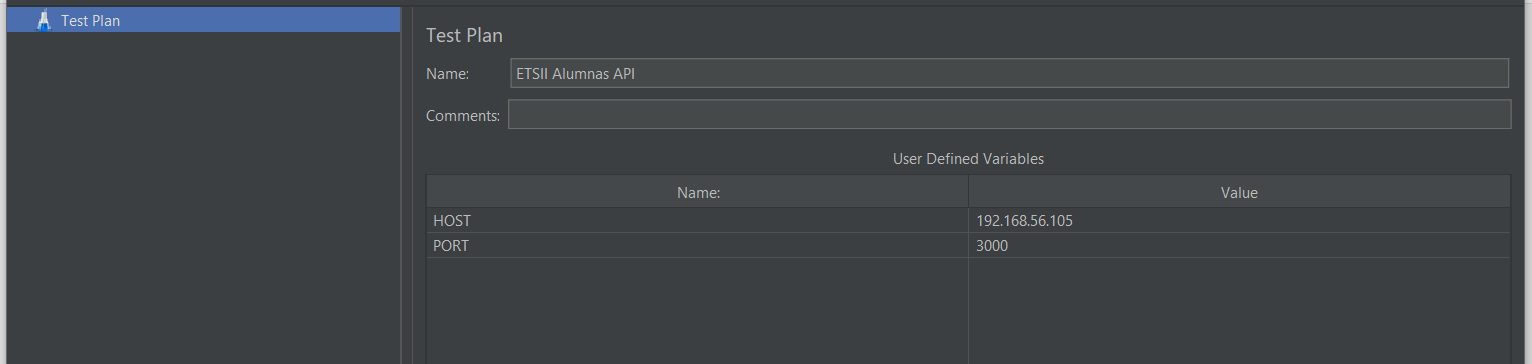
Cuando accedemos a la pagina web obtenemos dicho resultado:



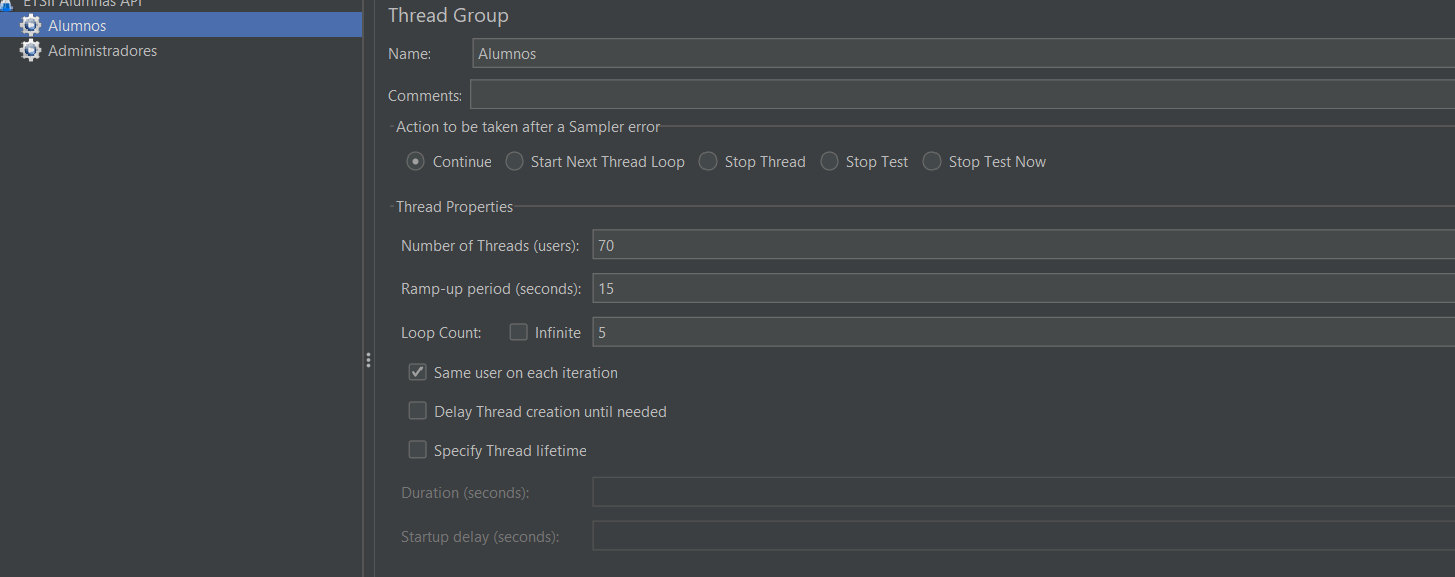
Después nos descargamos el Jmeter y abrimos su interfaz gráfica:

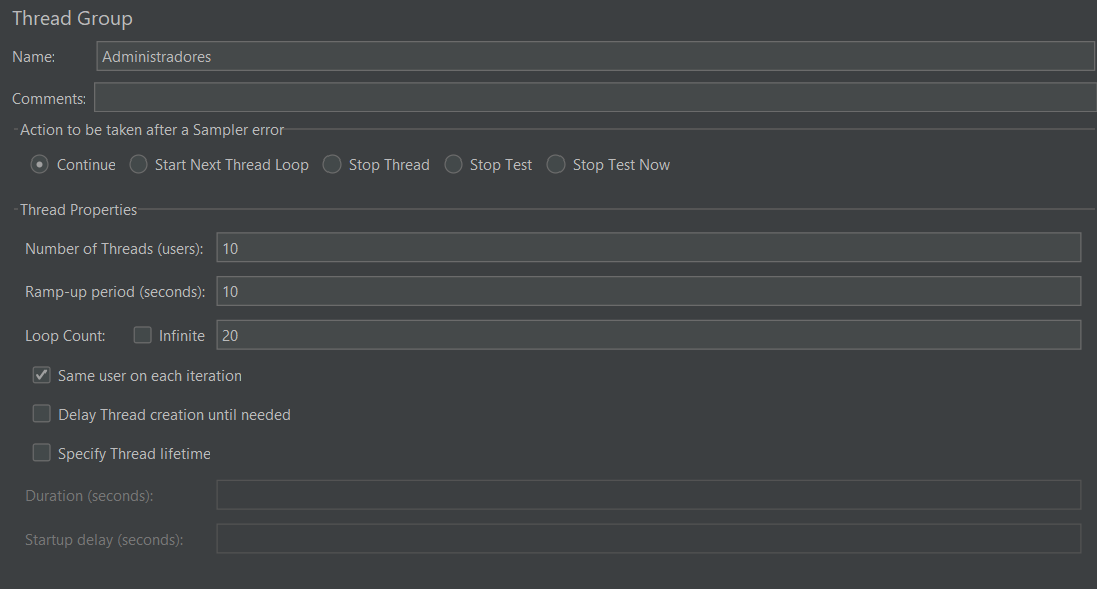


Luego lo que hacemos es parametrizar el host y el puerto como nos dice el enunciado del ejercicio:

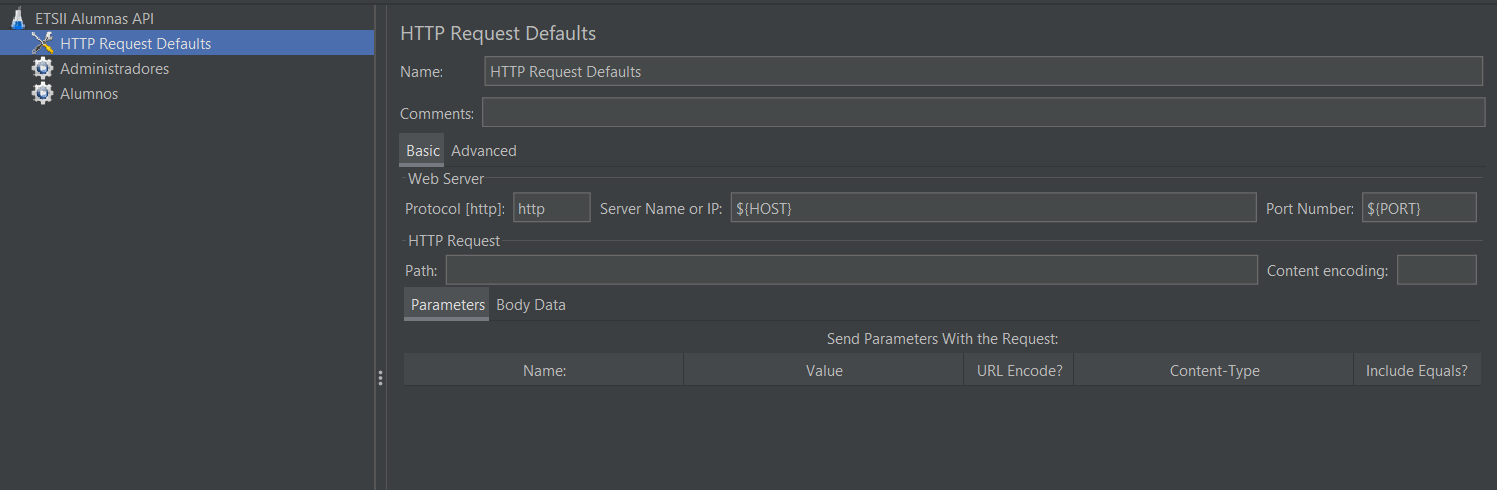


Nos creamos dos grupos de hebras, tanto la de alumnos como el de administradores:

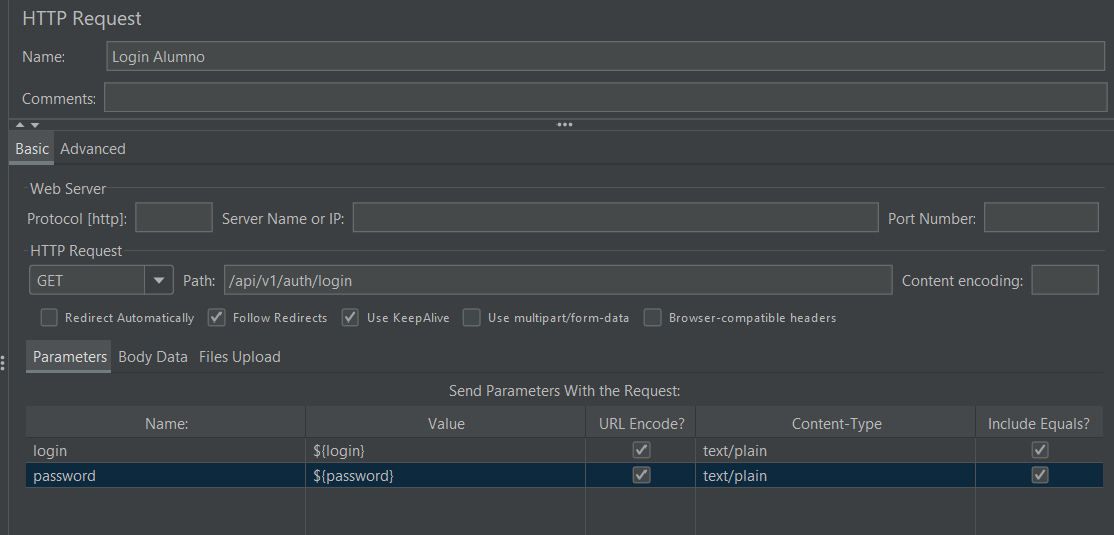




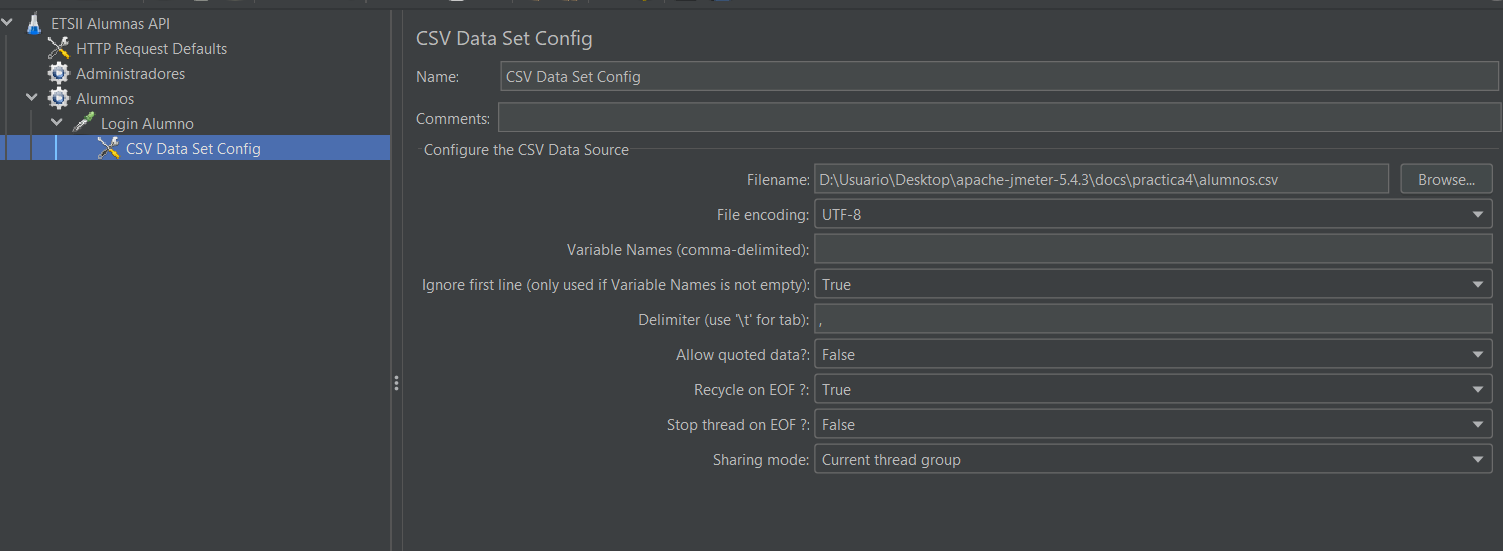
A continuación nos creamos unos valores por defecto para las peticiones a http, donde pondremos por defecto nuestro valores de nuestro puerto y de nuestra ip:



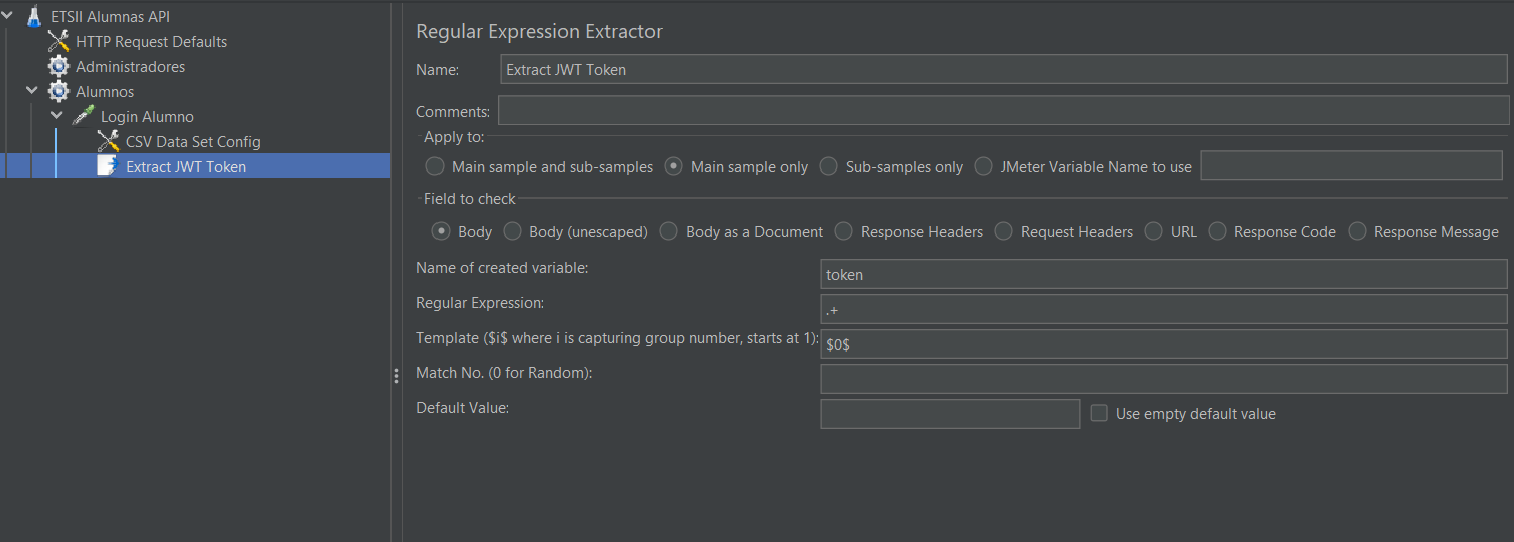
Después, nos creamos una simulación de inicio de sesión de los alumnos, donde utilizaremos el camino propuesto por la api, además los parámetros serán los que vengan por defecto:



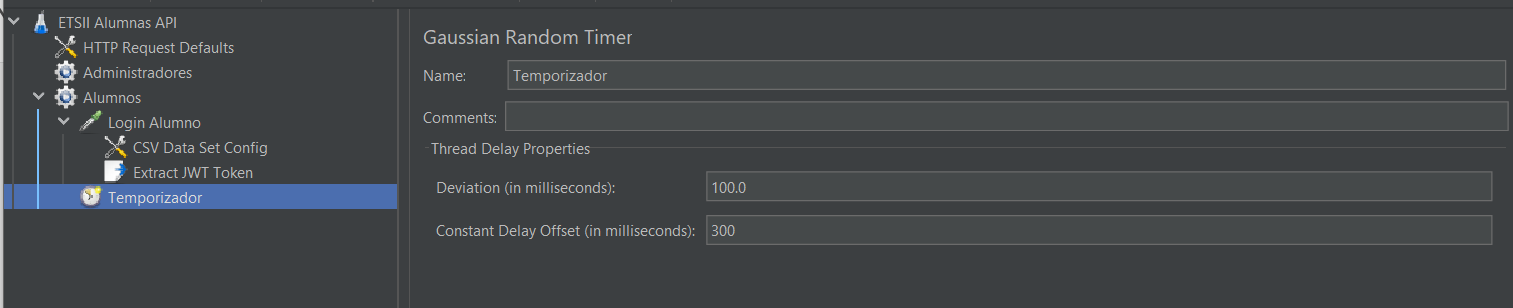
Añadimos el archivo csv de alumnos para su posterior uso, por lo que añadimos un archivo de configuración de csv y ponemos la ruta de dicho archivo:



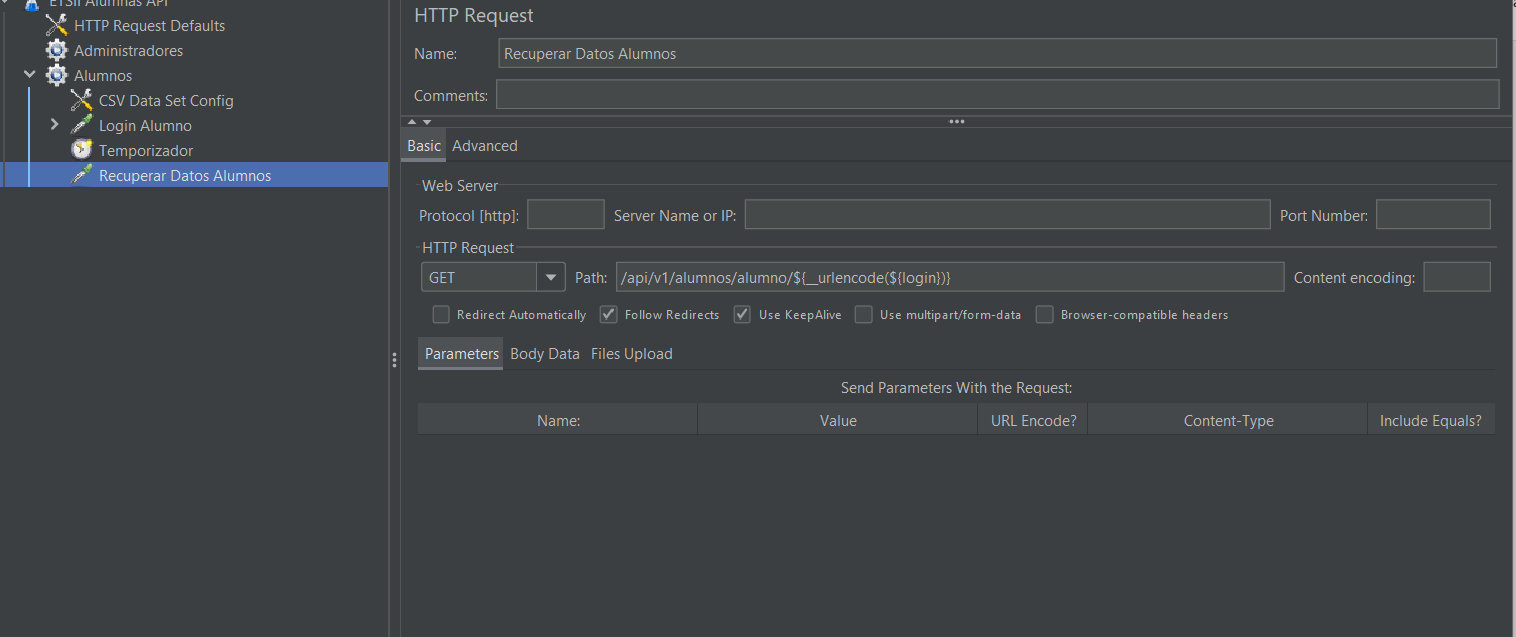
Cuando realicemos diferentes llamadas a la API, este devolverá un token JWT, por lo que necesitaremos hacer llamadas del tipo GET, para ello es necesario crearnos un patron para detectar lo que queremos guardar, nos creamos una expresión regular a la que llamaremos Extract JWT Token:



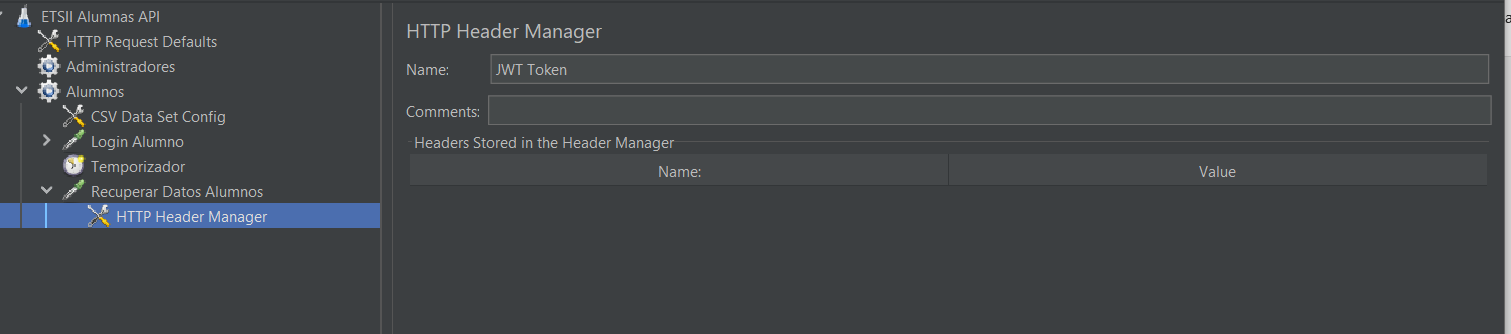
Siguiendo el guion, nos creamos un temporizador gaussiano y queda así:



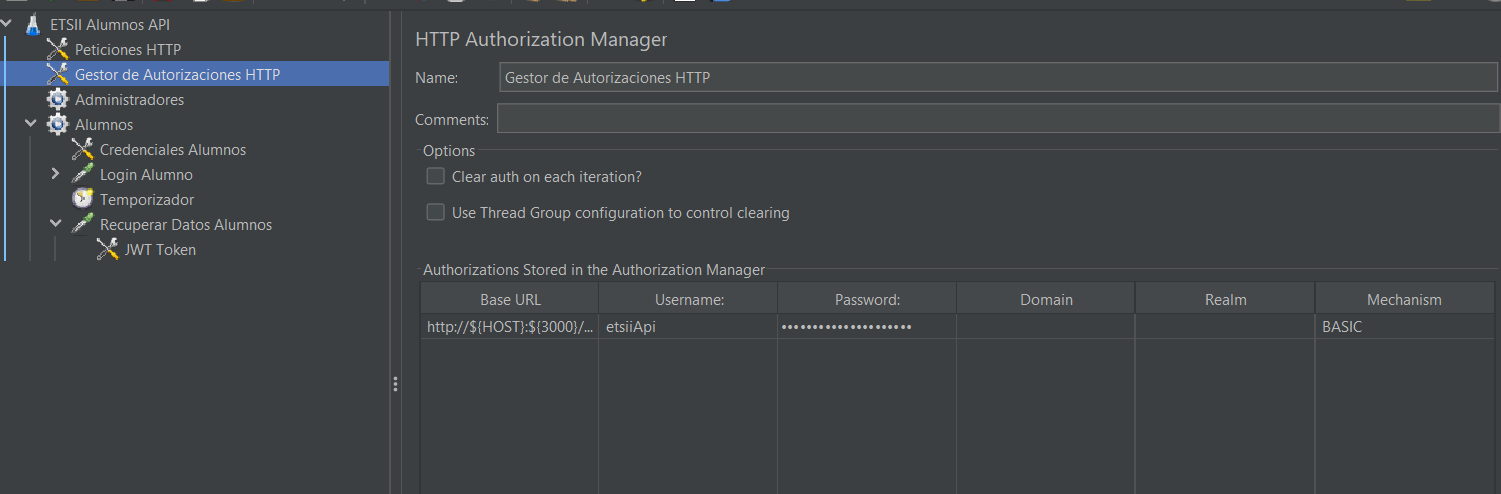
Nos creamos otro request http para poder recuperar los datos de los alumnos:



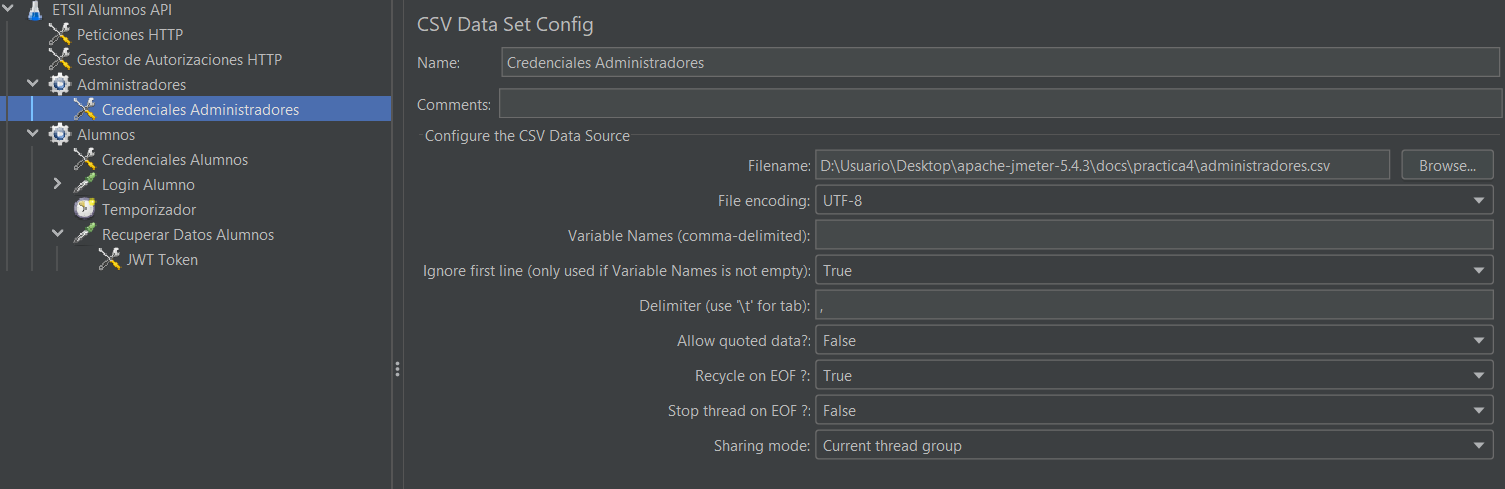
Por ultimo, en cuanto a la parte de alumnos es necesario autenticarse para poder realizar peticiones del tipo GET, por lo que nos hará falta crearnos un gestor de cabeceras de http:



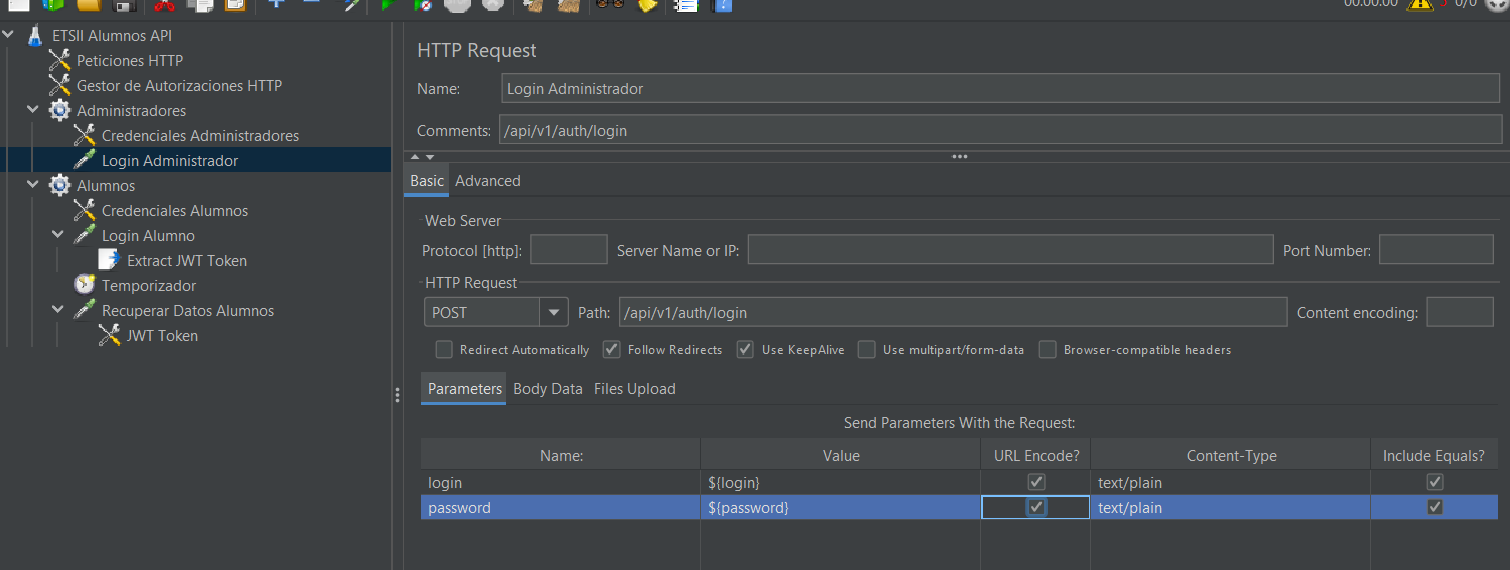
Ahora, nos creamos un gestor de autorizaciones de la utilización de la API, por lo que será necesario crearse dicho gestor con los siguientes parámetros:



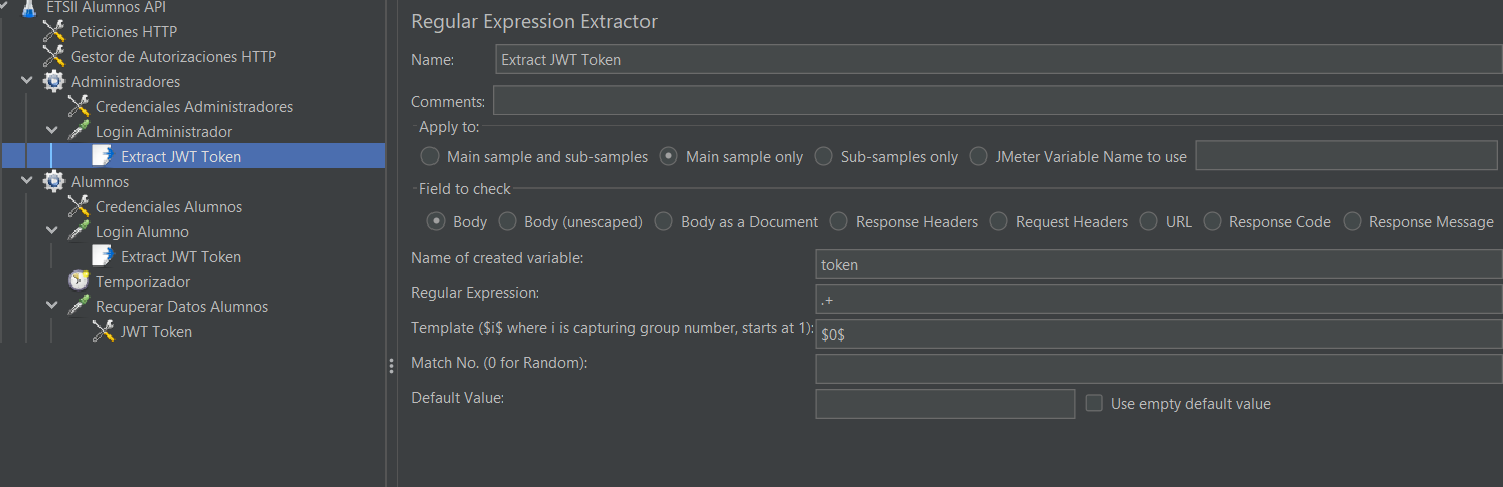
Ahora pasamos a la parte de administradores, y en primer lugar nos creamos un csv Data Set:



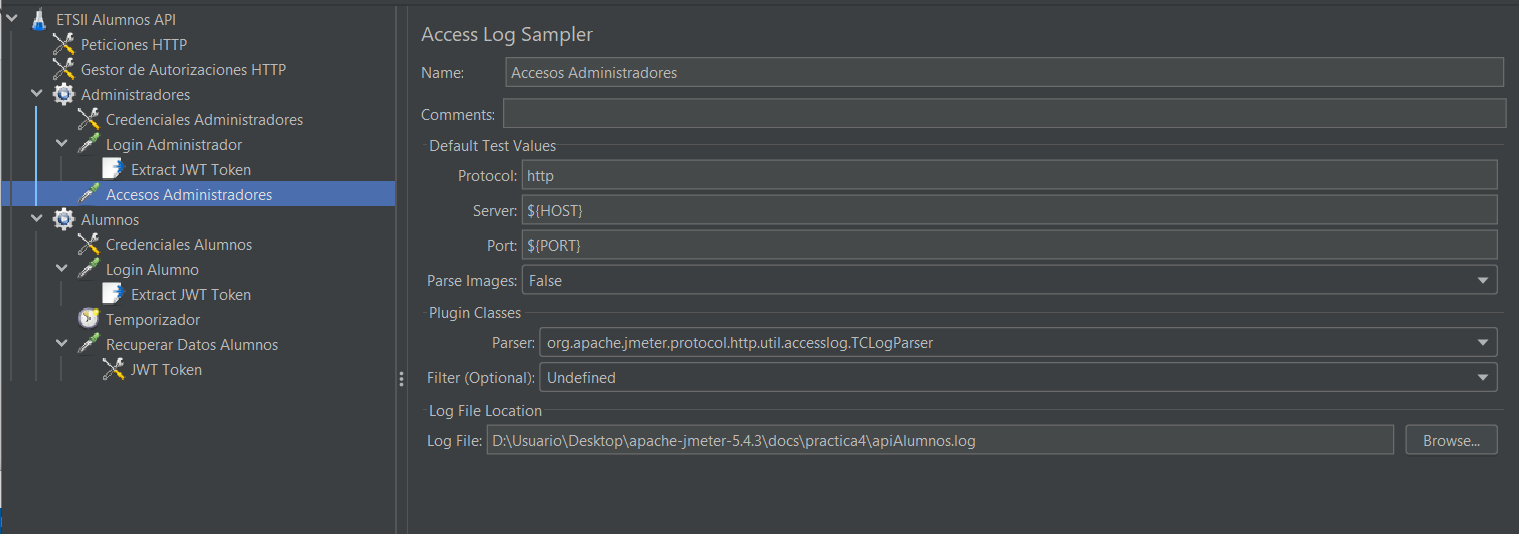
Posteriormente nos creamos una Peticion POST para el administrador, con el fin de que puedan iniciar sesión en la API:



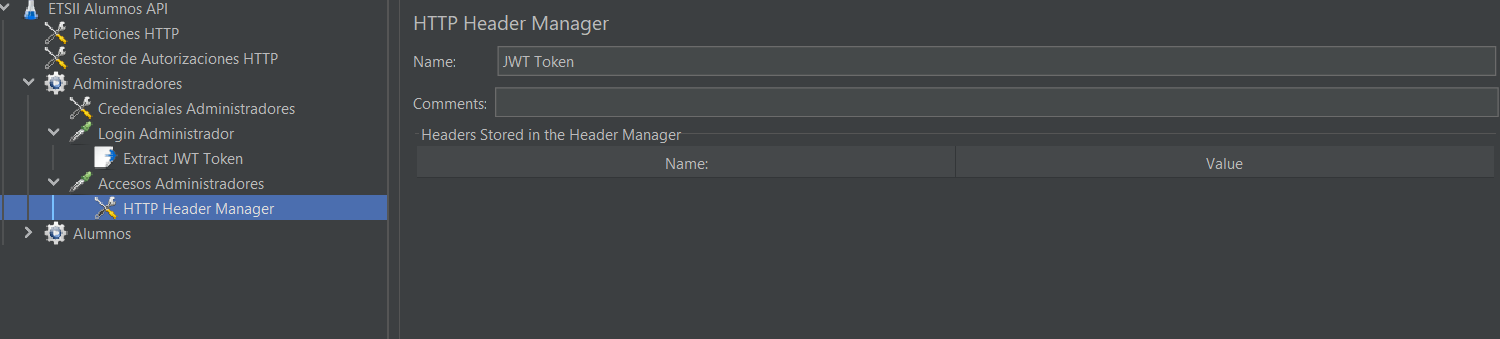
También debemos crearnos el extractor de Expresiones Regulares:



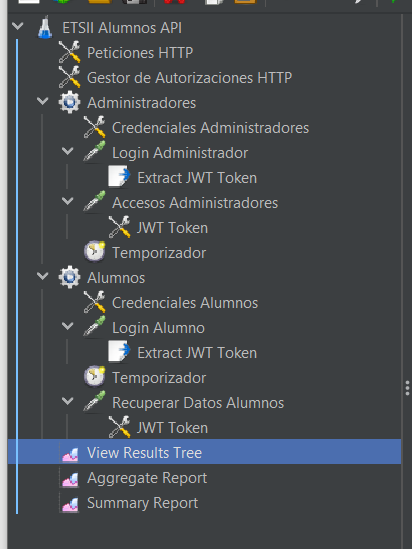
Nos creamos el Acceso de los Administradores, para obtener los datos de los diversos alumnos, podemos conseguir dicha funcionalidad creándonos un muestreador de los inicios de sesión:



También como hicimos con alumnos nos creamos un gestor de cabeceras http:

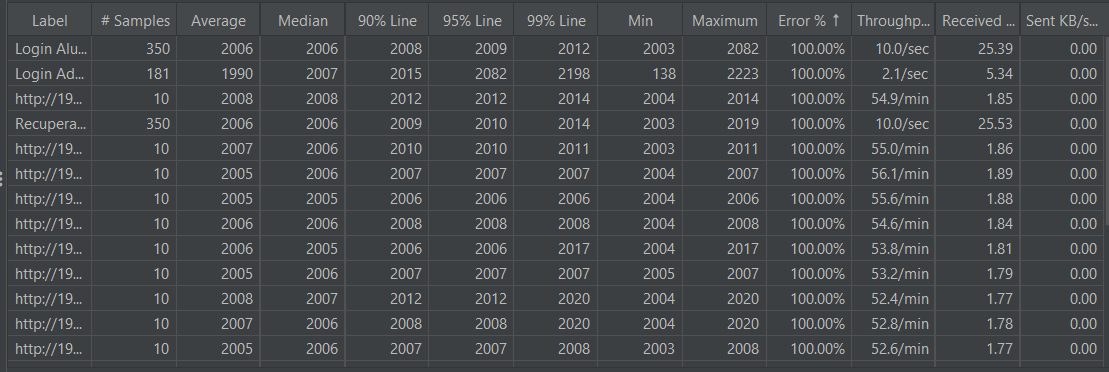


Por último, agregamos los listener para poder comprobar las llamadas a la API y las peticiones solicitadas tanto por los administradores como por los alumnos:

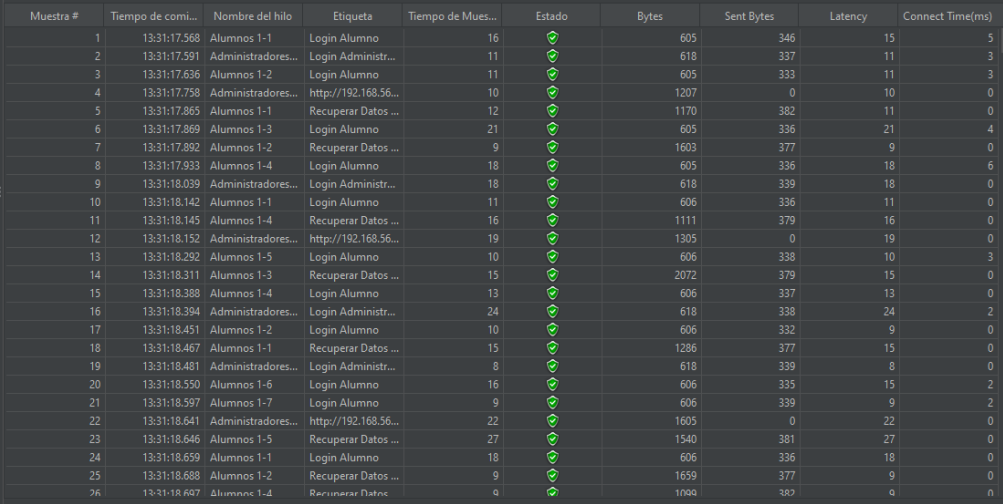


Y estos son los resultados:

* Aggregate Report



* Summary Tree



* Summary Report:

