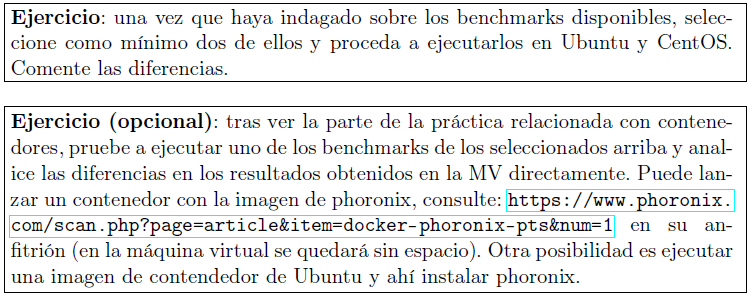
## BENCHMARKING Y AJUSTE DEL SISTEMA

## PHORONIX

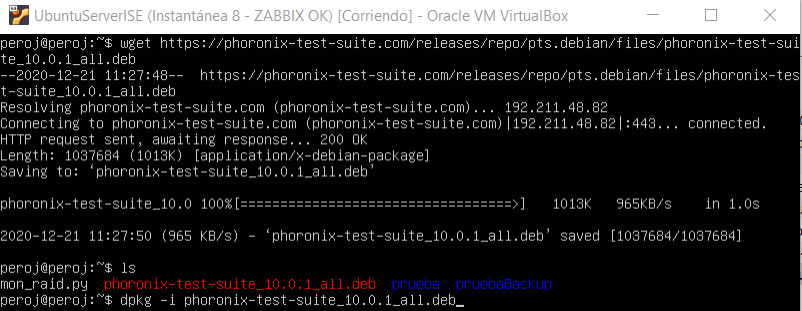


Documentación: <https://www.phoronix-test-suite.com/documentation/phoronix-test-suite.pdf>

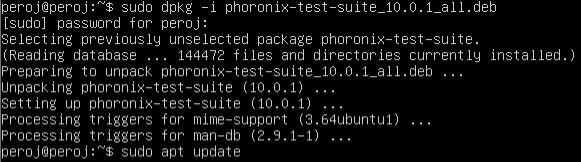
Phoronix es una plataforma que permite ejecutar un conjunto de benchmarks bajo agrupación openbenchmarking.org. En esta organización existen benchmarkings de todo tipo (videojuegos,cpu,memoria...) y mediante la herramienta Phoroxix podremos lanzar todos estos benchmarks.

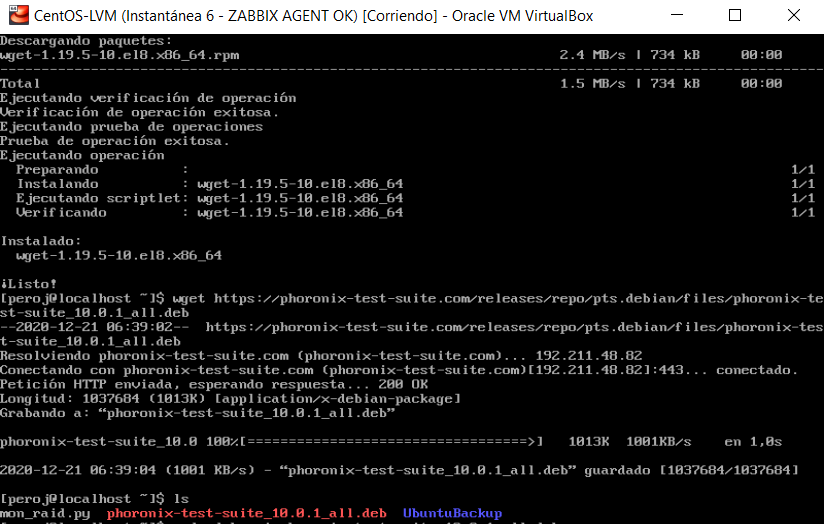
Para proceder con la instalación en las nuevas distribuciones Linux de 2020 no podemos hacerlo desde los gestores de paquetes dado que no contiene los repositorios. Por lo que, para instalarlo, tanto en Ubuntu Server como en CentOS, deberemos descargarnos los paquetes desde el siguiente enlace de Phoronix ejecutando los siguientes comandos:

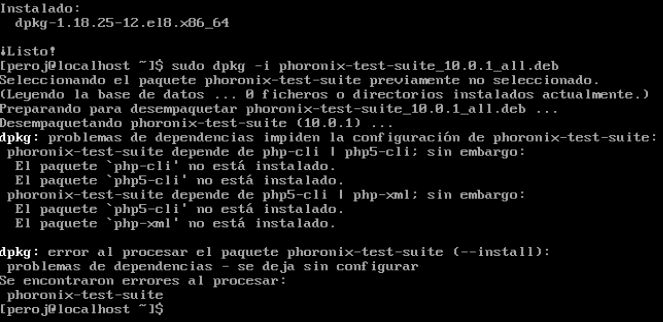
***wget*** [***https://phoronix-test-suite.com/releases/repo/pts.debian/files/phoronix-test-suite\_10.0.1\_all.deb***](https://phoronix-test-suite.com/releases/repo/pts.debian/files/phoronix-test-suite_10.0.1_all.deb)

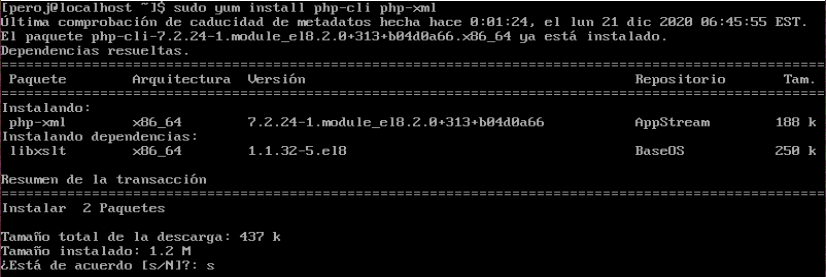
***dpkg -i \*.deb***

***apt update***

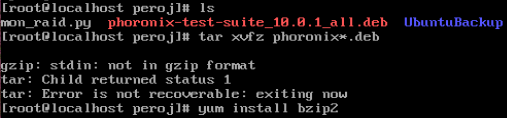


Procedemos de igual manera para la instalación en CentOS para posteriormente instalar y probar algunos benchmarks en ambas máquinas.

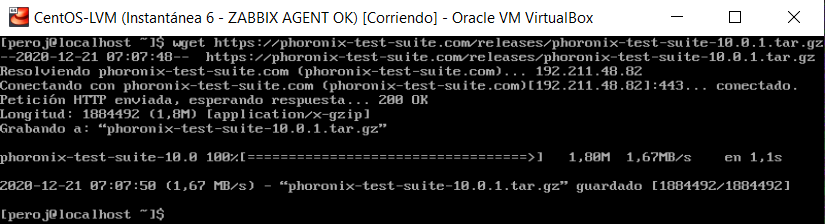
A parte de instalar las herramientas *wget* y *dpkg*, al hacer *dpkg -i* *phoronix...deb* vemos que en este caso nos da error debido a que no están instaladas las dependencias “*php-cli”* ni *“php-xlm”*.

Pasamos a instalarlas:

Hacemos un ***sudo yum update*** y volvemos a usar el ***dpkg -i***y vemos que nos sigue dando el mismo error, seguramente porque no es capaz de procesar ciertas dependencias. Por lo que probamos con ***tar xvfz*** y vemos que nos da error por el formato .zip. Por lo que pasamos a instalar la herramienta *bzip2*.

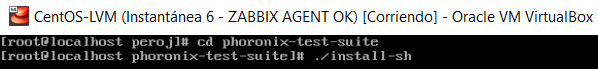


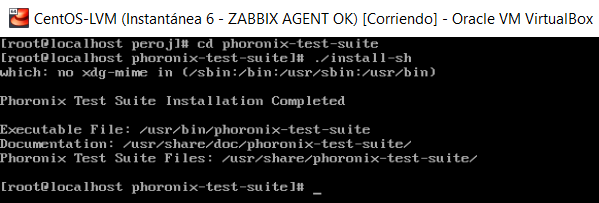
Nota: me acabo de dar cuenta de que descargué el *.deb* y procedí como en Ubuntu con *dpkg*, por lo que debo volver a descargar con *wget* el paquete pero esta vez en *.tar.gz* para proceder correctamente con ***tar xvfz phoronix\*.tar.gz***.

El paquete para CentOS está en este caso en: <https://phoronix-test-suite.com/releases/phoronix-test-suite-10.0.1.tar.gz>

Descomprimimos con ***tar xvfz phoronix\*.tar.gz***.

E instalamos con:





Después de echarle un ojo a los benchmarks, he decidido probar con *sudokut* que es un test dedicado a estresar la CPU mediante la resolución de sudokus y con *ramspeed* que pone a prueba el rendimiento de la memoria RAM del sistema.

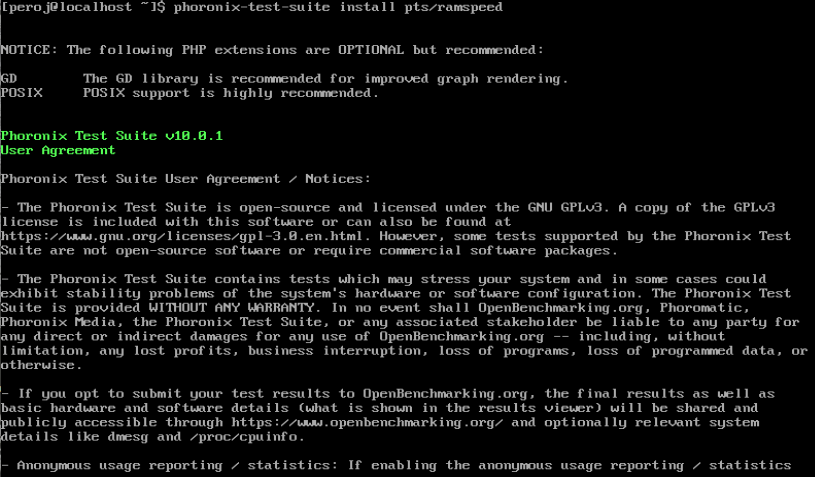
Es muy importante, al menos en este momento, no ejecutar ningún benchmark categorizado con *disk* dado a que puede estropear los discos duros físicos de nuestro pc (aunque estemos trabajando con máquinas virtuales).

**-En CentOS:**

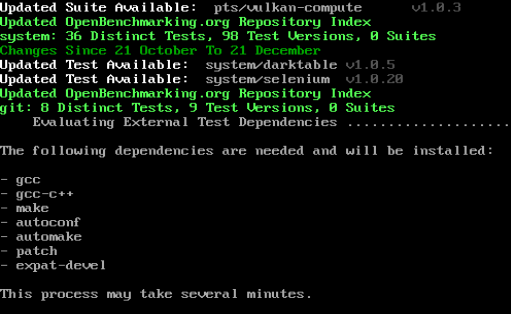
**-ramspeed:** [**http://www.alasir.com/software/ramspeed/**](http://www.alasir.com/software/ramspeed/)

Nos damos cuenta de que nos falta la extensión JSON para php. Por lo que la instalamos antes de instalar los test con *sudo* ***yum install php-json***.

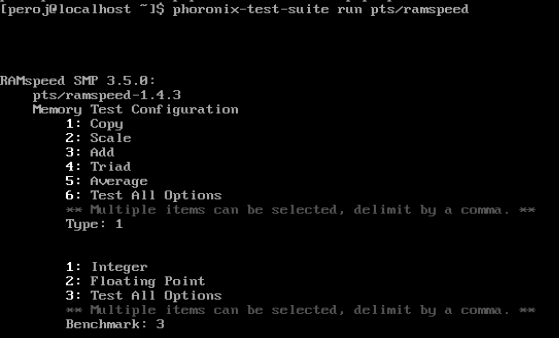
Nos descargamos el test ***ramspeed***:



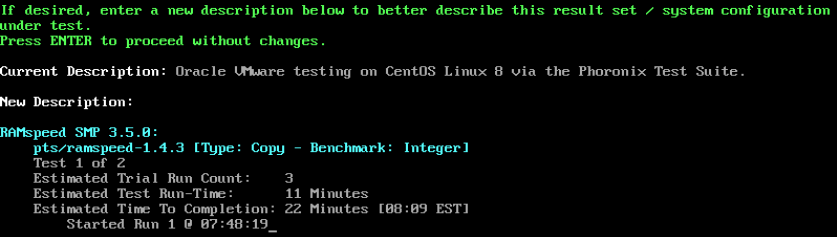
Aceptamos los términos de uso y recopilación de datos anónimos y comenzará a instalarse el test.

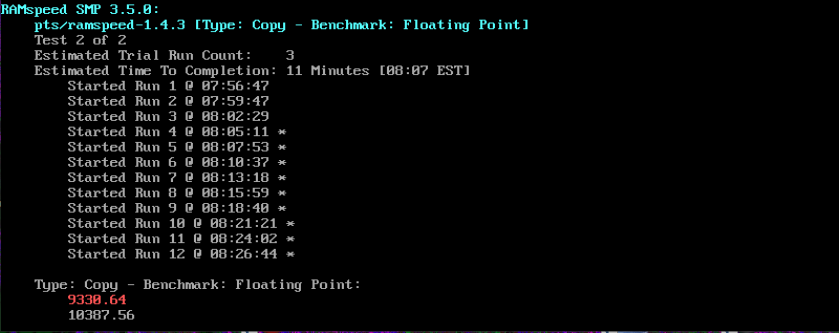


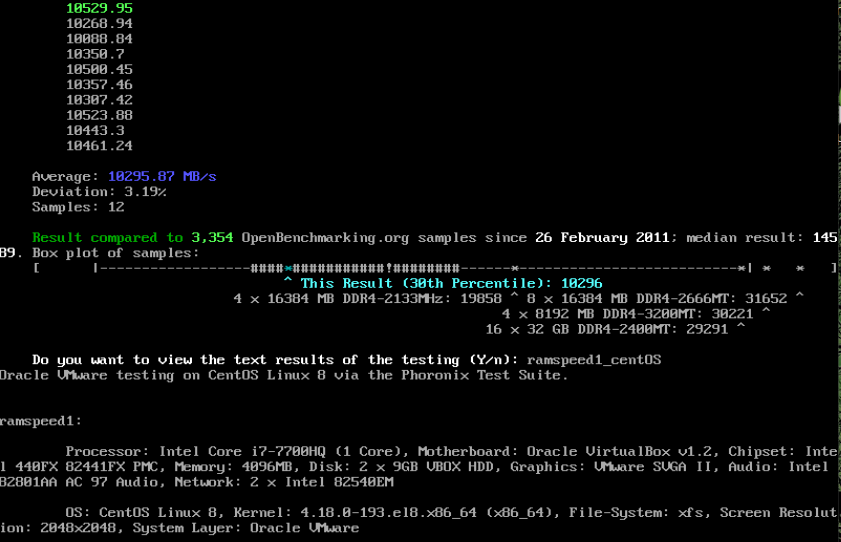
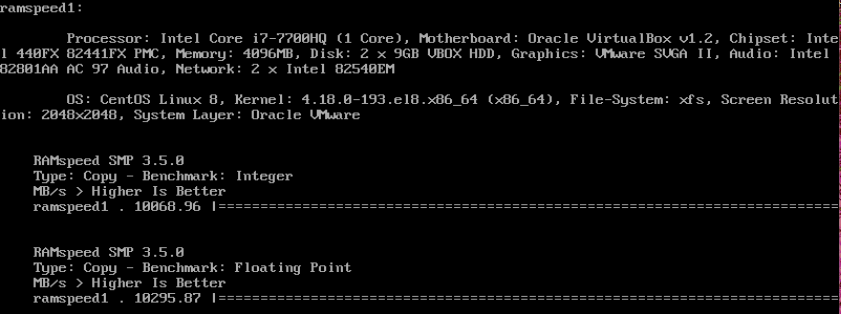
Una vez instalado lo ejecutamos como sigue y elegimos la primera de las opciones (por copia) y tanto de valores enteros como en coma flotante (3).



Nos enseña el sistema donde se lanzará el test y nos pregunta si queremos guardar los resultados del test, le decimos que si e introducimos el nombre del archivo y descripción si lo vemos necesario.



Ejecutará el benchmark durante unos minutos y nos mostrará los resultados.



Como podemos comprobar en este caso, en CentOS, este test copia más datos en coma flotante por segundo que valores enteros, con una diferencia de velocidad de unos 200 MB/s.

Podemos consultar el histórico de los resultados de los test realizados que se guarda por defecto en la siguiente carpeta:



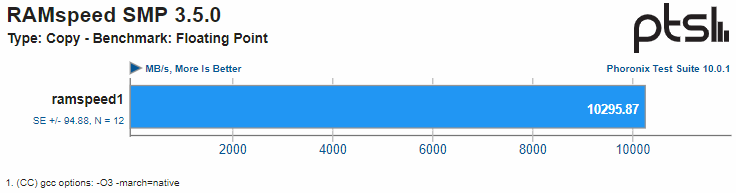
Podemos crear una carpeta en */var/www/html/phoronix* y copiar dentro los resultados de los test, y como tenemos el servidor web activo, si en el navegador del host buscamos la *ip/phoronix/test* podremos ver los resultados gráficamente.



Levantamos el servicio *httpd* y comprobamos desde el navegador de la máquina anfitrión:

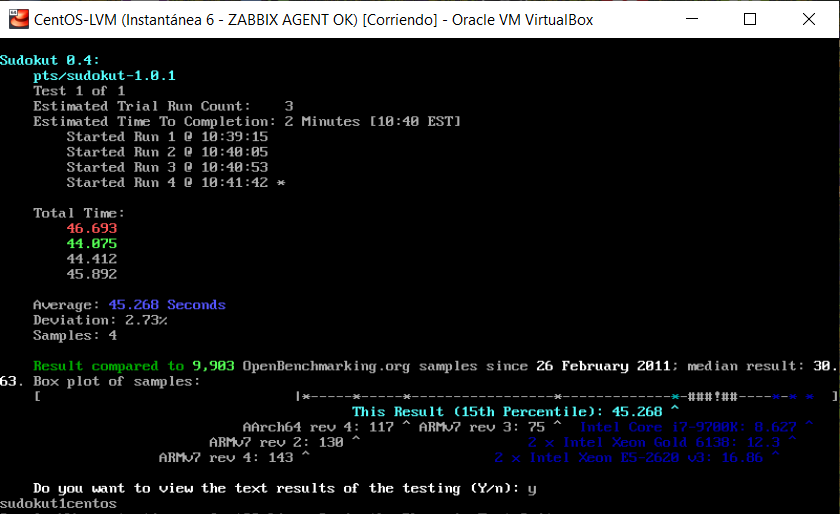


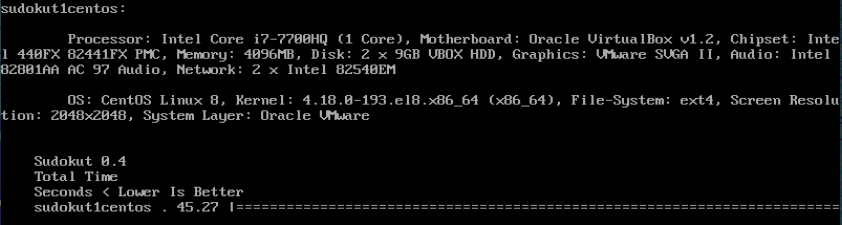
Vemos los resultados, por ejemplo de coma flotante:



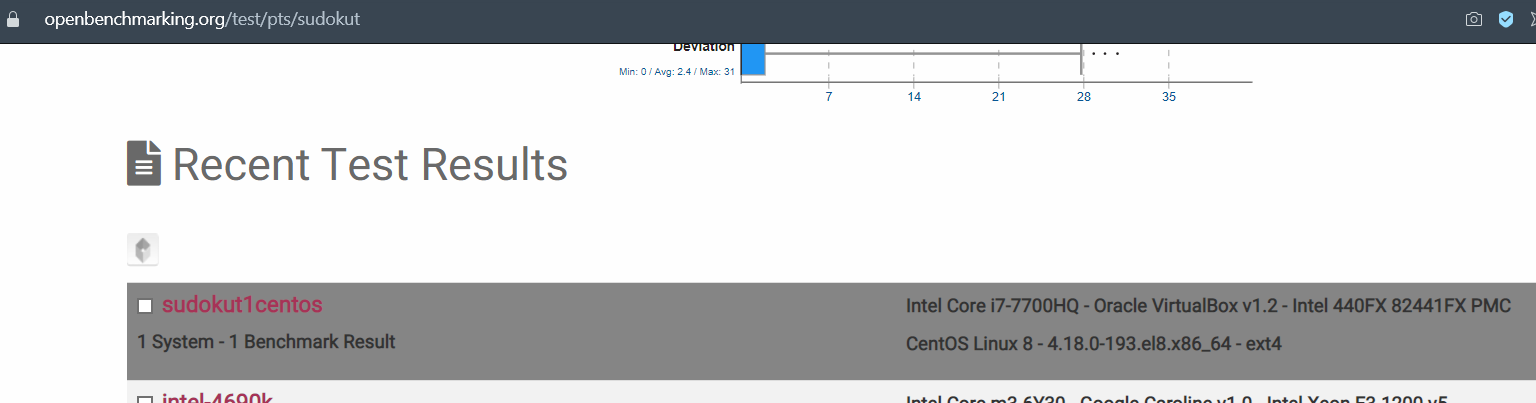
**-sudokut:** [**https://sourceforge.net/projects/sudokut/**](https://sourceforge.net/projects/sudokut/)

Procedemos con la instalación y ejecución del test como con el anterior caso. Instalamos con ***phoronix-test-suite install pts/sudokut*** y ejecutamos con la opción *run*.





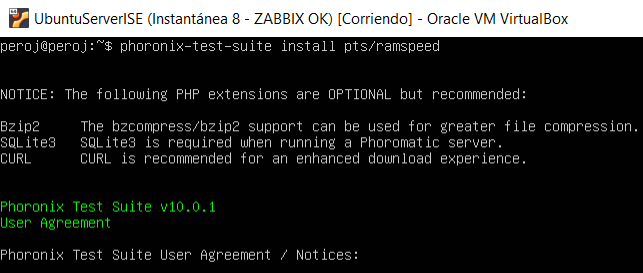
También podemos visualizar los resultados desde un navegador como en el caso anterior copiando los resultados del test. Pero tenemos la alternativa de poderlos ver desde la página de la organización de *openbenchmarking.org/test/pts/sudokut*.



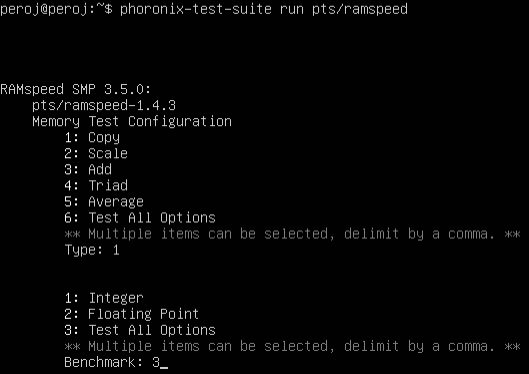
**-En Ubuntu Server:**

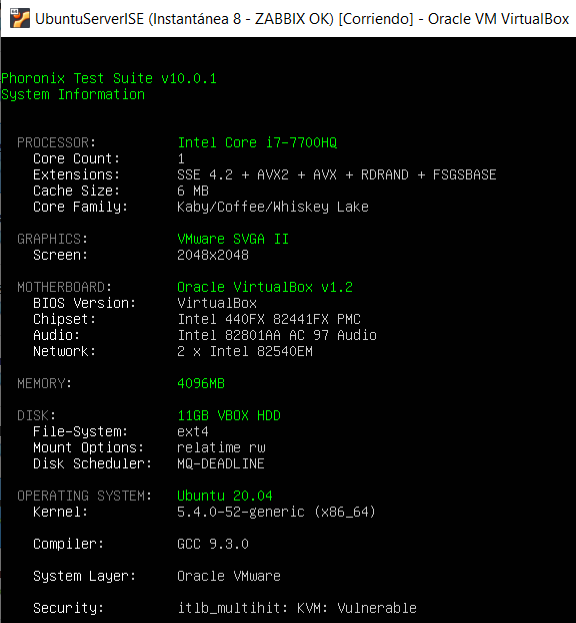
**-ramspeed:**

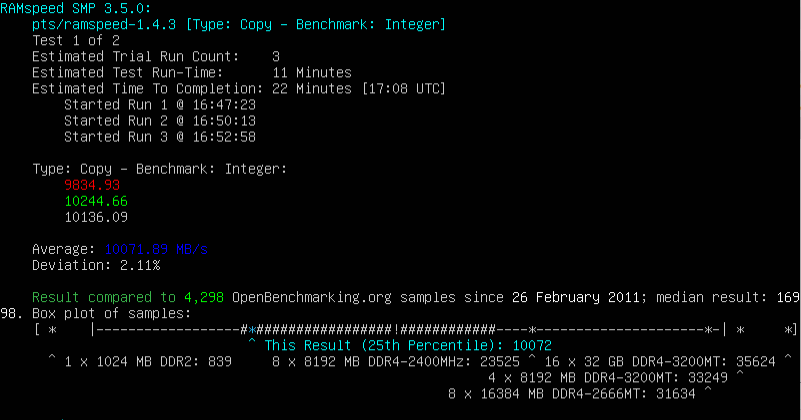
Lo instalamos con ***phoronix-test-suite install pts/ramspeed*** como hicimos anteriormente. Esto nos llevará un par de minutos después de introducir la contraseña de usuario.

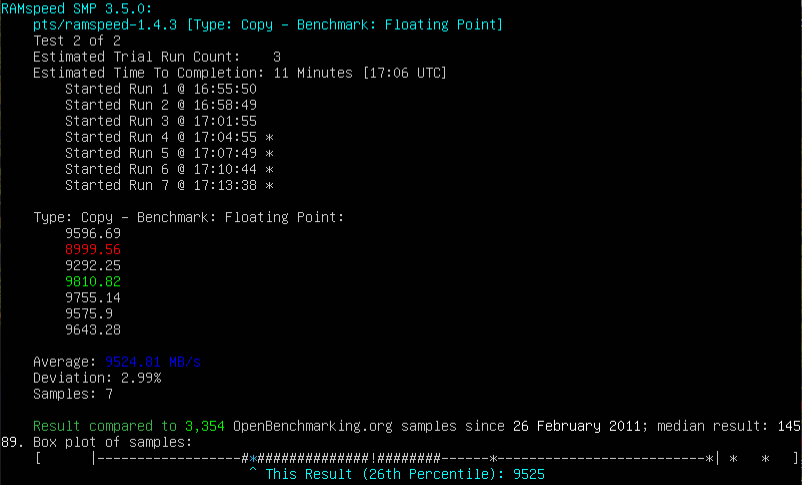


Y lo ejecutamos de igual manera con la opción ***run***y vemos también la información sobre el sistema donde se ejecuta:



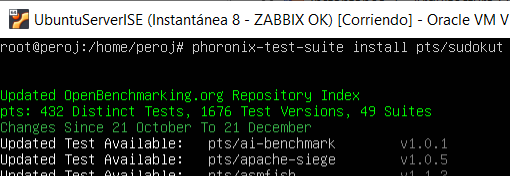


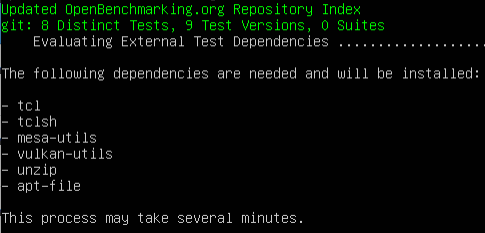
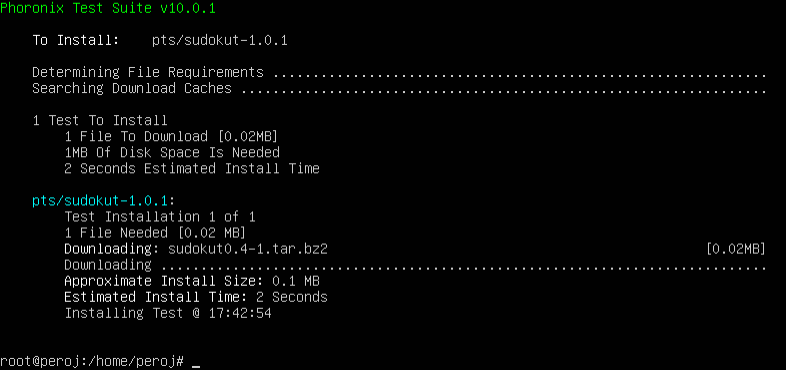
Comienza el test y esperamos unos 22 minutos hasta ver los resultados (también podremos verlos a través de *openbenchmarking.org* o desde nuestro navegador si tenemos el servidor web activo como hicimos en el caso anterior):



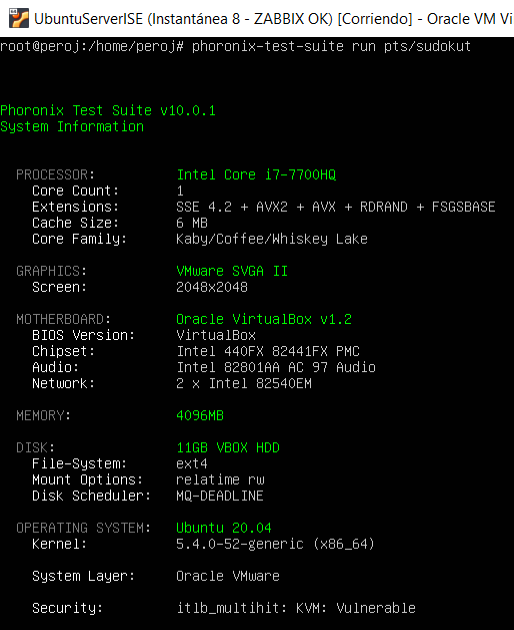
Vemos que en este caso es un poco mejor el test de memoria cuando copia datos enteros a cuando los copia en coma flotante.

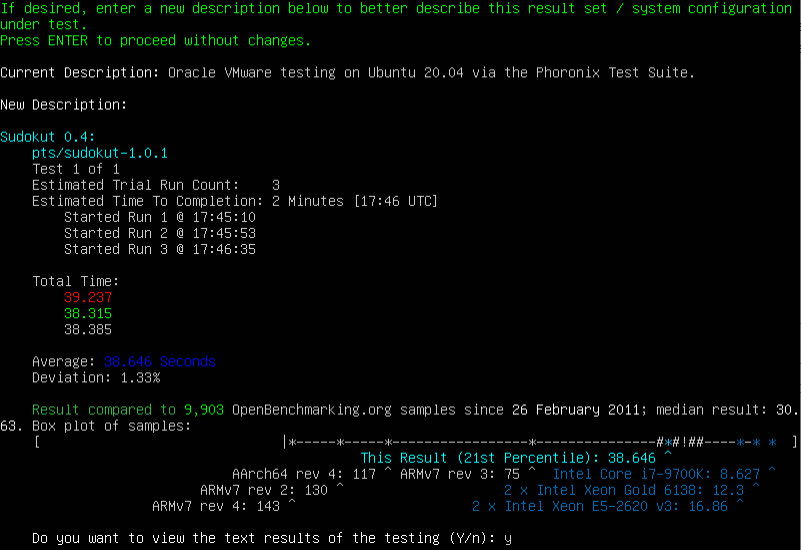
**-sudokut:**

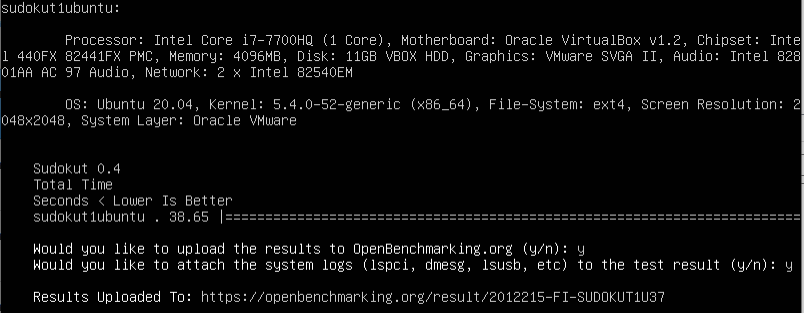




Una vez instalado lo ejecutamos y esperamos un par de minutos:







**-Comparando resultados**

**-ramspeed:**

CentOS:

Integer 10068.96 Mb/s

Floating Point 10295.87 Mb/s

Ubuntu:

Integer 10071.89 Mb/s

Floating Point 9524.81 Mb/s

Como podemos comprobar y sabiendo que cuanto mayores sean los resultados mejores serán en este tipo de test, en el caso de la ejecución del test para enteros, los resultados son muy parejos entre CentOS y Ubuntu Server. En el caso de las lecturas y escrituras de los datos en coma flotante tiene mejores resultados en CentOS, con una diferencia de unos 750 Mb/s, quedando así un poco por detrás Ubuntu Server en este aspecto del test *ramspeed*.

**-sudokut:**

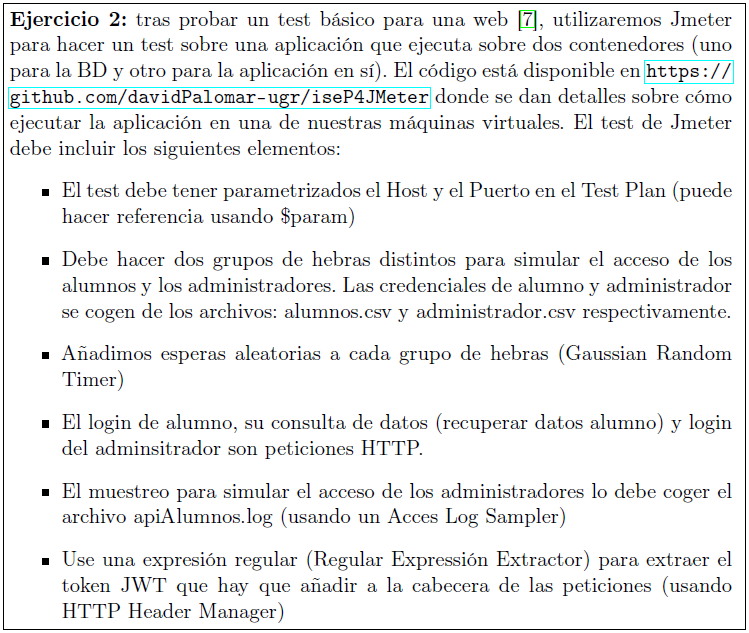
CentOS: 45.27 seconds

Ubuntu: 38.65 seconds

En este caso Ubuntu Server nos muestra mejores resultados en este test de estrés al procesador que en CentOS. Cuanto menor sea el tiempo mejor, dado que más rápido resuelve este test.

También podríamos fijarnos en los percentiles y tener una visión más global del rendimiento del sistema ante estos test.

## APACHE BENCHMARK Y JMETER



**Docker**

La MV, dada la infraestructura del sistema monta un SO para cada una de las máquinas virtuales con su hardware virtualizado. Sin embargo, docker no, docker usa la infraestructura del sistema y el SO del host para montar la interfaz de docker y aquí desplegar las aplicaciones, así las aplicaciones pueden tener conexión entre ellas de forma nativa (en MV no se podría de forma nativa, si a través de servidores web por ejemplo) y las aplicaciones comparten la misma memoria.

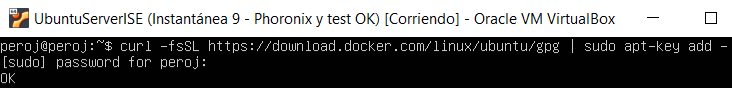
Gracias a docker se han popularizado mucho los microservicios que son pequeños servicios que se abren, hacen lo que tienen que hacer y se cierran, ahorrando así recursos, ya que no tiene que estar siempre activo en estado “ocioso” o esperando peticiones.

Dado al desarrollo de la tecnología web muchas veces no nos interesa tener muchos servicios a la vez, entonces, la solución sería tener los servicios principales activos y si llega una peticion que requiere otro servicio en concreto, se levanta este servicio con docker, se devuelve la petición y se vuelve a bajar el servicio (microservicios).

**Instalación de Docker en Ubuntu Server:**

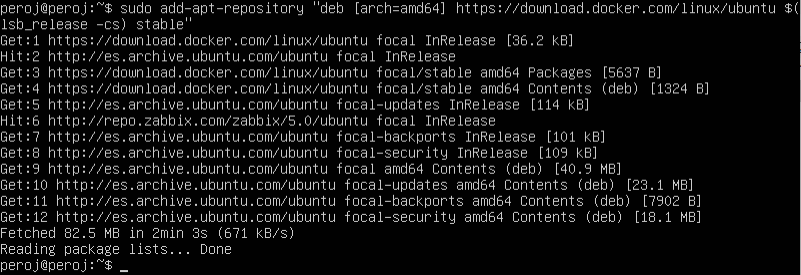
Añadimos llave GPG para validar el repositorio:

***curl -fsSL*** [***https://download.docker.com/linux/ubuntu/gpg***](https://download.docker.com/linux/ubuntu/gpg) ***| sudo apt-key add -***



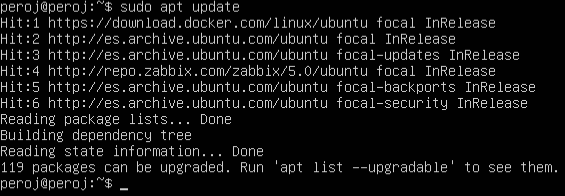
Añadimos repositorio (si no funciona, sustituir ***$(lsb\_release -cs)*** por ***focal***):

***sudo add-apt-repository “deb [arch=amd64]*** [***https://download.docker.com/linux/ubuntu***](https://download.docker.com/linux/ubuntu) ***$(lsb\_release -cs) stable”***



Una vez que tenemos el repositorio añadido, actualizamos los repositorios y procedemos con la instalación de docker.

***sudo apt update***



Buscamos el repositorio docker (community edition) y lo instalamos:

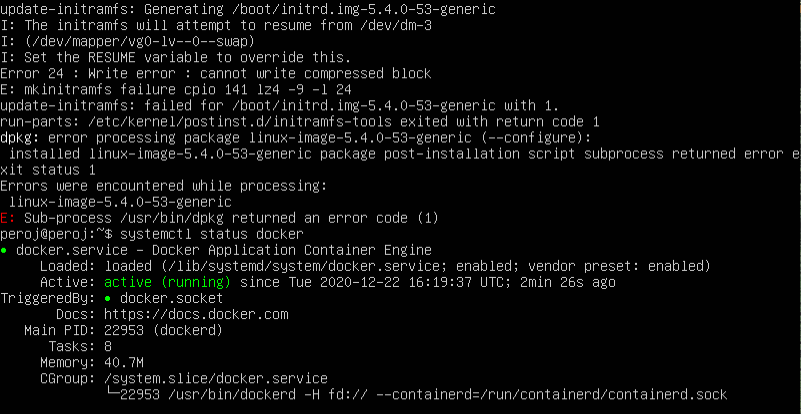
***apt search docker-ce***

***sudo apt install docker-ce***



Nos salta un error del *dpkg* por lo que volvemos atrás y lo hacemos con la opción *focal* como hemos visto anteriormente y ya si se instala (aunque me sigue dando algún problema *dpkg)*.

Comprobamos que el servicio está activo con ***systemctl***.



Para poder usar docker sin tener que ser root, debemos añadir nuestro usuario al grupo docker y volver a logearnos:

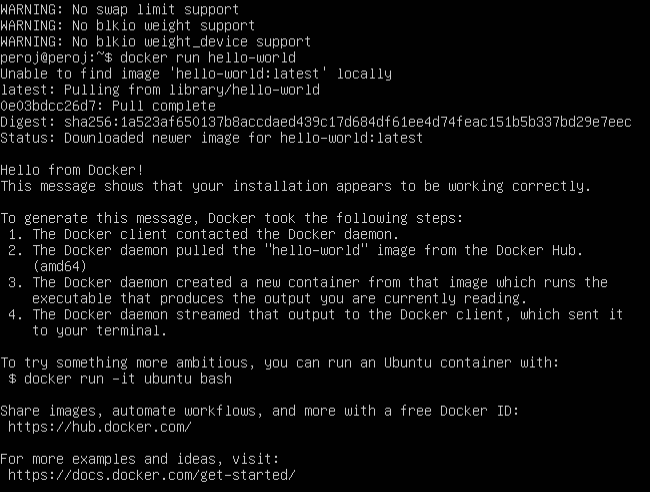
***sudo usermod -aG docker peroj***



Después de relogearnos comprobamos que está instalado con:

***docker info***

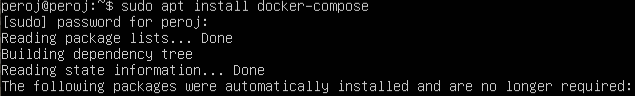
***docker run hello-word***



**Instalación de Docker Compose en Ubuntu Server:**

Para orquestar contenedores tenemos docker-compose (entre otras herramientas, véase buidah, kubernetes...) que nos permite desplegar aplicaciones con varios contenedores de modo que cada uno tenga una función. Los pasos para instalarlo son:

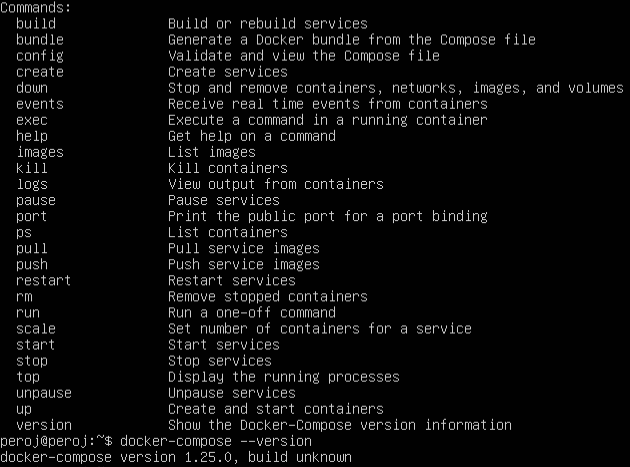
Como ya tenemos el repo configurado, instalamos con ***apt install docker-compose***.



Y probamos con:

***docker-compose***

***docker-compose --version***



**Instalación de la aplicación para el test con JMeter**

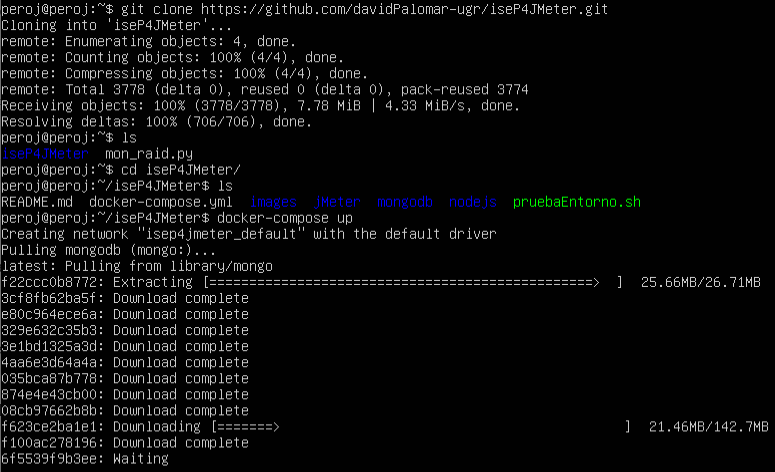
Una vez tengamos instalado docker y docker-compose comenzamos con el despliegue de la aplicación en nuestro Ubuntu Server, para ello, lo primero que debemos hacer es clonar el repositorio de la aplicación:

***git clone*** [***https://github.com/davidPalomar-ugr/iseP4JMeter.git***](https://github.com/davidPalomar-ugr/iseP4JMeter.git)

Tras esto, tendremos un directorio nuevo: iseP4JMeter al cual podremos acceder y levantar la aplicación por el puerto 3000 con docker-compose:

***cd iseP4JMeter***

***docker-compose up***



* Nociones básicas de la aplicación:

El ejercicio consiste en realizar una prueba de carga de una API Rest empleando JMeter.

La API se ha desarrollado empleando: MongoDB, NodeJS y Express

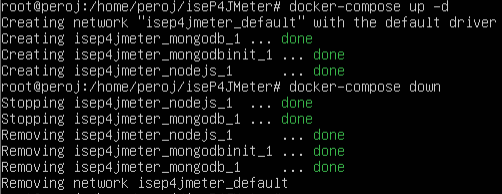
El servidor se distribuye en forma de una aplicación de contenedores Docker sobre Compose. Ambas aplicaciones deben estar instaladas para ejecutar el servidor.

Tras descargar el código, situarse en el directorio principal (al mismo nivel del archivo docker-compose.yaml) y ejecutar (en segundo plano):

***docker-compose up -d***

Para parar la aplicación ejecutar:

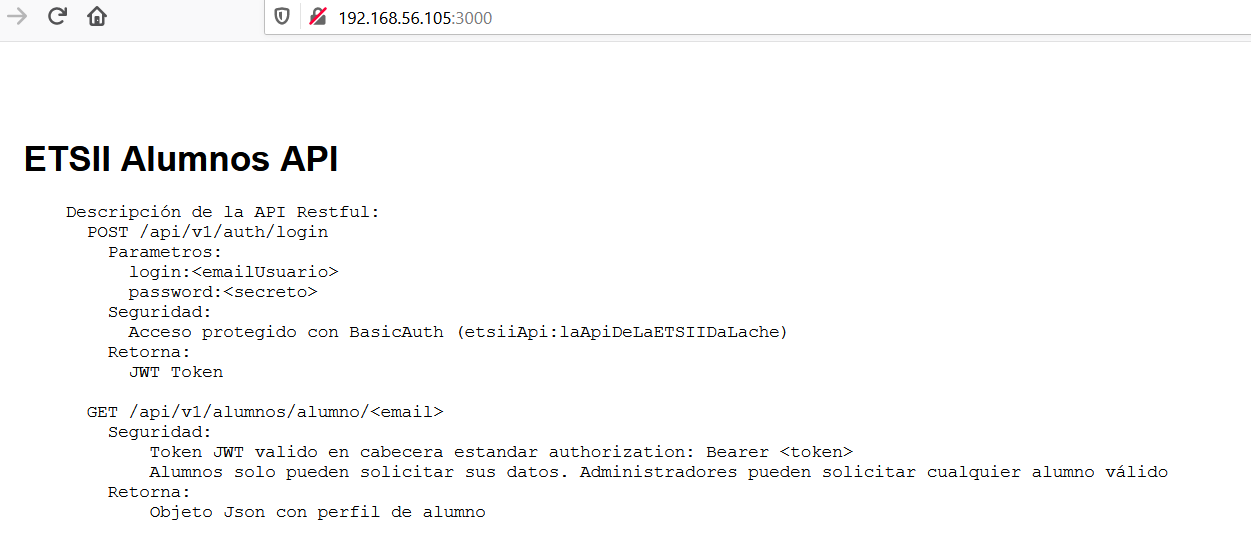
***docker-compose down***



Docker descargará las imágenes base y construirá las nuevas imágenes para la aplicación.

Accediendo con un navegador a http://<IPDockerContainer>:3000 (ej.- <http://localhost:3000>) se presenta la descripción básica de la api. Se tratan de dos métodos:

* /auth/login: Permite identificarse al usuario como Alumno o Administrador. El acceso a este servicio está protegido por Http BasicAuth. Una vez autenticado, se obtiene un [JWT](https://jwt.io) Token para ser empleado en el servicio de alumno.
* /alumnos/alumno: Devuelve el registro de calificaciones del alumno. Los administradores pueden consultar los datos de cualquier alumno. Los alumnos solo los propios. Se debe proporcionar un JWT válido (obtenido en el login) que portará la identidad (autenticación) y rol (autorización) del usuario.



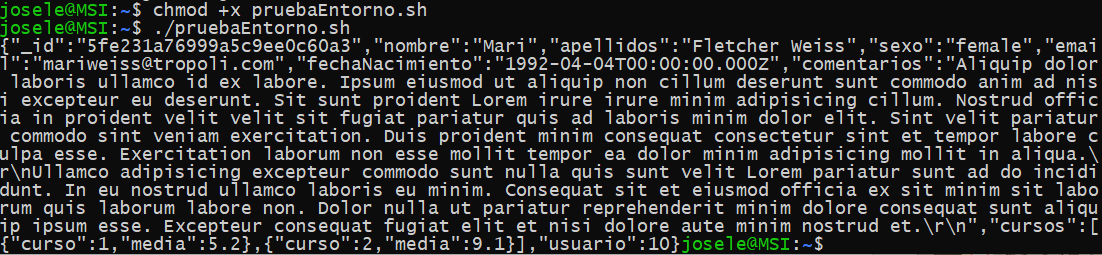
El proceso de consulta es el siguiente:

1. Identificarse en el servicio de login proporcionando credenciales de válidas de alumno o administrador. Obteniendo un token.
2. Solicitar los datos del propio alumno identificado (alumno) o de un grupo de alumnos (administrador).

Para una prueba más detallada de que el entorno funciona, ejecutamos el script desde el host:

**pruebaEntorno.sh**

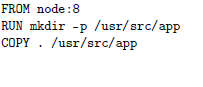
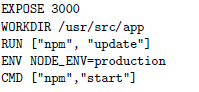
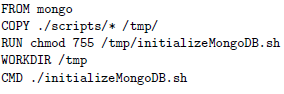
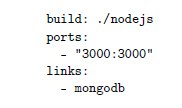
Este script contiene la secuencia descrita anteriormente en invocaciones a *curl*, por lo que describe las operaciones a realizar. La primera línea contiene la variable SERVER. Debe definirse a la IP donde corre el contenedor de Docker. Si todo está correctamente configurado, obtendrá el perfil de un alumno.

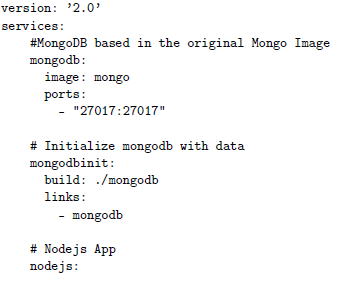


<dockerfile>:

En este caso el docker se ha creado de un docker previo de node (FROM). Y ha hecho una carpeta dentro del docker que se llama app(RUN) y lo ha copiado al docker a la carpeta app (COPY). Lo a expuesto el docker por el puerto 3000 (EXPOSE) ya que se entra por este puerto a este docker. Como directorio de trabajo ha puesto la carpeta que ha creado (WORKDIR). Actualiza npm(RUN), pone el entorno de node como producción(ENV) y lo arranca(CMD).

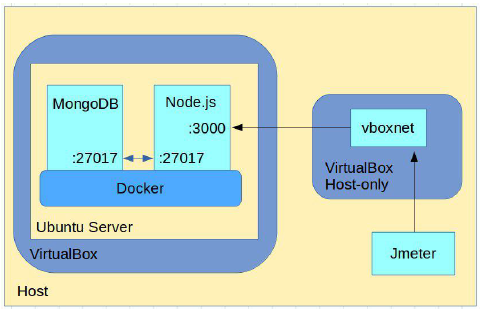
Esto es un dockerfile, como se construye un docker. Normalmente heredas de una imagen que ya existe en docker y ejecutas los comandos que tu quieres. Y se te queda como una especie de máquina virtual, un CONTENEDOR con esa configuración. Cuando tu haces el docker run lo que hace es arrancar esa configuración. Esto para node.

Para mongo igual. Hereda de mongo. Copia un script sh y le cambia los permisos y lo inicializa. En el script inicializa una base de datos de mongo.

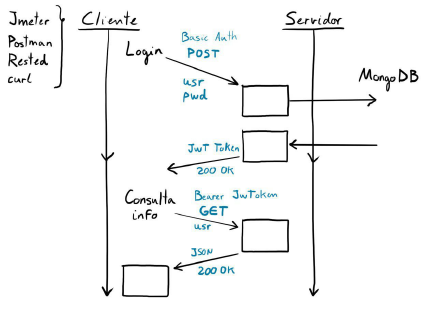
Archivo *compose:*

Para juntarlos lo que se hace es el docker compose. Una vez que están abiertos le dices que puertos quieres y cómo se inicializa.

Descripción de la aplicación:



Tenemos JMeter que manda las peticiones al servidor correspondiente (que es Ubuntu server). Ubuntu Server tiene con el docker levantado la aplicación de node y mongodb. JMeter va a hacer las peticiones al puerto 3000 e internamente ya están configurados los docker para que por el puerto 27017 node se conecta a mongo y mongo le devuelva la información (esta conexión está definida internamente).



En este caso, el cliente sería JMeter.

Primero mandamos al servidor una petición POST para que nos devuelva el token. El servidor internamente comprueba el usuario en la base de datos mongo y devuelve al servidor si todo ha ido bien un 200 con el token (en Consulta Info Jmeter recibe el token). Una vez que lo tenemos hacemos una petición GET usando el token y ya nos devolvería el servidor la información del usuario.

En JMeter se hacen 2 peticiones,una con el token y otra con la información y se va a devolver dos respuestas, una con el token si el login ha ido bien y otra respuesta con la información.

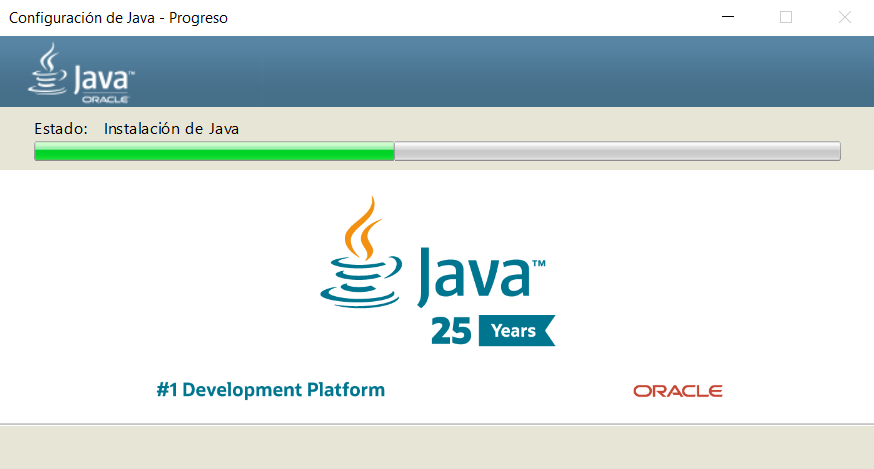
Instalaremos y configuraremos JMeter en la máquina anfitrión, Windows en mi caso y retomaremos a continuación en este punto.

**Instalación y configuración de JMeter**

Apache JMeter es un benchmark desarrollado en Java. Nos da una forma mucho más profesional para testear un servidor web ya que permite ejecutar varios procesos a la vez, varias hebras a la vez, simular situaciones reales a las que se puede ver sometido un servidor web en cualquier momento de su funcionamiento. Permite llevar a cabo test de carga muy reales, permitiendo crear concurrencia real en el sistema porque permite ejecutar varias hebras dentro de una cpu así como distribuir la creación de carga en varias máquinas.

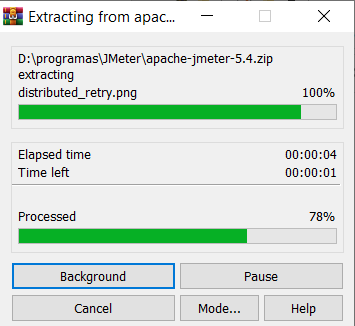
Jmeter es una aplicación Java por lo que requiere disponer de una máquina virtual java instalada (JVM). La forma más común de definir los test, y que emplearemos, es con la consola gráfica. Para evitar tener que instalar un servidor de Xwindows en las MVs, vamos a ejecutar JMeter en el ordenador anfitrión. Los pasos básicos son:

1º) Descargar e instalar máquina virtual java 11: <https://www.java.com/es/download/>

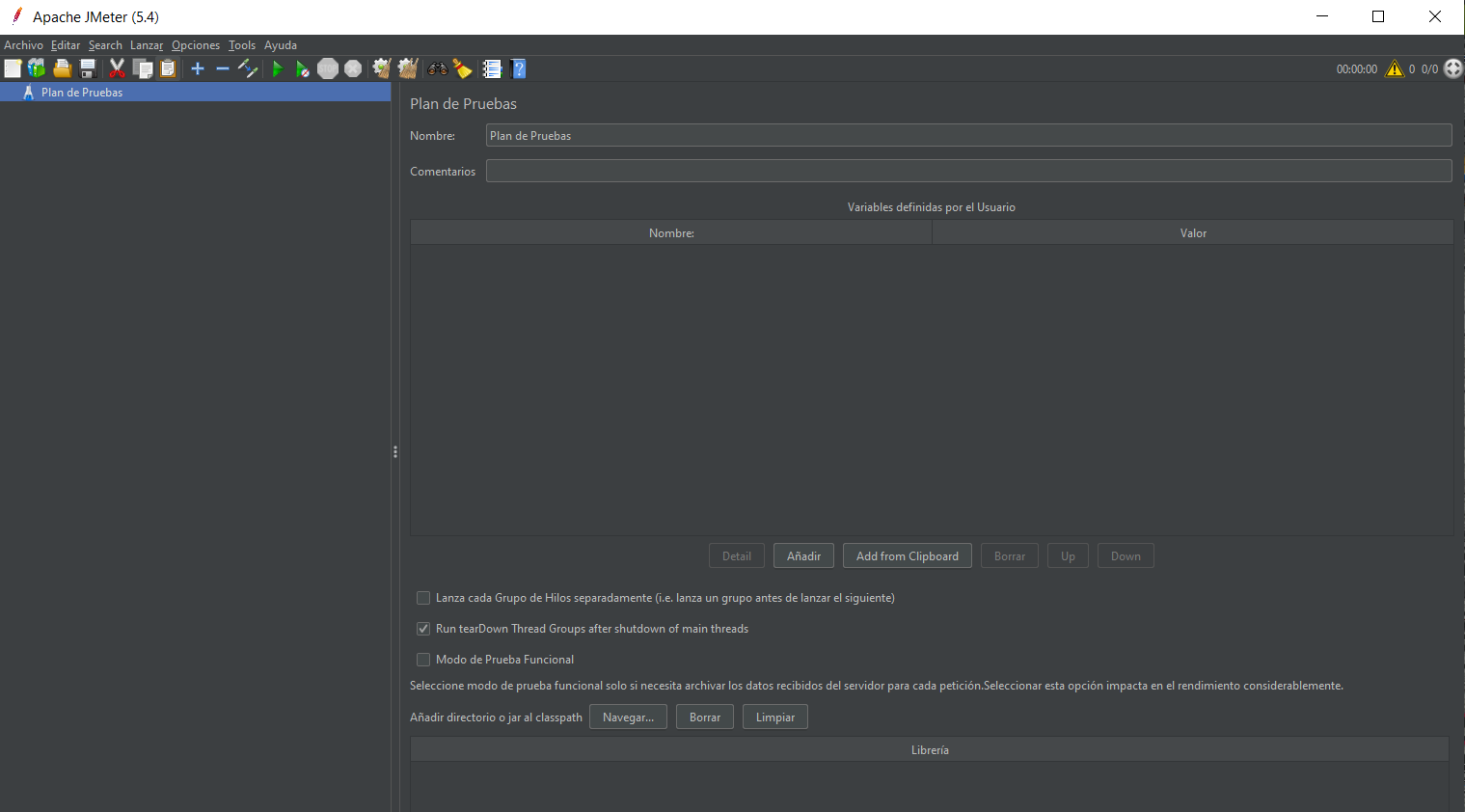


2º) Descargar JMeter desde un mirror cercano: <https://jmeter.apache.org/download_jmeter.cgi>

Para instalarlo en Windows, simplemente descargamos comprobando la versión correcta y el .zip lo descomprimimos en el directorio donde queramos instalarlo:



Desde el explorador de archivos (o el *cmd*)de Windows nos movemos al directorio *<jmeterlocation>\bin* y ejecutamos *jmeter.bat* o *jmeter.bat* .



3º) Seguir el tutorial para realizar la prueba de carga: <https://jmeter.apache.org/usermanual/build-web-test-plan.html>

**Configuración y prueba de carga contra iseP4JMeter**

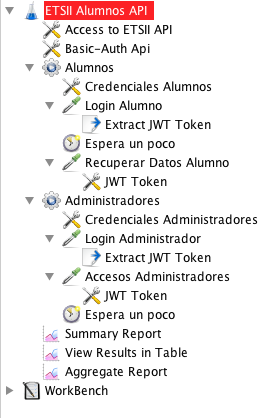
El subdirectorio JMeter contiene los archivos necesarios para realizar la sesión de prácticas:

* alumnos.csv: Archivo con credenciales de alumnos
* administradores.csv: Archivo con credenciales de administradores
* apiAlumno.log: Log de acceso Http en formato apache.

La prueba de JMeter debe:

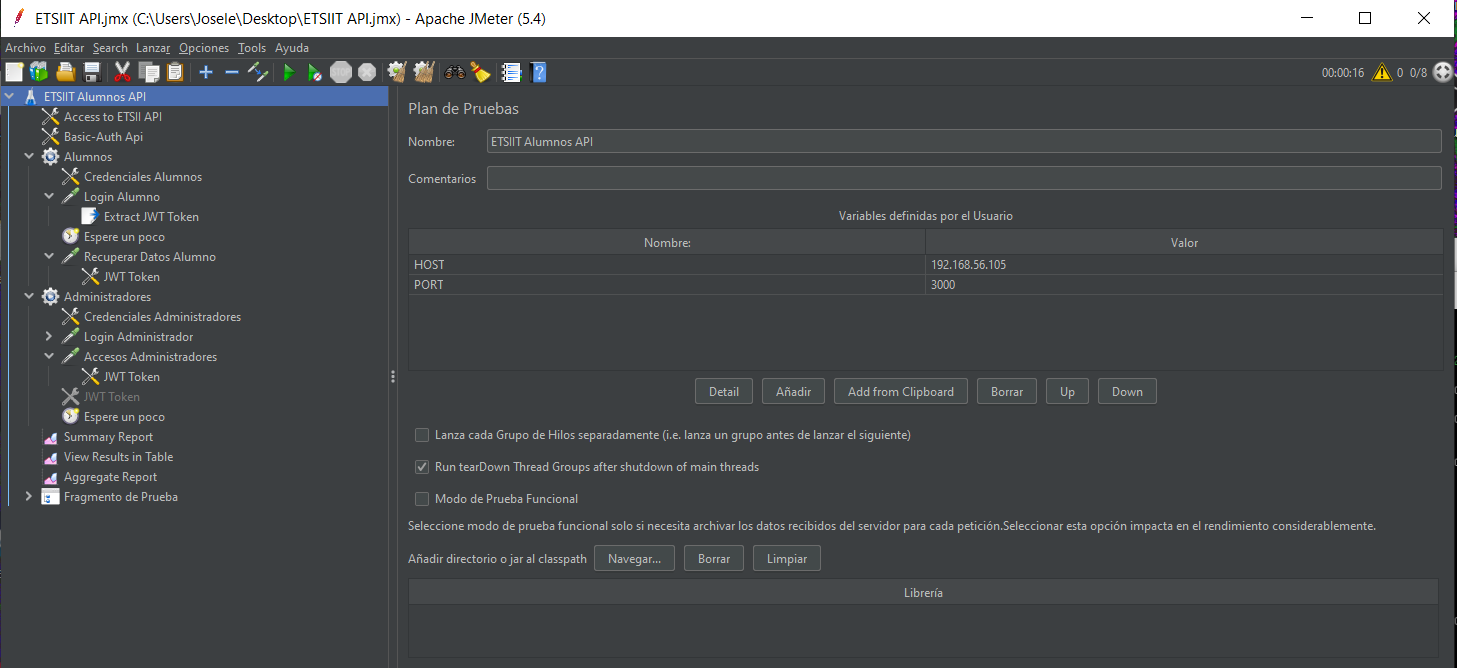
* Parametrizar el "EndPoint" del servicio mediante variables para la dirección y puerto del servidor. Emplee "User Defined Variables" del Test Plan.
* Definir globalmente las propiedades de los accesos Http y la Autenticacion Basic. Emplee HTTP Request Defatuls y HTTP Authorization Manager.
* Los accesos de alumnos y administradores se modelarán con 2 Thread Groups independientes. La carga de accesos de administradores se modelará empleando el registro de accesos del archivo apiAlumno.log

La imagen siguiente presenta un posible diseño de la carga:

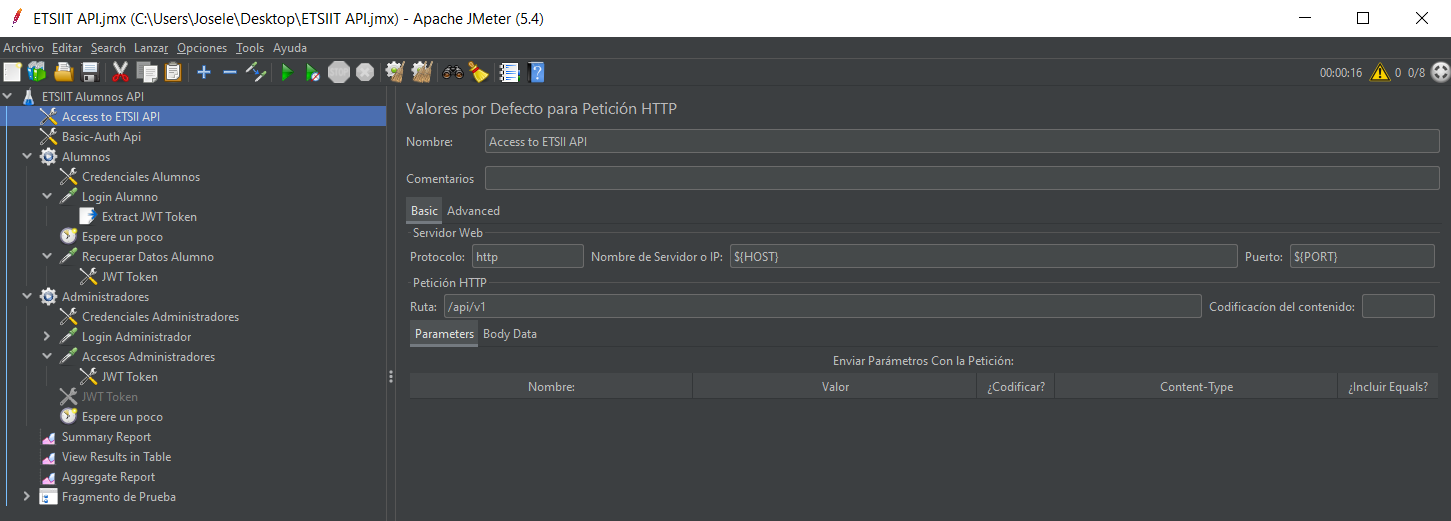
[](https://github.com/davidPalomar-ugr/iseP4JMeter/blob/master/images/jmeterLoadTest.png)

Procedemos siguiendo los pasos para la configuración de JMeter y lo requerido por la imagen anterior.

* Creamos el nuevo plan de pruebas llamado ETSIIT Alumnos API y añadimos dos variables nuevas: el host con la IP de UbuntuSever y el puerto 3000.



* Insertamos los valores por defecto para las peticiones HTTP que haremos posteriormente en:

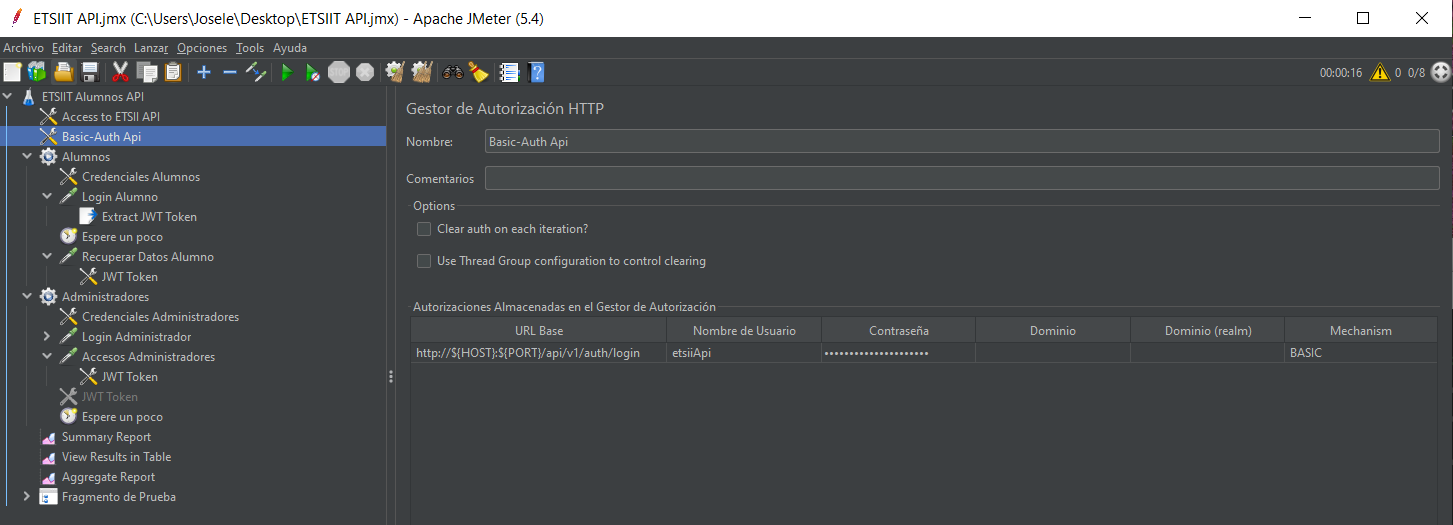
*Editar/Añadir/Elemento de configuración/Valores por defecto para petición HTTP.* Y establecemos los valores como sigue:

* Añadimos la autorización de la API en *ETSIIT\_API/Editar/Añadir/Elemento de configuracion/Gestor de autorización HTTP*

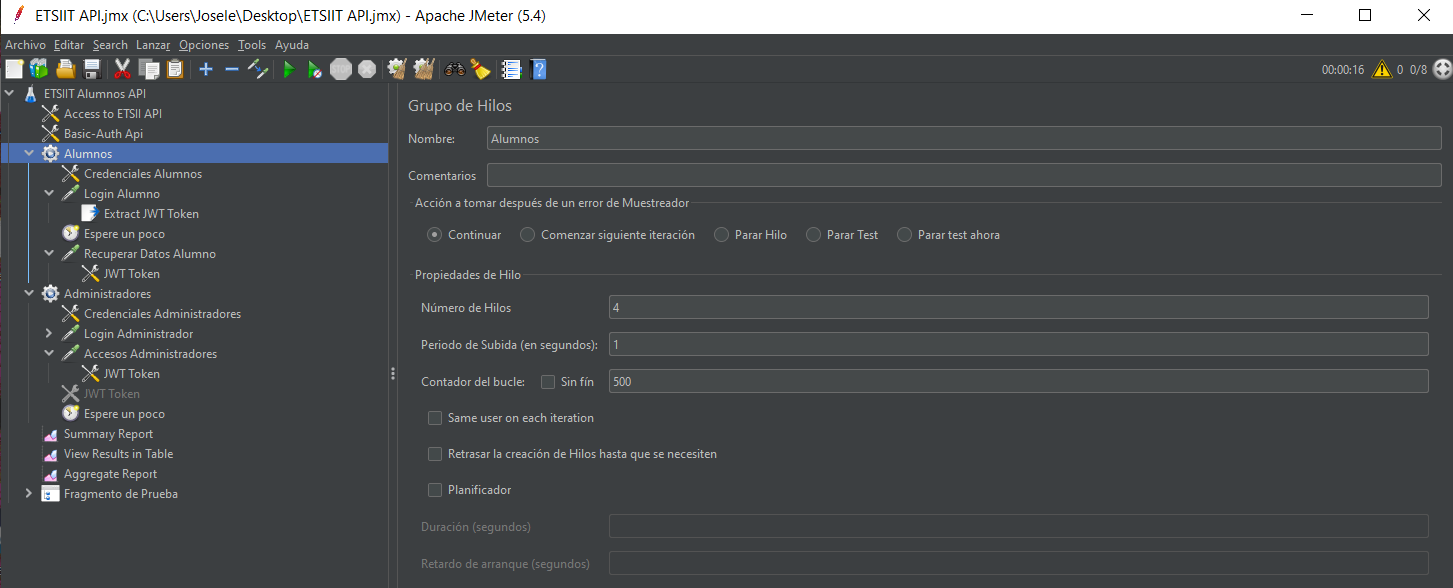
Rellenamos con los siguientes datos, como hemos visto anteriormente en la imágen del navegador:

URL Base: <https://$(HOST):$(PORT)/api/v1/auth/login>

Nombre: etsiiapi

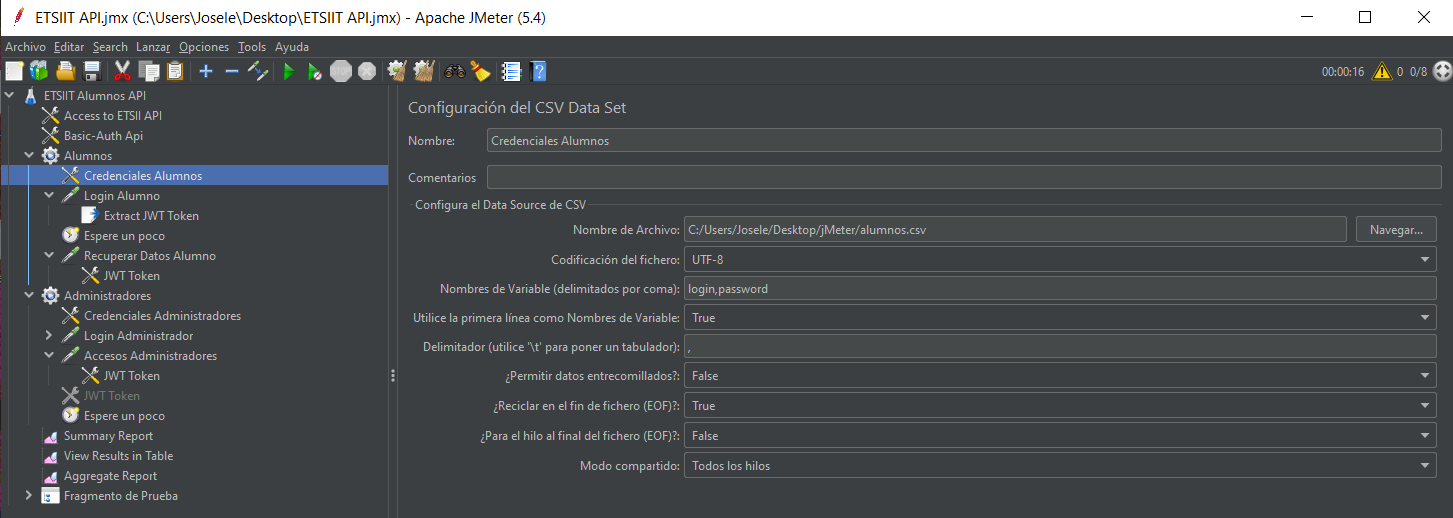
 Contraseña: laApiDeLaEtsiiDaLache

* **Alumnos**
* Añadimos la hebra (grupo de usuarios) de los alumnos como sigue:

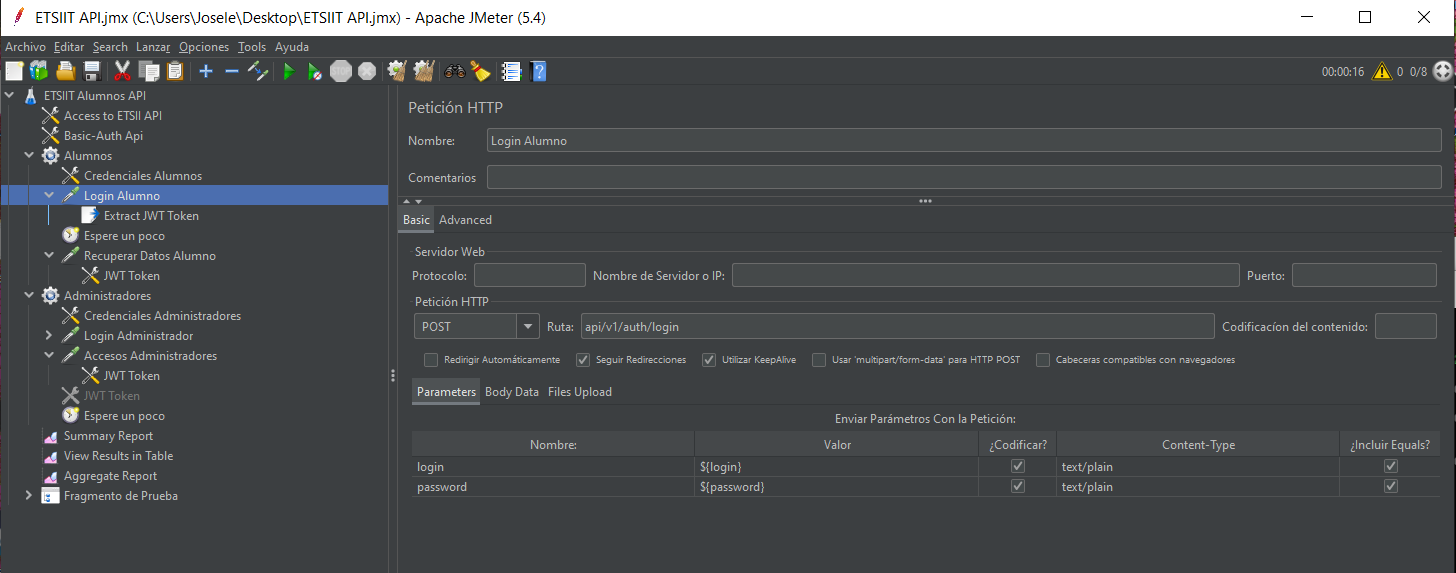
*ETSIIT Alumnos API/Editar/Añadir/Hilos(Usuarios)/Grupo de hilos*

* Configuramos las credenciales de los alumnos.

*Alumnos/Editar/Añadir/Elemento de configuración/Configuración del CSV Data Set*

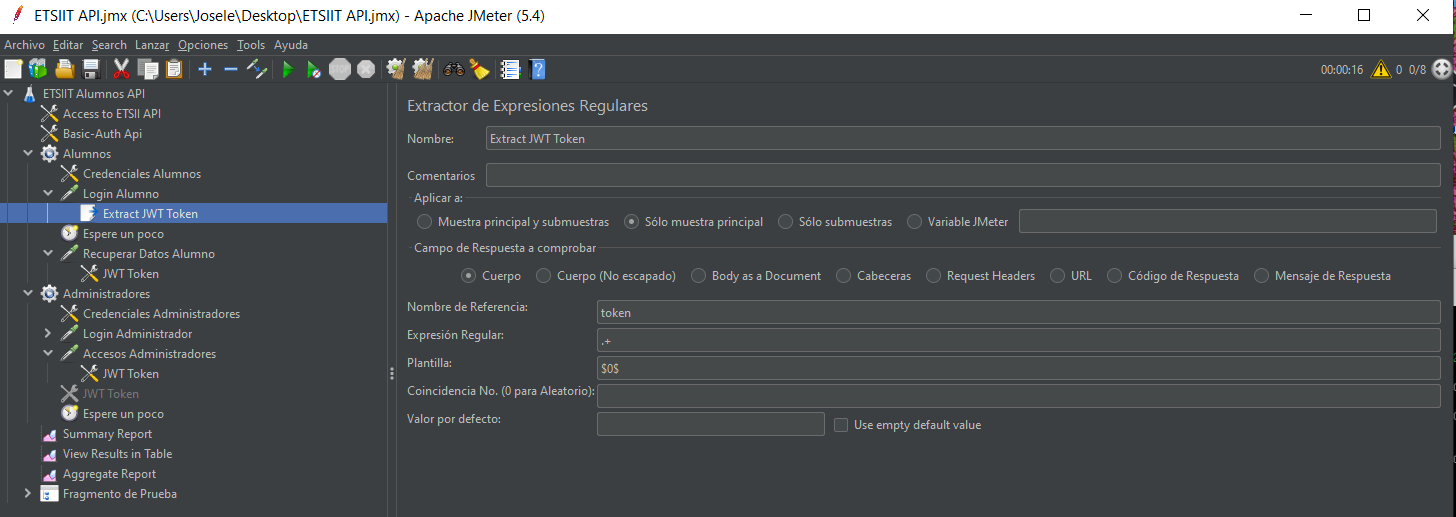


* Para poder logearnos, necesitamos el token del servidor, por lo que realizaremos una petición HTTP con el usuario y la contraseña. Una vez el servidor compruebe que los datos son correctos, nos devuelve el token.

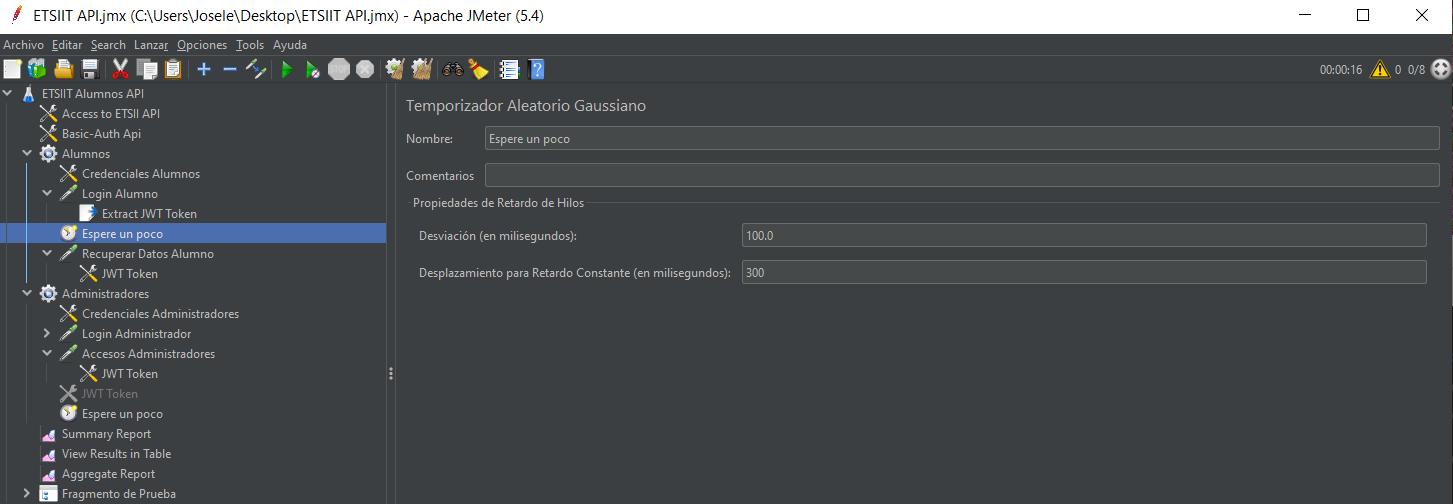
*Alumnos/Editar/Añadir/Muestreador/Peticion HTTP*

* Cuando realizamos la petición, necesitamos obtener el token que el servidor nos envía al comprobar los datos.

*Alumnos/Login Alumno/Añadir/Post Procesadores/Extractor de expresiones regulares*

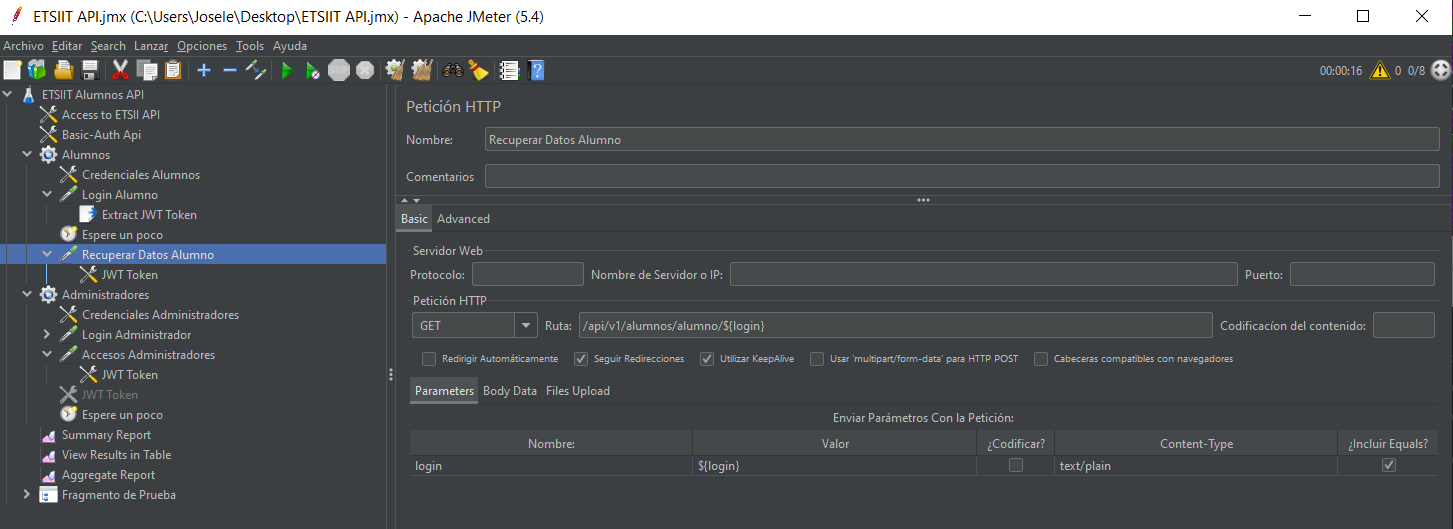


* Para hacer la simulación de los accesos de los alumnos más realista, creamos un temporizador aleatorio gaussiano.

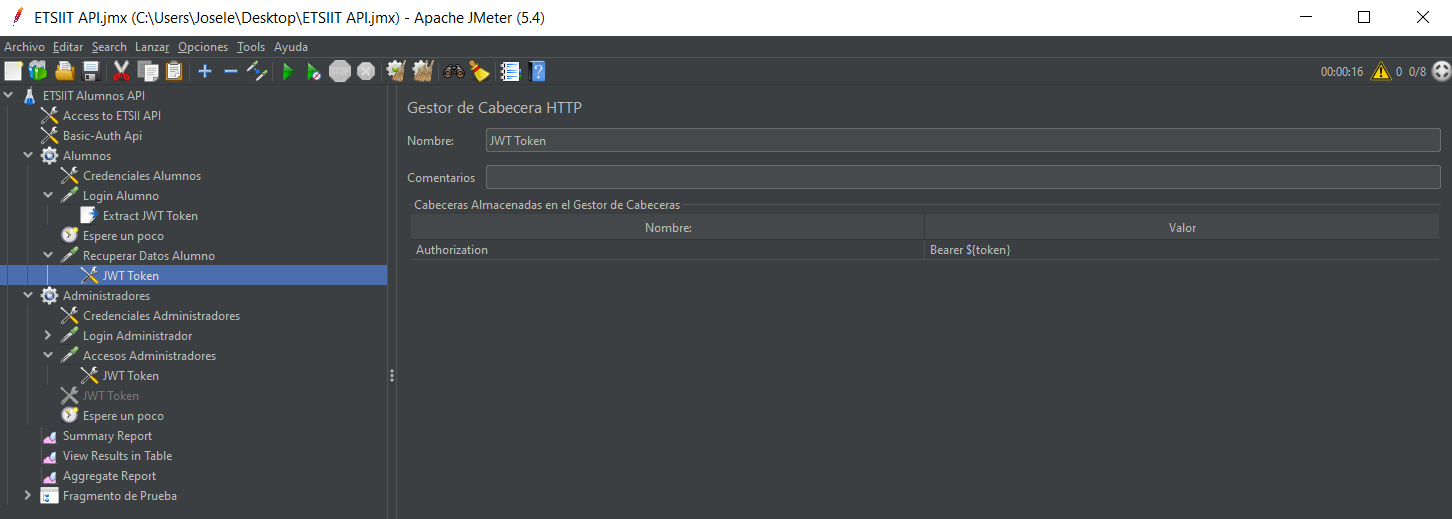
*Alumnos/Editar/Añadir/Temporizador Aleatorio Gaussiano*

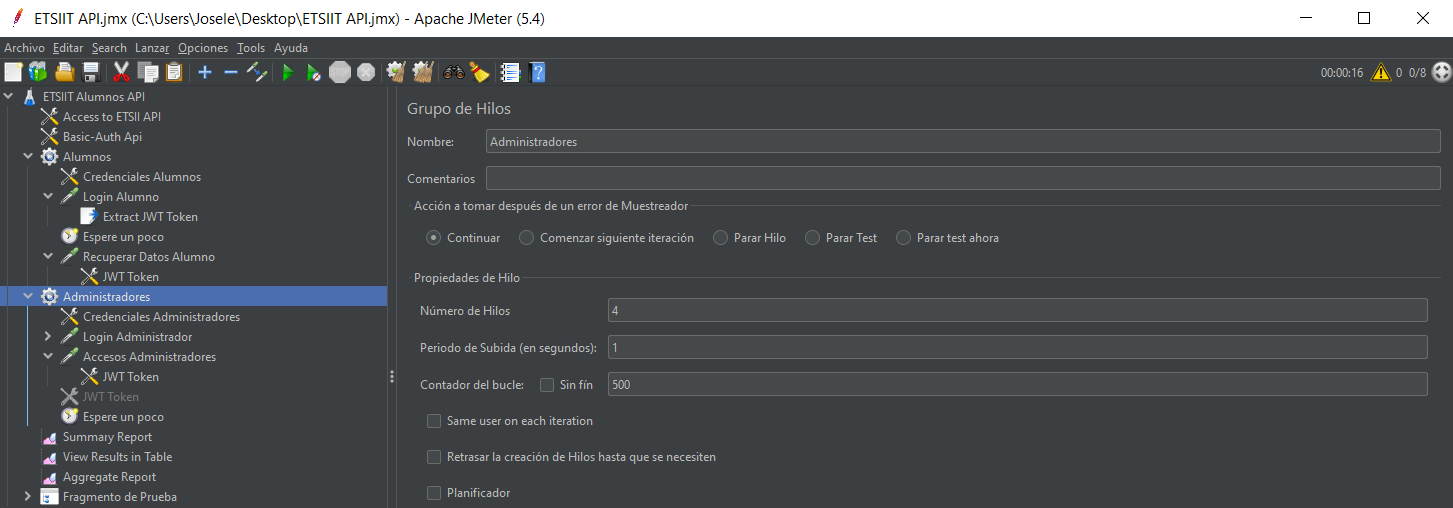
* Cuando tenemos el token para poder loguearnos, realizamos una petición al servidor con el usuario, la contraseña y el token que hemos obtenido anteriormente.

*Alumnos/Editar/Añadir/Muestreador/Peticion HTTP*

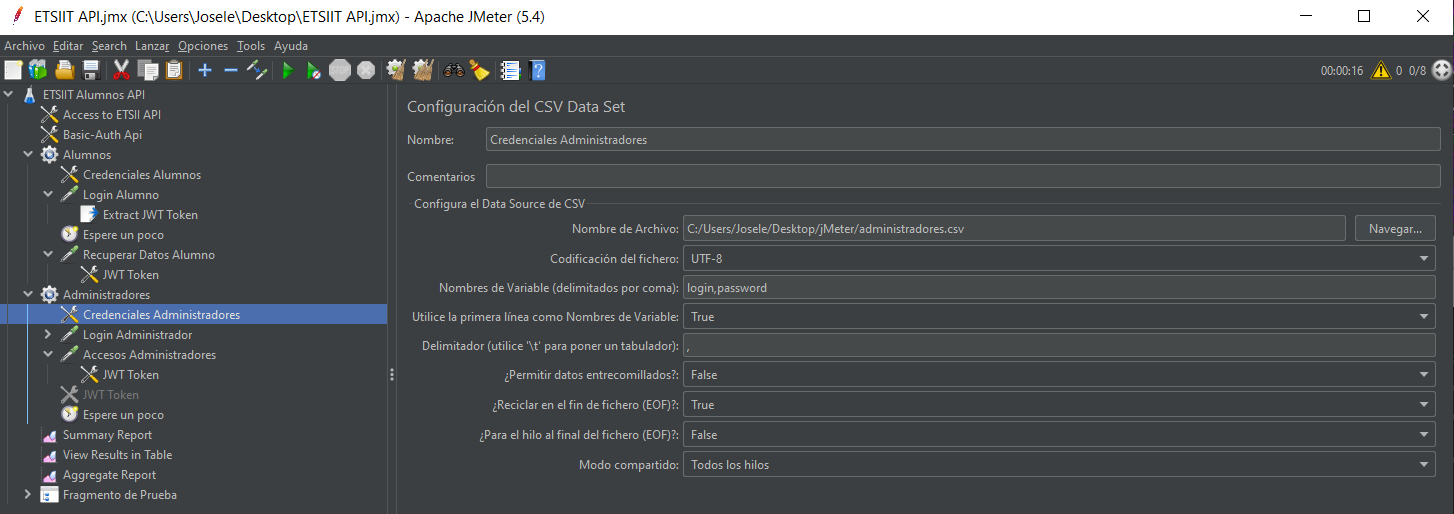


* Para resolver si al autenticarnos la aplicación nos redirige a otro nombre de dominio, entonces el token ya no sería útil, debemos mantener la sesión de la siguiente manera:

*Alumnos/recuperar Datos Alumno/Editar/Añadir/Elemento de configuración/Gestor de Cabecera HTTP*

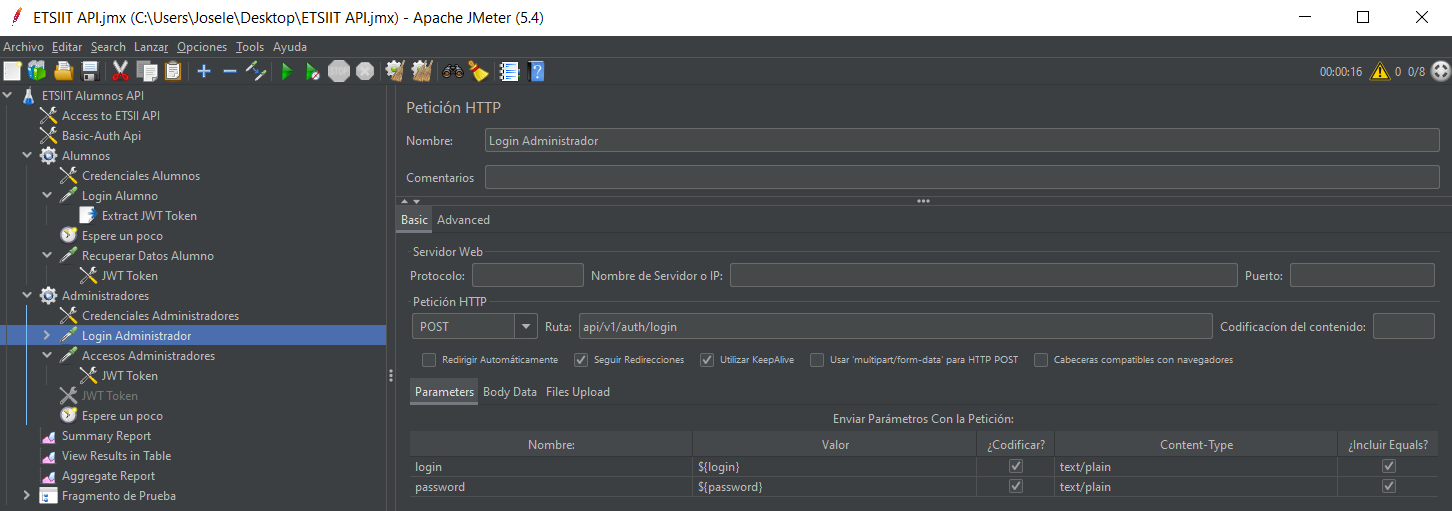
* **Administradores**
* Añadimos la hebra de los administradores
* Configuramos las credenciales de los administradores.

*Administradores/Editar/Añadir/Elemento de configuración/Configuración del CSV Data Set*



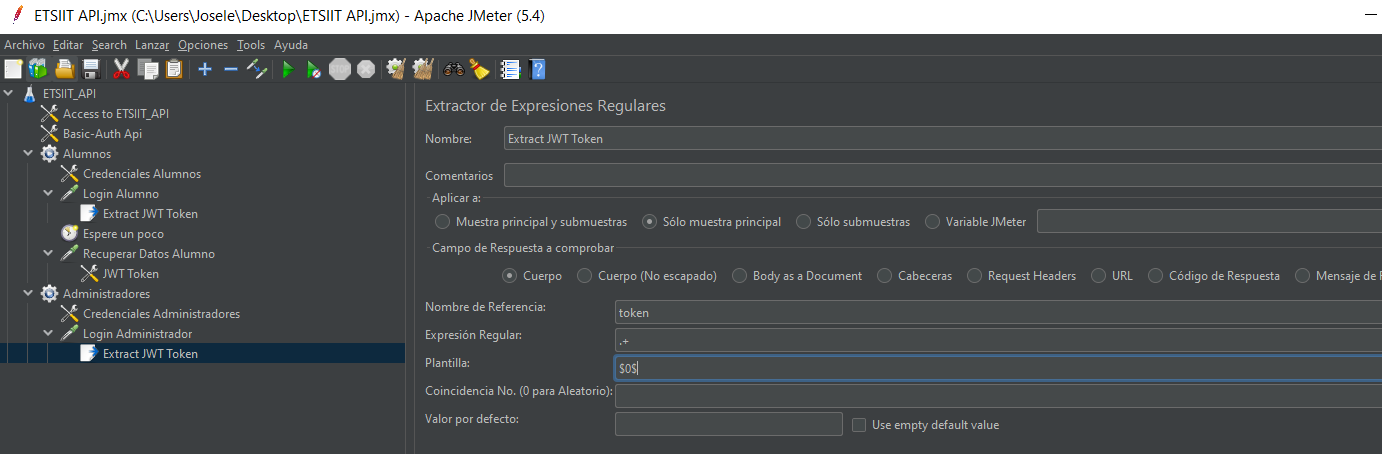
* Para poder logearnos, necesitamos el token del servidor, por lo que realizaremos una petición HTTP con el administrador y la contraseña. Una vez el servidor compruebe que los datos son correctos, nos devuelve el token.

*Administrador/Editar/Añadir/Muestreador/Peticion HTTP*

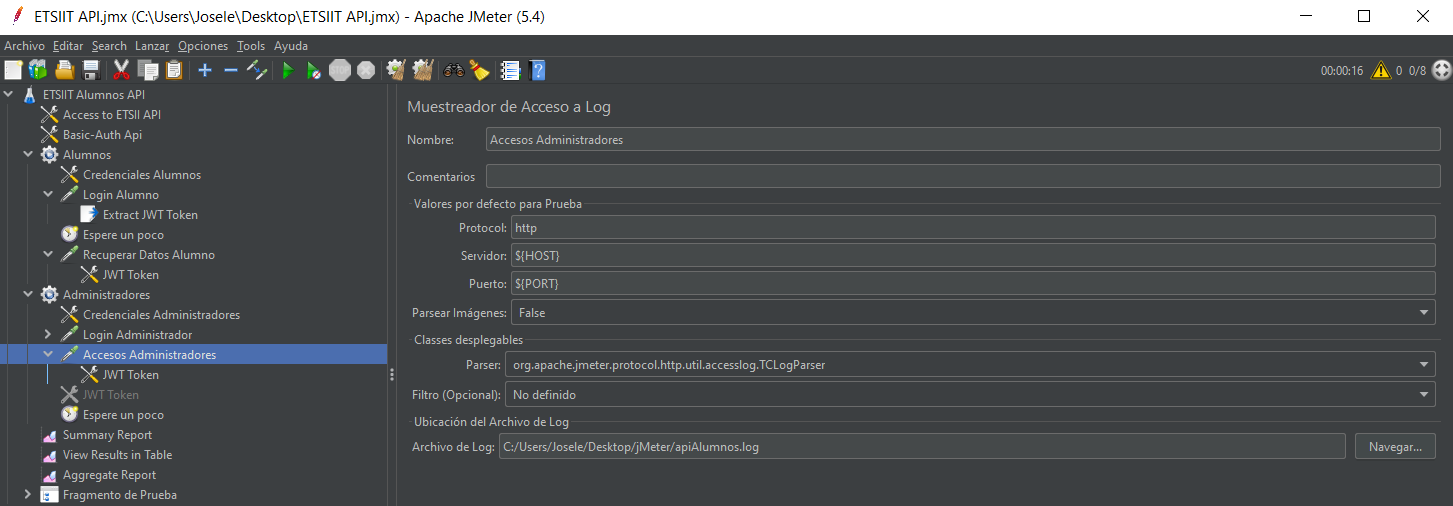


* Cuando realizamos la petición necesitamos obtener el token que el servidor nos envía al comprobar los datos.

*Administradores/Login Administradores/Añadir/Post Procesadores/Extractor de expresiones regulares*

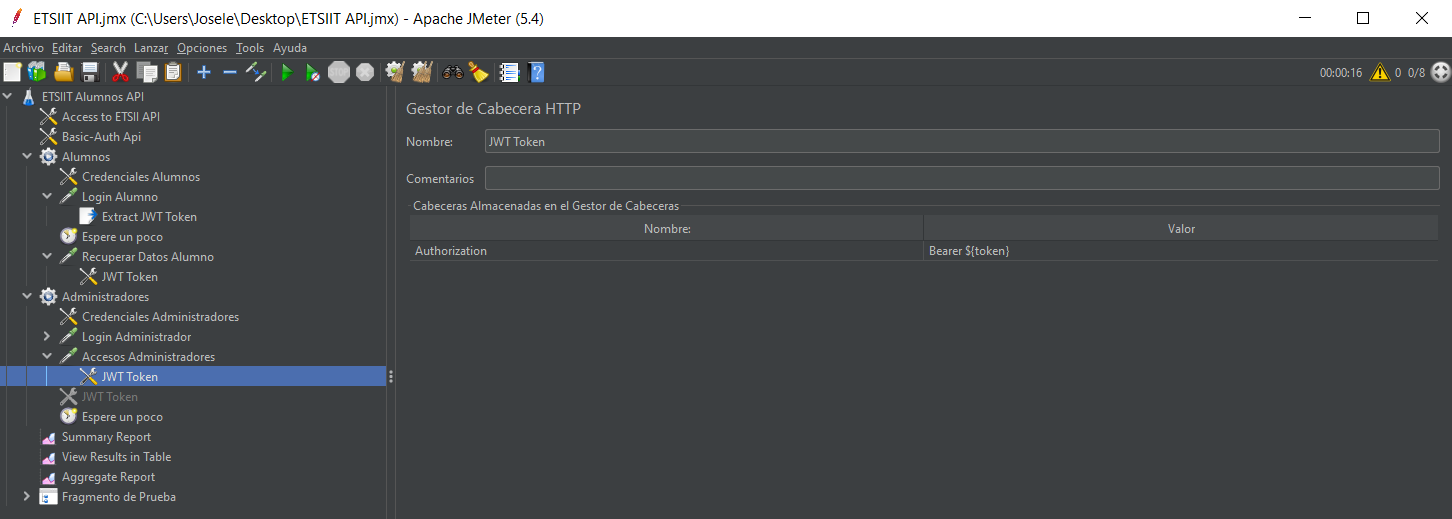


* Muestra el archivo .log de accesos al sistema.

*Administradores/Editar/Añadir/Muestreador/Muestreador de Acceso a Log*

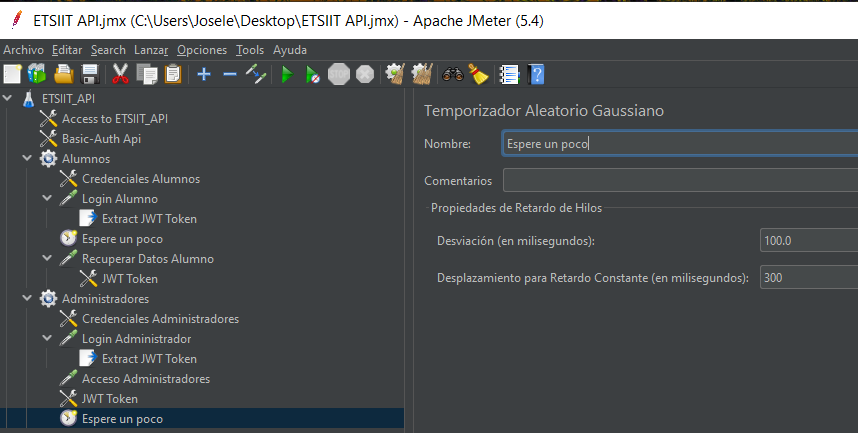
* Para resolver si autenticarnos la aplicación nos redirige a otro nombre de dominio, en lo que el token no sería útil.

*Administradores/Editar/Añadir/Elemento de configuración/Gestor de Cabecera HTTP*



* Para que la simulación de administradores accediendo sea más realista, creamos un temporizador aleatorio gaussiano.

*Administradores/Editar/Añadir/Temporizador/Temporizador Aleatorio Gaussiano*

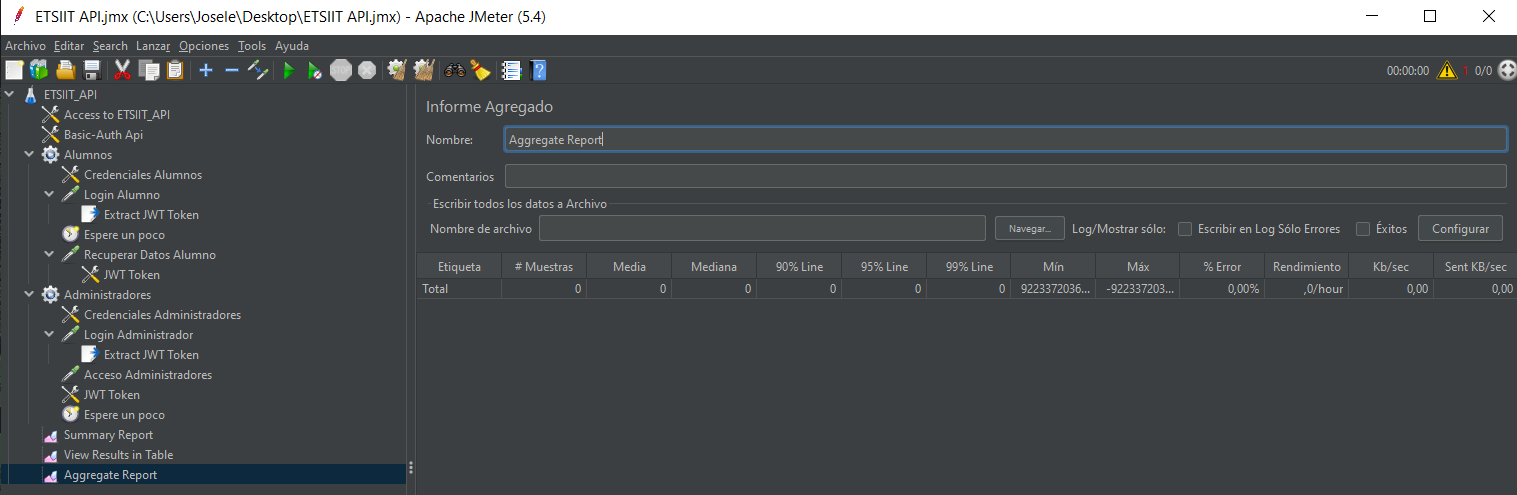


* **ETSIIT Alumnos API**
* Añadimos un reporte resumen, un informe agregado y la opción de ver los resultados en una tabla.

*ETSIIT Alumnos API/Editar/Añadir/Receptor/Reporte resumen*

*ETSIIT Alumnos API/Editar/Añadir/Receptor/Ver resultados en tabla*

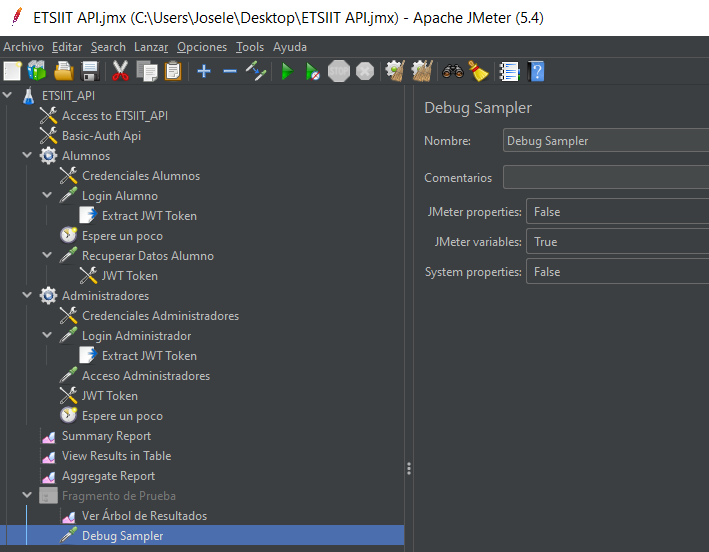
*ETSIIT Alumnos API/Editar/Añadir/Receptor/Informe agregado*



* Para realizar una prueba de la aplicación con estos parámetros:

*ETSIIT Alumnos API/Editar/Añadir/Fragmento de prueba/Fragmento de prueba*

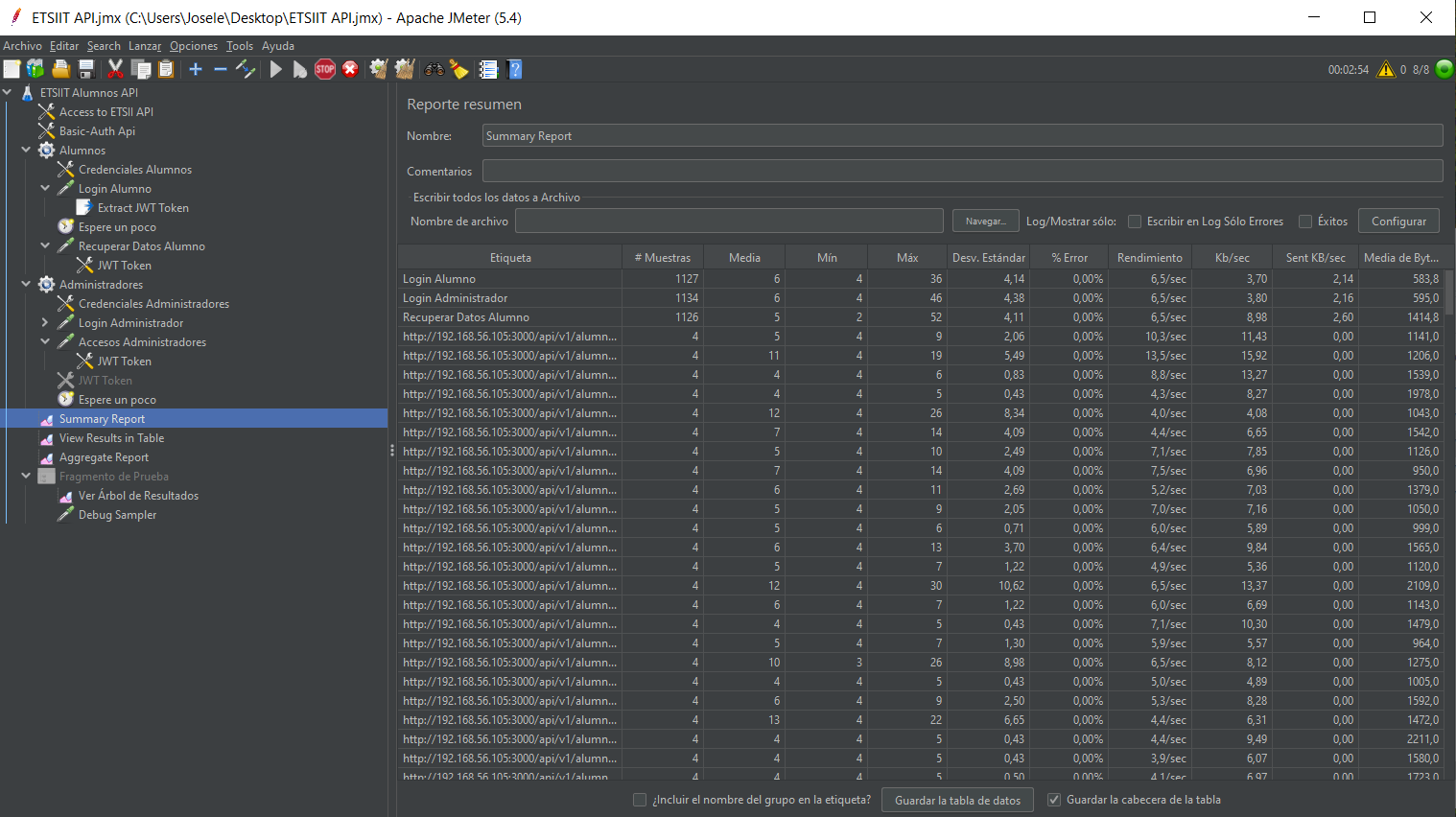
*Fragmento de prueba/Editar/Añadir/Receptor/Ver Árbol de Resultados*

*Fragmento de prueba/Editar/Añadir/Muestreador/Debug Sampler*

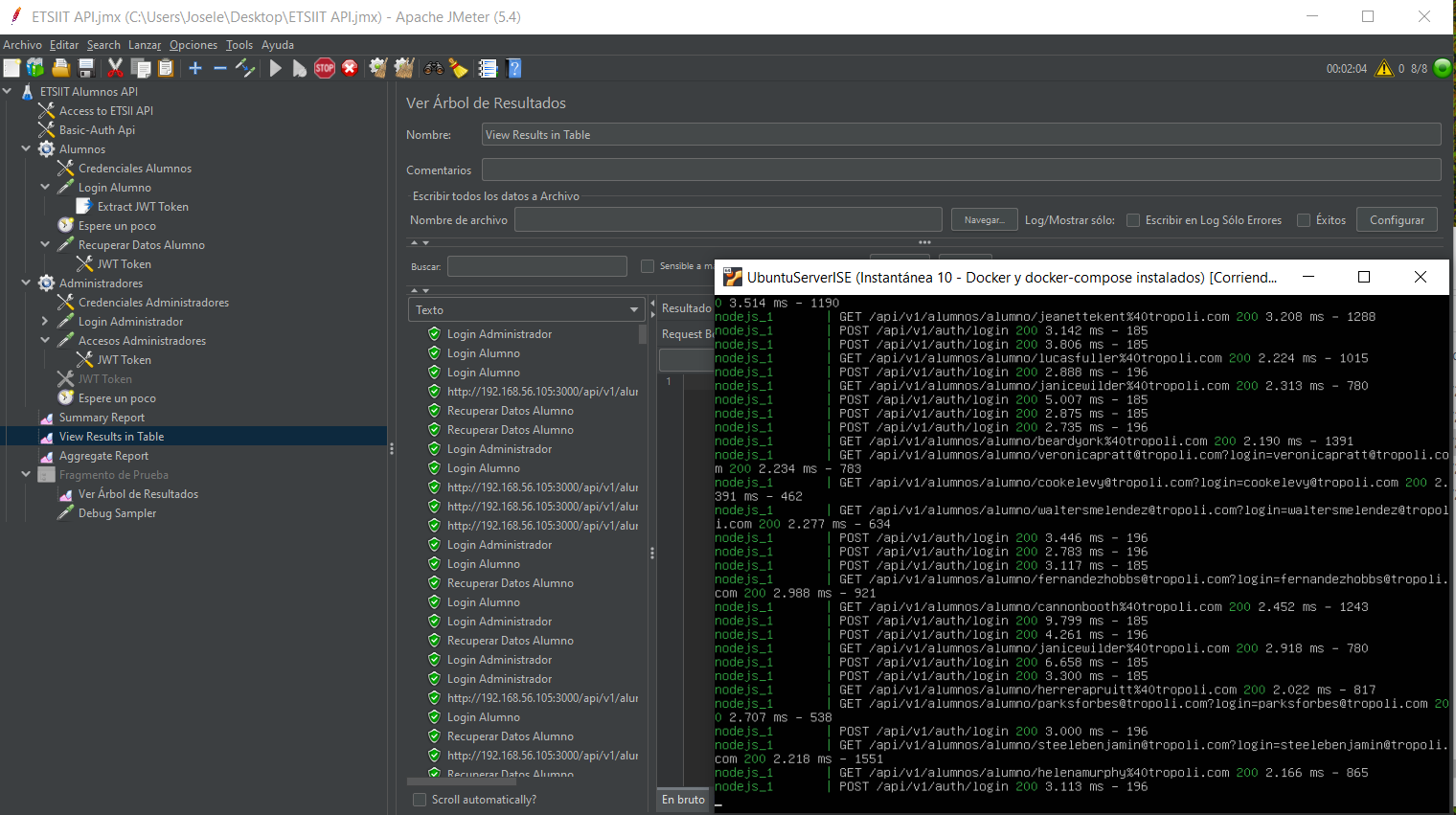
* **Ejecutar el programa**
* En Ubuntu Server levantamos los dockers:

***cd iseP4JMeter***

***docker-compose up***

* Ejecutamos JMeter (en el play/run verde)
* Paramos después de un tiempo (un minuto, por ejemplo)
* Examinamos las tablas del Summary Report:

Como podemos comprobar en la tabla, tenemos un error de 0% en cada petición.



Todo OK!!