

P4 ISE

Phoronix

Comunidad de código abierto, cualquiera puede subir un benchmark. Una vez descargado el benchmark, se debe de compilar y ejecutar. Algunos de ellos tienen dependencias que podría ser que no tuviéramos en nuestra máquina, así que se recomienda mirar sus requisitos antes de instalar. Desde línea de comandos podemos listar los benchmarks disponibles, y proceder con la descarga.

Vamos a instalar en Ubuntu (se hace de manera similar en Centos)

Para descargar e instalar la suite → `apt install phoronix-test-suite`

Phoronix divide entre tests individuales y suites. Las suites son un conjunto de benchmarks que alguien decidió unir con varias aplicaciones y darle un nombre. Normalmente agrupa las aplicaciones útiles para ciertas tareas relevantes. Nosotros ejecutaremos solamente tests individuales.



La primera vez que lo ejecutemos nos va a salir lo típico de aceptar la licencia y que si queremos compartir datos con ellos.

Comanditos útiles de `phoronix`

`phoronix-test-suite` → da una ayuda

`phornix-test-suite system-info` → informacion del equipo

`phoronix-test-suite system-sensors` → info sobre sensores

`phoronix-test-suite list-available-suites` → hace listado de las suites pero no instala nada. Se conecta con el repositorio de phoronix para listar solamente (no descarga ni instala). Básicamente nos lista las suites completas que existen dentro de `phoronix`.

Necesitamos `unzip` (`apt install unzip`) para extraer los paquetes del comando superior, ya que vienen comprimidos



Suites de tipo `system` evalúan equipos completos (benchmarks completos)

`phoronix-test-suite list-available-tests` Nombre, descripción y sobre que trata (graficos, disco, procesador, memoria...) Básicamente nos lista los tests individuales.

`phoronix-test-suite info nombrepaquete` nos da pila de info (autor, web para ver el codigo, documentacion)

Vamos a ejecutar un benchmark jeje

Podemos mediante `grep` buscar sobre que quieres hacer el benchmark, en este caso es un ejemplo y no arroja resultados :(

`phoronix-test-suite list-available-tests | grep -i Processor` y nos vamos a leer la info del elegido, en este caso ha sido (`-i` ignora entre mayúsculas y minúsculas)

`phoronix-test-suite info pts/compress-gzip`

```
josemanu@josemanuchido:~$ phoronix-test-suite info pts/compress-gzip

Phoronix Test Suite v5.2.1
Gzip Compression

Run Identifier: pts/compress-gzip-1.2.0
Profile Version: 1.2.0
Maintainer: Michael Larabel
Test Type: Processor
Software Type: Utility
License Type: Free
Test Status: Verified
Project Web-Site: http://www.gzip.org/
Estimated Run-Time: 452 Seconds
Download Size: 148.29 MB

Description: This test measures the time needed to archive/compress two copies of the Linux 4.13 kernel source tree using Gzip compression.

Test Installed: Yes
Last Run: 2020-06-01
Latest Run-Time: N/A
Times Run: 0
```

la bacanería de info que nos arroja er phoronix este iyo

Como no tiene gzip lo instala, `apt install gzip`

Ejecutamos: `phoronix-test-suite benchmark pts/compress-gzip`



Cuando ejecutas el comando anterior se procede con la instalación y ejecución posteriormente. Otra opcion es primero instalar mediante `install` posteriormente `run` Con el comando anterior se descarga, instalar y ejecuta.

Va tardar pila en mostrar los resultados, puedes tomarte un café mientras.



Cinebench es un benchmarck de CPU que evalua la capacidad para lanzar procesos en paralelo mientras compone una imagen, es muy bacano realmente

para ver los resultados → `phoronix-test-suite list-saved-results`

Crea un directorio con todos los parámetros.

A mi personalmente me está dando un error de memoria en la máquina virtual, así que lo estoy probando en mi máquina personal.

Al terminar la ejecución te dirá que si quieres subirlo a la database de phoronix para tener estadísticas mega chidas en su web.

```
josemanu@Jose-Manu:~$ phoronix-test-suite result-file-to-text resultaditos
resultaditos
Intel Core i7-7700HQ testing with a LENOVO Lenovo YOGA 720-15IKB and Intel Device 591b on Ubuntu 18.04 via the Phoronix Test Suite.

resultaditos:

Processor: Intel Core i7-7700HQ @ 2.80GHz (8 Cores), Motherboard: LENOVO Lenovo YOGA 720-15IKB, Chipset: Intel Xeon E3-1200 v6/7th, Memory: 8192MB, Disk: 100GB, Graphics: Intel Device 591b, Audio: Realtek ALC236, Network: Qualcomm Atheros QCA6174 802.11ac Wireless

OS: Ubuntu 18.04, Kernel: 4.15.0-101-generic (x86_64), Desktop: GNOME Shell 3.28.4, File-System: ext4, Screen Resolution: 1920x1080

Gzip Compression
Linux Source Tree Archiving To .tar.gz
Seconds
resultaditos .. 53.44 |=====
```

ejemplito de resultados (después de ejecutar el comando inferior)

Con el comando `phoronix-test-suite result-file-to-text nombredelresultado` lo convierte a texto este programa bien bacano realmente, que bacaneria.

Apache benchmark

Exclusivo sobre `http`, sirve para simular cargas. Realmente sirve para cargar cualquier server `http`

Viene instalado dentro de la instalacion de apache. asike ya lo tenemos

Y es muy facil ejecutar una prueba de carga de forma rapida

`ab -n 10 -c 5 http://ugr.es/` (10 peticiones) y 5 (serian los usuarios en paralelo) (por lo tanto cada usuario lanza 2 peticiones) Con esto enviamos peticiones `get` a la UGR y devuelve pila de info.

```
josemanu@josemanuchido: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>  
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/  
Licensed to The Apache Software Foundation, http://www.apache.org/  
  
Benchmarking ugr.es (be patient).....done  
  
Server Software:      nginx  
Server Hostname:      ugr.es  
Server Port:          80  
  
Document Path:        /  
Document Length:      162 bytes  
  
Concurrency Level:    5  
Time taken for tests:  0.150 seconds  
Complete requests:    10  
Failed requests:      0  
Non-2xx responses:    10  
Total transferred:    3400 bytes  
HTML transferred:     1620 bytes  
Requests per second:  66.64 [#/sec] (mean)  
Time per request:     75.035 [ms] (mean)  
Time per request:     15.007 [ms] (mean, across all concurrent requests)  
Transfer rate:        22.13 [Kbytes/sec] received  
  
Connection Times (ms)  
            min  mean[+/-sd] median   max  
Connect:    33   35   1.9    36    38  
Processing: 35   39   3.2    41    43  
Waiting:    34   38   3.2    40    41  
Total:      72   74   1.7    74    77  
  
Percentage of the requests served within a certain time (ms)  
 50%    74  
 66%    75  
 75%    75  
 80%    76  
 90%    77  
 95%    77  
 98%    77  
 99%    77  
100%    77 (longest request)  
josemanu@josemanuchido:~$
```

Nos devuelve pocos bytes, es raro, algo pasa, vamos a ver por que jeje `curl -v "web" (ugr en este caso)` y aqui vemos que lo que pasa es que es una web de

redirección (nos redirecciona a la nueva web `https` de la ugr, así que ahora probamos el comando de arriba pero con https → `ab -n 10 -c 5 https://ugr.es/`

Tenemos que entender todas las medidas. Así que vamos a entenderlas (voy a usar lo que nos arroja la versión con https (la imagen aquí abajito) 🤖)

```
josemanu@josemanuchido: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/  
Licensed to The Apache Software Foundation, http://www.apache.org/  
  
Benchmarking ugr.es (be patient).....done  
  
Server Software:      nginx  
Server Hostname:      ugr.es  
Server Port:          443  
SSL/TLS Protocol:     TLSv1.2,ECDHE-RSA-AES128-GCM-SHA256,2048,128  
  
Document Path:        /  
Document Length:      226 bytes  
  
Concurrency Level:    5  
Time taken for tests:  0.392 seconds  
Complete requests:    10  
Failed requests:      0  
Non-2xx responses:    10  
Total transferred:    4880 bytes  
HTML transferred:     2260 bytes  
Requests per second:  25.48 [#/sec] (mean)  
Time per request:     196.207 [ms] (mean)  
Time per request:     39.241 [ms] (mean, across all concurrent requests)  
Transfer rate:        12.14 [Kbytes/sec] received  
  
Connection Times (ms)  
      min      mean[+/-sd] median   max  
Connect:    124    148  18.6    153    171  
Processing:   31     41   9.2     42     54  
Waiting:     31     40   8.9     42     52  
Total:       166    189  25.2    186    225  
  
Percentage of the requests served within a certain time (ms)  
 50%    186  
 66%    187  
 75%    223  
 80%    224  
 90%    225  
 95%    225  
 98%    225  
 99%    225  
100%    225 (longest request)  
josemanu@josemanuchido:~$
```

Empezamos con el software en el que corre el servidor → nginx

Seguido del nombre del server → ugr.es

Y su puerto → 443

Protocolo de encriptación → un churraco de cosas

El path → estamos en la raíz manín.

Lo que ocupa el documento solicitado → 226 bytes

Peticiones concurrentes → en serio no lo sabes? si lo has elegido tu!!!! son 5 anda, que te veo perdido jeje

El tiempo que ha requerido cada peticion → 0,392 segundicos, na de na

Peticiones completadas → Las 10 que le dijimos que hicera

Peticiones falladas → ninguna bro

Respuestas que no han sido de código 2xx (OK) → en este ejemplo toas

Bytes transferidos en estas peticiones y los bytes referentes a HTML solamente → 4880 y 2260 respectivamente, vamos que $4880 - 2260 =$ bytes que son cualquier cosas menos HTML.

Peticiones por segundo → 25,48, quiere decir que podemos de media hacer 25,48 peticiones por segundo a la web de la ugr que nos va a poder dar servicio decente (al no ser que haya examen de CF, entonces hay 2048235023 personas conectadas y se peta)

Tiempo por petición → de media 192,207 segundicos

Tiempo por petición entre peticiones concurrentes → de media 39,241

El ratio de transferencia (KB por segundo) → 12,14

Ahora vemos una tabla con los tiempos, y varias estadísticas (media, desviación estándar, mediana, minimos y máximos) que hemos tardado en conectar, procesar y un tiempo de espera en dar servicio la ugr.

tiempo de conexion: tiempo que le lleva al proceso en abrir el socket (comunicacion tcp, handshaking)

tiempo de espera: tiempo que ha tardado en llegar el primer byte a nuestra machine (la web ha hecho sus calculos y luego nos envia cosas)

tiempo de procesamiento: tiempo que ha tardado en procesar to, vamos, en devolvernos la web enterita

tiempo total: la suma de to (no me digas 🤔)



Cuando lanzamos muchas la web detecta la IP y solo va a servir las peticiones poco a poco, en otras palabras, no se satura na ~~(la web dice: iyo que pollas pasa que hay un pesao que quiere que le enseñe mis cosas tantas veces???????, pues como creo que quiere saturarme le voy a enseñar to lo que tengo poco a poco, que tengo que darle servicio a más peñica no solo al pesao este oye.)~~

esto no simula el tiempo que pasa en user en la web, no es una carga realista así que se usa pa ver si aguanta peticiones a lo bruto, se usa por comodidad realmente

Todo esto era un tráiler, ahora comienza lo chido. Pilla palomitas y un poco de vaselina, que nos vamosssssss.

JMeter

simular carga realista? necesitamos enviar peticiones distribuidas

es una herramienta antigua pero funciona de locos

flood.io → subes una prueba de carga jmeter (admite otros) y ellos desde muchos laos del mundo simulan esa carga realista

podemos ejecutar muchos tipos de cargas → cargas sinteticas (basadas en la actividad de un usuario pero con modificaciones a nuestro antojo) y generadas aleatoriamente o cargas reales → recogidos de usuarios.

Los test de carga permiten reproducir cargas de trabajo sobre los sistemas para poder evaluar su capacidad para soportar cargas esperadas durante el funcionamiento de nuestra máquina.

Requisitos para nuestras pruebitas:

- ejecutar JMeter en nuestra máquina anfitriona
- tener java instalado (mínimo la versión 8)

Otras herramientas que hacen lo mismo y son reultraconocidas: Gatling y Locus

Prueba básica (no entregable) es por la ciencia.

Accedemos a la [web de descarga](#)

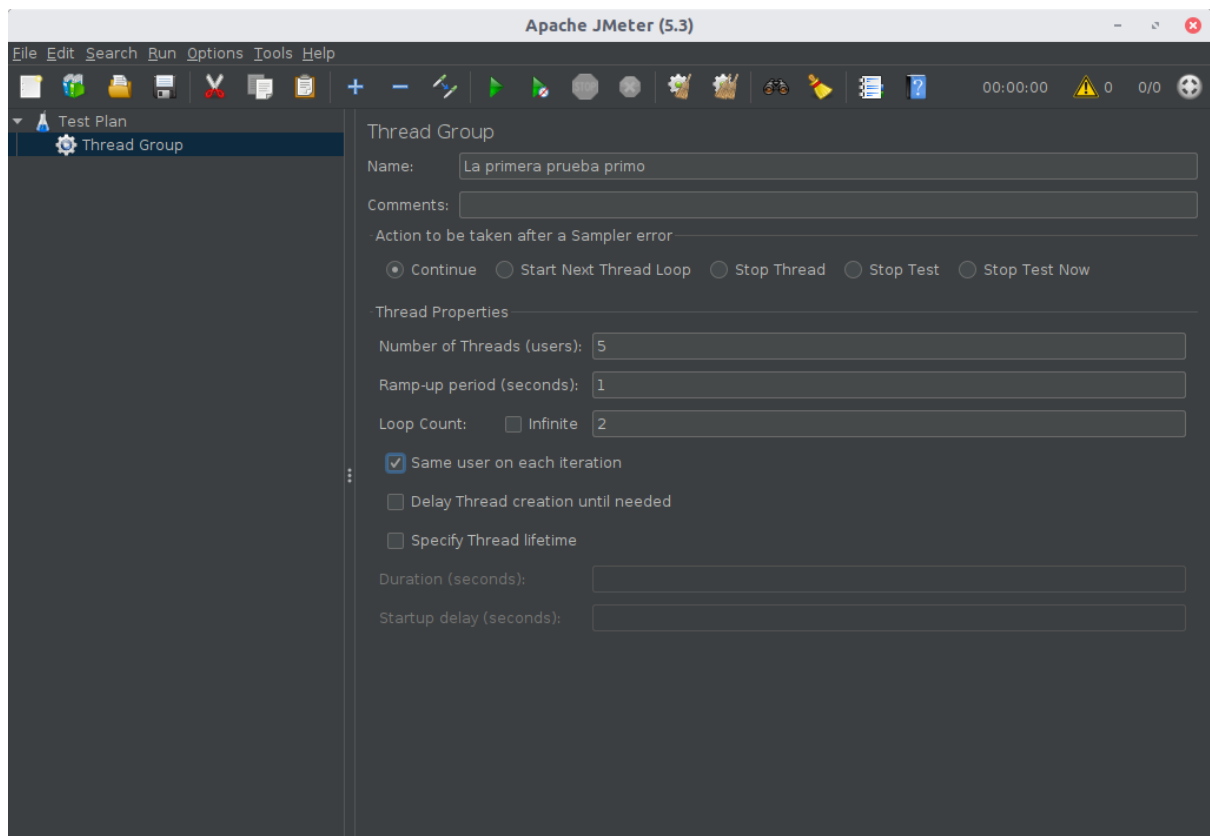
En la sección *binaries* clickamos sobre la que queramos, recomiendo la versión `.zip`. Una vez descargada la descomprimos. Arroja una carpeta, accedemos desde `cd rutadondeestelacarpeta/apache-jmeter-5.3/bin/` y lanzamos `./jmeter` y tachán, se abrió la wea. Si da algún error probablemente sea por la versión de java, chequeala con `java --version`

Una vez abierto vamos a seguir [este tutorial](#) pero voy a ir explicándolo yo poco a poco. Si prefieres no leer spoilers o hacerlo a tu rollo no sigas leyendo xd.

pero iyo, que vamos a hacer????? → vamos a crear 5 usuarios, que van a enviar peticiones a 2 páginas de la web de Jmeter. Estos van a hacer 2 peticiones y como vamos a decirles que repitan el proceso 2 veces lanzaremos la friolera de $5 \text{ usuarios} \cdot 2 \text{ peticiones/user} \cdot 2 \text{ veces} = 10$ peticiones HTTP, iyo iyo que locura compae.

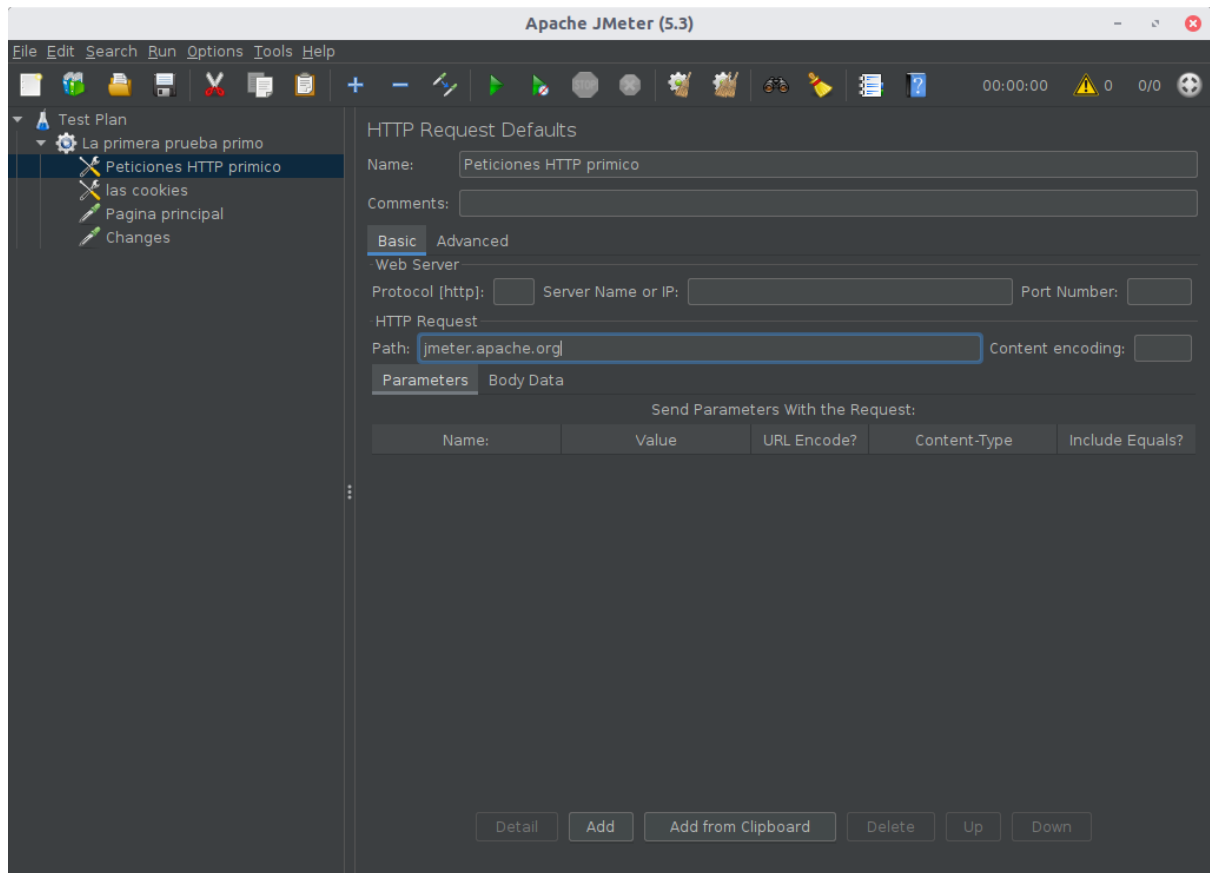
Comensemos:

1. Vamos a a crear usuarios, que necesitamos 5 loco. Para ello añadimos un *Thread Group* a nuestro plan. Aquí le diremos el número de usuarios que vamos a tener simulados y cuantas peticiones vamos a hacer. *Click derecho sobre Test Plan en la izquierda de la ventana de JMeter/add/Thread/Thread Group* se abrirá una ventanita con distintos parámetros que podemos mover. Debería de quedar finalmente algo así:

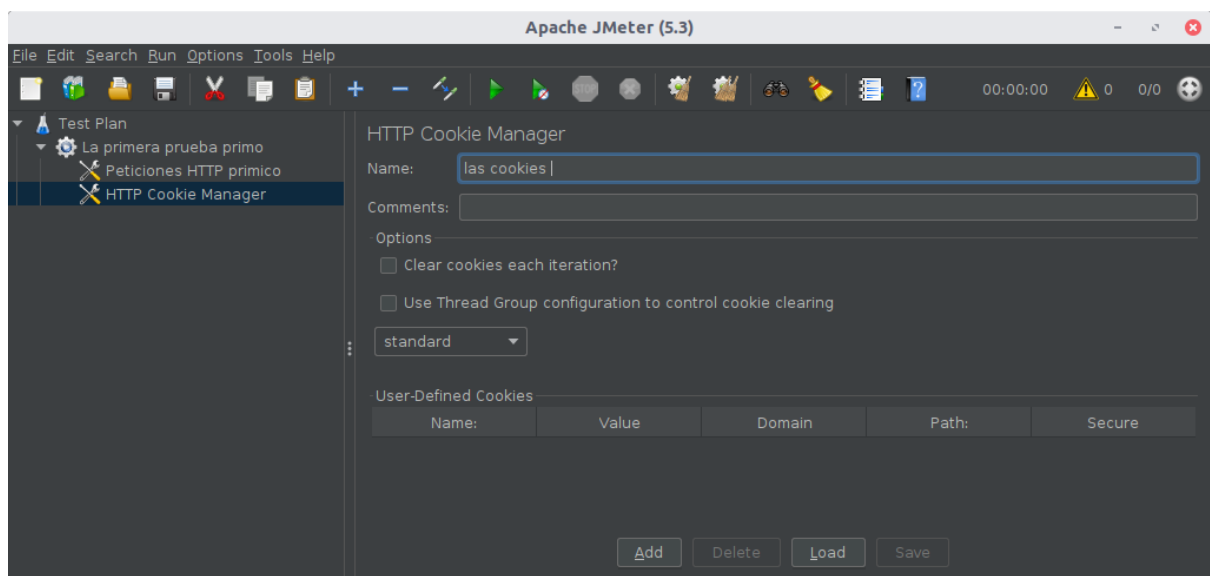


Donde 5 son los usuarios, 1 es el delay entre usuarios, es decir, en 1 segundo habrá ejecutado los 5 usuarios ($5/1=5$ usuarios/S) y 2 porque vamos a hacerlo 2 veces. La opción marcada es que en cada iteración sea el mismo usuario el que la realice.

2. Tenemos ahora que ver ahora las propiedades de las peticiones HTTP que vamos a hacer. *Click derecho sobre Thread Group/Add/Config Element/HTTP Request Defaults* y se nos abrirá una ventanita que tiene que quedar tal que así

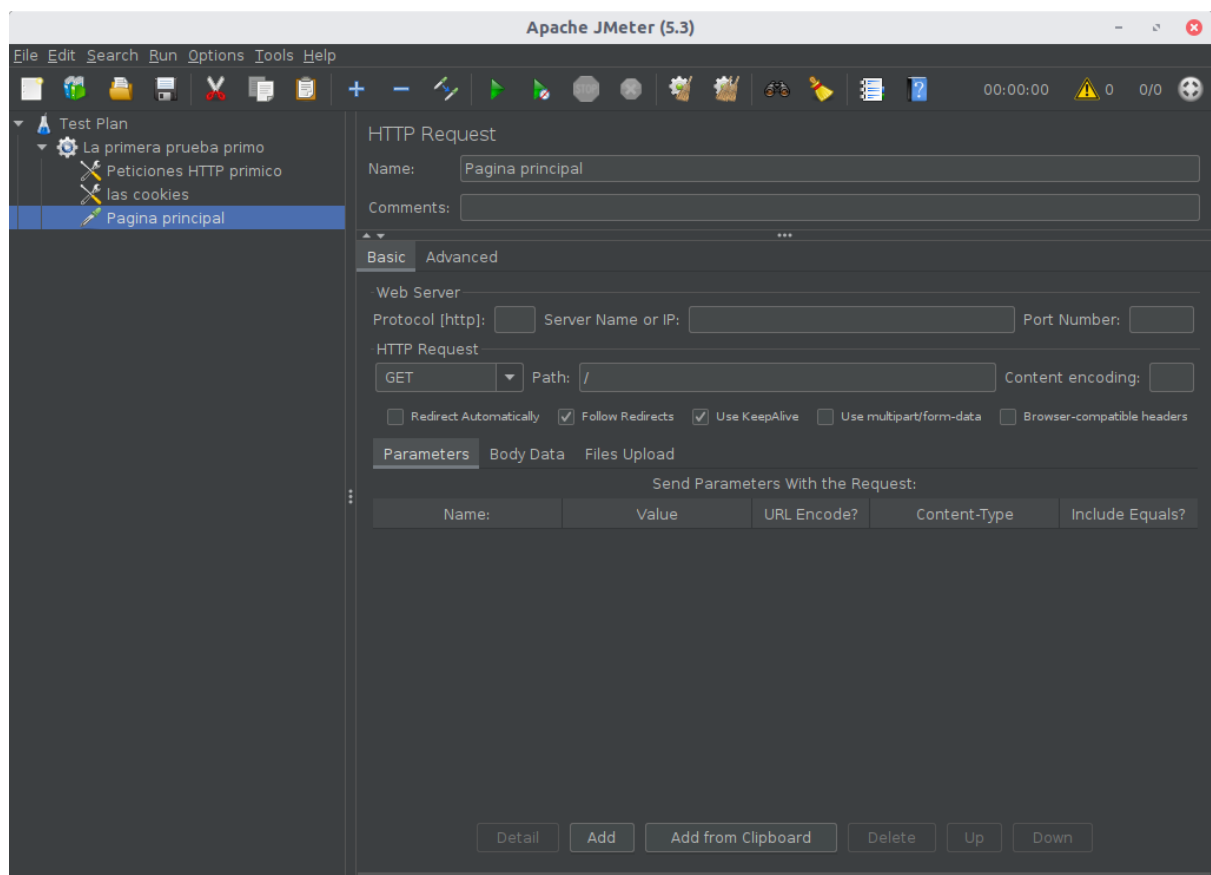


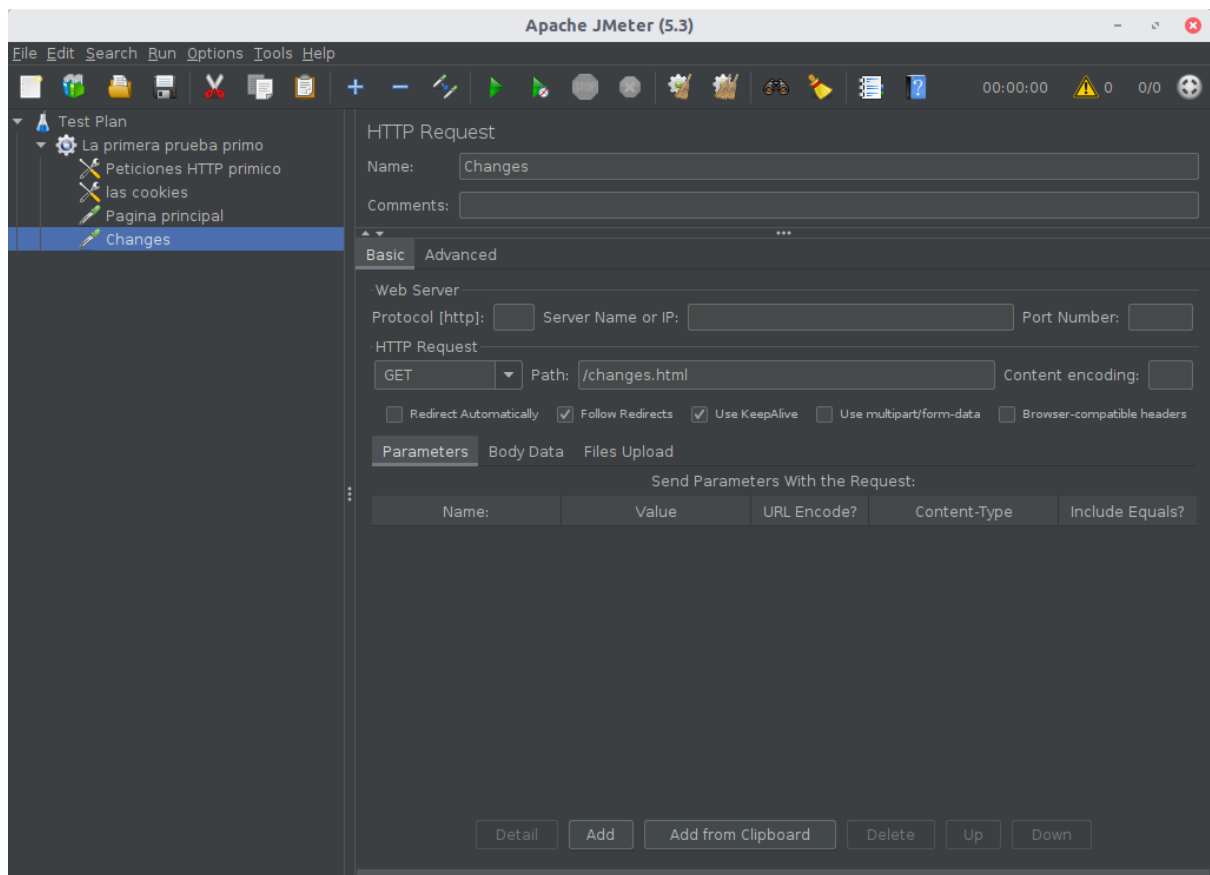
3. Vamos a añadir soporte para las cookies. Cada hebra va a tener su propia cookie que será usada por las diferentes peticiones HTTP. Para crear esta vaina *Click derecho en Thread Group/Add/Config Element/ HTTP Cookie Manager* y quedará algo tal que así



Todo por defecto, podremos marcar la opción de limpiar las cookies en cada iteración o usar la configuración que se le aplique por defecto a nuestro Thread Group, en este caso no vamos a usar ninguna, lo vamos a dejar por defecto.

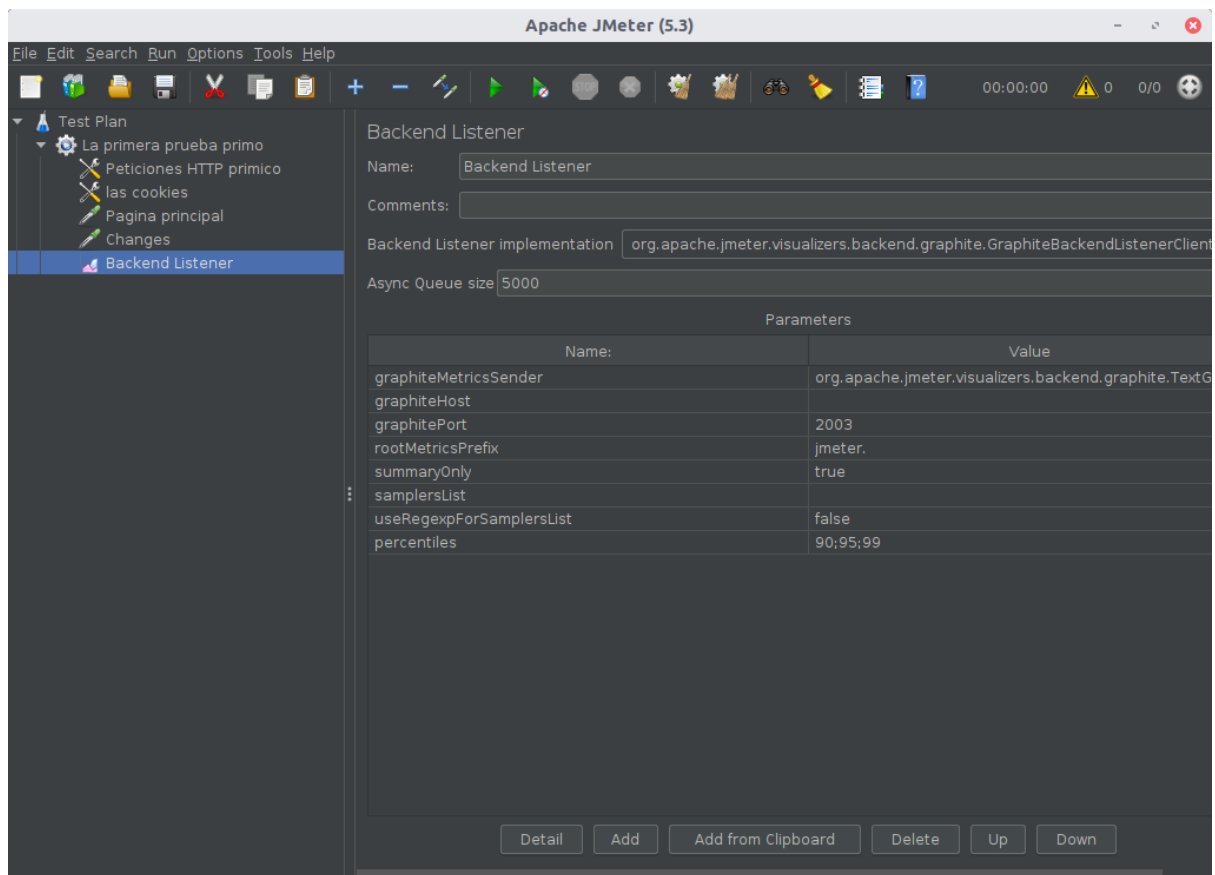
4. Vamos a añadir ahora las peticiones en cuestión, una será a la web de JMeter y la otra a la sección de cambios en las versiones de JMeter. Para hacerlo *Thread Group/Add/Sampler/HTTP Request*. Vamos a hacer eso dos veces, una por petición. Adjunto foto de como tiene que quedar cada una. Por orden:





En este caso son peticiones GET, pero podrían ser cualquiera de las existentes, depende de lo que queramos hacer.

5. Necesitamos ahora algo para poder auditar nuestros resultados. Para ello *Thread Group/Add/Listener/Backend Listener* Dejamos to por defecto.



6. Lo más probable es que esta prueba de error ya que el puerto de monitoreo por defecto (2003, que se ve en el backen listener) lo tendremos cerrado en el router y no podamos conectarnos.

Ahora si que comienza lo chido, esta parte se debe entregar

Comenzaremos instalando JMeter, si hicimos todo lo anterior lo tendremos ya en nuestra máquina, si no, tendremos que ir al paso 1. de lo anterior.

Después instalar Docker y Docker compose en **Ubuntu Server**.

Como vamos a tocar cositas de JMeter, Docker y Compose vamos a entrar en materia un poquito.

Que es docker?

Se trata de un contenedor (virtualización) que proporciona una capa adicional de abstracción y virtualización. Esto mejora la portabilidad con respecto a las máquinas virtuales. Su principal ventaja es que no necesitamos un hipervisor, ya que correremos directamente sobre nuestro hardware aquello que hay en el contenedor. No corremos un nuevo SO, si no las apps directamente.

Este proyecto sigue dos caminos, una parte más empresarial con múltiples opciones y otra con menos implementaciones pero libre.

Y docker compose qué es iyo? dímelos!!

Pues básicamente es una herramienta que nos permite definir y correr varias aplicaciones en un docker a la vez, usaremos un fichero para configurar todos los servicios de la aplicación. Así ahorramos pasos al tener un solo comando que nos va a activar toooooooooooooo lo que necesitemos. Ole ole lo caracole.

Docker compose sirve para orquestación de servicios. Sirve para proveer un servicio final el cual es un conjunto de numerosos servicios.

A la chicha

Hasta que indiquemos lo contrario estaremos en **Ubuntu Server**, recomiendo hacer `ssh` desde nuestro pc para poder copiar los comandos y no picar teclas como unos condenados.

1. Una vez instalado JMeter vamos a iniciar nuestra máquina Ubuntu Server y a instalar en ella Docker y Docker Compose. Vamo a darle. `apt-get update` después a instalar el chingo de dependencias que necesitamos, agárrate los machos que son tropecientos hermanico `apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common`
2. Ahora añadimos la key GPG oficial de Docker. (oye oye más despacio crack, para que se hace esto? nunca he tenido que hacer eso pa instalar na primico...)

La respuesta es sencilla → validaremos la integridad de lo que estamos instalando.

Bueno que sigo, vamos a añadirla de esta manera: `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -` Se puede chequear si hemos puesto la clave wena, aquí se ve como.

3. Nos traemos el repositorio con el comando:

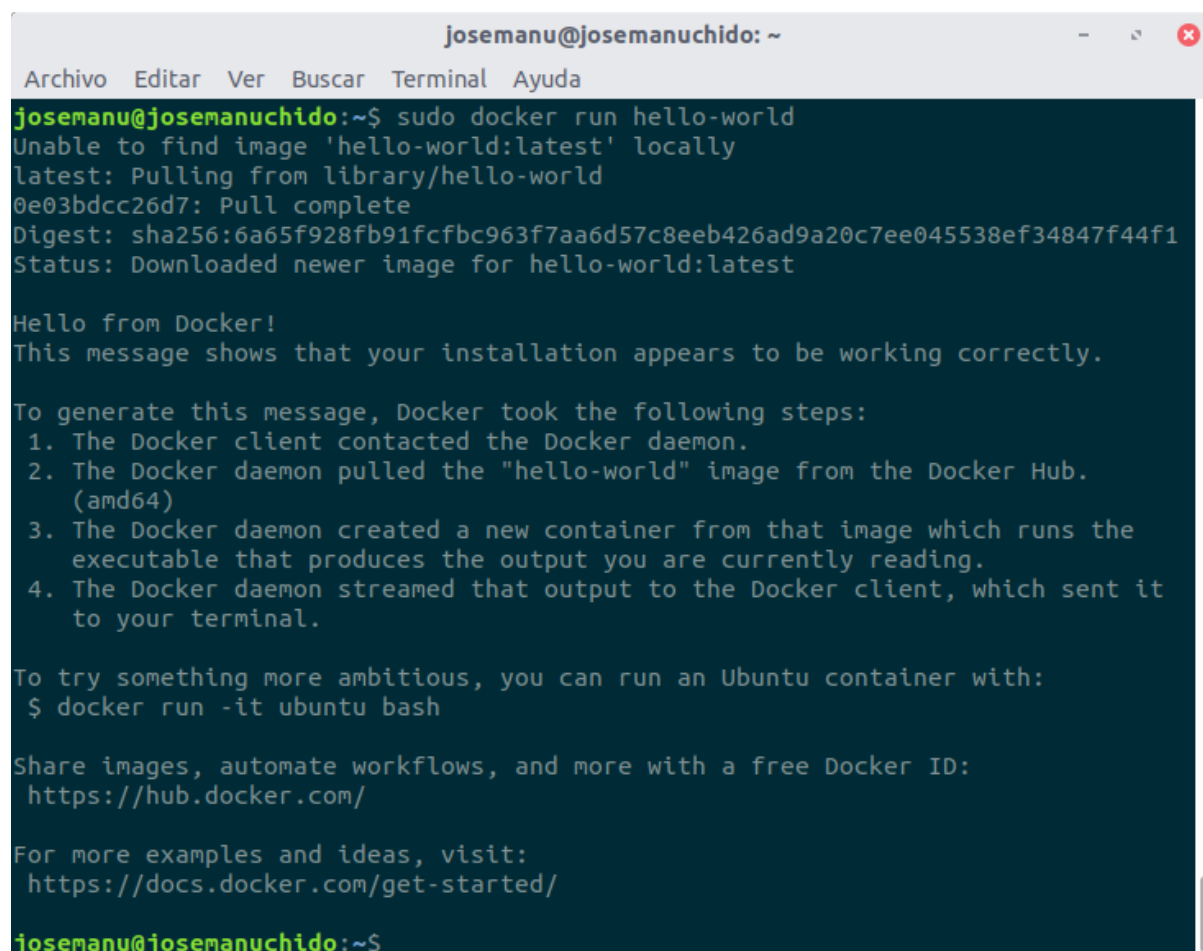
```
sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"
```

4. Estamos más que ready para instalar, así que vamos ya, que maemía que latazo la previa de este partido. `apt-get update` y `apt-get install docker-ce docker-ce-cli containerd.io`

5. Deberíamos ya tener en nuestra máquina el maravilloso Docker instalado. Ejecutaremos el comando `docker run hello-world`, es un contenedor que los desarrolladores incluyen para que podamos ver si funciona todo de manera chida.

6. A que no te ha salido? jajajaja soy reperverso, es que debes ser superuser, puedes añadir docker a nuestro sistema de superusuarios, o simplemente añadir `sudo` delante. Yo voy a optar por la segunda. Pero si te gustan los riesgos y quieres la primera, [click aquí](#) para un maravilloso tutorial de como hacerlo.

7. Ya debería de funcionar, te mostraré esto

A screenshot of a terminal window titled 'Josemanu@Josemanuchido: ~'. The terminal shows the command 'sudo docker run hello-world' being executed. The output indicates that the 'hello-world:latest' image was pulled from the Docker Hub and a container was successfully created and run. The container outputs a 'Hello from Docker!' message and a list of steps Docker took to generate the message. It also provides instructions on how to run an Ubuntu container and links to Docker's documentation and image sharing resources.

```
Josemanu@Josemanuchido: ~
Archivo Editar Ver Buscar Terminal Ayuda
josemanu@josemanuchido:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:6a65f928fb91fcfb963f7aa6d57c8eeb426ad9a20c7ee045538ef34847f44f1
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
josemanu@josemanuchido:~$ _
```

Vamos ahora a instalar Docker compose, enga.

1. Nos traemos el paquete `sudo curl -L`
`"https://github.com/docker/compose/releases/download/1.25.5/docker-`
`compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose`
2. Como vemos de la línea anterior, se va a instalar en `/usr/local/bin/docker-`
`compose` así que ya le estamos dando los permisos de ejecución necesarios
a esta cosa → `sudo chmod +x /usr/local/bin/docker-compose`
3. Pos yasta, ejecutamos `docker-compose --version` para ver la versión y probar
que todo está yendo sobre ruedas.

docker compose va por encima de docker para hacer orquestación, es decir,
arrancar una serie de imágenes (+ o - lo que dije arriba)

4. Ahora vamos a instalar la API que nos da el profe haciendo `git clone`
`https://github.com/davidPalomar-ugr/iseP4JMeter.git`

Esto simula una API REST → que chucha es esto??? Pues es la abreviatura de
Transferencia de Estado Representacional en inglés. Resumiendo, una
arquitectura para diseñar aplicaciones en red. Hoy en día todo lo que corra
sobre HTTP para obtener datos o indicar ejecución de operaciones sobre
datos es REST, cosas de los informáticos.

En la api usa mongodb que es una base de datos.

5. Bueno, una vez hecho lo anterior, vamos a darle caña a nuestra API, nos
vamos al directorio que acabamos de clonar `cd iseP4JMeter/` y una vez aquí
`docker-compose up`, ponle `sudo` que si no no te va a ir, xd.

Se van a descargar pila de cosas (imágenes base y construirá las nuevas
imágenes para la aplicación)

Si queremos parar la aplicación pues `docker-compose down`

6. Una vez haya sido creado todo y toda la pesca, va a quedar algo así, se van
a quedar los `logs` en línea de comandos para verlos IRT (in real time bro):


```
josemanu@josemanuchido: ~/IseP4JMeter
Archivo  Editar  Ver  Buscar  Terminal  Ayuda

mongodbinit_1 |      "numIndexesAfter" : 2,
mongodbinit_1 |      "ok" : 1
mongodbinit_1 |  }
mongodb_1      | 2020-06-01T16:04:26.144+0000 I NETWORK [conn9] end connection
172.18.0.4:39730 (1 connection now open)
mongodb_1      | 2020-06-01T16:04:26.148+0000 I NETWORK [listener] connection
accepted from 172.18.0.4:39732 #10 (2 connections now open)
mongodb_1      | 2020-06-01T16:04:26.149+0000 I NETWORK [conn10] received client
metadata from 172.18.0.4:39732 conn10: { driver: { name: "mongo-go-driver", v
ersion: "v1.2.1" }, os: { type: "linux", architecture: "amd64" }, platform: "go1
.12.17" }
mongodb_1      | 2020-06-01T16:04:26.150+0000 I NETWORK [listener] connection
accepted from 172.18.0.4:39734 #11 (3 connections now open)
mongodb_1      | 2020-06-01T16:04:26.151+0000 I NETWORK [conn11] received client
metadata from 172.18.0.4:39734 conn11: { driver: { name: "mongo-go-driver", v
ersion: "v1.2.1" }, os: { type: "linux", architecture: "amd64" }, platform: "go1
.12.17", application: { name: "mongoimport" } }
mongodbinit_1 | 2020-06-01T16:04:26.152+0000      connected to: mongodb://mongodb/
mongodbinit_1 | 2020-06-01T16:04:26.155+0000      20 document(s) imported successf
ully. 0 document(s) failed to import.
mongodb_1      | 2020-06-01T16:04:26.156+0000 I NETWORK [conn11] end connectio
n 172.18.0.4:39734 (2 connections now open)
mongodb_1      | 2020-06-01T16:04:26.156+0000 I NETWORK [conn10] end connectio
n 172.18.0.4:39732 (1 connection now open)
IseP4JMeter_mongodbinit_1 exited with code 0
nodejs_1       | GET / 200 10.364 ms - 843
nodejs_1       | GET /stylesheets/style.css 200 5.622 ms - 111
nodejs_1       | GET /favicon.ico 404 6.677 ms - 41
```

Si no quieres que se queden los `logs` mandalo a segundo plano con `docker-compose -d up`

7. Ahora desde el navegador de nuestro PC anfitrión podremos poner `ipUbuntuServer:3000` (que es el puerto en el que corremos esta API de docker) y saldrá esto:

ETSII Alumnos API

Descripción de la API Restful:

POST /api/v1/auth/login

Parámetros:

login:<emailUsuario>

password:<secreto>

Seguridad:

Acceso protegido con BasicAuth (etsiiApi:laApiDeLaETSIIaLache)

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 vál

Retorna:

Objeto Json con perfil de alumno

Todo está yendo sobre ruedas. En esa web tenemos la info sobre nuestra app.

Tenemos dos métodos, uno POST, que sería el `/auth/login` y unos GET que es `/alumnos/alumno`

Podemos observar que `/auth/login` permite identificar al usuario como Alumno o administrador. Este servicio está protegido por HTTP BasicAuth



HTTP BasicAuth es el método más sencillo de autenticación, no es demasiado seguro ya que se codifica en Base64 y es fácil revertir la codificación y pillar las contraseñas. No requiere cookies. Se suele utilizar para proteger servicios que sacamos a internet pública ya que en cuanto abrimos un puerto y monitorizamos vemos que sufrimos ataques (ruido de internet), en palabras de hackerman "hay máquinas intentando reclutar nuestra máquina para botnets (que son redes de ordenadores infectados que pueden ser controladas remotamente para hacer ataque DDoS (~~por favor hacker ruso que lee esto déjame tranquilo~~)). Es por ello que para filtrar este ruido se pasan unas credenciales fijadas y así te quitan a los pesaos de los hackers (los más tontitos o vagos).

Una vez identificad nos devuelve un JWT (json web token) que usaremos para utilizar el servicio de alumno.



Una aproximación family friendly de que es un JWT sería algo así como la TUI, tanto los profes como los estudiantes la tienen, pero los estudiantes tienen el rango de estudiante y no pueden abrir las clases, cosa que los profes sí. Siendo más técnico: el JWT te identifica en la web para saber que permisos tienes y qué cosas puedes y no hacer en nuestro servicio. **XQ USAMOS ESTO TAMBIÉN?** → Un api rest no guarda sesión en el servidor, no vemos que hace el user dentro de nuestra machine. Para mantenerla tenemos cookies pero la api rest no deja el uso de cookies, por eso se usa el token con la info a a mantener durante la sesión

El método GET (`/alumnos/alumno`) devuelve el registro de calificaciones del alumno. Si tu JWT dice que eres admin puedes ver el registro que te plazca, ahora bien, si eres un alumno solo podras ver tus propias calificaciones.

Nuestra API es un servicio global que está compuesto de una base de datos (Mongodb) y corre sobrenode.js



Mongodb es de esquema free y puede añadir columnas en caliente. Además tiene optimización de escritura. Mongodb mantiene la coherencia. Es una base de datos no relacional.



Normalmente el tiempo en peticiones de lectura > escritura.

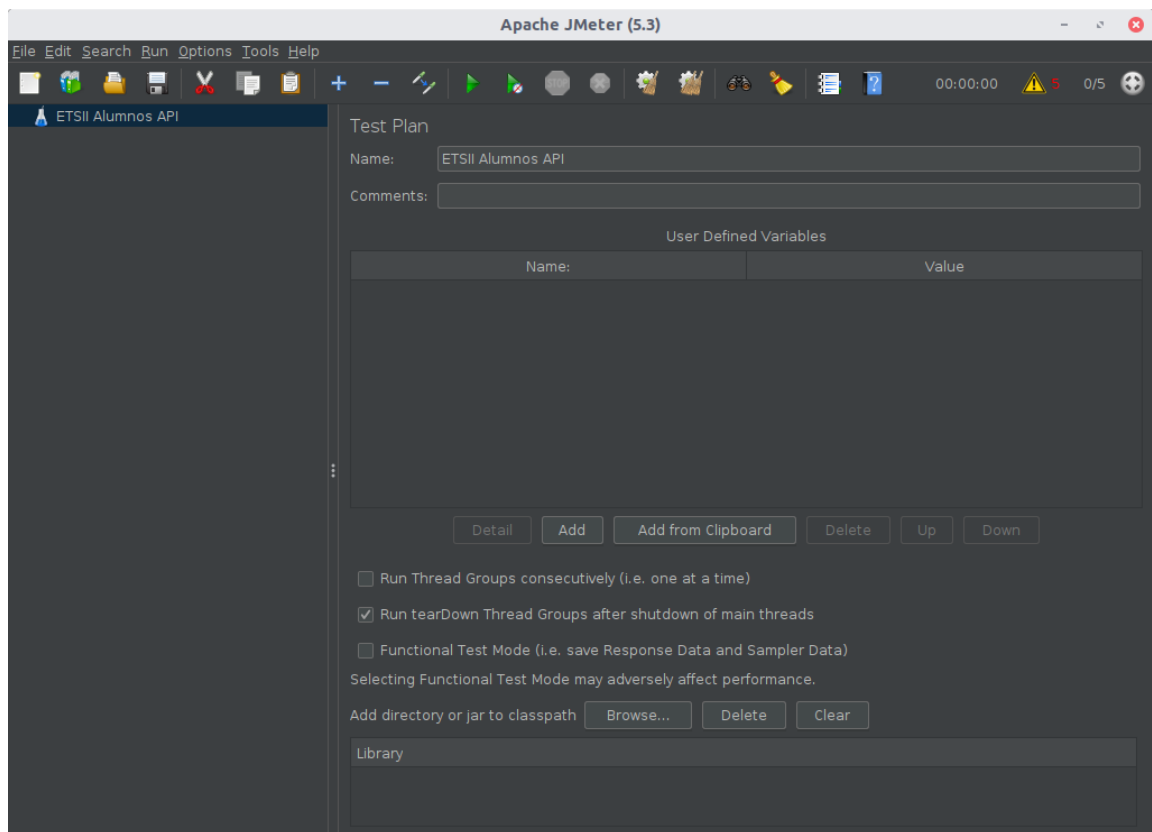
Compose levanta servicios. Vamos, que aquí indicamos los servicios que queremos correr y Docker se encarga de ir creando las imágenes y las acciones programadas sobre estos servicios. No entra en las competencias ver los códigos de Compose así que voy a pasar muchísimo de ello.

Bueno, basta ya de parafernalia, vamos a ver si la instalación, los dockers y toda esta vaina funciona bien manin. Para ello tenemos un scrip que nos ha pasado el profe.

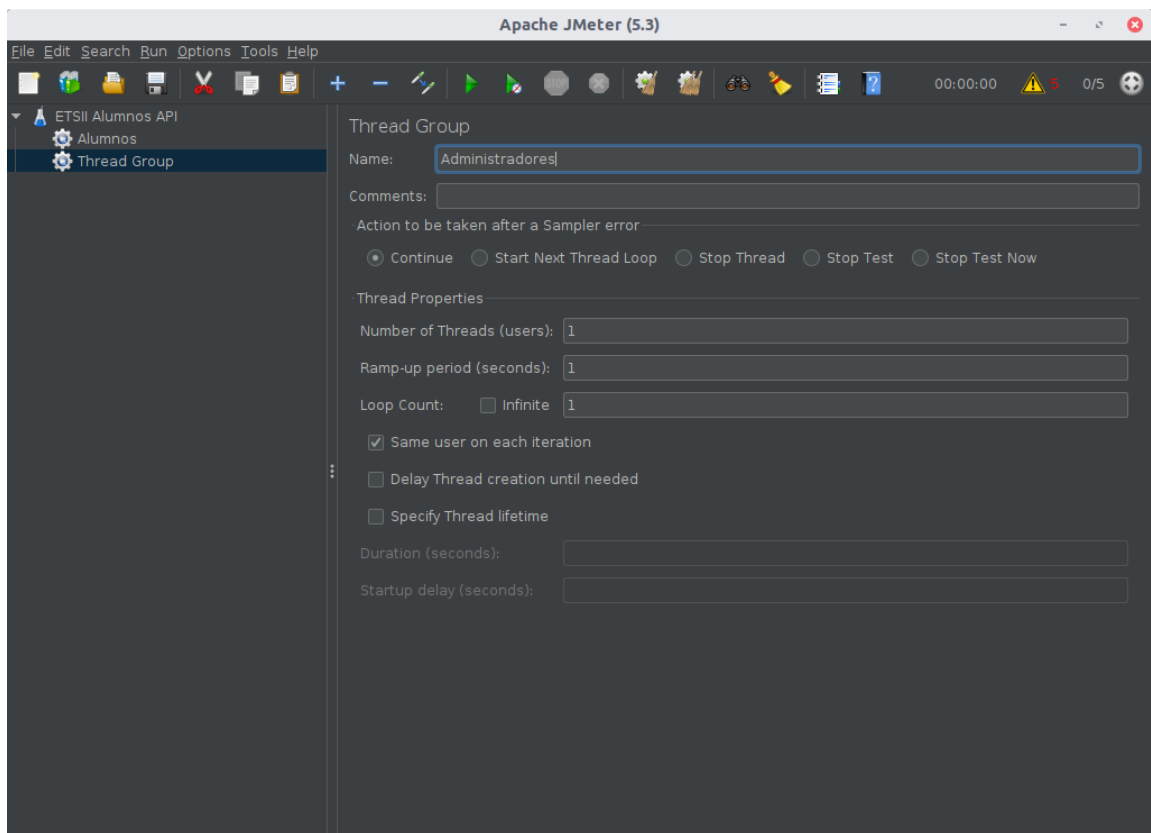
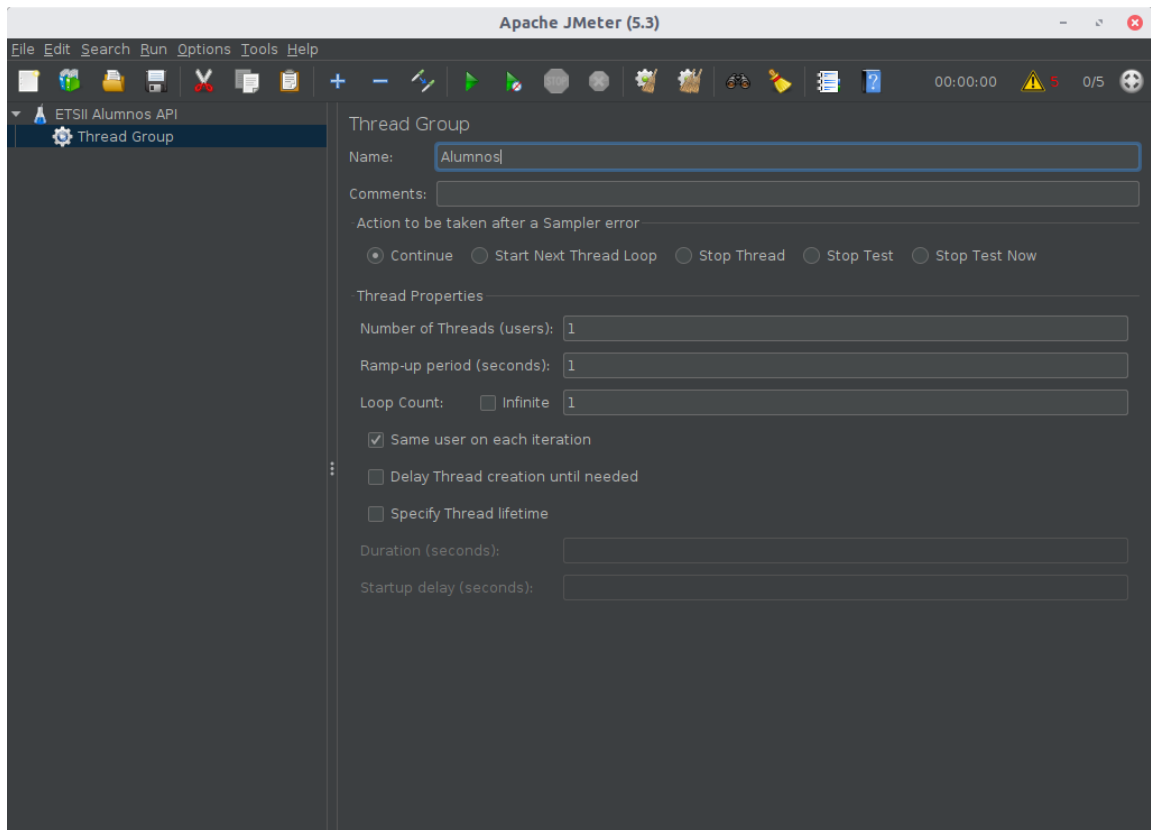
En línea de comandos lanzamos `./pruebaEntorno.sh`

Si nos devuelve lo que parece ser el perfil de una persona, nombre, apellidos, etc todo está bien. Si no, algo falló, volver a los pasitos anteriores.

Podemos hacer un cambio en el script, para que quede así:



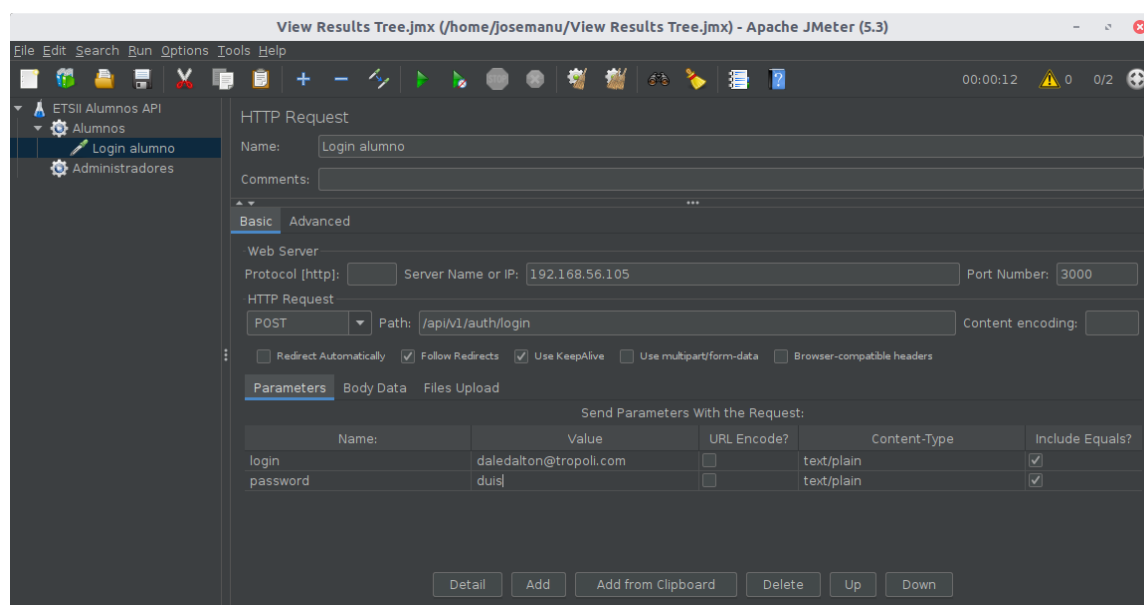
3. Va a quedar así, ahora, igual que hicimos en el ejemplo anterior vamos a crear grupos de usuarios. En este ejercicio necesitamos 2, alumnos y administradores. Así que vamos a crear los 2 de la siguiente manera: *Click derecho sobre el plan/Add/Thread Group*, Haremos esto dos veces, una para Alumnos y otra para Administradores. De la siguiente manera:



Vamos a dejar todo por defecto, la primera línea que no la expliqué antes es para ver que pasará en la hebra en caso de error. Dejamos la opción

por defecto que es *Continue* es decir, me sua la polla si hay error, yo voy palante como los de Alicante.

4. *Click derecho sobre Alumnos/Add/Sampler/HTTP Request* Ahora nombramos como Login alumno, pasamos la IP de nuestro servidor y el puerto en el que funciona. Cambiamos el método a POST, y ponemos el path que en la API nos indica que es el necesario para hacer login. A su vez buscamos dentro de la documentación pasada por el profe el login de un alumno random y se lo pasamos como parámetros. Para ello haremos click en *Add*, en mi caso quedó así:



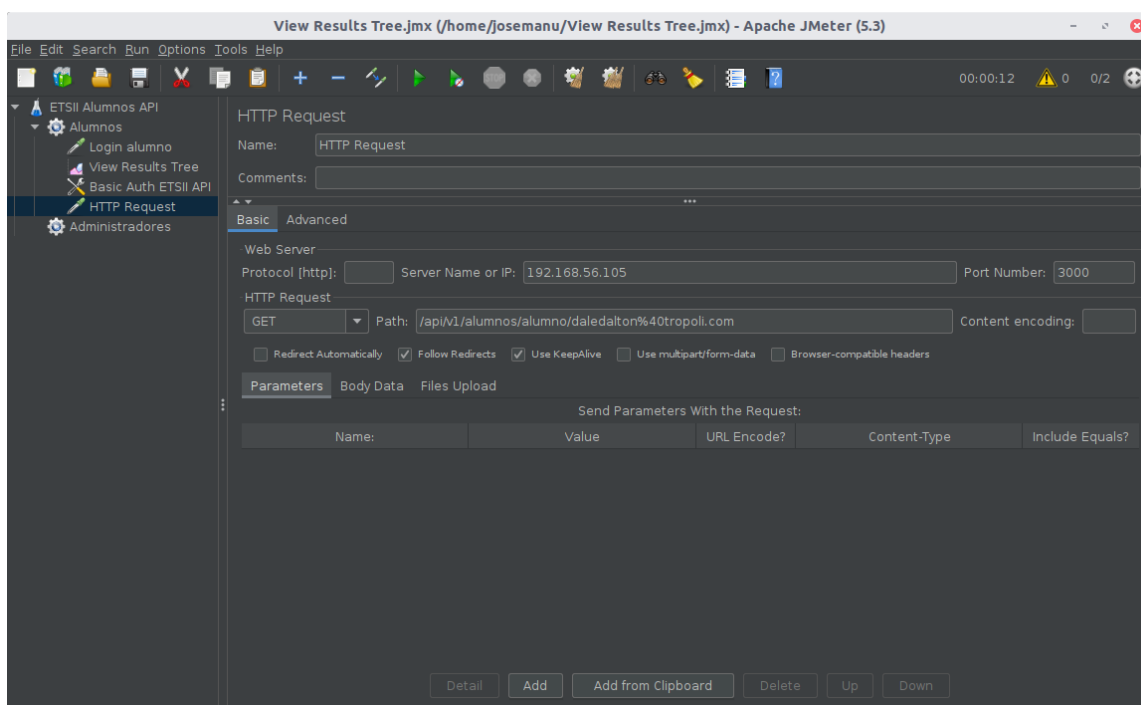
5. Vamos a comprobar si funciona. *Click derecho en alumnos/Add/Listener/View Results Tree* y ahora click al triangulito verde de arriba, para ejecutar nuestro plan. Nos mostrará abajo a la izquierda un simbolo de error, si hacemos click sobre el vemos que error nos ha dado. Nos dará un 401, Unauthorized. Claro, no le hemos pasado el BasicAuth...

6. Para añadirlo, muy fácil → *Click derecho sobre alumnos/Add/Config Element/HTTP Authorization Manager* Y le vamos a dar los siguientes valores. El nombre que queramos, yo le he dado uno bastante explícito, luego click en *Add* para pasar nuestra URL (ip del server) el Username (etsiiApi) y la pass (laApiDeLaETSII DaLache) Todo esto está en la página principal de nuestro servidor, vamos que no me lo he sacado de la chistera.

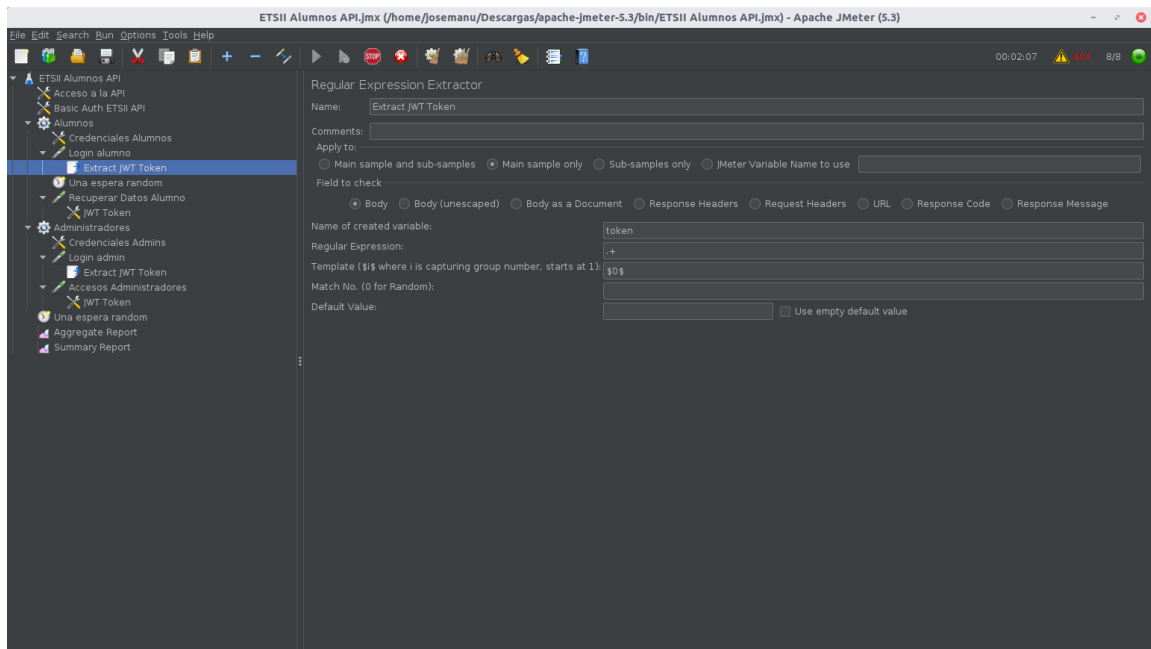
7. Si volvemos sobre nuestro *View Results Tree* y clickamos ahora el botón de *Start* no nos dará error. Oleeeeeeee

8. Si hacemos click sobre nuestro login ya OK, y vamos a la pestaña Response data de la derecha nos da el JWT, que podeis consultar en la web jwt.io.

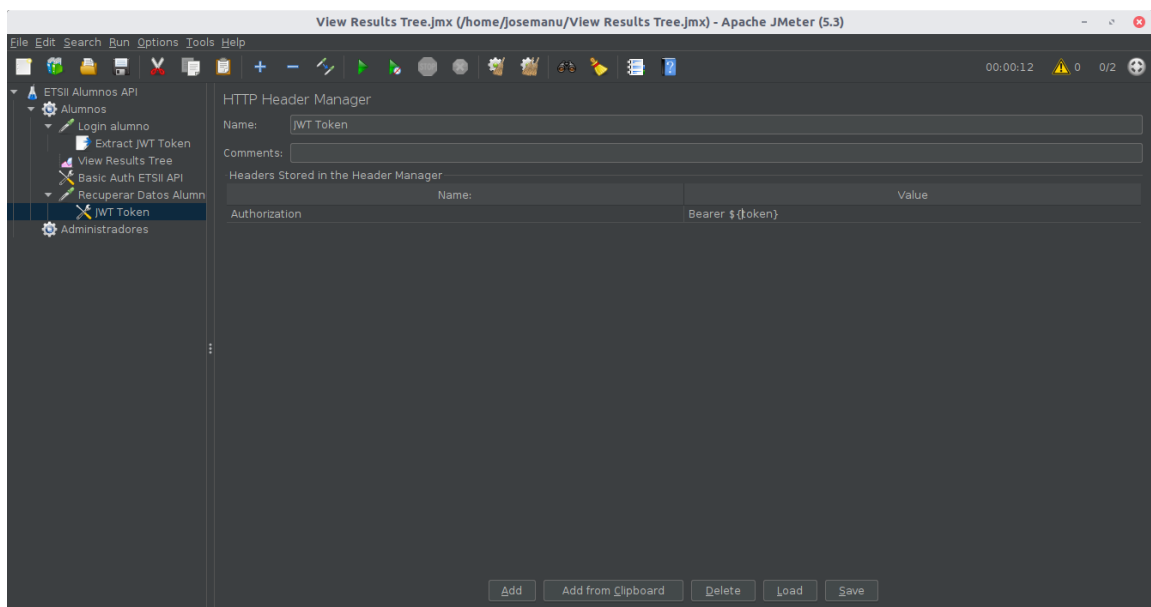
9. Hemos conseguido logearnos, ahora tenemos que conseguir el acceso a la parte en la que podemos ver nuestras calificaciones. Guay, vamos a hacer *Click derecho sobre alumnos/Add/Sampler/HTTP Request* Tenemos que meter la IP, el puerto, método GET y ahora vamos a ir a la sección de alumnos `/api/v1/alumnos/alumno`, más específicamente a la de nuestro alumno de ejemplo. Añadiremos `/emaildenuestroalumno` **NO PODEMOS OLVIDAR CAMBIAR LA @ POR %40 (COSAS DE LA CODIFICACIÓN QUE TIENE JMETER)**



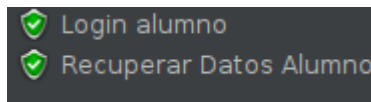
10. Muy bien, pero realmente como sabemos que somos un alumno que tiene acceso a las notas? Tenemos que extraer el JWT y pasarselo a esta última operación. Vamos a ello 😊 *Click derecho sobre login/Add/Post Processors/Regular Expression Extractor* cambiamos el nombre por algo más explícito, abajo le damos un nombre a la variable, y el resto de parámetros lo que hacen es conseguir que cojamos la clave completa



11. Ya hemos conseguido extraer el token, ahora se lo debemos pasar al método de obtener las calificaciones. Para ello *Click derecho en Recuperar Datos Alumno*(el último HTTP Request hecho)/*Add/Config Element/HTTP Header Manager*. Click en *Add* y añadimos esto. → **Bearer** es porque la cabecera sigue esta estructura y para pasar el parámetro debe de ser con **`\${nombrevariable}`** ,poniendo en medio el nombre de variable que le dimos en el paso anterior (en mi caso *token*)

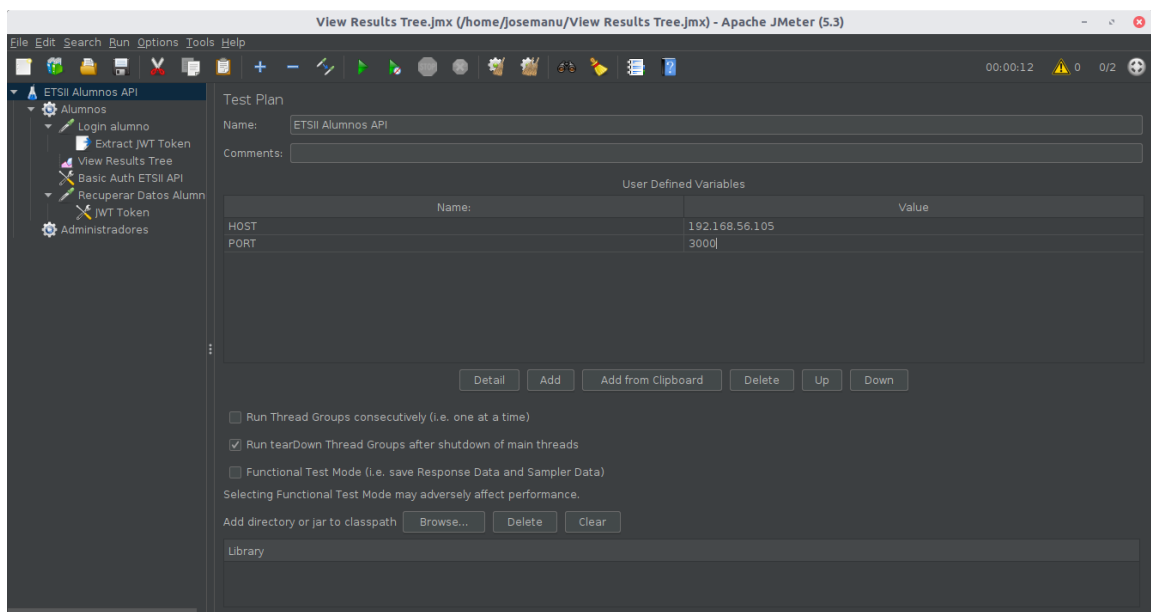


12. Vamos a *Results Tree* para de nuevo ejecutar y ver como nuestras 2 peticiones dan un resultado satisfactorio



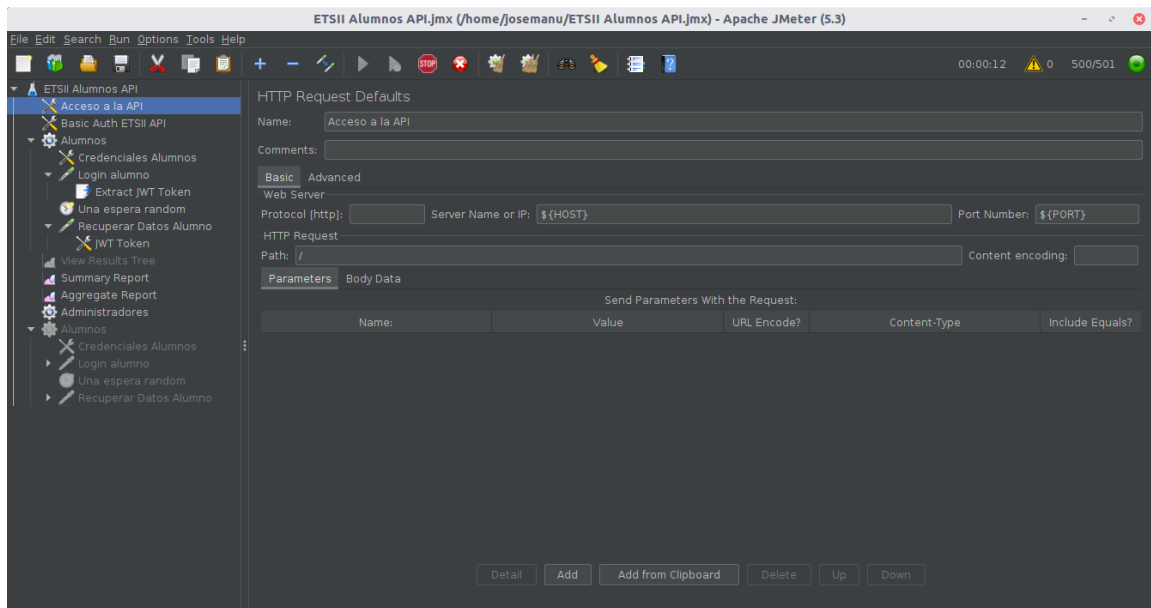
si no sale esto repasaremos los pasos anteriores, pues tendremos algún error

13. iyo, pero vamos a parametrizar, que es mejor. Nos vamos a ETSII Alumnos API y damos clic, añadimos dos variables tal que así. En HOST damos la ip del servidor que tengamos.



14. Ahora en todos los sitios que hemos puesto la ip y el puerto lo cambiamos por `${HOST}` y `${PORT}` respectivamente. Recomiendo volver a ejecutar la prueba para ver que después de cambiarlo todo sigue funcional.

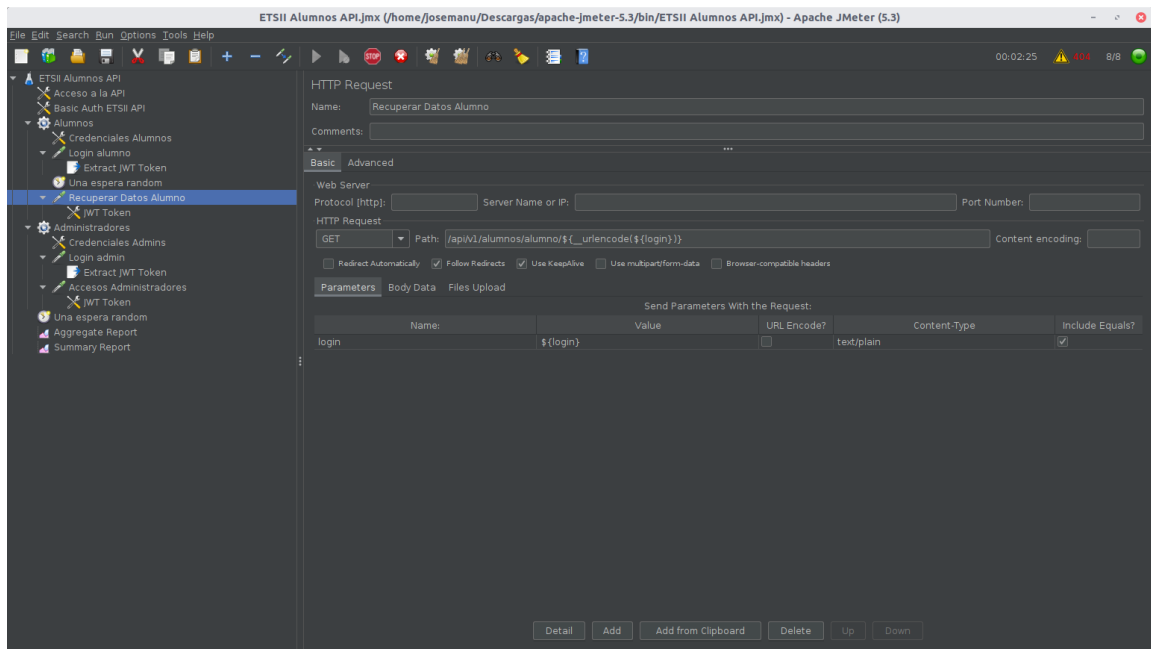
Pero mejor aún, podemos crear un HTTP Request Default (click derecho sobre Alumnos/Add/Config Element/HTTP Request Default) tal que así:



Así solo ponemos las variables ahí y del resto de sitios (HTTP Request) podemos quitarlas y no poner na

15. Bueno, ahora tenemos la simulación de un alumno entrando a nuestra API, pero el profe nos ha pasado un .csv con unos cuantos alumnos, vamos a pasarselo para que tengamos algo más realista. *Click derecho sobre alumnos/Add/Config Element/ Dats Set Config* Ahi le damos un nombre, en filename le pasamos la ruta relativa. Vamos, que nos descargemos el archivo `alumnos.csv` en nuestra máquina y lo metamos en la ruta donde tenemos jMeter (`...../jMeter/bin/alumnos.csv`) y le pasamos ahí `./alumnos.csv` . Listo, ahora el mismo documento lleva cabeceras que automáticamente le van a dar el nombre de login y password a las variables. Guay, ahora vamos a quitar nuestro usuario de pruebas (lo hemos usado en *login alumno* y *Recuperar Datos Alumno*) y cambiamos los valores por `${login}` en el caso del login y por `${password}` . *Recuperar datos alumno* le pasamos el login a un path, pues ahí también ponemos nuestra variable login nueva pero ahi la pasaremos como `${__urlencode(${login})}` También añadiremos una variable login.

Quedara esto:



Recomiendo de nuevo hacer una prueba yendo a *View Results Tree* para asegurar que sigue saliendo el bonito simbolo verde.

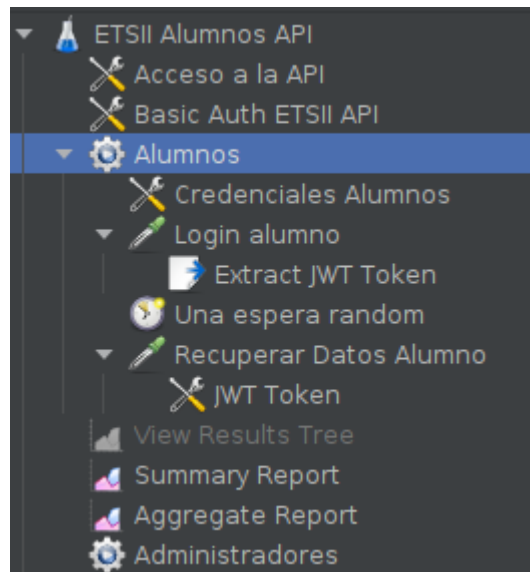


No me iba a ir sin explicar que es el `urlencode` ese turbio. Antes dije que por cosas de codificación el jmeter no tiene ni idea de que es una arroba(@) y tenemos que pasarle %40 que lo interpreta como la @. Pues con `urlencode` nos aseguramos de que ocurra esto de manera automática.

16. Vamos a tomarnos una pausa para reestructurar nuestro árbol de la izquierda, que vaya lío hay montado. También vamos a añadir una espera random (para simular que el usuario esta por ahi haciendo cosas) y dos nuevos *listener* para obtener más datos en nuestra prueba.

Para añadir la espera → *click derecho alumnos/add/timer/Gaussian Random Timer* y le damos una desviación de 500 y un Delay de 3000

Para añadir los nuevos auditores de resultados → *Click derecho sobre alumnos/add/listener/Summary Report* y despues *Aggregate Report*

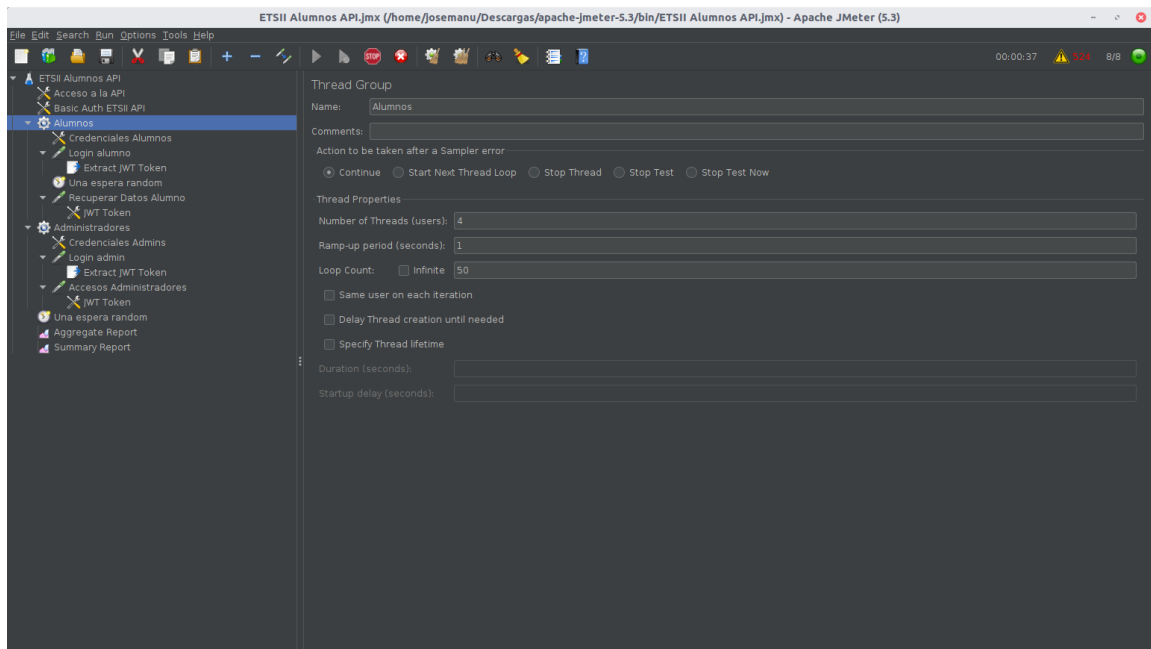


Yo lo pondría así ya que lo primero que tenemos que hacer es decirle a donde accedemos, despues, Basic Auth(para saber que no somos bots) (hay que usarlo luego también para Admin), despues todas las operaciones que tienen que ver con alumnos, primero el documento con todos los datos, despues el login, dentro del login extraemos su JWT, Sacamos nuestros datos, pero para ello le tenemos que pasar el JWT y finalmente el resultado. Esto cuando toquemos Administradores irá abajo para sacar resultados de todo. Pero de momento a los admins los tenemos olvidados.

Última prueba para ver si la vaina funciona. **AHORA VAMOS A DESACTIVAR VIEW RESULTS TREE** ya que es un test pesado y como vamos a darle caña al servidor esto nos retrasaría la vida misma. (Click derecho sobre el elemento y disable)

Antes de seguir, vamos a hacer una prueba real de que todo funciona con los alumnos de manera correcta. Luego ya nos pasaremos a los admins.

- Nos ponemos sobre *Alumnos*, clickamos y ponemos 4 hebras, 1 el tiempo en el que vamos a tener ya todas las hebras volando y en las veces que lo vamos a repetir, 50



Abre una pestaña en tu navegador con `http://ipserver:3000/status` para ver gráficas ultrachidas mientras petamos el server.

Dale click a la flecha verde y a esperaraaaaaaaaaaaaaaaaaaaaaaar

Si tenemos un terminal con los `logs` como dijimos arriba vas a ver como se mueve eso sin parar.

Listo, si has llegado aquí todo ha ido de putisimos locos

Vamos ahora a configurar los admins.

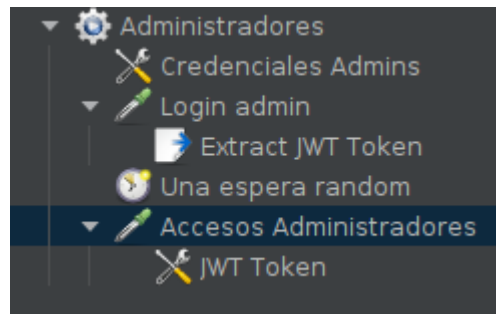
life hack 1 link mega megasecreto nos va a ahorrar tiempo

Click derecho sobre el administradores que creamos antes, remove.

Click derecho sobre alumnos, copy, click derecho sobre ETSII Alumnos API paste

Ahora tenemos 2 alumnos, el segundo vamos a modificarlo para que quede acorde a los admins.

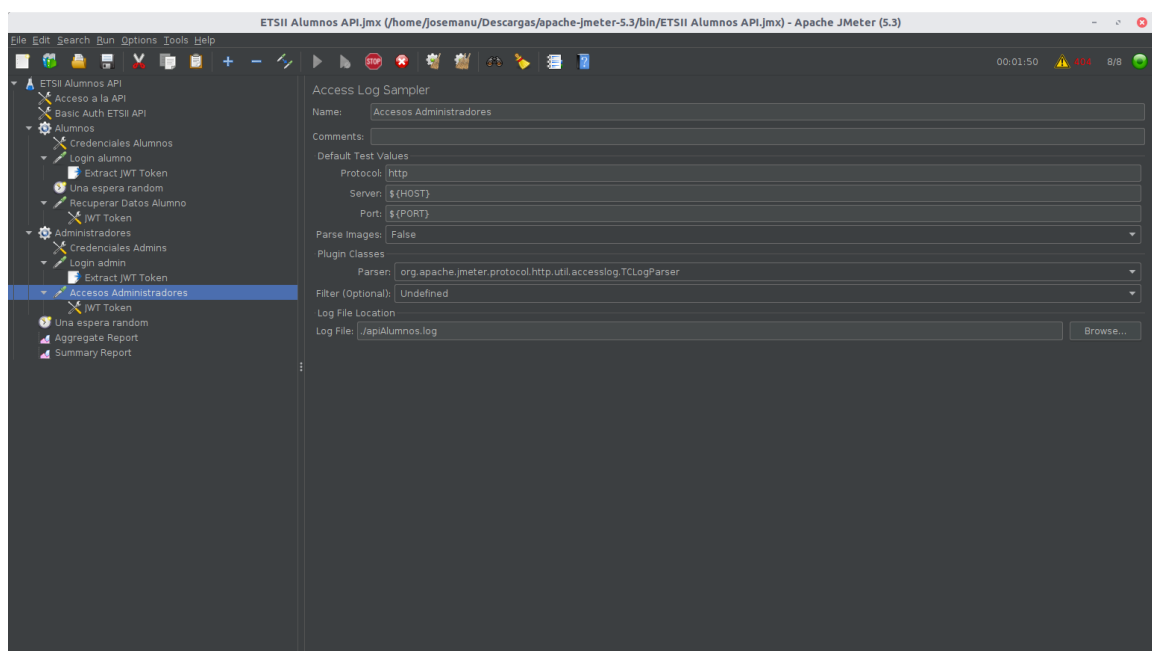
Cambiamos los nombres de todo al estilo del git del profe, algo así:



Aparte de cambiar los nombres, en credenciales, haremos lo mismo que hicimos en su homónimo Alumnos pero con el archivo `administradores.csv`

Otra cosa a cambiar son los accesos, ahora no vamos a consultar nuestras notas (~~más que nada porque no tenemos xd, somos admins~~) por un archivo `.log` que nos ha pasado el profe simulando lo que haría un admin en un día normalico.

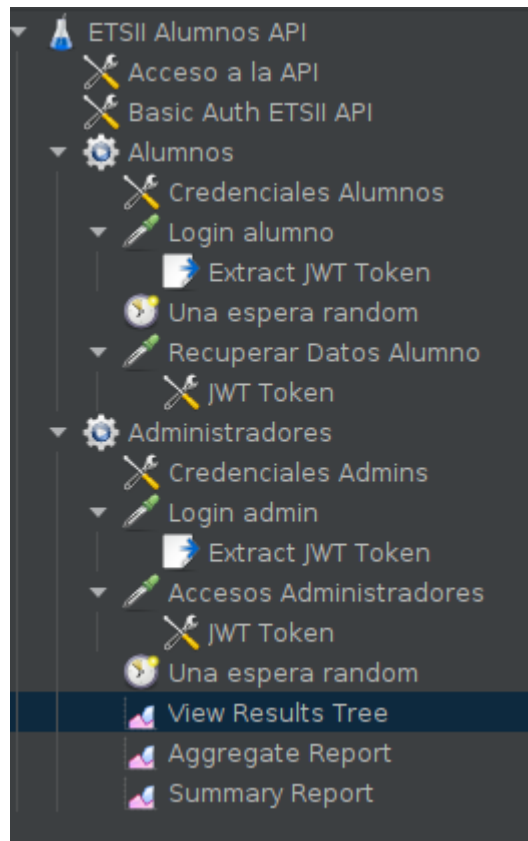
Para ello, *Click derecho sobre administradores/Add/Sampler Access Log Sampler* lo vamos a modificar para que quede así:



Ahí hemos pasado el archivo `.log` que nos ha pasado el profe y previamente hemos metido en `...../jMeter/bin`

Le metemos ahora el token (JWT Token) a esta operación arrastrando el elemento sobre el nombre de esta operación a la izquierda y eliminamos la operación que antes tenía el Token.

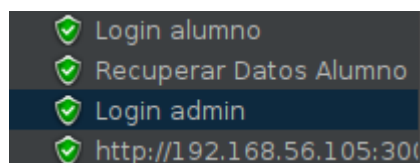
Reestructuramos para que quede así



Vamos a hacer una pruebita simple como antes

Click sobre Alumnos y Admins y ponemos todo a 1

Activamos de nuevo el view results tree y click a star. Debe de darnos ahora 4 simbolitos verdes



Funciona, así que vamos a darle una prueba tocha ya o klk, ponemos tanto en alumnos como en admins 4, 1, 50 (como antes)

Tenemos que desctivar View Results Tree

Ahora vamos a crear un archivo de reultados desde el terminal.

Nos vamos a la carpeta donde tenemos jmeter

```
./jmeter -n -t archivo.jmx(./archivo.jmx si lo tienes en la misma carpeta que jmeter) -l results.jtl -JHOST=ipservidor -JPORT=puerto
```

- `-n` para trabajar nongui (sin interfaz gráfica)

- `-t` para indicar que lo siguiente que viene es un test file
- `-l` para indicar que lo siguiente es el archivo donde guardar los resultados

Aquí podemos ver muchas más opciones para la lista de comandos como vemos si es correcto lo arrojado? lo vemos en jmeter. → añadimos un `listener, summary report` y abrimos el resultado, a ver si es valido y ahí se verán los resultados.

Y ESTO ES LO QUE SE ENTREGA → `.zip` con `archivo.jmx` y con `results.jtl`

Algunas cosas interesante en JMeter (ya no hay que entregar na)

En jmeter hay una parte que se usa para hacer un registro de los movimientos de los usuarios dentro de la web (ver donde se mete y to la pesca). Consiste en instalar un proxy entre la web a acceder y nuestro navegador, de esta manera en un archivo se registra todos los movimientos del usuario. vamos, su navegación por la web.

Para hacer esto partimos de un proyecto vacío en Jmeter, *Click derecho/add/non-test-element/ HTTP Test Script Recorder*, le ponemos un nombre

Vamos a usar firefox para esta manualidad, supongo que se podrá en otros navegadores pero el profesor lo explicó en firefox y es el que voy a usar

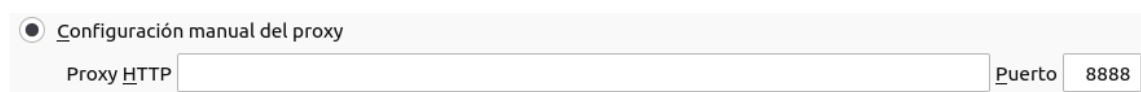
Accedemos a la web de apache (ya que no es https)

Queremos que nuestro navegador acceda a través de jmeter y no directamente desde mi pc hasta la web pasando por el router (en otras palabras, que jmeter sea un man-in-the-middle), entonces en jmeter → *click derecho sobre el test plan/add/thread* le ponemos un nombre y de nuevo, *click derecho sobre el thread group/add/logic controller/recording controller*

Click sobre lo primero que creamos (HTTP Test Script Recorder), click en Target controller y en el desplegable que sale seleccionamos lo que acabamos de crear (Recordin Controller)

Damos click en *start* y parece un aviso de que jmeter ha generado un certificado. Ahora mismo no se usa. Nos da el puerto en el que se lanza la

confi del proxy de jmeter (8888) y dentro de firefox → configuración
→ general → configuración de red → configuración manual de proxy → y le
ponemos el puerto 8888 en Proxy HTTP.

A screenshot of the 'Configuración manual del proxy' (Manual Proxy Configuration) window in Firefox. The 'Configuración manual del proxy' radio button is selected. Below it, the 'Proxy HTTP' label is followed by an empty text input field. To the right, the 'Puerto' (Port) label is followed by a text input field containing the number '8888'.

Con esto no pedimos info directamente a la internet, si no a un proxy y desde ese proxy (que en nuestro ejemplo es nuestro pc) ya salimos a la red global.

HASTA AQUI NO HE CONSEGUIDO HACER FUNCIONAR ESTO ÚLTIMO, DE AQUÍ PARA ABAJO SON APUNTES DE CLASE POR SI CAE EN EL EXAMEN, YA QUE NO HE CONSEGUIDO QUE FUNCIONE :(

Supuestamente se deben de generar bajo el script recorder las cabeceras propias del la web

En todas las peticiones se crea la misma cabecera, con esto podemos recrear unos logs de un usuario en nuestra web pero realmente nos capta bastante basurilla

Esto solo funciona en http, que pasa con los https???????????

Con https, tenemos más seguridad, nos llega la info cifrada. Si accedemos con el proxy http se va a detectar como un posible atacante man in the middle

Ahora vamos a intalar el certificado propio de jmeter del que se habló antes en preferencias de firefox, certificados, import, y buscamos en directorio bin de jmeter el certificado

activamos lo que nos sale y ale

ahora recargamos la web https y ya funciona bb.

Caso especial: https corre sobre tls, existen varios estandar del tls, hay algunas webs que no funciona esto que hemos hecho por incompatibilidad con el protocolo. La web de la ugr por ejemplo

nos vamos pa la web de la ugr a ver que sale,

Dice que la conexion ha fallado. Que pasa? que la ugr ha configurado la version 1.4 del protocolo TLS que autentifica no solo al server y autentifica al cliente, es decir, dice yo soy la ugr, y yo soy firefox (esto gracias a un

certificado de firefox) pero a ugr le llega el certificado jmeter, que no es aceptado por la ugr (en nuestro ejemplo super simple sería: nuestra petición llega a la UGR, ugr dice: hola, soy ugr, tu eres jmeter, mmmmmmmmm no te conozco, no te voy a abrir la puerta, sorry). Pa arreglarlo ponemos `about:config` en la barra de direcciones, aceptamos, buscamos `tls, security.tls.version.max 4`

Cambiamos el 4 por 3

Y ya debería de funcionar.

Ole.