EJERCICIO OBLIGATORIO: JMETER

https://github.com/davidPalomar-ugr/iseP4JMeter
En el README.md sale todo lo que tenemos que hacer. Recomienda hacerlo en Ubuntu que da menos problemas que Rocky. Hay puntos que no se dicen en el README que se dan por sabidos (arrancar los servicios y eso).

Lo que vamos a hacer es una prueba de carga que se ejecuta sobre dos contenedores (uno para la BD y otro para la aplicación en sí) atacando una API Restftul (Resource State Transfer).

1. Instalar DockerEngine y DockerCompose en MV Ubuntu

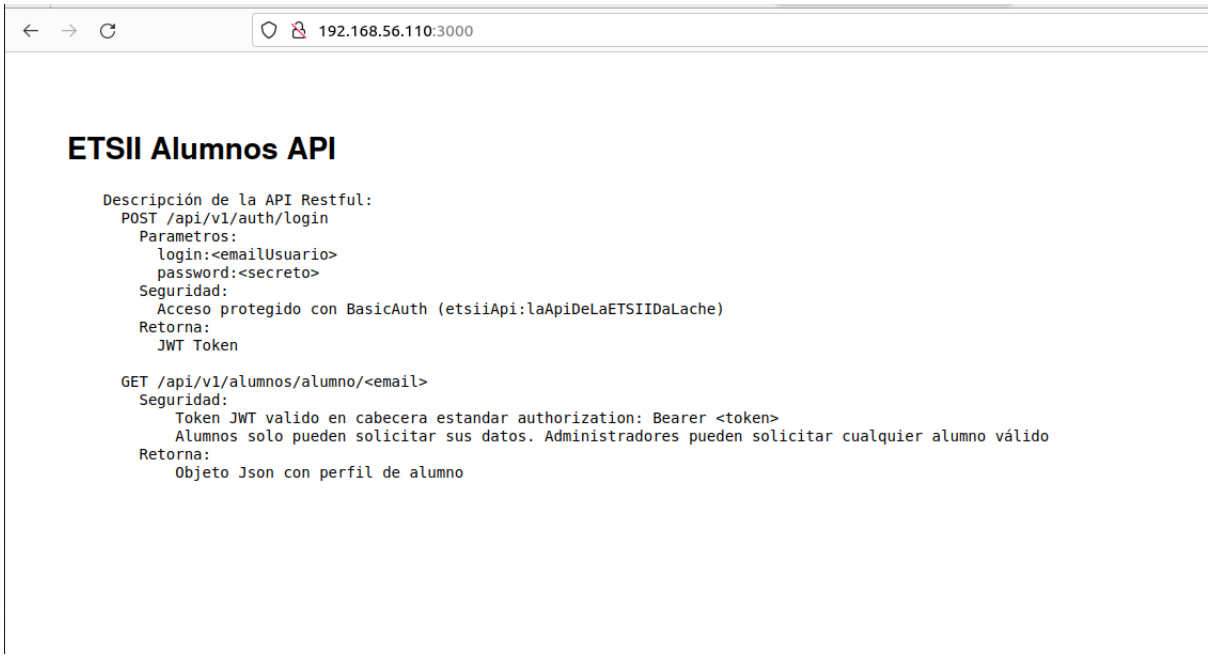
2. En la MV clonamos el repositorio del ejercicio

3. Una vez instalado:
cd iseP4JMeter/ : meternos en el repositorio
docker compose up : arranca el servicio y lo tenemos que dejar ahí esperando (ya lo siguiente lo hacemos desde otra terminal, nos conectamos por ssh).

Si queremos parar el servicio: docker compose down

4. Desde otra terminal de la misma MV, o conectándonos con ssh:
cd iseP4JMeter/
./pruebaEntorno.sh : En la terminal de la MV Ubuntu veremos los cambios: POST y GET.

5. Habilitamos el puerto 3000:
sudo ufw allow 3000
y desde un navegador nos metemos en la página http://IP_UBUNTU:3000 y vemos la página:



Esta página nos dice que haciendo: POST /api/v1/auth/login podemos hacer el login siguiendo las instrucciones de abajo (login y password). **Lo haremos más adelante en el test de jMeter**.

Tras hacer el login nos va a devolver un token JWT que tienen forma de una cadena de caracteres, esto nos sirve para identificarnos. Este token es el que indicaremos a la hora de hacer la petición GET.
Si nos hemos autenticado como Alumno, solo podremos solicitar nuestros datos, y si ha sido como Admin, podremos acceder a los datos de cualquier alumno.

En el apartado Seguridad, nos asigna un protocolo BasicAuth que es muy antiguo y ya no se utiliza casi para autenticación, aunque sí lo usa por ejemplo Zabbix para eliminar ruido (esto es, peticiones que se hacen para dar por culo básicamente y que el servidor tenga que estar rechazándolas).

6. En la MV Ubuntu:
nano -w ./pruebaEntorno.sh

Podemos ver en el archivo:

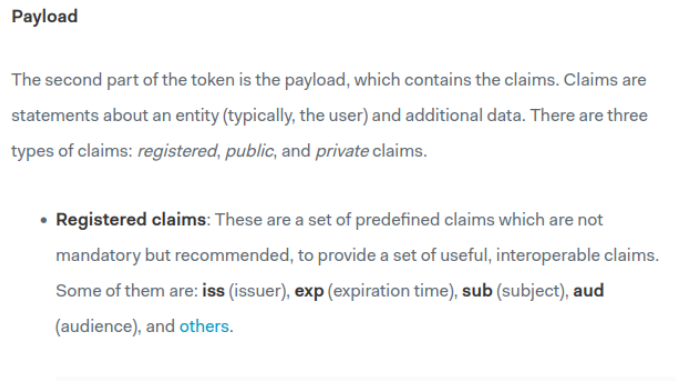
- SERVER localhost : aquí podríamos poner otra IP de otro servidor donde corre el contenedor de Docker.
- TOKEN -> hacemos llamada a curl : sirve para hacer llamadas http. Los parámetros son:
  -u : usuario de proxy <user:password>
  -d : datos que queremos enviar
  -H : es uno de los pocos formatos estandarizados.
  -X POST://$SERVER:3000/api/vi/auth/login : -X es para request. Hacemos una petición a una página web (endpoint indicado).

De esta manera, el token es el resultado de hacer el POST y loggearnos.

7. Nos salimos y lo ejecutamos:
./pruebaEntorno.sh : nos devuelve el resultado del GET. Para ver mejor el contenido nos vamos a la página de jsonformatter.org y copiamos lo que nos ha devuelto.

8. En el archivo de pruebaEntorno.sh añadimos que imprima el token: echo “TOKEN: ${TOKEN}” y esto nos lo pide el ejercicio.

9. Vamos a la página de jwt.io/introduction, bajamos y vamos al apartado de Payload, en concreto Registered claims: nos dice datos importantes del token.



Payload

The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional data. There are three types of claims: *registered*, *public*, and *private* claims.

- **Registered claims**: These are a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. Some of them are: **iss** (issuer), **exp** (expiration time), **sub** (subject), **aud** (audience), and others.

En la página https://datatracker.ietf.org/doc/html/rfc7519#section-4.1 tenemos más información.
Vamos ahora a la sección Debugger donde podemos hacer cosas para verificar firmas. Copiamos el Token obtenido. Vemos que nos muestra el payload, tenemos que entender los atributos (el nbf es más técnico y menos importante).

IMPORTANTE: el token no está cifrado por lo que nunca debemos compartir nada privado con él. Lo que sí está es firmado porque lo que nos interesa es saber que no ha sido modificado por nadie (man-in-the-middle).
En nuestro caso lo cifraremos con una clave secreta que está en GitHub en el proyecto > nodejs > config > config.json. Nos sale una contraseña que es compalaApiDeLaETSIIDaLachertida.

10. Docker Compose:
Lo que hace DockerCompose es orquestar contenedores, esto es, automatizar de la mayoría de las operaciones necesarias para ejecutar cargas de trabajo y servicios en contenedores.

Nos vamos en GitHub a iseP4JMeter > docker-compose.yaml. Este documento es muy importante.
Nos salen varios contenedores:
- MONGODBINIT : cuando arrancamos el servicio por primera vez “inyecta” los datos en el contenedor MongoDB y luego muere. Lo normal es que los contenedores no contengan datos permanentes (solo provisionales) así que lo que hacemos es tener los datos en otra parte.
- MONGODB : Ponemos un puerto.
- NODEJS : Expone a través de su puerto 3000 dos endpoints que son /login y /alumno.

Tenemos que tener en cuenta que los contenedores están aislados, no tienen comunicación entre sí por lo que hay que creársela.
Normalmente un contenedor no sabe la IP de otro equipo que quiere atacar, por eso se usan nombres simbólicos. Por ejemplo si nos vamos al archivo config.json vemos que donde debería ir la IP en la primera línea, pone “mongodb”. Es como una especie de DNS privado, lo cual es una ventaja de este servicio.

De vuelta al archivo : docker-compose.yaml.

Lo primero que se hace es arrancar la imagen de mongo. Esto es un repositorio donde la gente publica sus contenedores.
Luego hace build ./nodejs. En este directorio hay un archivo Dockerfile que monta la imagen de este contenedor. Su contenido es de la forma:

- FROM : basándose en una imagen ya existente node:16...stretch (ya nadie crea una imagen desde cero),
- RUN : ejecuta un comando, en nuestro caso crea un directorio
- COPY : copia todo el contenido del directorio actual en el directorio creado
- WORKDIR : indica un directorio de trabajo
- RUN : instala npm, que es el sistema de gestión de paquetes de node.js
- ENV activa un flag para decir que el servidor va a realizar operaciones en producción y no en desarrollo (no es muy importante)
- CMD : arranca el servicio

En la página hub.docker.com podemos encontrar registros de contenedores ya creados.
Ejemplo: Buscamos “tomcat” en el buscador. Le damos al primero de un gato. En el apartado tags nos podemos descargar una versión concreta del contenedor.
En los contenedores se suelen usar distribuciones linux específicos que lo único que tienen es un kernel y un sistema de archivos súper reducido porque su objetivo es que sean lo más ligeros posibles.

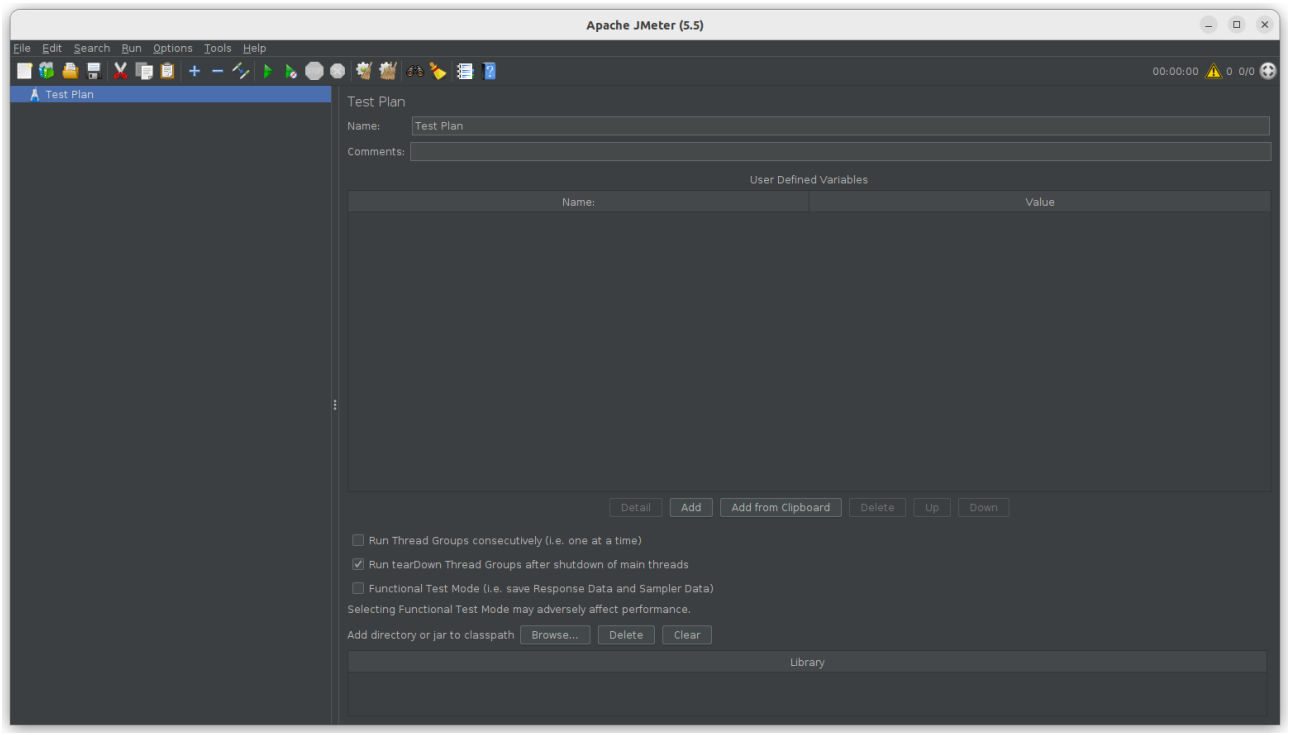
Ya sabemos cómo se arranca el servicio y cómo se monta. Vamos a empezar a trabajar con JMeter.

JMETER

Es un sistema de carga que lanza muchas hebras sobre nuestra máquina que se van a identificar y van a lanzar peticiones.
El profe recomienda que lo corramos sobre nuestro ordenador anfitrión por temas de eficiencia. Para ello vamos a necesitar Java >= 8.

Lo que vamos a hacer es atacar los dos endpoints /login y /alumno.

Primero instalamos jMeter en el anfitrión en la página:
https://jmeter.apache.org/download_jmeter.cgi (en concreto el zip binario comprimido).
Luego descomprimimos y en la carpeta /bin tenemos la aplicación que la ejecutamos con java:
java -jar ./Downloads/apache-jmeter-5.5/bin/ApacheJMeter.jar

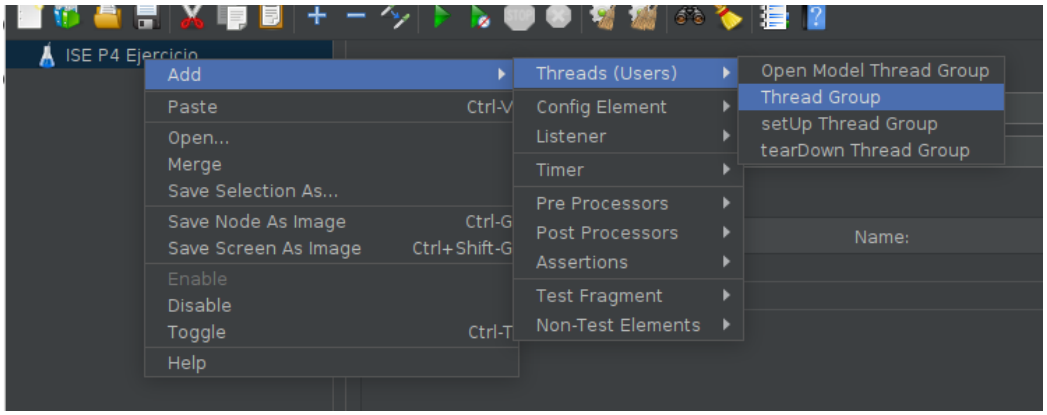


EN EL EJERCICIO NO PONER EN TODO EXACTAMENTE LOS MISMOS NOMBRES QUE NOS DICEN.

1. Test Plan es la base de un proyecto de Jmeter. Tenemos que completar los datos:
Name = Ejercicio P4 ISE
Comments = Ejercicio de la alumna Elsa Rodríguez
Para cambiar el idioma Options -> Language

En Test Plan podemos crear una especie de **variables globales** que sirvan para todos los tests que hagamos.
Añadir en la tabla: HOST = IPUbuntu y PORT = 3000.
Podemos ahora poner en las peticiones ${PORT} y ${HOST} en el puerto y en el Server. De esta manera si cambia la IP o el puerto, solo lo tenemos que cambiar en este archivo.

2. Ahora vamos a crear dos grupos de hebras distintos se utiliza para modelar grupos de usuarios que tienen comportamientos similares. En nuestro caso serán Alumnos y Administradores.
Boton derecho (del archivo que se ha creado al aceptar lo del Test Plan) > **Thread Group**



Alumnos:
Name = Alumnos
Numer of threads = 10. Si nuestro ordenador corre menos hebras, esto es un problema. Lo que se hace normalmente es usar la opción de Run > ? para instalar JMeter en muchos ordenadores y correrlos todos en local. Entonces sí que podríamos ejecutar muchas hebras. Con la herramienta flood.io le podemos decir "Quiero que corras 'x' hebras en ordenadores de Asia, otras 'y' hebras en Rusia, etc".
Ramp-up period = 1 (No es realista poner 0)
Loop count = 5 : Número de veces que se van a ejecutar las hebras
El resto lo dejamos como está.

Administradores:
Name = Administradores
Numer of threads = 3
Ramp-up period = 1
Loop count = 10

ESTO NO ES PARTE DEL EJERCICIO

3. Hacemos una prueba de http. Botón derecho sobre Alumnos y Add > Sampler > **HTTP Request**:
Name = Pagina Home de UJA
Server name or IP = www.ujaen.es (la IP siempre **sin el protocolo** (http o https)). Si la dirección tiene cosas con /.../... esto lo tenemos que poner en el apartado "path"
HTTP Request : GET

Para ver los resultados: doble click en "View Results Tree" en el apartado de "Text": Si le damos a una vemos que se han hecho dos llamadas por cada petición.

4. Botón derecho en Alumnos > Add > Sampler > HTTP Request :
Hacemos lo mismo que antes poniendo una URL que esté mal.

Al ejecutar vemos que la primera nos sale bien y para la segunda nos da error.

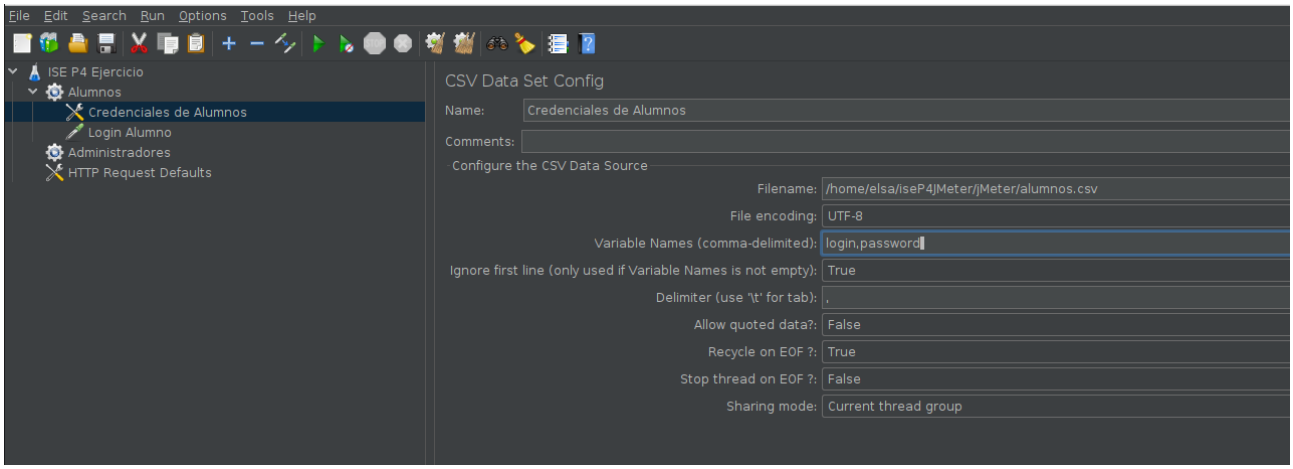
---

5. Ahora vamos a empezar a crear tests. Para no tener que escribir ${PORT} y ${HOST} en todos los tests podemos crear **valores por defecto** en Add > Config Element > HTTP Request Defaults.

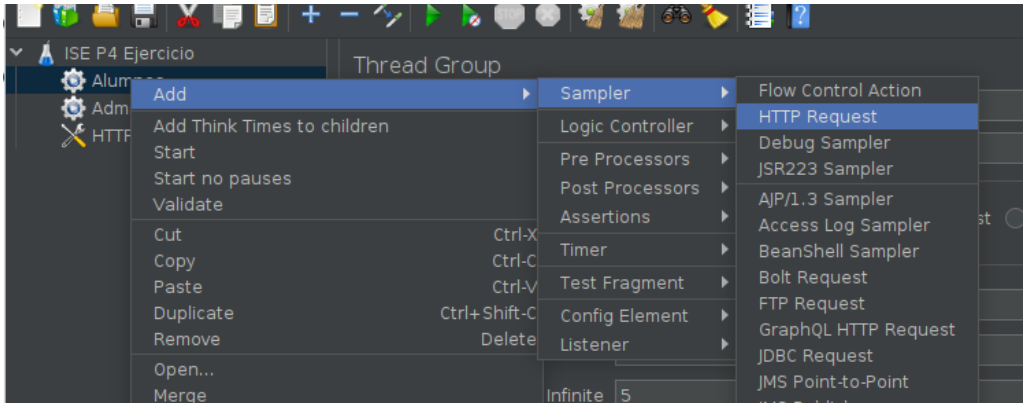
De esta manera, si dejamos en blanco estos huecos en los tests, tomará estos valores.

Name = Valores por defecto de la prueba de carga
Añadir el host y el puerto: ${HOST} y ${PORT}: ya para todos los demás se usarán esos sin ni siquiera tener que especificarlos.
Añadir que las peticiones se harán con el protocolo http.

6. Antes de poder hacer la petición de POST Login Alumnos, debemos obtener las credenciales de los alumnos.
Botón derecho sobre Login Alumnos > Add > Config Element > CSV Data Set.
Name = Datos de Alumnos
Filename = ruta al archivo alumnos.csv
De esta manera en el test HTTP Request del Login Alumnos podemos poner como parámetros ${login} y coge todos los emails de alumnos. Y lo mismo con ${password}.

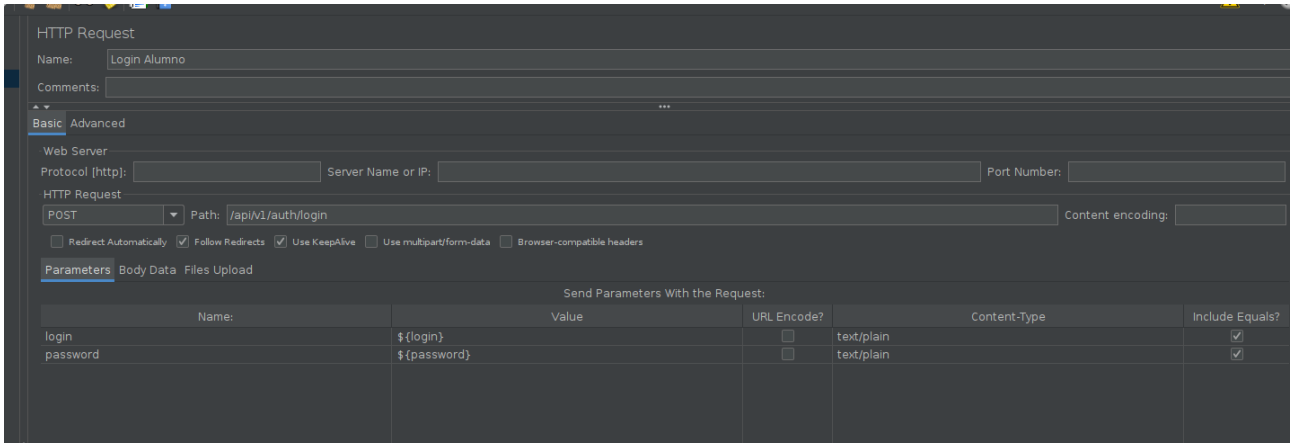


7. Ahora hacemos una HTTP Request **POST** sobre Alumnos para el Login:

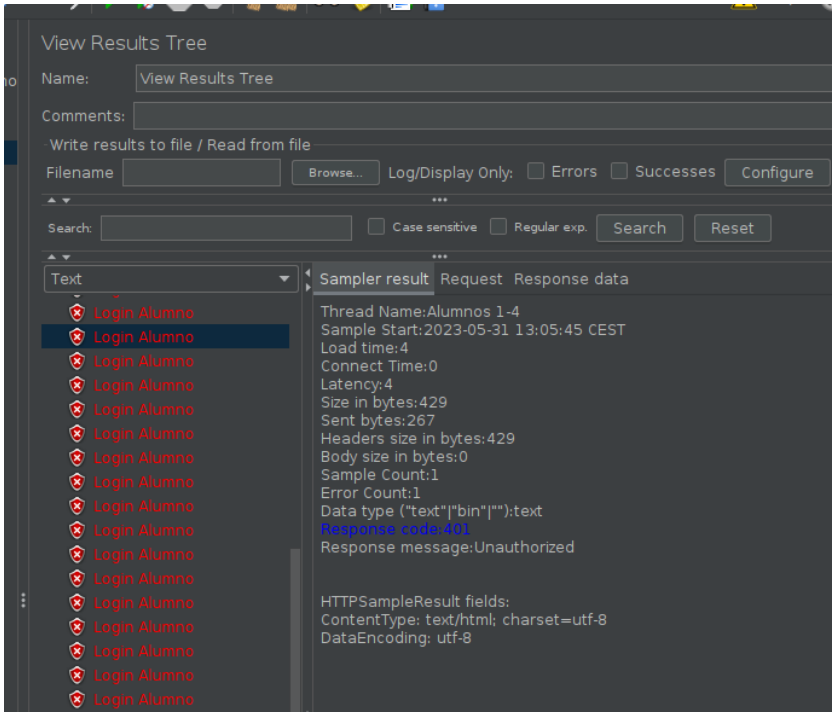


Name = Login Alumnos
Path = /api/v1/auth/login
HTTP Request = POST

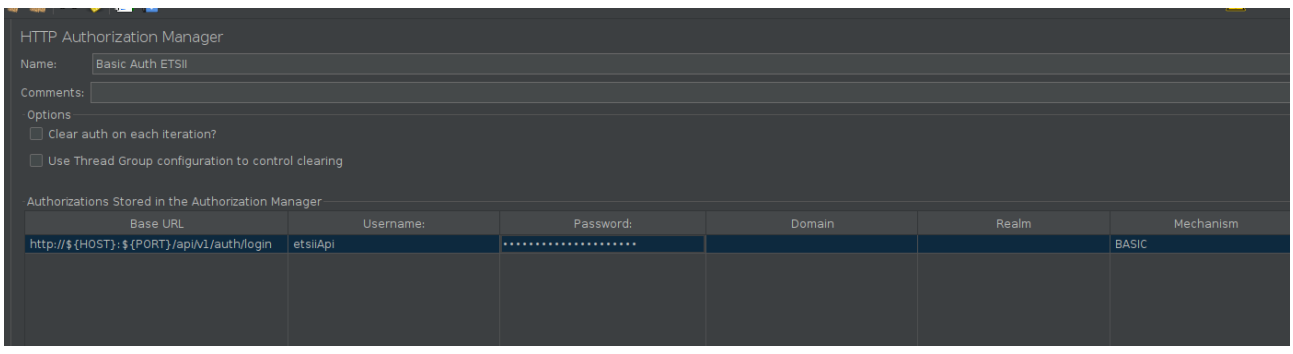
Parámetros:
- login | ${login}
- password | ${password}



8. Botón derecho en el HTTP Request creado > Listener > “Agregate Report” y “View Results Tree” : se utilizan para editar y depurar la prueba de carga, NO para ejecutarla.
Run > Start :
File = pruebajmeter . Esto es lo que vamos a tener que entregar en el examen.
Vemos que nos da un error de Unauthorized:

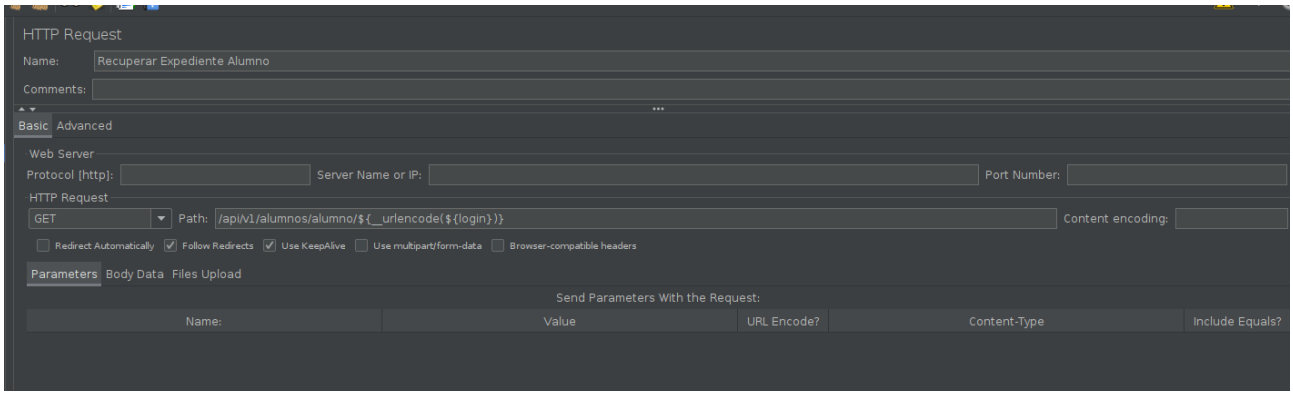


9. Botón derecho y Add > Config Element > HTTP Authorization Manager:
Name = Basic Auth ETSII
Parámetros : los que vienen en el archivo config.json



Ejecutamos y ya funciona bien.

10. Añadimos otra HTTP Request **GET** para recuperar los datos de alumnos:
Name = Recuperar Expediente Alumno
Path = /api/v1/alumnos/alumno/{email que hayamos puesto en los params de Login Alumnos cambiando @ por %}.
HTTP Request = GET



Al ejecutar falla de nuevo porque no está autorizado ya que no le hemos pasado la cabecera del token.

11. Tenemos que guardar el token JWT que nos ha devuelto la petición POST. Para comprobar si una cadena es un token JWT lo hacemos mediante expresiones regulares.

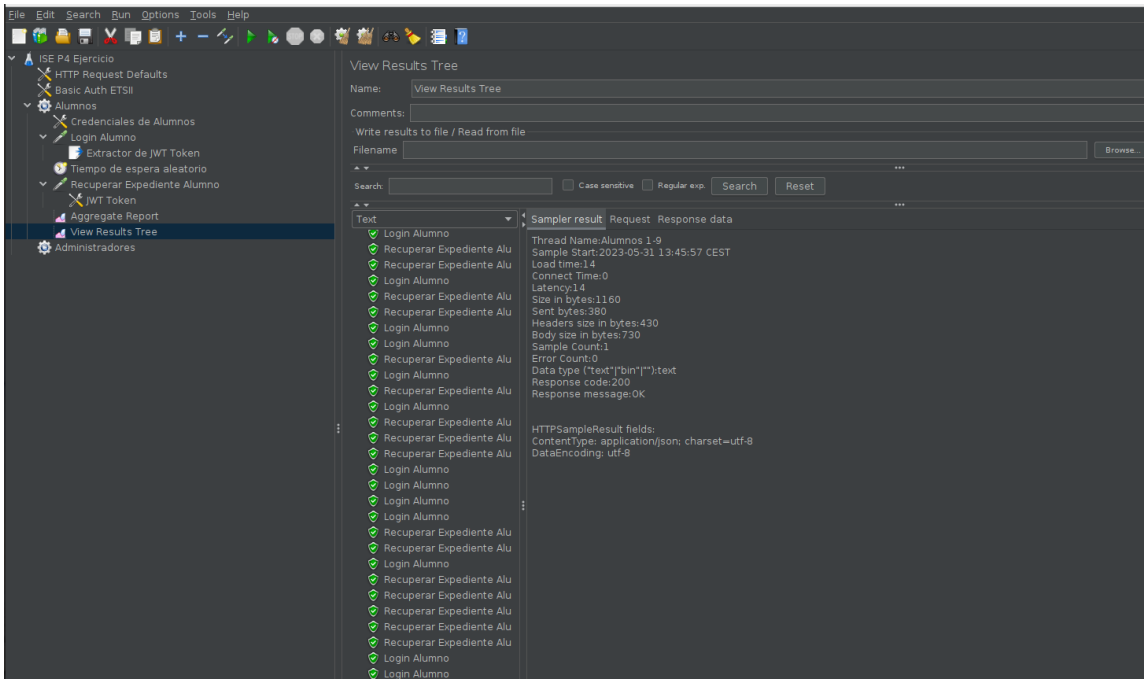
Botón derecho sobre Login Alumnos > Add > Post-Processor > Regular Expression Extractor.
Name = Extractor de JWT Token
Name of created variable = token
Regular expression = ,+
Template = 0

12. Botón derecho sobre Recuperar Expediente Alumno > Add > Config Element > HTTP Header Manager:
Name = JWT Token
En la tabla:
Name = La cabecera tiene que ser la misma que la del archivo pruebaEntorno, que es “Authorization” es nuestro caso.
Value = Bearer ${token}

13. Añadimos un tiempo de espera aleatorio.
Click derecho en Alumnos: Add > Timer > Gaussian Random Timer: para añadir esperas aleatorias. Podemos modificar los parámetros (los ms) para aparentar el comportamiento real de los usuarios, simulando pausas aleatorias desde 0ms a los ms especificados.

Lo situamos entre las peticiones POST y GET.

Si ejecutamos ahora ya funciona bien.



14. Ahora debemos hacer exactamente lo mismo para administradores: crear “Credenciales de Administradores” y “Login Administradores” con el extractor del token

15. Ahora desde Administradores queremos pedir los expedientes de los alumnos:
Add > Sampler > Access Log Sampler :
Name = Accesos Administradores
Protocol = http
Server = ${HOST}
Port = ${PORT}
Log file = ruta al archivo apiAlumnos.log

Podemos deshabilitar los sampler dandole a botón derecho > Deshabilitar, esto es mejor que eliminarlos directamente por si luego queremos usarlos.

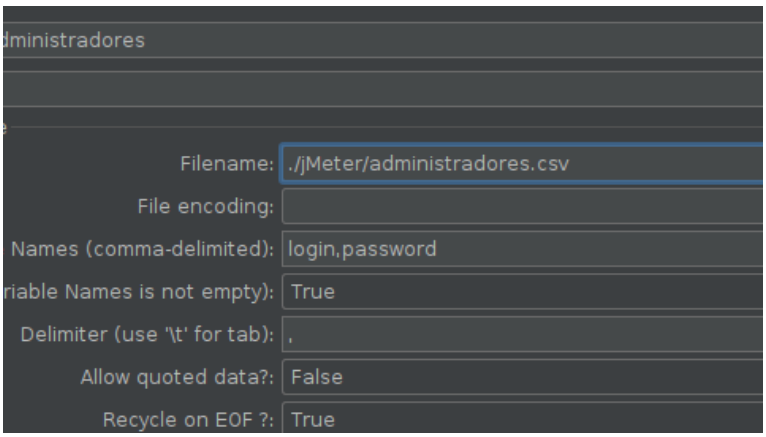
16. Creamos de nuevo el gestor de cabecera que usará el token JWT igual que hicimos para Alumnos.

Ejecutamos y todo funciona correctamente.

15. Ahora vamos a hacer la prueba de carga.
IMPORTANTE: Deshabilitamos el View Results Tree para hacer la prueba de carga.

Nos vamos a la consola donde estamos corriendo jmeter. Ejecutamos jmeter. Nos dice que NO usemos la GUI para ejecutar pruebas de carga reales y nos dice cómo hay que hacerlo con CLI.
pwd
ls

IMPORTANTE: los paths a los archivos los ponemos relativos. TENEMOS QUE CORREGIRLO en el archivo CSV Data Config en el campo “Filename”.



Sintaxis : jmeter -n -t [jmx file] -l [results file] -e -o [Path to web report folder]
ruta_jmeter -n -t p4_clase.jmx -l resultados.jtl
ls : nos ha generado el archivo de resultados
jmeter : Abrimos el entorno gráfico y abrimos el archivo de resultados.

Lo que nos va a pedir en el examen es el archivo .jtx de la prueba y el archivo de resultados .jtl

Mientras, todo esto se va mostrando en la terminal de Ubuntu que está corriendo el Docker. Si hacemos una petición y no aparece ahí, es que algo estamos haciendo mal.

APACHE BENCHMARK

Se instala por defecto con el httpd aunque se puede instalar a parte. Se utiliza para hacer pruebas de carga sobre servidores http.

Su comando es **ab**. En clase solo veremos las dos opciones siguientes:
-n requests : Número de peticiones de peticiones que se van a lanzar en total (por defecto 1)
-c concurrency: Número de peticiones que se van a lanzar a la vez (por defecto es 1)

Ejemplo: 10 llamadas con una concurrencia de 2 hebras
ab -n 10 -c 2 http://www.ugr.es/ (Nivel 2 de concurrencia)

```
(base) elsa@elsarm-Inspiron-5370:~$ ab -n 10 -c 2 http://www.ugr.es/
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking www.ugr.es (be patient).....done


Server Software:        nginx
Server Hostname:        www.ugr.es
Server Port:            80

Document Path:          /
Document Length:        162 bytes

Concurrency Level:      2
Time taken for tests:   0.042 seconds
Complete requests:      10
Failed requests:        0
Non-2xx responses:      10
Total transferred:      3440 bytes
HTML transferred:       1620 bytes
Requests per second:    238.54 [#/sec] (mean)
Time per request:       8.384 [ms] (mean)
Time per request:       4.192 [ms] (mean, across all concurrent requests)
Transfer rate:          80.14 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        3    3   0.7      3       5
Processing:     3    4   0.4      3       4
Waiting:        3    3   0.4      3       4
Total:          6    7   1.0      7       8
ERROR: The median and mean for the processing time are more than twice the standard
       deviation apart. These results are NOT reliable.

Percentage of the requests served within a certain time (ms)
  50%      7
  66%      7
  75%      7
  80%      8
  90%      8
  95%      8
  98%      8
  99%      8
 100%      8 (longest request)
(base) elsa@elsarm-Inspiron-5370:~$
```

En el protocolo http:
- Si la respuesta es correcta: el código de respuesta es 2xx (200 y algo).
- Cuando nos responde algo que no es 2xx, CUIDADO! 4xx y 5xx ya son errores.

El comando anterior nos devuelve:
‘Non-2xx responses: 10’ → Nos dice que las 10 llamadas no han sido éxitosas :(

Hacemos una llamada con curl:
curl -v http://www.ugr.es : Veremos que en location, aparece https (se redirige hacia https) pero supuestamente apache benchmark no soporta las redirecciones. Es decir, hay que hacer:
ab -n 10 -c 2 https://www.ugr.es/ : Ya funciona bien

```
(base) elsa@elsarm-Inspiron-5370:~$ ab -n 10 -c 2 https://www.ugr.es/
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking www.ugr.es (be patient).....done


Server Software:        nginx
Server Hostname:        www.ugr.es
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES128-GCM-SHA256,2048,128
Server Temp Key:        ECDH P-384 384 bits
TLS Server Name:        www.ugr.es

Document Path:          /
Document Length:        97241 bytes

Concurrency Level:      2
Time taken for tests:   0.448 seconds
Complete requests:      10
Failed requests:        0
Total transferred:      1031514 bytes
HTML transferred:       972410 bytes
Requests per second:    22.33 [#/sec] (mean)
Time per request:       89.558 [ms] (mean)
Time per request:       44.779 [ms] (mean, across all concurrent requests)
Transfer rate:          2249.58 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       32   42   8.3     44      57
Processing:    21   33   9.5     33      52
Waiting:        5   18  10.2     14      41
Total:         64   75  11.4     78      98

Percentage of the requests served within a certain time (ms)
  50%     78
  66%     79
  75%     81
  80%     88
  90%     98
  95%     98
  98%     98
  99%     98
 100%     98 (longest request)
```

IMPORTANTE : Nos puede preguntar métricas y percentiles (CONOCER TODAS LAS MÉTRICAS DE APACHE BENCHMARK).

Algunas métricas interesantes son:

- Connect → Tiempo que tardamos en hacer el handshaking de tcp para abrir una conexión permanente con el servidor.
- Processing → Tiempo en que la conexión ha estado abierta.
- Waiting → Tiempo que transcurre desde que le mandamos la petición get hasta que recibimos el primer byte de respuesta.
- Total → Tiempo desde que solicitamos la conexión por TCP hasta que la conexión se cierra. Normalmente es la suma de Connect y Processing (se obvia el Waiting).

Percentiles → Porcentaje de peticiones servidas en un intervalo de tiempo. Es decir, cuando por ejemplo decimos que un servidor es capaz de procesar 20000 peticiones por minuto, o que el tiempo de latencia es de 145ms, etc.
Cuando hacemos afirmaciones así, estamos diciendo que la latencia está en el percentil x%.

Nos quedamos con la latencia de 95% y 98%

Apache Benchmark produce el siguiente report donde aparecen las métricas (es un ejemplo):

```
Server Software:        AmazonS3
Server Hostname:       <SOME_HOST>
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES128-GCM-SHA256,2048,128
TLS Server Name:        <SOME_HOST>

Document Path:          /
Document Length:        45563 bytes

Concurrency Level:      2
Time taken for tests:   3.955 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      4625489 bytes
HTML transferred:       4556300 bytes
Requests per second:    25.29 [#/sec] (mean)
Time per request:       79.094 [ms] (mean)
Time per request:       39.547 [ms] (mean, across all concurrent requests)
Transfer rate:          1142.21 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       40   53   8.1     51      99
Processing:    12   24   9.4     23      98
Waiting:        5   14  10.6     12      95
Total:         57   77  15.0     75     197

Percentage of the requests served within a certain time (ms)
  50%     75
  66%     77
  75%     80
  80%     81
  90%     85
  95%     92
  98%    116
  99%    197
 100%    197 (longest request)
```

Aquí podemos ver que en 80 ms se sirven el 75% de peticiones.

Apachebenchmark es muy sencillo, pero por esto no se pueden hacer pruebas de carga muy complejas como hacíamos con JMeter. Sirve sobre todo para pruebas Smoketest (= revisión rápida de un producto software para verificar que funciona y no tiene defectos evidentes).

PHORONIX

Aplicación para correr benchmarks colgados en openbenchmark.org, que es una página similar a GitHub donde la gente sube proyectos opensource.

Para insalarlo, la forma más cómoda es buscar en Docker Hub la imagen:

phoronix/pts Docker image

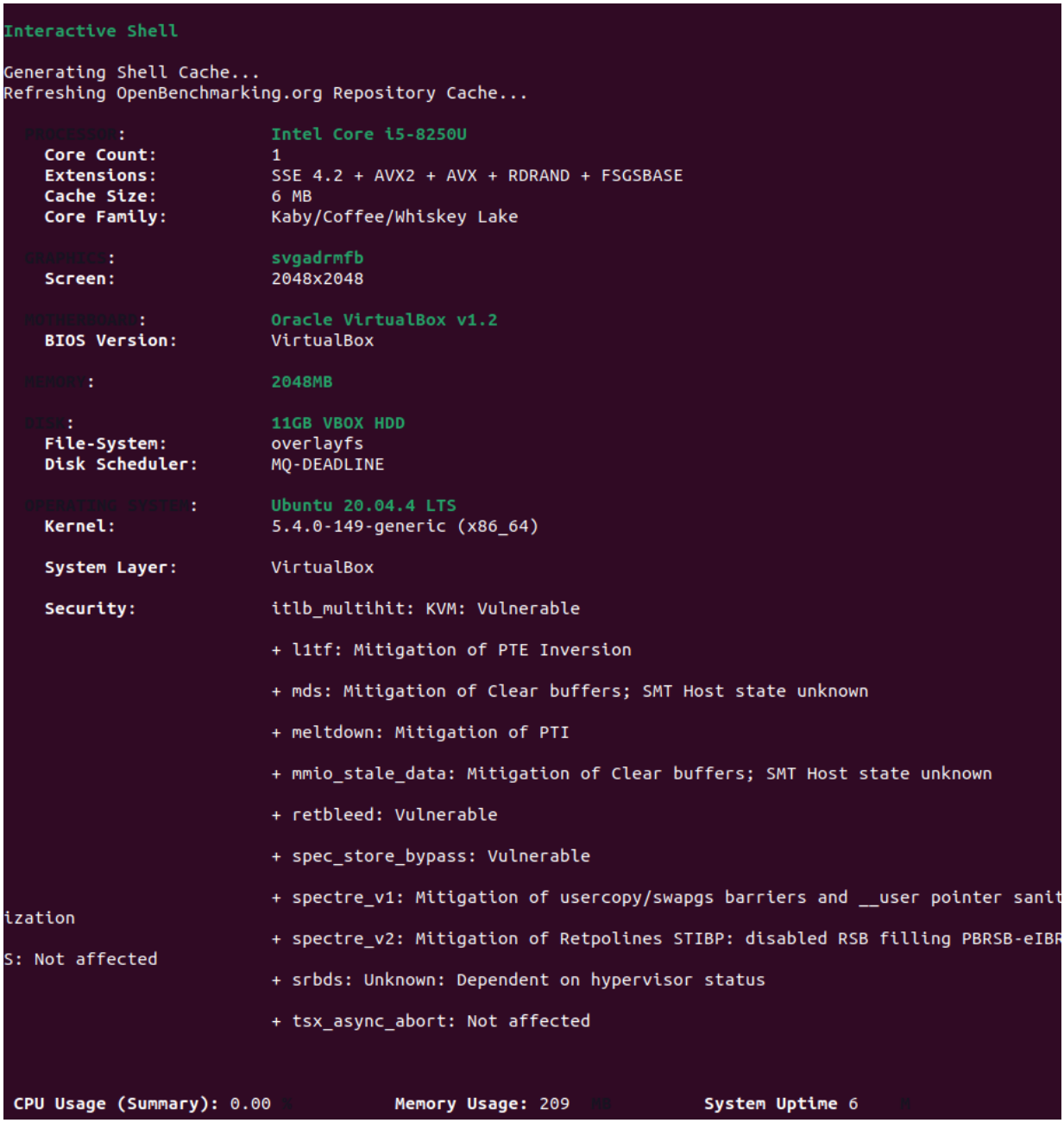
(puede correr Phoronix sobre Docker porque los contenedores no virtualizan, se corre directamente sobre la estructura y el sistema operativo).
Para ello HAY QUE TENER DOCKER INSTALADO.

Hacemos en la MV Ubuntu:
sudo docker pull phoronix/pts
sudo docker run -it phoronix/pts : para correr docker con interfaz interactiva. Nos aparecen primero algunos cambios en los tests (tests actualizado y nuevos desde el 13 de febrero):

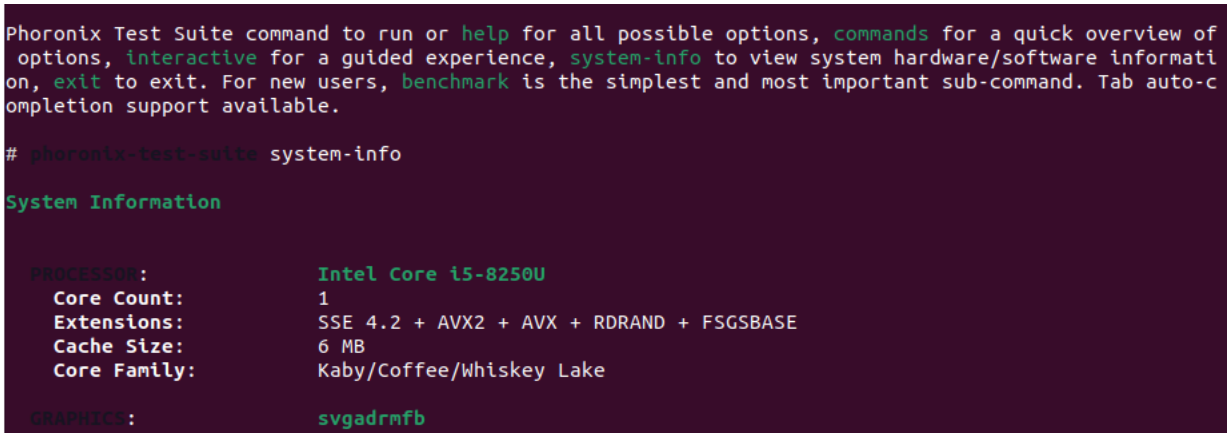
```
^Cerodriguez@ubuise:~sudo o docker run -it phoronix/pts

Updated OpenBenchmarking.org Repository Index
pts: 522 Distinct Tests, 2250 Test Versions, 56 Suites
Available Changes From 13 February 2022 To 2 June
New Test:       pts/3dmark                v1.0.0   3DMark Wild Life Extreme
Updated Test:   pts/ai-benchmark          v1.0.2   AI Benchmark Alpha
Updated Test:   pts/aircrack-ng           v1.3.0   Aircrack-ng
Updated Test:   pts/aom-av1               v3.4.0   AOM AV1
Updated Test:   pts/apache                v3.0.0   Apache HTTP Server
Updated Test:   pts/astcenc               v1.4.0   ASTC Encoder
Updated Test:   pts/avifenc               v1.3.0   libavif avifenc
New Test:       pts/axs2mlperf            v1.0.0   axs2mlperf
Updated Test:   pts/batman-knight         v1.0.1   Batman: Arkham Knight
Updated Test:   pts/blender               v3.5.0   Blender
Updated Test:   pts/blosc                 v1.2.0   C-Blosc
Updated Test:   pts/brl-cad               v1.4.0   BRL-CAD
Updated Test:   pts/build-erlang          v1.2.0   Timed Erlang/OTP Compilation
Updated Test:   pts/build-ffmpeg          v4.0.0   Timed FFmpeg Compilation
Updated Test:   pts/build-godot           v1.0.0   Timed Godot Game Engine Compilation
Updated Test:   pts/build-linux-kernel    v1.15.0  Timed Linux Kernel Compilation
Updated Test:   pts/build-llvm            v1.5.0   Timed LLVM Compilation
Updated Test:   pts/build-mplayer         v1.5.0   Timed MPlayer Compilation
Updated Test:   pts/build-nodejs          v1.3.0   Timed Node.js Compilation
Updated Test:   pts/build-php             v1.0.0   Timed PHP Compilation
New Test:       pts/build-python          v1.0.0   Timed CPython Compilation
Updated Test:   pts/build-wasmer          v1.2.0   Timed Wasmer Compilation
Updated Test:   pts/build2                v1.2.0   Build2
Updated Test:   pts/chia-vdf              v1.1.0   Chia Blockchain VDF
New Test:       pts/clickhouse            v1.2.0   ClickHouse
Updated Test:   pts/cloudsuite-da         v1.2.0   CloudSuite Data Analytics
Updated Test:   pts/cloudsuite-ga         v1.1.0   CloudSuite Graph Analytics
Updated Test:   pts/cloudsuite-ma         v1.1.0   CloudSuite In-Memory Analytics
Updated Test:   pts/cloudsuite-ms         v1.0.1   CloudSuite Media Streaming
Updated Test:   pts/cloudsuite-ws         v1.1.0   CloudSuite Web Serving
Updated Test:   pts/clpeak                v1.1.0   clpeak
New Test:       pts/cockroach             v1.0.2   CockroachDB
Updated Test:   pts/compress-7zip         v1.10.0  7-Zip Compression
Updated Test:   pts/compress-zstd         v1.4.0   Zstd Compression
Updated Test:   pts/couchdb               v1.2.0   Apache CouchDB
Updated Test:   pts/cpuminer-opt          v1.4.0   Cpuminer-Opt
Updated Test:   pts/csgo                  v1.7.2   Counter-Strike: Global Offensive
New Test:       pts/cyberpunk2077         v1.0.1   Cyberpunk 2077
Updated Test:   pts/daphne                v1.1.0   Darmstadt Automotive Parallel Heterogeneous Suite
Updated Test:   pts/dav1d                 v1.13.0  dav1d
Updated Test:   pts/ddnet                 v1.3.0   DDraceNetwork
New Test:       pts/deeprec               v1.0.2   DeepRec
New Test:       pts/deepsparse            v1.3.2   Neural Magic DeepSparse
Updated Test:   pts/draco                 v1.4.0   Google Draco
New Test:       pts/dragonflydb           v1.0.0   Dragonflydb
Updated Test:   pts/embree                v1.4.0   Embree
Updated Test:   pts/encode-flac           v1.0.1   FLAC Audio Encoding
Updated Test:   pts/encode-opus           v1.4.0   Opus Codec Encoding
New Test:       pts/encodec               v1.0.1   EnCodec
```

Luego nos aparece información de la shell:



Y por último podemos escribir comandos:
> system-info : Me da la información del ordenador anfitrión



```
    Screen:               2048x2048

  MOTHERBOARD:            Oracle VirtualBox v1.2
    BIOS Version:         VirtualBox

  MEMORY:                 2048MB

  DISK:                   11GB VBOX HDD
    File-System:          overlayfs
    Disk Scheduler:       MQ-DEADLINE

  OPERATING SYSTEM:       Ubuntu 20.04.4 LTS
    Kernel:               5.4.0-149-generic (x86_64)

    System Layer:         VirtualBox

    Security:             itlb_multihit: KVM: Vulnerable

                          + l1tf: Mitigation of PTE Inversion

                          + mds: Mitigation of Clear buffers; SMT Host state unknown

                          + meltdown: Mitigation of PTI

                          + mmio_stale_data: Mitigation of Clear buffers; SMT Host state unknown

                          + retbleed: Vulnerable

                          + spec_store_bypass: Vulnerable

                          + spectre_v1: Mitigation of usercopy/swapgs barriers and __user pointer san
itization
                          + spectre_v2: Mitigation of Retpolines STIBP: disabled RSB filling PBRSB-eI
BRS: Not affected
                          + srbds: Unknown: Dependent on hypervisor status

                          + tsx_async_abort: Not affected


 CPU Usage (Summary): 0.00 %          Memory Usage: 198  MB          System Uptime 10    m

Phoronix Test Suite command to run or help for all possible options, commands for a quick overview of
 options, interactive for a guided experience, system-info to view system hardware/software informati
on, exit to exit. For new users, benchmark is the simplest and most important sub-command. Tab auto-c
ompletion support available.

# phoronix-test-suite
```

Suites : son agrupaciones varios benchmark disponibles.
> phoronix-test-suite <opcion>
> list-available-suites : Para ver las suites disponibles

```
# phoronix-test-suite list-available-suites

Available Suites

  pts/audio-encoding              - Audio Encoding                System
  pts/av1                         - AV1                           System
  pts/bioinformatics              - Bioinformatics                System
  pts/browsers                    - Web Browsers                  System
  pts/cad                         - CAD                           System
  pts/chess                       - Chess Test Suite              Processor
  pts/compilation                 - Timed Code Compilation        System
  pts/compression                 - Compression Tests             Processor
```

Nos salimos y ejecutamos:
docker ps : Nos da los contenedores de Docker que están ejecutándose en el anfitrión con sus ids (a mí no me sale ninguno ya que no tengo).

```
erodriguez@ubuise:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
erodriguez@ubuise:~$
```

docker exec -it <container ID> /bin/sh : Para conectarnos a un contenedor.
No es común ejecutar una shell en un contenedor, por ello hay ciertos contenedores en los que no puedes hacer ls. Este no es el caso, aquí sí podemos.

Nos volvemos a conectar al docker de phoronix:
sudo docker run -it phoronix/pts
> list-available-tests : Nos da la lista de los benchmarks disponibles (Hay que bajarlos y ejecutarlos, aquellos que te hagan falta).

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

Available Tests

pts/ai-benchmark            AI Benchmark Alpha                      System
pts/aircrack-ng             Aircrack-ng                             Processor
pts/amg                     Algebraic Multi-Grid Benchmark          Processor
pts/aobench                 AOBench                                 Processor
pts/aom-av1                 AOM AV1                                 Processor
pts/apache                  Apache HTTP Server                      System
pts/apache-siege            Apache Siege                            System
pts/appleseed               Appleseed                               System
pts/arrayfire               ArrayFire                               Processor
pts/askap                   ASKAP                                   System
pts/asmfish                 asmFish                                 Processor
pts/astcenc                 ASTC Encoder                            System
pts/avifenc                 libavif avifenc                         Processor
pts/axs2mlperf              axs2mlperf                              System
pts/basemark                Basemark GPU                            System
pts/basis                   Basis Universal                         System
pts/blake2                  BLAKE2                                  Processor
pts/blender                 Blender                                 System
pts/blogbench               BlogBench                               Disk
pts/blosc                   C-Blosc                                 Processor
pts/bork                    Bork File Encrypter                     Processor
pts/botan                   Botan                                   Processor
pts/brl-cad                 BRL-CAD                                 System
pts/build-apache            Timed Apache Compilation                Processor
pts/build-clash             Timed Clash Compilation                 Processor
```

El profe elige un test en concreto llamado pts/compress-gzip y:
> install pts/compress-gzip : Baja el benchmark pero no lo ejecuta
> run pts/compress-gzip : Lo ejecuta

Otra opción es:
> benchmark pts/compress-gzip : Lo baja y lo ejecuta

En mi caso, me dice que no hay espacio suficiente para ejecutar este test.