

Nombre del estudiante: Daniel Alconchel Vázquez
Semestre: 2
Año Académico: 2023/2024



UNIVERSIDAD
DE GRANADA

E.T.S. de Ingenierías Informática y de Telecomunicación
Práctica 5: Benchmarking
Servidores Web de Altas Prestaciones

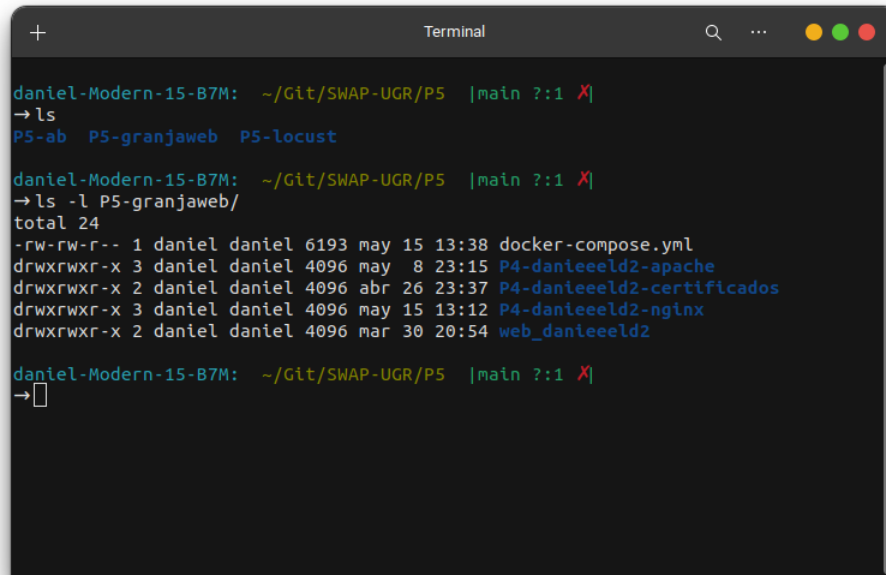
Índice

1. Tareas Básicas	2
1.1. Creación del espacio de trabajo Benchmarking	2
1.2. Implementación de Apache Benchmark	2
1.3. Implementación con Locust	3
1.4. Verificación y Pruebas del escenario con Benchmarking	3
1.4.1. Apache Benchmark	3
1.4.2. Locust	7
2. Tareas Avanzadas	9
2.1. Desarrollar tareas avanzadas en Locustfile.py	9
2.2. Crear escenarios multicontenedor con algún CMS	13
2.3. Ejecución y análisis de carga sobre CMS	21

1. Tareas Básicas

1.1. Creación del espacio de trabajo Benchmarking

Como siempre, comenzamos creando la estructura de directorios que nos indica el guión de prácticas:



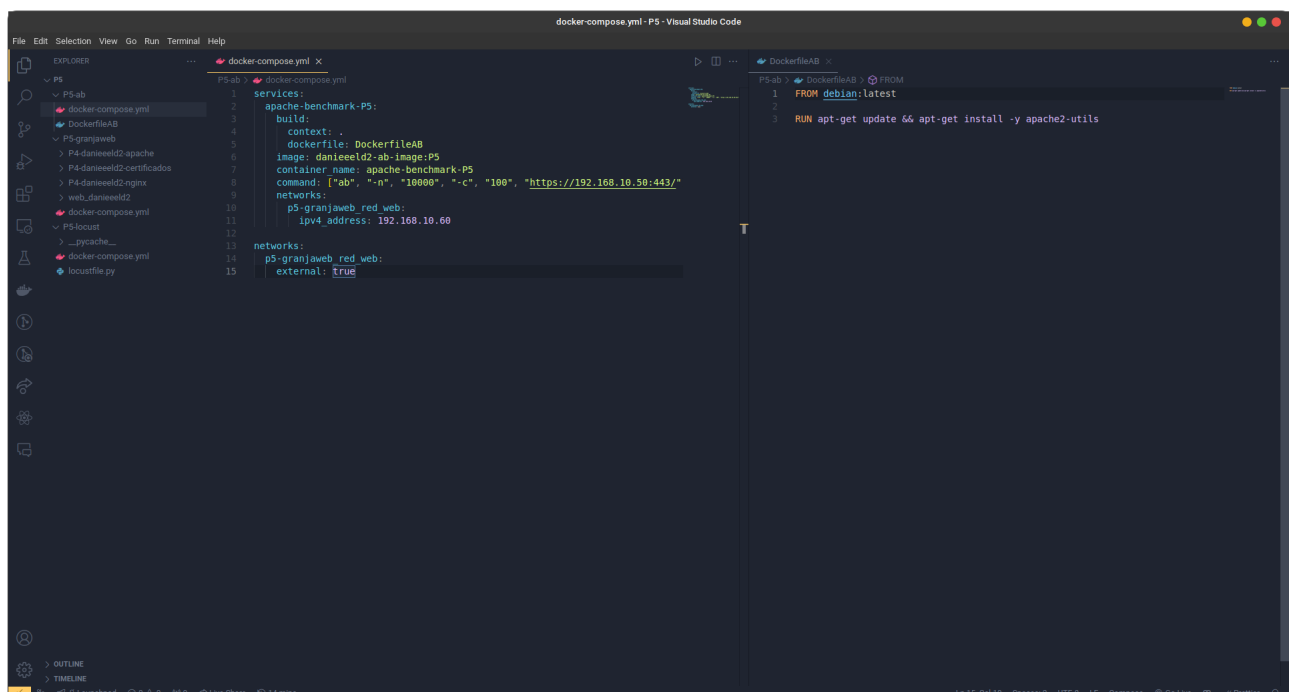
```
daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P5 |main ? :1 X|
→ ls
P5-ab P5-granjaweb P5-locust

daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P5 |main ? :1 X|
→ ls -l P5-granjaweb/
total 24
-rw-rw-r-- 1 daniel daniel 6193 may 15 13:38 docker-compose.yml
drwxrwxr-x 3 daniel daniel 4096 may 8 23:15 P4-danieeld2-apache
drwxrwxr-x 2 daniel daniel 4096 abr 26 23:37 P4-danieeld2-certificados
drwxrwxr-x 3 daniel daniel 4096 may 15 13:12 P4-danieeld2-nginx
drwxrwxr-x 2 daniel daniel 4096 mar 30 20:54 web_danieeld2

daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P5 |main ? :1 X|
→
```

1.2. Implementación de Apache Benchmark

Nuevamente, seguimos las instrucciones del guión y creamos el `DockerfileAB` y su respectivo `docker-compose.yml` tal y como nos indica el guión de prácticas:



```
docker-compose.yml - P5 - Visual Studio Code

EXPLORER
P5
├── P5-ab
│   ├── docker-compose.yml
│   └── DockerfileAB
├── P5-granjaweb
│   ├── P4-danieeld2-apache
│   ├── P4-danieeld2-certificados
│   ├── P4-danieeld2-nginx
│   └── web_danieeld2
├── docker-compose.yml
├── P5-locust
├── ...pycache...
├── docker-compose.yml
└── locustfile.py

1 services:
2   apache-benchmark-P5:
3     build:
4       context: .
5       dockerfile: DockerfileAB
6     image: danieeld2-ab-image-p5
7     container_name: apache-benchmark-P5
8     command: ["ab", "-n", "10000", "-c", "100", "https://192.168.10.50:443/"]
9     networks:
10      p5-granjaweb_red_web:
11        ipv4_address: 192.168.10.60
12
13 networks:
14   p5-granjaweb_red_web:
15     external: true

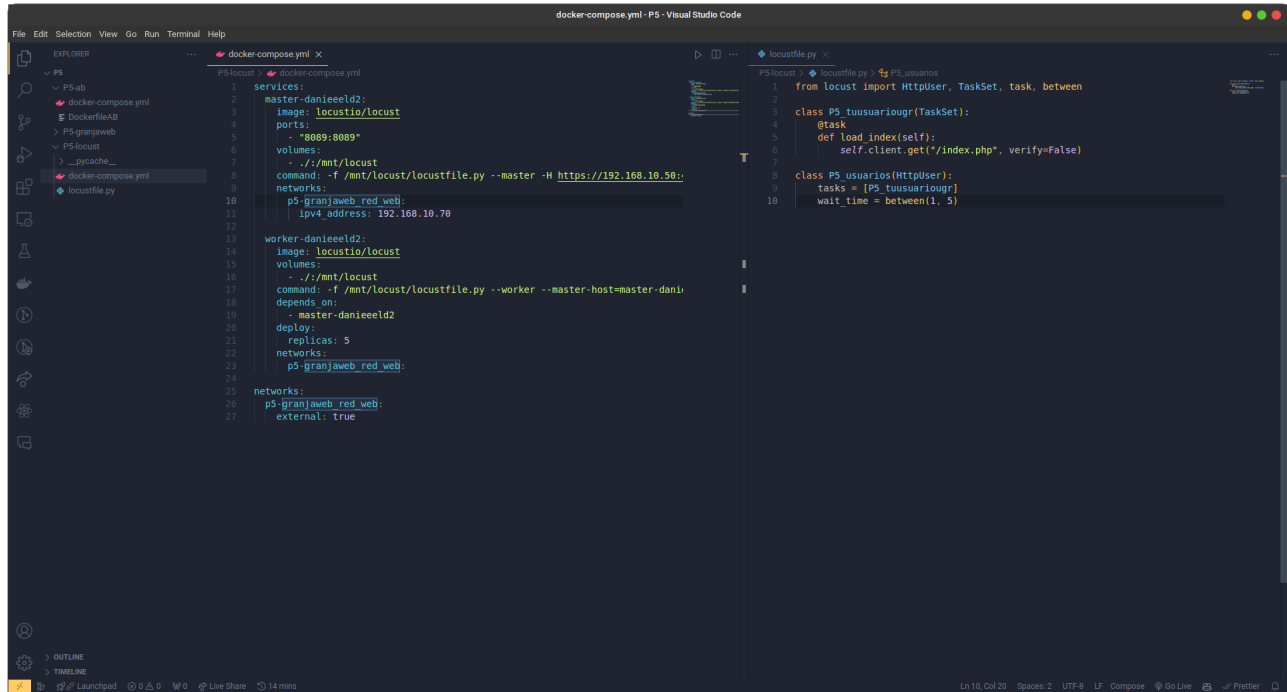
DockerfileAB
FROM debian:latest
RUN apt-get update && apt-get install -y apache2-utils
```

Como vemos, bastante sencillo y, además, no tenemos que construir la imagen, ya que se encarga el propio `docker-compose.yml` de hacerlo.

Nota: Como las redes las estamos creando desde el `docker-compose` de la granja, esta toma como prefijo el nombre de la carpeta en la que está situada el `docker-compose.yml`, por lo que no podemos indicar directamente `red_web`, sino que tenemos que indicar `p5-granjaweb_red_web`.

1.3. Implementación con Locust

Vamos a continuar siguiendo las instrucciones del guión. Nos toca crear el `docker-compose.yml` para Locust y su respectivo archivo de configuración, `locustfile.py`.



The screenshot shows the Visual Studio Code interface with two files open. The left pane shows the Explorer view with a file tree containing `PS-ab`, `docker-compose.yml`, `locustfile.py`, and `locustfile.py`. The main editor displays the `docker-compose.yml` file with the following content:

```
1 services:
2   master-danieeld2:
3     image: locustio/locust
4     ports:
5       - "8089:8089"
6     volumes:
7       - ./mnt/locust
8     command: -f /mnt/locust/locustfile.py --master -H https://192.168.10.50:
9     networks:
10      p5-granjaweb_red_web:
11        ipv4_address: 192.168.10.70
12
13   worker-danieeld2:
14     image: locustio/locust
15     volumes:
16       - ./mnt/locust
17     command: -f /mnt/locust/locustfile.py --worker --master-host=master-dani
18     depends_on:
19       - master-danieeld2
20     deploy:
21       replicas: 5
22     networks:
23      p5-granjaweb_red_web:
24
25 networks:
26   p5-granjaweb_red_web:
27     external: true
```

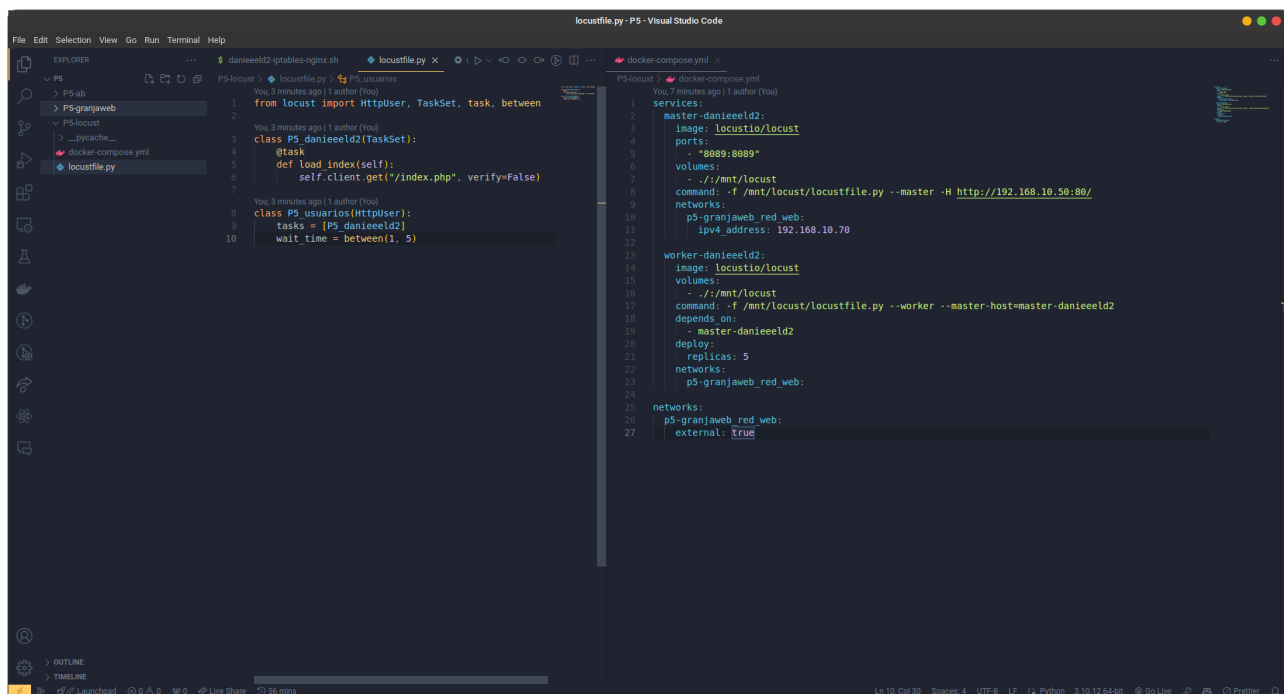
The right pane shows the `locustfile.py` file with the following content:

```
1 from locust import HttpUser, TaskSet, task, between
2
3 class P5_tuusuariougr(TaskSet):
4     @task
5     def load_index(self):
6         self.client.get("/index.php", verify=False)
7
8 class P5_usuario(HttpUser):
9     tasks = [P5_tuusuariougr]
10    wait_time = between(1, 5)
```

1.4. Verificación y Pruebas del escenario con Benchmarking

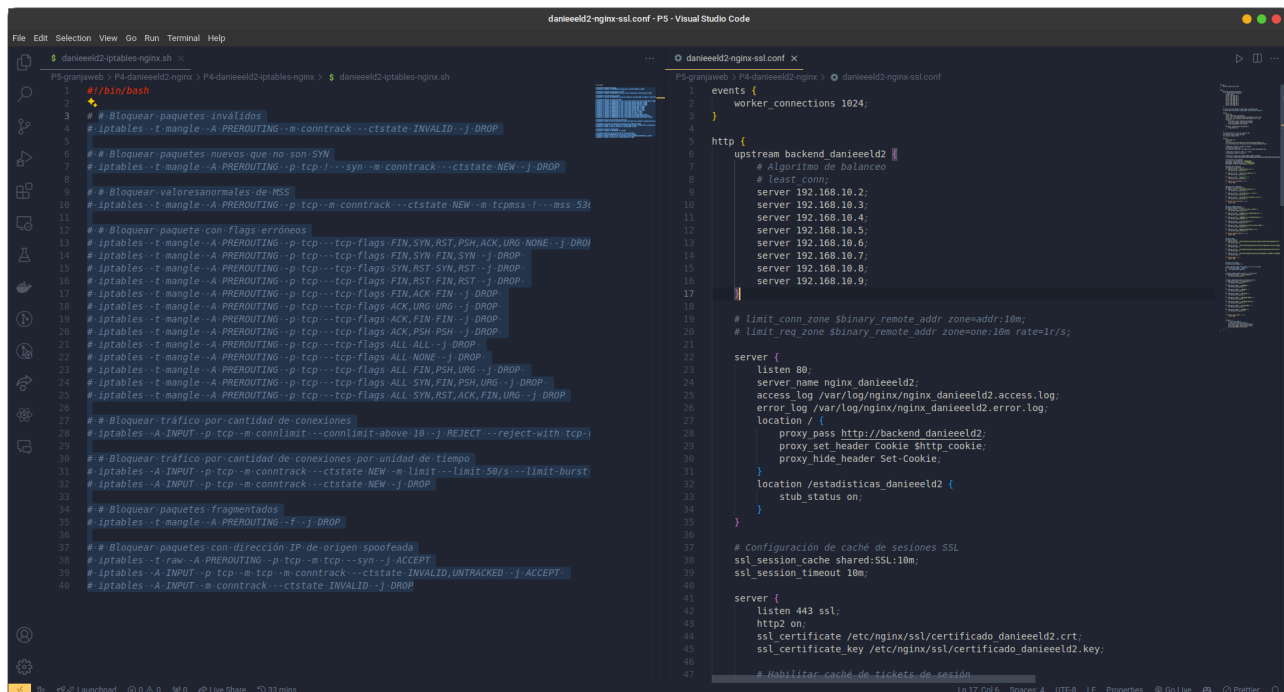
1.4.1. Apache Benchmark

Comenzamos desplegando la granja web. Para ello, accedemos a la carpeta correspondiente y ejecutamos `docker compose up -d`. Una vez estén todos los servicios levantados, vamos a la carpeta de AB y levantamos su `docker-compose.yml`:



Como se puede apreciar en la imagen, hay muchas peticiones que están dando error. Esto se debe a que en la práctica anterior configuré políticas de seguridad para ataques DDoS y, esencialmente hablando, el test este es parecido a este tipo de ataque, ya que lo que hace es mandar muchas peticiones en concurrencia en un corto periodo de tiempo.

Vamos a desactivar las configuraciones avanzadas de seguridad y probaremos de nuevo. Las configuraciones que voy a desactivar son las que aparecen comentadas a continuación:



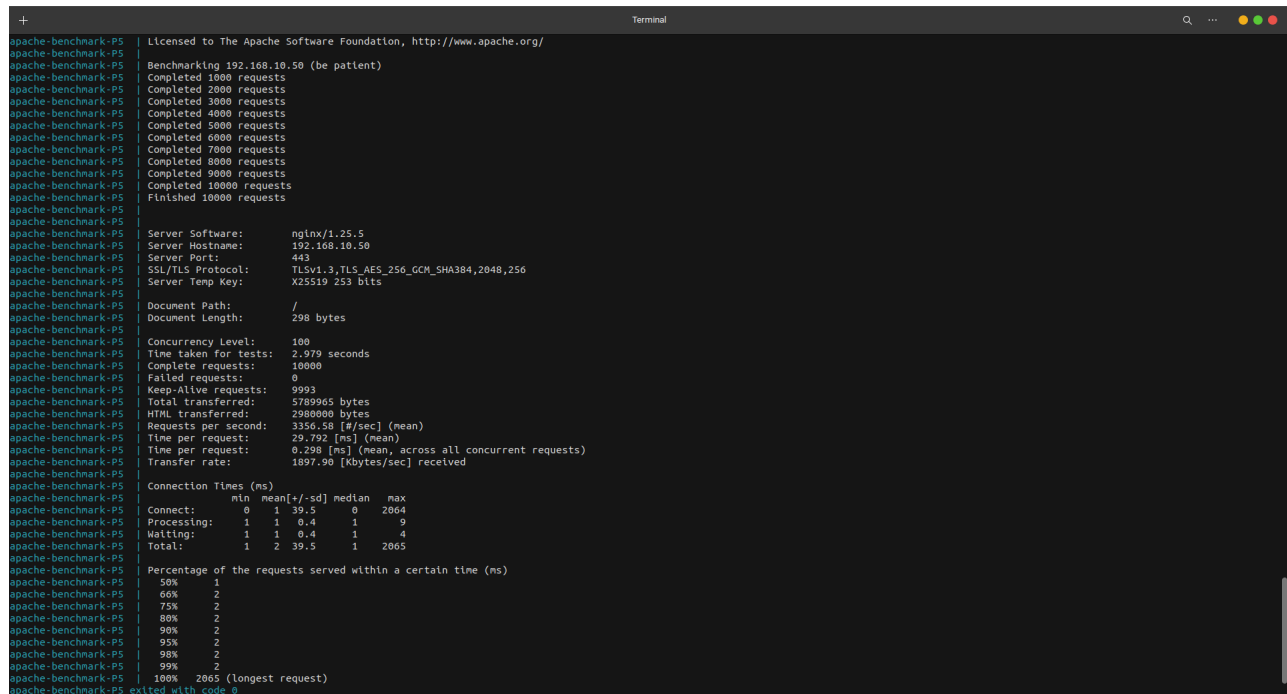
Por un lado, desactivamos toda la configuración de iptables del balanceador y, en su fichero de configuración, desactivamos `limit_conn_zone` y `limit_req_zone`. Con esto ya debería funcionar sin problemas el test.

Nota: Se me olvidó también indicar que no comprobase el certificado *SSL* (ya que este es autofirmado y no es seguro). En caso de no indicar esto, veremos lo siguiente en el test de carga:

```
apache-benchmark-P5 | SSL read failed (5) - closing connection
apache-benchmark-P5 | SSL read failed (5) - closing connection
apache-benchmark-P5 | SSL read failed (5) - closing connection
apache-benchmark-P5 | SSL read failed (5) - closing connection
apache-benchmark-P5 | SSL read failed (5) - closing connection
apache-benchmark-P5 | SSL read failed (5) - closing connection
apache-benchmark-P5 | SSL read failed (5) - closing connection
apache-benchmark-P5 | SSL read failed (5) - closing connection
apache-benchmark-P5 | SSL read failed (5) - closing connection
```

Para ello, basta con incluir el flag *-k*.

Vamos a probar ahora:



```
+ Terminal
apache-benchmark-P5 | Licensed to The Apache Software Foundation, http://www.apache.org/
apache-benchmark-P5 | Benchmarking 192.168.10.50 (be patient)
apache-benchmark-P5 | Completed 1000 requests
apache-benchmark-P5 | Completed 2000 requests
apache-benchmark-P5 | Completed 3000 requests
apache-benchmark-P5 | Completed 4000 requests
apache-benchmark-P5 | Completed 5000 requests
apache-benchmark-P5 | Completed 6000 requests
apache-benchmark-P5 | Completed 7000 requests
apache-benchmark-P5 | Completed 8000 requests
apache-benchmark-P5 | Completed 9000 requests
apache-benchmark-P5 | Completed 10000 requests
apache-benchmark-P5 | Finished 10000 requests
apache-benchmark-P5 |
apache-benchmark-P5 | Server Software:      nginx/1.25.5
apache-benchmark-P5 | Server Hostname:     192.168.10.50
apache-benchmark-P5 | Server Port:         443
apache-benchmark-P5 | SSL/TLS Protocol:    TLSv1.3,TLS_AES_256_GCM_SHA384,2048,256
apache-benchmark-P5 | Server Temp Key:      X25519 253 bits
apache-benchmark-P5 |
apache-benchmark-P5 | Document Path:       /
apache-benchmark-P5 | Document Length:     298 bytes
apache-benchmark-P5 |
apache-benchmark-P5 | Concurrency Level:    100
apache-benchmark-P5 | Time taken for tests: 2.979 seconds
apache-benchmark-P5 | Complete requests:    10000
apache-benchmark-P5 | Failed requests:      0
apache-benchmark-P5 | Keep-Alive requests:  9993
apache-benchmark-P5 | Total transferred:    5789965 bytes
apache-benchmark-P5 | HTML transferred:    2980000 bytes
apache-benchmark-P5 | Requests per second:  3350.58 [#/sec] (mean)
apache-benchmark-P5 | Time per request:     29.792 [ms] (mean)
apache-benchmark-P5 | Time per request:     0.298 [ms] (mean, across all concurrent requests)
apache-benchmark-P5 | Transfer rate:        1897.90 [Kbytes/sec] received
apache-benchmark-P5 |
apache-benchmark-P5 | Connection Times (ms)
apache-benchmark-P5 | min  mean[+/-sd] median  max
apache-benchmark-P5 | Connect:  0   1 39.5   0 2064
apache-benchmark-P5 | Processing: 1   1  0.4   1   9
apache-benchmark-P5 | Waiting:   1   1  0.4   1   4
apache-benchmark-P5 | Total:    1   2 39.5   1 2065
apache-benchmark-P5 |
apache-benchmark-P5 | Percentage of the requests served within a certain time (ms)
apache-benchmark-P5 | 50%    1
apache-benchmark-P5 | 60%    2
apache-benchmark-P5 | 75%    2
apache-benchmark-P5 | 80%    2
apache-benchmark-P5 | 90%    2
apache-benchmark-P5 | 95%    2
apache-benchmark-P5 | 98%    2
apache-benchmark-P5 | 99%    2
apache-benchmark-P5 | 100% 2865 (longest request)
apache-benchmark-P5 |
apache-benchmark-P5 | apache-benchmark-P5 exited with code 0
```

Voy a volver a activar las configuraciones de seguridad indicadas previamente, para poder observar la diferencia:

```
+ Terminal
apache-benchmark-PS | Completed 4000 requests
apache-benchmark-PS | Completed 5000 requests
apache-benchmark-PS | Completed 6000 requests
apache-benchmark-PS | Completed 7000 requests
apache-benchmark-PS | Completed 8000 requests
apache-benchmark-PS | Completed 9000 requests
apache-benchmark-PS | Completed 10000 requests
apache-benchmark-PS | Finished 10000 requests
apache-benchmark-PS |
apache-benchmark-PS | Server Software:      nginx/1.25.5
apache-benchmark-PS | Server Hostname:     192.168.10.50
apache-benchmark-PS | Server Port:         443
apache-benchmark-PS | SSL/TLS Protocol:    TLSv1.3,TLS_AES_256_GCM_SHA384,2048,256
apache-benchmark-PS | Server Temp Key:      X25519 253 bits
apache-benchmark-PS |
apache-benchmark-PS | Document Path:       /
apache-benchmark-PS | Document Length:     298 bytes
apache-benchmark-PS |
apache-benchmark-PS | Concurrency Level:    100
apache-benchmark-PS | Time taken for tests: 3.008 seconds
apache-benchmark-PS | Complete requests:    10000
apache-benchmark-PS | Failed requests:      9996
apache-benchmark-PS |   (Connect: 0, Receive: 0, Length: 9996, Exceptions: 0)
apache-benchmark-PS | Non-2xx responses:    9996
apache-benchmark-PS | Keep-Alive requests:  9992
apache-benchmark-PS | Total transferred:    3740780 bytes
apache-benchmark-PS | HTML transferred:    1970404 bytes
apache-benchmark-PS | Requests per second:  3324.39 [#sec] (mean)
apache-benchmark-PS | Time per request:     30.081 [ms] (mean)
apache-benchmark-PS | Time per request:     0.301 [ms] (mean, across all concurrent requests)
apache-benchmark-PS | Transfer rate:         1214.43 [Kbytes/sec] received
apache-benchmark-PS |
apache-benchmark-PS | Connection Times (ms)
apache-benchmark-PS |   min  mean[+/-sd] median  max
apache-benchmark-PS | Connect:    0    14.5    0  1026
apache-benchmark-PS | Processing:  0    1 29.9    0  2000
apache-benchmark-PS | Waiting:    0    1 29.9    0  2000
apache-benchmark-PS | Total:      0    1 33.3    0  2001
apache-benchmark-PS |
apache-benchmark-PS | Percentage of the requests served within a certain time (ms)
apache-benchmark-PS |   50%    0
apache-benchmark-PS |   60%    0
apache-benchmark-PS |   75%    0
apache-benchmark-PS |   80%    0
apache-benchmark-PS |   90%    0
apache-benchmark-PS |   95%    0
apache-benchmark-PS |   98%    0
apache-benchmark-PS |   99%    0
apache-benchmark-PS |  100%  2001 (longest request)
apache-benchmark-PS |
apache-benchmark-PS | apache-benchmark-PS exited with code 0
danjel@Modern-15-B7M: ~/.../PS/PS-ab |main 7:1 A|
~
```

Podemos ver que las configuraciones de seguridad de la práctica anterior funcionan correctamente, ya que impide que se completen casi todas las solicitudes.

Antes de hacer un análisis más exhaustivo, vamos a probar a mandarle peticiones a `http://192.168.10.50:80/` para poder comparar resultados. Para ello, simplemente indicamos esta dirección en el `docker-compose.yml`

```
+ Terminal
apache-benchmark-PS | Benchmarking 192.168.10.50 (be patient)
apache-benchmark-PS | Completed 1000 requests
apache-benchmark-PS | Completed 2000 requests
apache-benchmark-PS | Completed 3000 requests
apache-benchmark-PS | Completed 4000 requests
apache-benchmark-PS | Completed 5000 requests
apache-benchmark-PS | Completed 6000 requests
apache-benchmark-PS | Completed 7000 requests
apache-benchmark-PS | Completed 8000 requests
apache-benchmark-PS | Completed 9000 requests
apache-benchmark-PS | Completed 10000 requests
apache-benchmark-PS | Finished 10000 requests
apache-benchmark-PS |
apache-benchmark-PS | Server Software:      nginx/1.25.5
apache-benchmark-PS | Server Hostname:     192.168.10.50
apache-benchmark-PS | Server Port:         80
apache-benchmark-PS |
apache-benchmark-PS | Document Path:       /
apache-benchmark-PS | Document Length:     298 bytes
apache-benchmark-PS |
apache-benchmark-PS | Concurrency Level:    100
apache-benchmark-PS | Time taken for tests: 2.684 seconds
apache-benchmark-PS | Complete requests:    10000
apache-benchmark-PS | Failed requests:      0
apache-benchmark-PS | Keep-Alive requests:  9992
apache-benchmark-PS | Total transferred:    4839960 bytes
apache-benchmark-PS | HTML transferred:    2980000 bytes
apache-benchmark-PS | Requests per second:  3725.17 [#sec] (mean)
apache-benchmark-PS | Time per request:     26.844 [ms] (mean)
apache-benchmark-PS | Time per request:     0.268 [ms] (mean, across all concurrent requests)
apache-benchmark-PS | Transfer rate:         1760.71 [Kbytes/sec] received
apache-benchmark-PS |
apache-benchmark-PS | Connection Times (ms)
apache-benchmark-PS |   min  mean[+/-sd] median  max
apache-benchmark-PS | Connect:    0    1 32.5    0  2057
apache-benchmark-PS | Processing:  0    1  0.3    1    6
apache-benchmark-PS | Waiting:    0    1  0.3    1    5
apache-benchmark-PS | Total:      0    2 32.5    1  2058
apache-benchmark-PS |
apache-benchmark-PS | Percentage of the requests served within a certain time (ms)
apache-benchmark-PS |   50%    1
apache-benchmark-PS |   60%    1
apache-benchmark-PS |   75%    1
apache-benchmark-PS |   80%    1
apache-benchmark-PS |   90%    1
apache-benchmark-PS |   95%    2
apache-benchmark-PS |   98%    2
apache-benchmark-PS |   99%    2
apache-benchmark-PS |  100%  2058 (longest request)
apache-benchmark-PS |
apache-benchmark-PS | apache-benchmark-PS exited with code 0
danjel@Modern-15-B7M: ~/.../PS/PS-ab |main 7:1 A|
~
```

Análisis de los Resultados: Para empezar, comentar que este test no es muy realista, ya que parece un ataque DDoS y vemos como hemos tenido que deshabilitar configuraciones para probar nuestra web. Por otro lado, se puede apreciar como la configuración con HTTPS es mucho más pesada que la HTTP. No solo

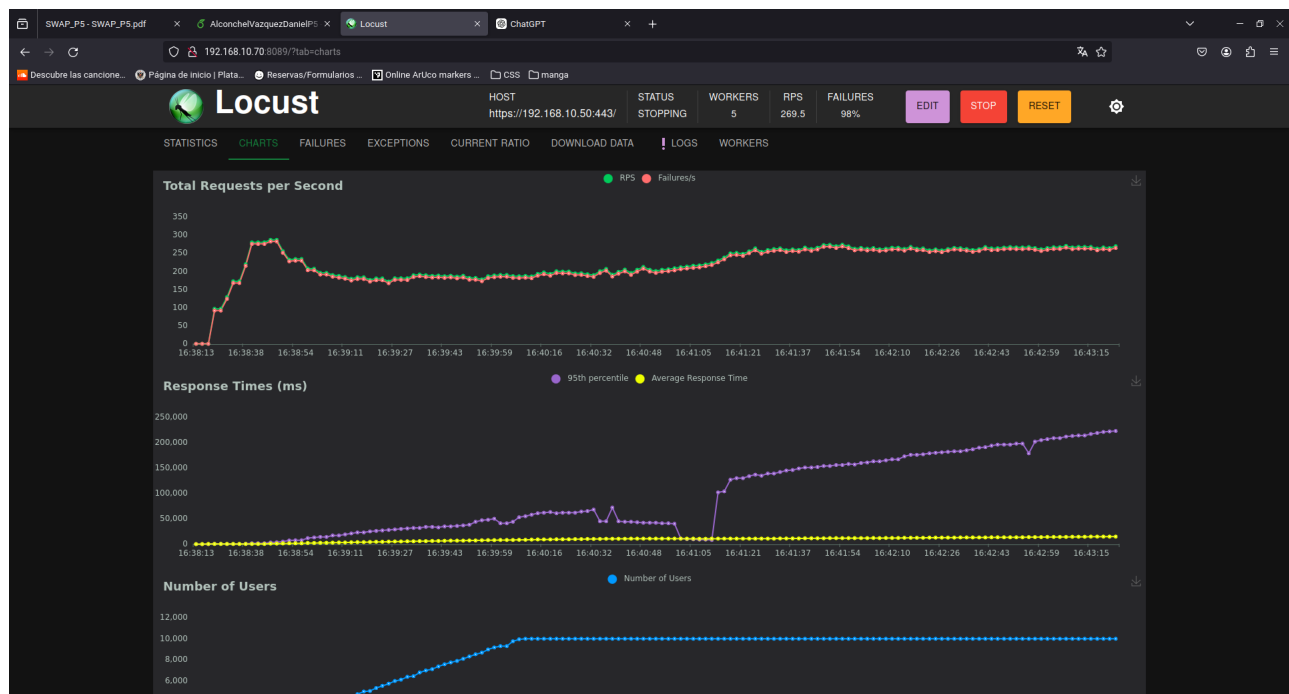
el tiempo medio por petición es más alto, sino que además las cabeceras que envía son más pesadas, ya que contienen información sobre las **cookies**, **sesión** y datos de la **secure layer** que hemos habilitado.

1.4.2. Locust

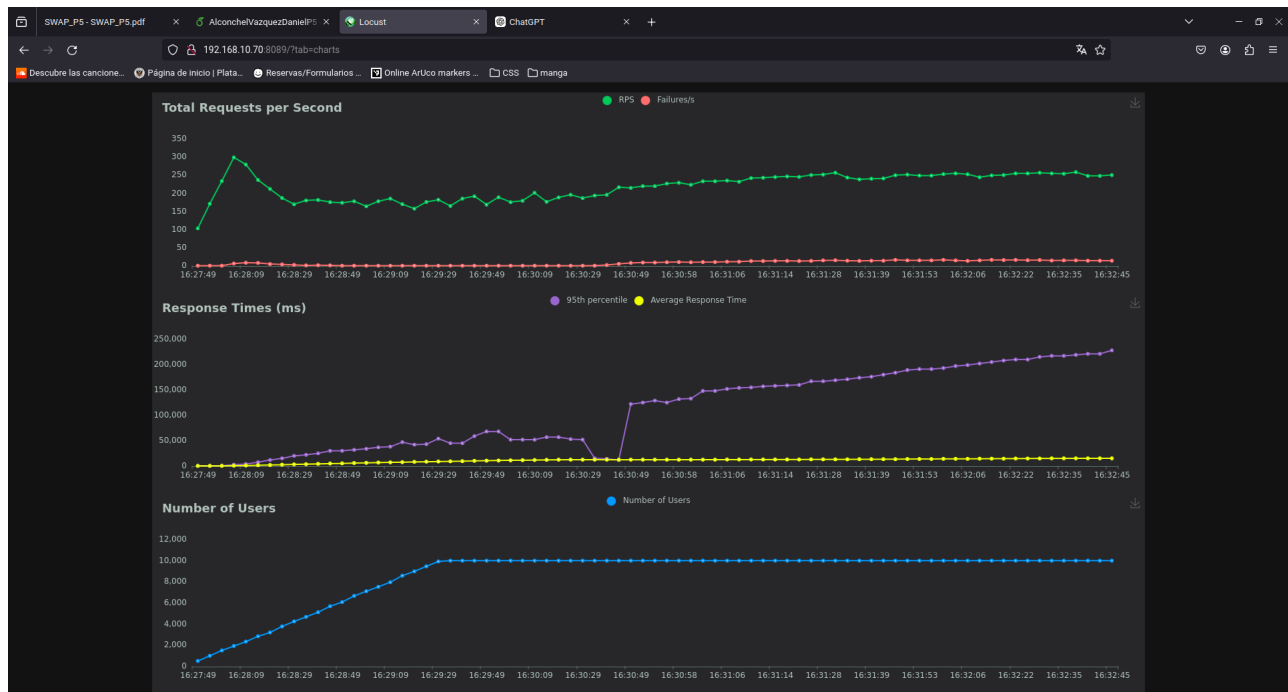
Repetimos ahora el proceso pero con **Locust**. A continuación adjunto algunas capturas bastante interesantes:

# Failures	Method	Name	Message
9719	GET	//index.php	"ConnectionResetError(104, 'Connection reset by peer')"
563473	GET	//index.php	"ConnectionRefusedError(111, 'Connection refused')"
1461	GET	//index.php	"RemoteDisconnected('Remote end closed connection without response')"
893	GET	//index.php	"SSLError(8, 'EOF occurred in violation of protocol (_ssl.c:2427)')"
3	GET	//index.php	"ConnectTimeoutError(curllib3.connection.HTTPSConnection object at 0x...: 'Connection to 192.168.10.50 timed out. (connect timeout=None)'"

En esta primera captura, vemos como debido a las políticas de seguridad de **iptables** y de límite de zonas de memoria que configuré en la práctica anterior, el test dá bastantes errores, ya que rechaza la conexión de los trabajadores. De hecho, podemos observar la gráfica:



Vamos a comentar la script de **iptables** y las zonas de memoria como en el ejercicio anterior. (**Nota:** No tengo montado un volumen para la script de **iptables** debido a como se ejecuta el **entrypoint** de la imagen, por lo que hay que reconstruir la misma):



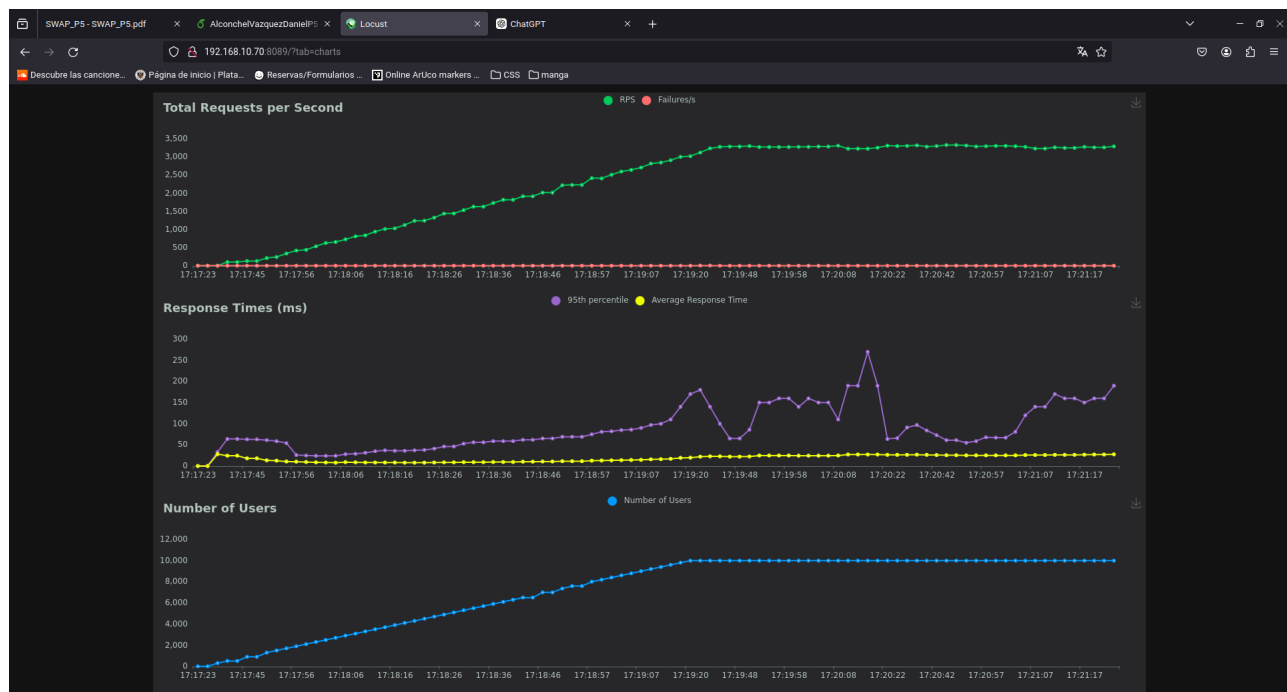
Vemos que los escasos fallos que da no son culpa del balanceador, sino porque los trabajadores alcanzan el límite máximo de cpu:

Logs

- [2024-05-23 14:27:29.106] 485a0c74c4f5/INFO/locust.main: Starting web interface at http://0.0.0.0:8089
- [2024-05-23 14:27:29.113] 485a0c74c4f5/INFO/locust.main: Starting Locust 2.26.0
- [2024-05-23 14:27:29.371] 485a0c74c4f5/INFO/locust.runners: Worker 6394ea1a51b9_eb2834be0e2c49d28d8d04ba17abd471 (index 0) reported as ready. 1 workers connected.
- [2024-05-23 14:27:29.556] 485a0c74c4f5/INFO/locust.runners: Worker f1629ab78eab_fed714bbc6974a30b609927153f45496 (index 1) reported as ready. 2 workers connected.
- [2024-05-23 14:27:29.782] 485a0c74c4f5/INFO/locust.runners: Worker 0d41977ef862_166e6a738c374da09be10123a10f26ad (index 2) reported as ready. 3 workers connected.
- [2024-05-23 14:27:29.949] 485a0c74c4f5/INFO/locust.runners: Worker 6a1e00d0a8e1_08d1c873039f46b78b574ed1b5a535ac (index 3) reported as ready. 4 workers connected.
- [2024-05-23 14:27:30.152] 485a0c74c4f5/INFO/locust.runners: Worker 8e0c6ea6894c_77fedf1cb2a241228711aeb89b0db832 (index 4) reported as ready. 5 workers connected.
- [2024-05-23 14:27:44.238] 485a0c74c4f5/INFO/locust.runners: Sending spawn jobs of 10000 users at 100.00 spawn rate to 5 ready workers
- [2024-05-23 14:28:00.274] 485a0c74c4f5/WARNING/locust.runners: Worker 6394ea1a51b9_eb2834be0e2c49d28d8d04ba17abd471 (index 0) exceeded cpu threshold (will only log this once per worker)
- [2024-05-23 14:28:00.423] 485a0c74c4f5/WARNING/locust.runners: Worker 6a1e00d0a8e1_08d1c873039f46b78b574ed1b5a535ac (index 3) exceeded cpu threshold (will only log this once per worker)
- [2024-05-23 14:28:00.512] 485a0c74c4f5/WARNING/locust.runners: Worker 0d41977ef862_166e6a738c374da09be10123a10f26ad (index 2) exceeded cpu threshold (will only log this once per worker)
- [2024-05-23 14:28:00.650] 485a0c74c4f5/WARNING/locust.runners: Worker f1629ab78eab_fed714bbc6974a30b609927153f45496 (index 1) exceeded cpu threshold (will only log this once per worker)
- [2024-05-23 14:28:00.867] 485a0c74c4f5/WARNING/locust.runners: Worker 8e0c6ea6894c_77fedf1cb2a241228711aeb89b0db832 (index 4) exceeded cpu threshold (will only log this once per worker)

Worker	State	# users	CPU usage	Memory usage
0e876c45e1ab_242578475f064b9ebd4a8c054b3f1522	running	2000	100.3	515.45 MB
1d53c0c0493a3_0e6f88ae794b4778b6ed6a798b09258	running	2000	100.2	483.88 MB
2df50b090933_94a71007edee47528415661428b501ef	running	2000	100.1	515.64 MB
84089242711d_754789d77c4f40afb9ebfd3a3lea406a	running	2000	100.2	519.62 MB
dac4c2b736ae_d19896cb41445329a07d4cdad57737c	running	2000	100.3	518.14 MB

Antes de sacar conclusiones, vamos a repetirlo pero mandando las peticiones a <http://192.168.10.50:80/>



Análisis de los datos: Como podemos observar, al ser una simple petición GET tampoco hay una diferencia abismal entre usar HTTP y HTTPS. Lo que si es cierto, que gracias a las mejoras que introducimos de uso de sesión y cookies, el tiempo de respuesta de HTTPS es más bajo a medida que aumenta el tiempo que transcurre haciendo peticiones. Debido a la simplicidad de nuestra web, no es una diferencia abismal, pero se puede notar como la curva del tiempo medio de respuesta en HTTP crece algo más rápido que en HTTPS.

2. Tareas Avanzadas

2.1. Desarrollar tareas avanzadas en Locustfile.py

Nota: Para este apartado he realizado primero la tarea siguiente, para saber como planificar el test.

Vamos a comenzar preguntándole a ChatGPT:

- **ChatGPT:** (Le paso el fichero de locust.py original). Quiero modificar el test de carga para que haga también consultas sql. Tengo una BD llamada usuarios donde almaceno nombre y apellidos. Quiero que se hagan también peticiones de inserción o de obtención de tuplas de la bd.
- **ChatGPT:**

1. Instalar un módulo para gestionar conexiones sql

```
pip install mysql-connector-python
```

2. Modificar el archivo de locust.py:

```
from locust import HttpUser, TaskSet, task, between
import mysql.connector
import random
```

```

class P5_danieeeeld2(TaskSet):
    def on_start(self):
        # Configurar la conexión a la base de datos MySQL
        self.conn = mysql.connector.connect(
            host="tu_host",
            user="tu_usuario",
            password="tu_contraseña",
            database="usuarios"
        )
        self.cursor = self.conn.cursor()

    def on_stop(self):
        # Cerrar la conexión a la base de datos al finalizar
        self.cursor.close()
        self.conn.close()

    @task
    def load_index(self):
        self.client.get("/index.php", verify=False)

    @task
    def insert_user(self):
        # Inserción de un usuario en la base de datos
        nombre = f"Nombre{random.randint(1, 1000)}"
        apellidos = f"Apellido{random.randint(1, 1000)}"
        query = "INSERT INTO usuarios
            (nombre, apellidos) VALUES (%s, %s)"
        self.cursor.execute(query, (nombre, apellidos))
        self.conn.commit()

    @task
    def select_users(self):
        # Obtención de tuplas de la base de datos
        self.cursor.execute("SELECT nombre,
            apellidos FROM usuarios LIMIT 10")
        rows = self.cursor.fetchall()
        for row in rows:
            print(f"Nombre: {row[0]}, Apellidos: {row[1]}")

class P5_usuarios(HttpUser):
    tasks = [P5_danieeeeld2]
    wait_time = between(1, 5)

```

Análisis de la salida: La salida proporcionada por la IA es prácticamente correcta, salvo que tenemos que definir todas las tareas como `@task` para que funcione, por lo que, haciendo los cambios convenientes, nos queda como:

```

from locust import HttpUser, TaskSet, task, between
import mysql.connector
import random

```

```

class P5_danieeeld2(TaskSet):
    @task
    def load_index(self):
        self.client.get("/index.php", verify=False)

    @task
    def load_web(self):
        self.client.get("/webAvanzada.php", verify=False)

    @task
    def insert_user(self):
        # Configurar la conexion a la base de datos MySQL
        self.conn = mysql.connector.connect(
            host="192.168.20.50",
            user="wordpress",
            password="wordpress",
            database="wordpress"
        )
        self.cursor = self.conn.cursor()
        # Insercion de un usuario en la base de datos
        nombre = f"Nombre{random.randint(1, 1000)}"
        apellidos = f"Apellido{random.randint(1, 1000)}"
        query = "INSERT INTO usuarios (nombre, apellidos) VALUES (%s, %s)"
        self.cursor.execute(query, (nombre, apellidos))
        self.conn.commit()
        # Cerrar la conexion a la base de datos al finalizar
        self.cursor.close()
        self.conn.close()

    @task
    def select_users(self):
        # Configurar la conexion a la base de datos MySQL
        self.conn = mysql.connector.connect(
            host="192.168.20.50",
            user="wordpress",
            password="wordpress",
            database="wordpress"
        )
        self.cursor = self.conn.cursor()
        # Obtencion de tuplas de la base de datos
        self.cursor.execute("SELECT nombre, apellidos FROM usuarios LIMIT 10")
        rows = self.cursor.fetchall()
        for row in rows:
            print(f"Nombre: {row[0]}, Apellidos: {row[1]}")
        # Cerrar la conexion a la base de datos al finalizar
        self.cursor.close()
        self.conn.close()

class P5_usuarios(HttpUser):
    tasks = [P5_danieeeld2]
    wait_time = between(1, 5)

```

Una vez creado el nuevo `locust.py`, tenemos que modificar el `docker-compose.yml` para que se conecte a la red-servicios y pueda hacer estas peticiones a la BD. Además, necesitamos instalar `mysql-connector-python`.

1. Creación del nuevo DockerfileLocust:

```
FROM locustio/locust
```

```
RUN pip install mysql-connector-python
```

2. Nuevo docker-compose.yml:

```
services:
  master-danieeeld2:
    build:
      context: .
      dockerfile: DockerfileLocust
    image: danieeeld2-locust-image:P5
    ports:
      - "8089:8089"
    volumes:
      - ./:/mnt/locust
    command: -f /mnt/locust/locustfile.py --master
              -H https://192.168.10.50:443/
    networks:
      p5-granjaweb_red_web:
        ipv4_address: 192.168.10.70
      p5-granjaweb_red_servicios:
        ipv4_address: 192.168.20.70

  worker-danieeeld2:
    build:
      context: .
      dockerfile: DockerfileLocust
    image: danieeeld2-locust-image:P5
    volumes:
      - ./:/mnt/locust
    command: -f /mnt/locust/locustfile.py --worker
              --master-host=master-danieeeld2
    depends_on:
      - master-danieeeld2
    deploy:
      replicas: 5
    networks:
      p5-granjaweb_red_web:
      p5-granjaweb_red_servicios:

networks:
  p5-granjaweb_red_web:
    external: true
  p5-granjaweb_red_servicios:
    external: true
```

Con todo esto, ya esta listo para usar el nuevo test de carga.

2.2. Crear escenarios multicontenedor con algún CMS

Vamos a comenzar preguntándole a ChatGPT:

- **Yo:** Tengo la siguiente granja web configurada en un escenario docker (*Le paso el archivo docker-compose de la granja web*). ¿Cómo puedo modificar el escenario de la granja web para usar un CMS como drupal o wordpress?

- **ChatGPT:**

1. Crear un contenedor para la base de datos:

```
db:
  image: mysql:5.7
  container_name: db
  volumes:
    - db_data:/var/lib/mysql
  networks:
    - red_servicios
  environment:
    MYSQL_ROOT_PASSWORD: your_root_password
    MYSQL_DATABASE: wordpress
    MYSQL_USER: wordpress_user
    MYSQL_PASSWORD: wordpress_password
```

2. Crear contenedores para WordPress:

```
wordpress1:
  image: wordpress:latest
  container_name: wordpress1
  depends_on:
    - db
  cap_add:
    - NET_ADMIN
  volumes:
    - ./web_danieeeeld2:/var/www/html
  environment:
    WORDPRESS_DB_HOST: db:3306
    WORDPRESS_DB_USER: wordpress_user
    WORDPRESS_DB_PASSWORD: wordpress_password
    WORDPRESS_DB_NAME: wordpress
  networks:
    red_web:
      ipv4_address: 192.168.10.2
    red_servicios:
      ipv4_address: 192.168.20.2

wordpress2:
  image: wordpress:latest
  container_name: wordpress2
  depends_on:
```

```

    - db
cap_add:
    - NET_ADMIN
volumes:
    - ./web_danieeeeld2:/var/www/html
environment:
    WORDPRESS_DB_HOST: db:3306
    WORDPRESS_DB_USER: wordpress_user
    WORDPRESS_DB_PASSWORD: wordpress_password
    WORDPRESS_DB_NAME: wordpress
networks:
    red_web:
        ipv4_address: 192.168.10.3
    red_servicios:
        ipv4_address: 192.168.20.3

# Repite para otros contenedores de WordPress
# como wordpress3, wordpress4, etc.
# ...

```

Análisis de la salida: Me parece una respuesta acertada, ya que si observamos sigue exactamente la misma estructura que propone *Wordpress* en su página de DockerHub.

Para realizar este apartado, vamos a rescatar la **script** de creación de **docker-compose** de prácticas anteriores y vamos a modificarla de la siguiente manera:

```

#!/bin/bash

if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <number_of_instances>"
    exit 1
fi

num_instances=$1

cat <<EOF >docker-compose.wordpress.yml
version: '4.0'

services:
    db:
        image: mysql:8.0
        container_name: db
        environment:
            MYSQL_RANDOM_ROOT_PASSWORD: root
            MYSQL_DATABASE: wordpress
            MYSQL_USER: wordpress
            MYSQL_PASSWORD: wordpress
        networks:
            red_servicios:
                ipv4_address: 192.168.20.50
EOF

for ((i=1; i<=$num_instances; i++)); do
    cat <<EOF >>docker-compose.wordpress.yml
web$i:
    image: wordpress:latest

```

```

    container_name: web$i
    cap_add:
      - NET_ADMIN
    volumes:
      - ./web_danieeeld2:/var/www/html
      - ./P4-danieeeld2-certificados:/etc/apache2/ssl
      # - ./P4-danieeeld2-certificados/certificado_subCA.crt:/etc/apache2/ssl/
      certificado_danieeeld2.crt
      # - ./P4-danieeeld2-certificados/certificado_subCA.key:/etc/apache2/ssl/
      certificado_danieeeld2.key
      - ./P4-danieeeld2-apache/danieeeld2-apache-ssl.conf:/etc/apache2/sites-available
      /danieeeld2-apache-ssl.conf
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
    networks:
      red_web:
        ipv4_address: 192.168.10.$((i+1))
      red_servicios:
        ipv4_address: 192.168.20.$((i+1))

```

EOF

done

```

cat <<EOF >>docker-compose.wordpress.yml
balanceador-nginx:
  image: danieeeld2-nginx-image:p4
  container_name: balanceador-nginx
  cap_add:
    - NET_ADMIN
  ports:
    - "80:80"
    - "443:443"
  command: ['nginx', '-g', 'daemon off;']
  volumes:
    - ./P4-danieeeld2-nginx/danieeeld2-nginx-ssl.conf:/etc/nginx/nginx.conf
    - ./P4-danieeeld2-certificados:/etc/nginx/ssl
    # - ./P4-danieeeld2-certificados/certificado_subCA.crt:/etc/nginx/ssl/
    certificado_danieeeld2.crt
    # - ./P4-danieeeld2-certificados/certificado_subCA.key:/etc/nginx/ssl/
    certificado_danieeeld2.key
  networks:
    red_web:
      ipv4_address: 192.168.10.50
  depends_on:

```

EOF

```

for ((i=1; i<=$num_instances; i++)); do
  echo "      - web$i" >>docker-compose.wordpress.yml
done

```

```

cat <<EOF >>docker-compose.wordpress.yml

```

```

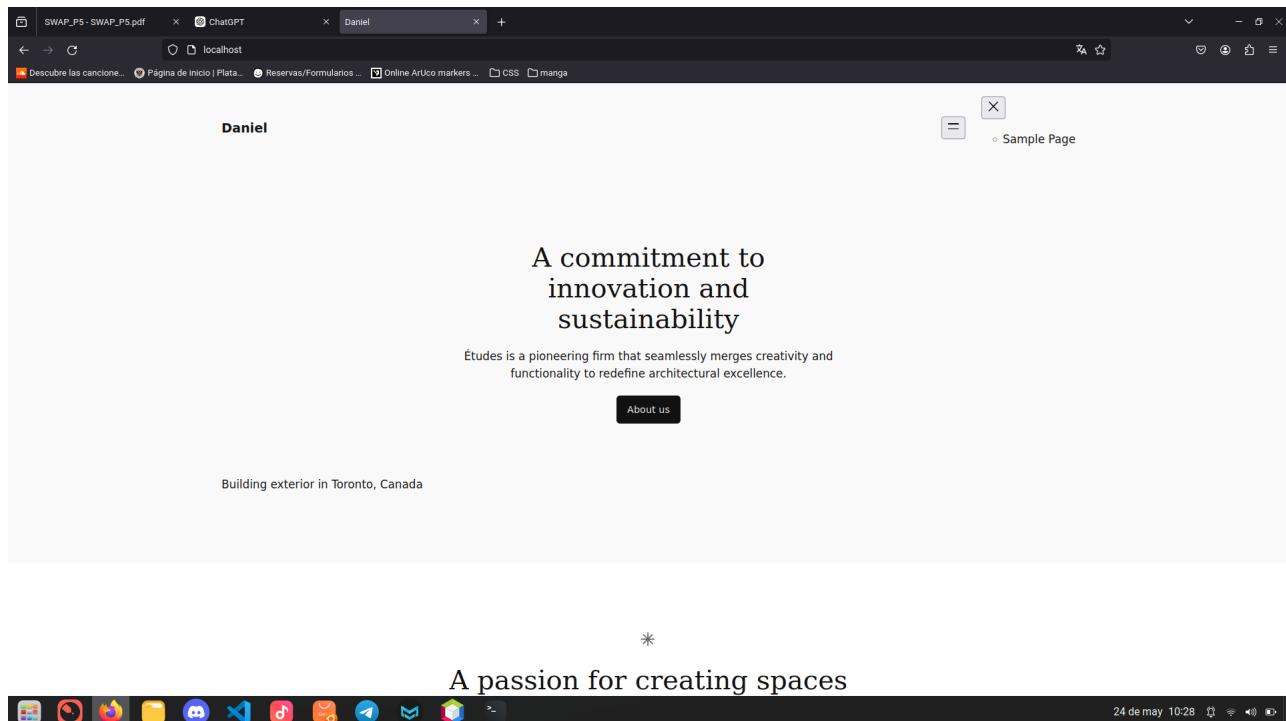
networks:
  red_web:
    driver: bridge
  ipam:
    config:
      - subnet: 192.168.10.0/24

```

```
red_servicios:
  driver: bridge
  ipam:
    config:
      - subnet: 192.168.20.0/24
EOF
```

The screenshot shows a terminal window on the left and a web browser on the right. The terminal displays the output of running `docker compose -f docker-compose.wordpress.yml up`. It shows the creation of several containers: `Network p5-granjaweb_red_servicios`, `Network p5-granjaweb_red_web`, `Container db`, `Container web1`, `Container web4`, `Container web5`, `Container web6`, `Container web3`, `Container web2`, and `Container balanceador-nginx`. The terminal also shows the initialization of the MySQL database and the Apache web server. The web browser on the right shows the WordPress installation progress page, with the title `SWAP-danieeeld2` and the Apache server IP address `192.168.10.5`.

Como vemos, sigue funcionando correctamente. A continuación lo que voy a hacer es desactivar el volumen para que pueda acceder al proceso de instalación de wordpress:



Como observamos, ya tendríamos una web más avanzada. El problema es que estos datos no son persistentes al reiniciar el escenario, por lo que vamos a crear un nuevo fichero **php** que se encargue de crear una tabla básica en la bd y que genere un formulario para insertar datos y que permita consultar las tuplas insertadas.

Como siempre, vamos a comenzar preguntándole a ChatGPT:

- **Yo:** Crea un fichero php que establezca una conexión a una bd, una vez establecida la conexión, comprueba si esta creada la tabla usuarios (dicha tabla contiene dos columnas, una para el nombre y otra para apellidos). En caso de no estar creada la tabla, debe crearla. Una vez creada la tabla, genera un formulario que permite insertar tuplas en la tabla y, acto seguido, las muestra abajo del formulario.
- **ChatGPT:**

```
<?php
// Datos de conexión a la base de datos
$servername = "localhost";
$username = "usuario";
$password = "contraseña";
$dbname = "nombre_bd";

// Crear conexión
$conn = new mysqli($servername, $username, $password, $dbname);

// Verificar conexión
if ($conn->connect_error) {
    die("Conexión fallida: " . $conn->connect_error);
}

// Verificar si la tabla usuarios existe
$table_check = "SELECT 1 FROM usuarios LIMIT 1";
$table_result = $conn->query($table_check);

if (!$table_result) {
    // Si la tabla no existe, crearla
    $create_table_query = "CREATE TABLE usuarios (
        id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
        nombre VARCHAR(30) NOT NULL,
        apellidos VARCHAR(30) NOT NULL
    )";

    if ($conn->query($create_table_query) === TRUE) {
        echo "Tabla usuarios creada exitosamente.<br>";
    } else {
        echo "Error al crear la tabla: " . $conn->error;
    }
}

// Procesar formulario de inserción
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $nombre = $_POST["nombre"];
```

```

$apellidos = $_POST["apellidos"];

// Insertar datos en la tabla
$insert_query = "INSERT INTO usuarios (nombre, apellidos)
VALUES ('$nombre', '$apellidos')";

if ($conn->query($insert_query) === TRUE) {
    echo "Datos insertados correctamente.<br>";
} else {
    echo "Error al insertar datos: " . $conn->error;
}
}

// Mostrar formulario de inserción
?>
<form method="post" action="<?php echo $_SERVER["PHP_SELF"]; ?>">
    Nombre: <input type="text" name="nombre"><br>
    Apellidos: <input type="text" name="apellidos"><br>
    <input type="submit" value="Insertar">
</form>

<?php
// Mostrar datos de la tabla usuarios
$select_query = "SELECT * FROM usuarios";
$result = $conn->query($select_query);

if ($result->num_rows > 0) {
    echo "<h2>Datos de la tabla usuarios:</h2>";
    echo "<table border='1'><tr><th>ID</th><th>Nombre</th>
        <th>Apellidos</th></tr>";
    while ($row = $result->fetch_assoc()) {
        echo "<tr><td>" . $row["id"] . "</td><td>" . $row["nombre"] .
            "</td><td>" . $row["apellidos"] . "</td></tr>";
    }
    echo "</table>";
} else {
    echo "No hay datos en la tabla usuarios.";
}

// Cerrar conexión
$conn->close();
?>

```

Análisis de la salida: La salida propuesta es en general correcta, aunque vamos a realizar algunos cambios de estructuración y comprobaciones, además de darle algo de formato:

```

<?php
$conexion = new mysqli("192.168.20.50", "wordpress", "wordpress", "wordpress");

// Verificar la conexion
if ($conexion->connect_error) {

```

```

        die("Error de conexion: " . $conexion->connect_error);
    }

    $salida = "";

    // Verificar si la tabla existe, si no existe, crearla
    $tabla = "usuarios";
    $verificar_tabla = $conexion->query("SHOW TABLES LIKE '$tabla'");
    if($verificar_tabla->num_rows == 0) {
        // La tabla no existe, entonces la creamos
        $crear_tabla = "CREATE TABLE $tabla (
            id INT(6) UNSIGNED AUTOINCREMENT PRIMARY KEY,
            nombre VARCHAR(30) NOT NULL,
            apellidos VARCHAR(30) NOT NULL
        )";

        if ($conexion->query($crear_tabla) == TRUE) {
            $salida = "Tabla creada exitosamente";
        } else {
            $salida = "Error al crear la tabla: " . $conexion->error;
        }
    }

    if(isset($_POST['nombre']) && isset($_POST['apellidos'])) {
        $nombre = $_POST['nombre'];
        $apellidos = $_POST['apellidos'];

        $sql = "INSERT INTO $tabla (nombre, apellidos) VALUES ('$nombre', '$apellidos')";
        if ($conexion->query($sql) == TRUE) {
            $salida = "Registro insertado exitosamente";
        } else {
            $salida = "Error al insertar registro: " . $conexion->error;
        }
    }

    $resultado = $conexion->query("SELECT * FROM usuarios");
    ?>

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background-color: #f4f4f4;
        }

        form {
            background-color: #fff;
            padding: 20px;

```

```

        border-radius: 5px;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }

    form input[type="text"],
    form input[type="submit"] {
        width: 100%;
        padding: 10px;
        margin-bottom: 10px;
        border: 1px solid #ccc;
        border-radius: 4px;
        box-sizing: border-box;
    }

    form input[type="submit"] {
        background-color: #4caf50;
        color: white;
        cursor: pointer;
    }

    form input[type="submit"]:hover {
        background-color: #45a049;
    }
    table {
        width: 50%;
        margin: 0 auto;
        border-collapse: collapse;
    }

    th, td {
        padding: 8px;
        text-align: left;
        border-bottom: 1px solid #ddd;
    }

    th {
        background-color: #f2f2f2;
    }
</style>
</head>

<body>
    <form method="post" action="">
        Nombre: <input type="text" name="nombre"><br>
        Apellidos: <input type="text" name="apellidos"><br>
        <input type="submit" value="Insertar">
    </form>

    <?php
        echo "<table>";
        echo "<tr><th>Nombre</th><th>Apellidos</th></tr>";

        while ($fila = $resultado->fetch_assoc()) {
            echo "<tr>";
            echo "<td>" . $fila['nombre'] . "</td>";
            echo "<td>" . $fila['apellidos'] . "</td>";
            echo "</tr>";
        }

        echo "<tr><td colspan='2'>Direccion IP del servidor: " . $_SERVER['SERVER_ADDR'] . "</td></tr>";
    </?php>

```

```

        if (isset($salida) && $salida != "") {
            echo "<tr><td colspan='2'>$salida</td></tr>";
        }
        echo "</table>";
    ?>

    <footer>
        <?php echo $salida; ?>
    </footer>

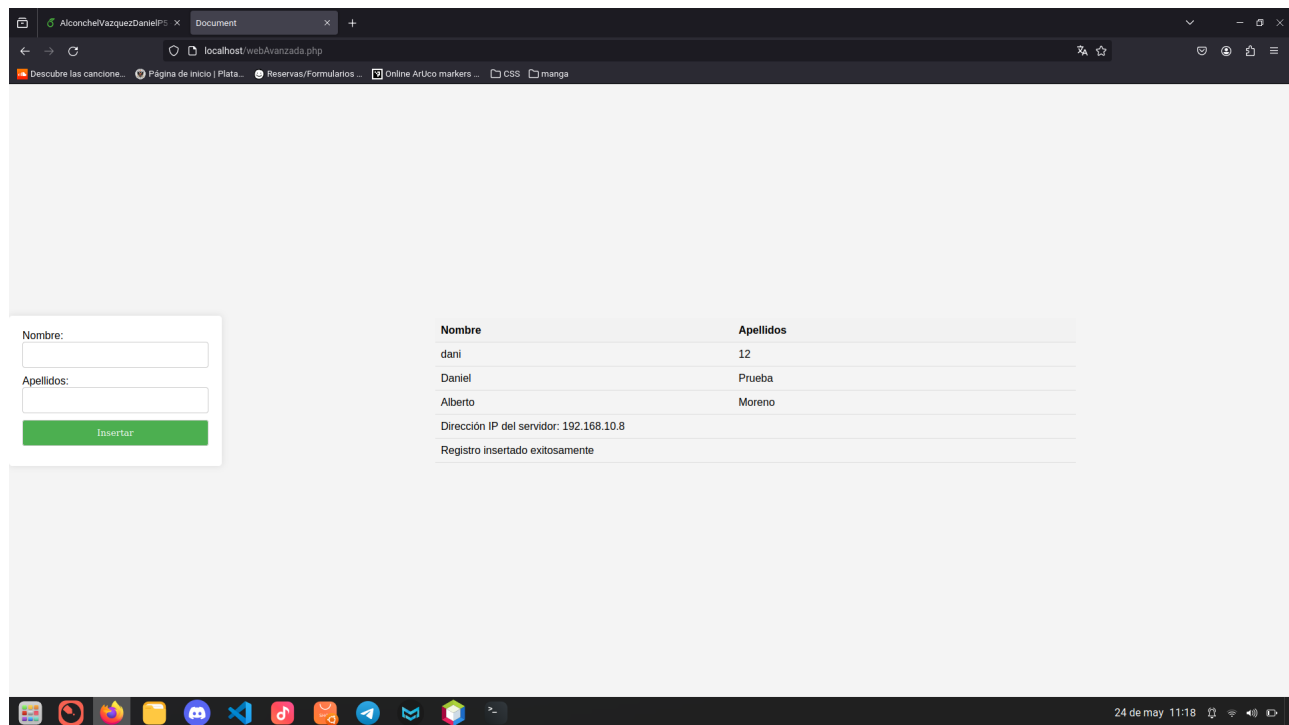
</body>

</html>

<?php
$conexion->close();
?>

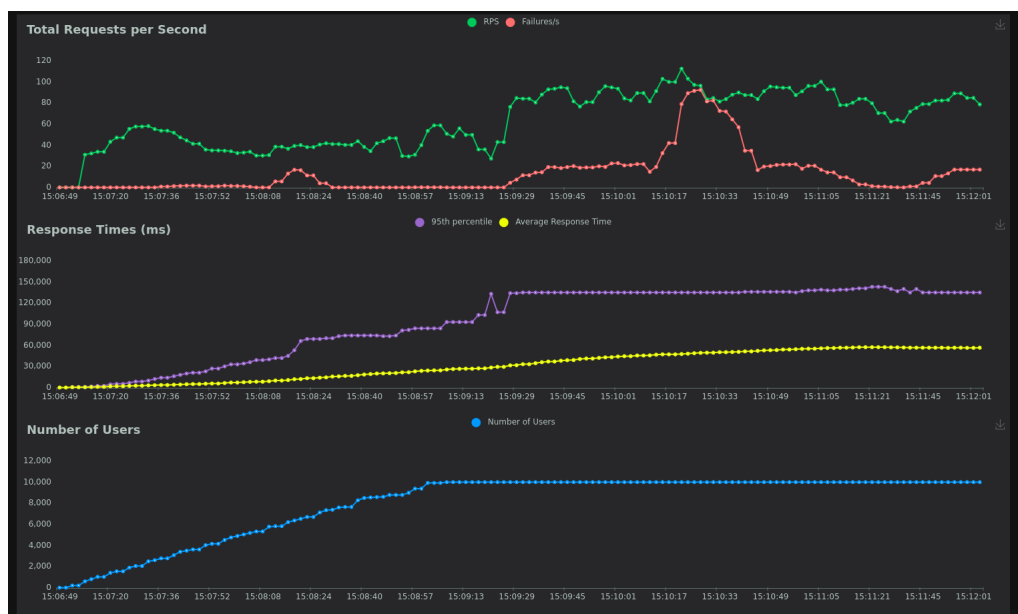
```

Finalmente, si accedemos a esta página como `localhost/webAvanzada.php` (ya sea usando `http` o `https`), veremos lo siguiente:



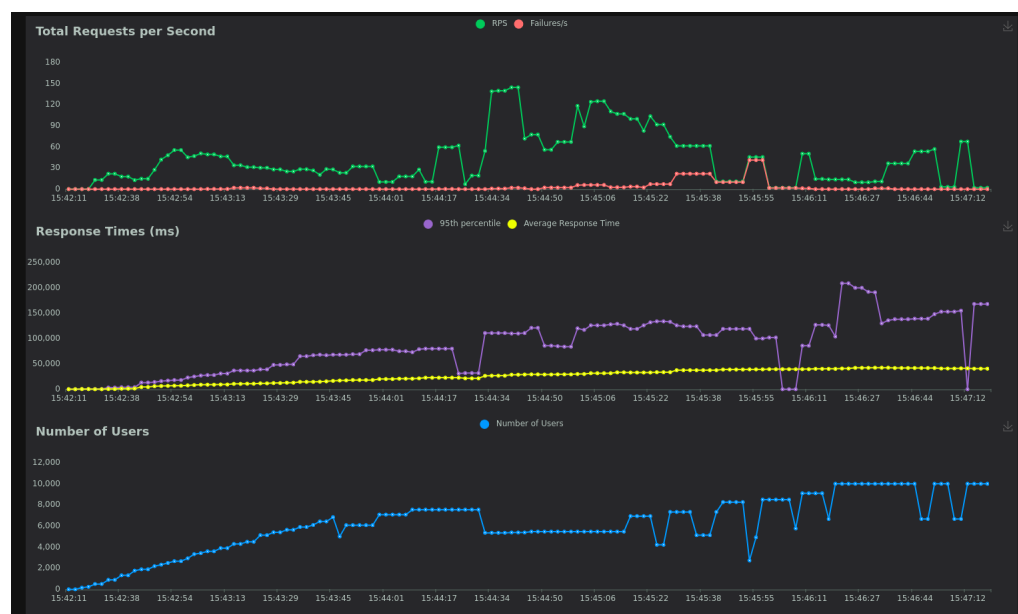
2.3. Ejecución y análisis de carga sobre CMS

Como siempre, he desconectado las configuraciones avanzadas de seguridad del servidor `HTTPs` para que funcione de forma correcta el test. Si ejecutamos el nuevo test de carga para este servidor, obtenemos los siguientes datos:



# Failures	Method	Name	Message
25	GET	/index.php	"RemoteDisconnected('Remote end closed connection without response')"
780	GET	/index.php	"SSL_ERROR(SSLError(8, '[SSL: UNEXPECTED_EOF_WHILE_READING] EOF occurred in violation of protocol (_ssl.c:1006)'))"
1	GET	/index.php	"ConnectionResetError(104, 'Connection reset by peer')"
1218	GET	/index.php	"ConnectTimeoutError(curllib3.connection.HTTPSConnection object at 0x...: 'Connection to 192.168.10.50 timed out. (connect timeout=None)'"
23	GET	/webAvanzada.php	"RemoteDisconnected('Remote end closed connection without response')"
815	GET	/webAvanzada.php	"SSL_ERROR(SSLError(8, '[SSL: UNEXPECTED_EOF_WHILE_READING] EOF occurred in violation of protocol (_ssl.c:1006)'))"
1168	GET	/webAvanzada.php	"ConnectTimeoutError(curllib3.connection.HTTPSConnection object at 0x...: 'Connection to 192.168.10.50 timed out. (connect timeout=None)'"

Antes de analizarlo, vamos a probar con el servidor HTTP:



# Failures	Method	Name	Message
822	GET	//index.php	"RemoteDisconnected(Remote end closed connection without response)"
4	GET	//index.php	"ConnectionResetError(104, 'Connection reset by peer')"
858	GET	//webAvanzada.php	"RemoteDisconnected(Remote end closed connection without response)"
5	GET	//webAvanzada.php	"ConnectionResetError(104, 'Connection reset by peer')"

Análisis de las gráficas: Parte de las peticiones fallidas es porque los trabajadores alcanzan el máximo de consumo de cpu permitido. Podemos observar como, pasado un tiempo, las peticiones **HTTPS** tiende a estabilizarse más que las realizadas con **HTTP**, aunque el consumo medio de ambas es bastante parecida. En general los fallos que han presentado ambas es por la saturación de la cpu y al pedir las páginas, mientras que en la gestión de la BD no ha habido fallos.