UNIVERSIDAD DE GRANADA

Servidores Web de Altas Prestaciones

Práctica 5: Benchmarking

Miguel Torres Alonso

Mayo, 2024

Índice

1. Tareas Básicas - B1: Configuración del entorno de benchmarking

Para este apartado vamos a crear la estructura de directorios y archivos necesaria que se nos pide:

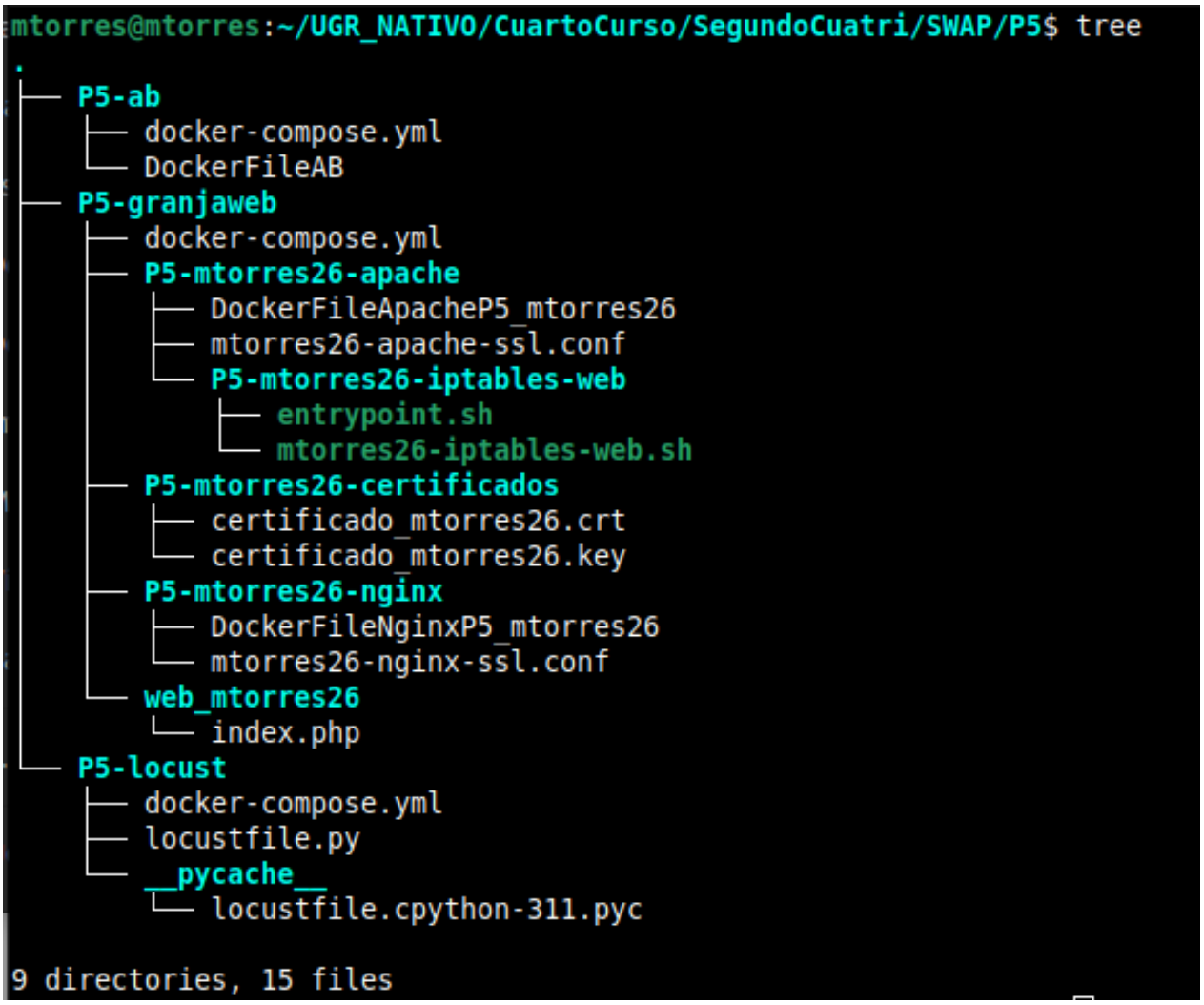

Figura 1: Estructura de árbol de directorios de esta práctica.

2. Tareas Básicas - B2: Implementación con Apache Benchmark

2.1. Consulta realizada a la IA:

Genera el código de un Dockerfile que use Apache Benchmark.

2.2. Respuesta de la IA:

```
FROM apache/httpd:2.4.48

```

```
RUN apt-get update && apt-get install -y apache2-utils

WORKDIR /usr/local/bin

COPY ab /usr/local/bin/

CMD ["ab"]
```

2.3. Análisis detallado:

La IA nos proporciona un Dockerfile semi-correcto, pues usa una imagen errónea de apache/httpd que no es necesaria para este apartado y fija directorios de trabajo que tampoco son necesarios, por tanto vamos a usar el DockerfileAB proporcionado en el guión que, además, es más simple.

Realizamos la creación del Dockerfile (listing 1) que contendrá el contenedor de Apache Benchmark y que se nos proporciona en el guión:

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

Listing 1: DockerFileAB.

2.4. Consulta realizada a la IA:

Genera el código de un docker-compose.yml que utilize una imagen con Apache Benchmark instalado, una dirección IP 192.168.10.60 y que al ejecutarse realice 10.000 peticiones con concurrencia de 100 a la dirección 192.168.10.50.

2.5. Respuesta de la IA:

```
version: "3.8"

services:
  ab:
    image: ab
    network_mode: host
    command: ["-c", "100", "-n", "10000", "http://192.168.10.50/"]
```

Listing 2: Respuesta de la IA.

2.6. Análisis detallado:

La IA ha acertado en el comando necesario para realizar las peticiones al balanceador correctamente pero lo demás no es correcto, ni siquiera ha proporcionado la configuración para que el AB tenga de dirección IP lo que le hemos pedido en la consulta. Procedemos entonces a usar el docker-compose.yml del guión.

El docker-compose.yml encargado de levantar el servicio con Apache Benchmark tiene el siguiente aspecto:

```
services:
  apache-benchmark-P5:
    image: mtorres26-ab-image:p5
    container_name: apache-benchmark-P5
    build:
      context: .
      dockerfile: DockerFileAB
    command:
      [
        "ab",
```

```
        "-n",
        "10000",
        "-c",
        "100",
        "https://192.168.10.50:443/"
      ]
    networks:
      red_web:
        ipv4_address: 192.168.10.60

networks:
  red_web:
    external: true
```

Listing 3: docker-compose.yml para el AB.

Si accedemos al log del servicio de Apache Benchmark, observamos los resultados de su ejecución como vemos en las siguientes figuras (2 y 3).

```
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P5/P5-ab$ docker logs apache-benchmark-P5
This is ApacheBench, Version 2.3 <$Revision: 1913912 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.10.50 (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests


Server Software:        nginx/1.25.5
Server Hostname:        192.168.10.50
Server Port:            443
SSL/TLS Protocol:       TLSv1.3,TLS_AES_256_GCM_SHA384,2048,256
Server Temp Key:        X25519 253 bits

Document Path:          /
Document Length:        299 bytes

Concurrency Level:      100
Time taken for tests:   14.504 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      4800000 bytes
HTML transferred:       2990000 bytes
Requests per second:    689.48 [#/sec] (mean)
Time per request:       145.037 [ms] (mean)
Time per request:       1.450 [ms] (mean, across all concurrent requests)
```

Figura 2: Log del servicio de Apache Benchmark.

```
Concurrency Level:      100
Time taken for tests:   14.504 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      4800000 bytes
HTML transferred:       2990000 bytes
Requests per second:    689.48 [#/sec] (mean)
Time per request:       145.037 [ms] (mean)
Time per request:       1.450 [ms] (mean, across all concurrent requests)
Transfer rate:          323.19 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        4  136  19.0    132     198
Processing:     5    8   8.0      7     108
Waiting:        4    8   7.7      7     106
Total:         16  144  17.7    139     209

Percentage of the requests served within a certain time (ms)
  50%    139
  66%    142
  75%    145
  80%    148
  90%    162
  95%    192
  98%    203
  99%    206
 100%    209 (longest request)
```

Figura 3: Log del servicio de Apache Benchmark.

3. Tareas Básicas - B3: Implementación con Locust

3.1. Consulta realizada a la IA:

Genera el código de un docker-compose.yml que un servicio master con la imagen oficial de Locust que actuará como nodo maestro y usará el archivo locustfile.py. Otro servicio worker con 5 replicas y que sigan las órdenes del servicio master

3.2. Respuesta de la IA:

```yaml
version: "3.8"

services:
  master:
    image: locustio/locust
    ports:
      - "8089:8089"
    volumes:
      - ./locustfile.py:/mnt/locust/locustfile.py
    command: -f /mnt/locust/locustfile.py --master
  worker:
    image: locustio/locust
    volumes:
      - ./locustfile.py:/mnt/locust/locustfile.py
    depends_on:
      - master
    command: -f /mnt/locust/locustfile.py --worker --master-host master
    scale: 5
```

Listing 4: Respuesta de la IA.

3.3. Análisis detallado:

La IA proporciona una respuesta casi correcta, a pesar de que hemos obviado ciertos datos como la direccón IP para que la consulta no sea especialmente larga. Sin embargo, falla en el comando replicas". Lo demás es muy similar a lo proporcionado en el guión.

El docker-compose.yml necesario para ejecutar Locust con un nodo master y múltiples workers nos lo proporcionan en el guión y es el siguiente:

```yaml
services:
  master-mtorres26:
    image: locustio/locust
    ports:
      - "8089:8089"
    volumes:
      - ./:/mnt/locust
    command: -f /mnt/locust/locustfile.py --master -H https://192.168.10.50:443/
    networks:
      red_web:
        ipv4_address: 192.168.10.70

  worker-mtorres26:
    image: locustio/locust
    volumes:
      - ./:/mnt/locust
    command: -f /mnt/locust/locustfile.py --worker --master-host master-mtorres26
    depends_on:
      - master-mtorres26
    deploy:
      replicas: 5
    networks:
      - red_web

networks:
  red_web:
    external: true
```

Listing 5: docker-compose.yml para Locust.

Para poder realizar peticiones HTTP y HTTPS con Locust es necesario un archivo *locustfile.py*:

```python
from locust import HttpUser, TaskSet, task, between

class P5_mtorres26(TaskSet):
    @task
    def load_index(self):
        self.client.get("/index.php", verify=False)

class P5_usuarios(HttpUser):
    tasks = [P5_mtorres26]
    wait_time = between(1, 5)
```

Listing 6: locustfile.py

En la figura 4 vemos el log del nodo máster de Locust.

```
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P5/P5-locust$ docker logs p5-locust-master-mtorres26-1
[2024-05-27 08:47:46,551] 5de36cca62f2/INFO/locust.main: Starting web interface at http://0.0.0.0:8089
[2024-05-27 08:47:46,575] 5de36cca62f2/INFO/locust.main: Starting Locust 2.28.0
[2024-05-27 08:47:47,019] 5de36cca62f2/INFO/locust.runners: Worker 3a3c64bcbed3_24eb84636c3648da80cb8eca3674a1a8 (index 0) reported as ready. 1 workers connected.
[2024-05-27 08:47:47,360] 5de36cca62f2/INFO/locust.runners: Worker 632aa57c254e_63d202d6e86d460086ef25ca90c3960c (index 1) reported as ready. 2 workers connected.
[2024-05-27 08:47:47,617] 5de36cca62f2/INFO/locust.runners: Worker 475d09c97e49_ae67596681704307835354dcaecc3ec8 (index 2) reported as ready. 3 workers connected.
[2024-05-27 08:47:47,886] 5de36cca62f2/INFO/locust.runners: Worker f4dc8c81716d_231fe1f496c44630b39649864edb7ce9 (index 3) reported as ready. 4 workers connected.
[2024-05-27 08:47:48,149] 5de36cca62f2/INFO/locust.runners: Worker 3fa24db6b0ed_8cf40e4d5ab048f781675fdd3e751ef8 (index 4) reported as ready. 5 workers connected.
```

Figura 4: Log del nodo máster de Locust.

4. Tareas Básicas - B4: Ejecución de pruebas de carga

Ahora queda la ejecución de la granja web y la monitorización de la misma con las dos herramientas que hemos configurado.

En la figura 5 observamos la granja web desplegada correctamente.

```
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P5/P5-granjaweb$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS
     NAMES
b2999d30781d   mtorres26-nginx-image:p5       "/docker-entrypoint.…"   5 seconds ago   Up 2 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp, :::443->443/t
cp   balanceador-nginx
e8fe3236247b   mtorres26-apache-image:p5      "/entrypoint.sh apac…"   5 seconds ago   Up 3 seconds   80/tcp, 443/tcp
     web5
ef9bec02b725   mtorres26-apache-image:p5      "/entrypoint.sh apac…"   5 seconds ago   Up 3 seconds   80/tcp, 443/tcp
     web4
42aaeb72252d   mtorres26-apache-image:p5      "/entrypoint.sh apac…"   5 seconds ago   Up 3 seconds   80/tcp, 443/tcp
     web7
228540e0dfcd   mtorres26-apache-image:p5      "/entrypoint.sh apac…"   5 seconds ago   Up 3 seconds   80/tcp, 443/tcp
     web1
7bbd50b0673c   mtorres26-apache-image:p5      "/entrypoint.sh apac…"   5 seconds ago   Up 3 seconds   80/tcp, 443/tcp
     web2
f9aff9e83948   mtorres26-apache-image:p5      "/entrypoint.sh apac…"   5 seconds ago   Up 3 seconds   80/tcp, 443/tcp
     web8
5962f26fc8fc   mtorres26-apache-image:p5      "/entrypoint.sh apac…"   5 seconds ago   Up 3 seconds   80/tcp, 443/tcp
     web3
92f61b7fa20f   mtorres26-apache-image:p5      "/entrypoint.sh apac…"   5 seconds ago   Up 3 seconds   80/tcp, 443/tcp
     web6
```

Figura 5: Procesos correspondientes a los contenedores web desplegados.

Vamos a ejecutar Apache Benchmark con 50.000 peticiones, primero con HTTPS y posteriormente con HTTP al balanceador. Las podemos ver en su log correspondiente como el que se muestra en la figura 6 y 7.

```
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P5/P5-ab$ docker logs apache-benchmark-P5
This is ApacheBench, Version 2.3 <$Revision: 1913912 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.10.50 (be patient)
Completed 5000 requests
Completed 10000 requests
Completed 15000 requests
Completed 20000 requests
Completed 25000 requests
Completed 30000 requests
Completed 35000 requests
Completed 40000 requests
Completed 45000 requests
Completed 50000 requests
Finished 50000 requests


Server Software:        nginx/1.25.5
Server Hostname:        192.168.10.50
Server Port:            443
SSL/TLS Protocol:       TLSv1.3,TLS_AES_256_GCM_SHA384,2048,256
Server Temp Key:        X25519 253 bits

Document Path:          /
Document Length:        299 bytes

Concurrency Level:      100
Time taken for tests:   93.467 seconds
Complete requests:      50000
Failed requests:        0
Total transferred:      24000000 bytes
HTML transferred:       14950000 bytes
Requests per second:    534.95 [#/sec] (mean)
```

Figura 6: Logs de Apache Benchmark con peticiones HTTPS al balanceador.

A continuación vamos a cambiar en el docker-compose.yml el comando para realizar las peticiones sobre HTTP en lugar de HTTPS. Esto simplemente se hace cambiando la línea que corresponde. Se muestran los logs con los cambios efectuados comentados anteriormente en las figuras 8 y 9.

Son muy notables las diferencias entre tiempos de HTTP y HTTPS, pues el tiempo medio de petición con HTTPS es de 187 milisegundos aproximadamente, mientras que con HTTP el tiempo medio de petición es de 9 milisegundos aproximadamente, esto es unas 20 veces menor. Esto es obvio, con HTTPS es normal

```
Requests per second:    534.95 [#/sec] (mean)
Time per request:       186.934 [ms] (mean)
Time per request:       1.869 [ms] (mean, across all concurrent requests)
Transfer rate:          250.76 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        2  177  54.8    153     364
Processing:     5   10   4.9      8     123
Waiting:        1   10   4.8      8     120
Total:         11  187  57.3    161     384

Percentage of the requests served within a certain time (ms)
  50%    161
  66%    169
  75%    178
  80%    191
  90%    306
  95%    318
  98%    335
  99%    348
 100%    384 (longest request)
```

Figura 7: Logs de Apache Benchmark con peticiones HTTPS al balanceador.

```
mtorres@mtorres:~/UGR_NATIVO/CuartoCurso/SegundoCuatri/SWAP/P5/P5-ab$ docker logs apache-benchmark-P5
This is ApacheBench, Version 2.3 <$Revision: 1913912 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.10.50 (be patient)
Completed 5000 requests
Completed 10000 requests
Completed 15000 requests
Completed 20000 requests
Completed 25000 requests
Completed 30000 requests
Completed 35000 requests
Completed 40000 requests
Completed 45000 requests
Completed 50000 requests
Finished 50000 requests


Server Software:        nginx/1.25.5
Server Hostname:        192.168.10.50
Server Port:            443

Document Path:          /
Document Length:        255 bytes

Concurrency Level:      100
Time taken for tests:   4.741 seconds
Complete requests:      50000
Failed requests:        0
Non-2xx responses:      50000
Total transferred:      20350000 bytes
HTML transferred:       12750000 bytes
Requests per second:    10545.69 [#/sec] (mean)
Time per request:       9.483 [ms] (mean)
Time per request:       0.095 [ms] (mean, across all concurrent requests)
```

Figura 8: Logs de Apache Benchmark con peticiones HTTP al balanceador.

que aumente el tiempo medio de petición porque es necesario hacer comprobaciones del certificado SSL.

Ahora, accedemos a la interfaz gráfica de Locust a través del puerto en el que se encuentra, `http://localhost:8089/`, y ejecutamos el servicio con 500 usuarios en concurrencia (figura 10). Disponemos de un apartado en el que se nos muestra el log (figura 11) y vemos que se han simulado 500 usuarios.

También es posible ver varios gráficos donde se refleja el comportamiento de la granja web con un máximo de 500 usuarios en concurrencia (figura 12), con tiempos de respuesta (figura 13) y número de peticiones por segundo (figura 14).

Procedemos a hacer lo mismo sobre HTTP y comparar tiempos medios de petición (figura 15):

No existe una gran diferencia entre tiempos usando HTTP o HTTPS, siendo en ambas el tiempo medio de respuesta de 6 milisegundos aprox. y el número de peticiones por segundo de 160 aprox. en ambas también.

```
Time per request:       9.483 [ms] (mean)
Time per request:       0.095 [ms] (mean, across all concurrent requests)
Transfer rate:          4191.50 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    4   0.6      4       7
Processing:     1    5   0.9      5       9
Waiting:        0    4   1.0      4       9
Total:          5    9   0.8      9      14

Percentage of the requests served within a certain time (ms)
  50%      9
  66%     10
  75%     10
  80%     10
  90%     10
  95%     11
  98%     12
  99%     12
 100%     14 (longest request)
```

Figura 9: Logs de Apache Benchmark con peticiones HTTP al balanceador.

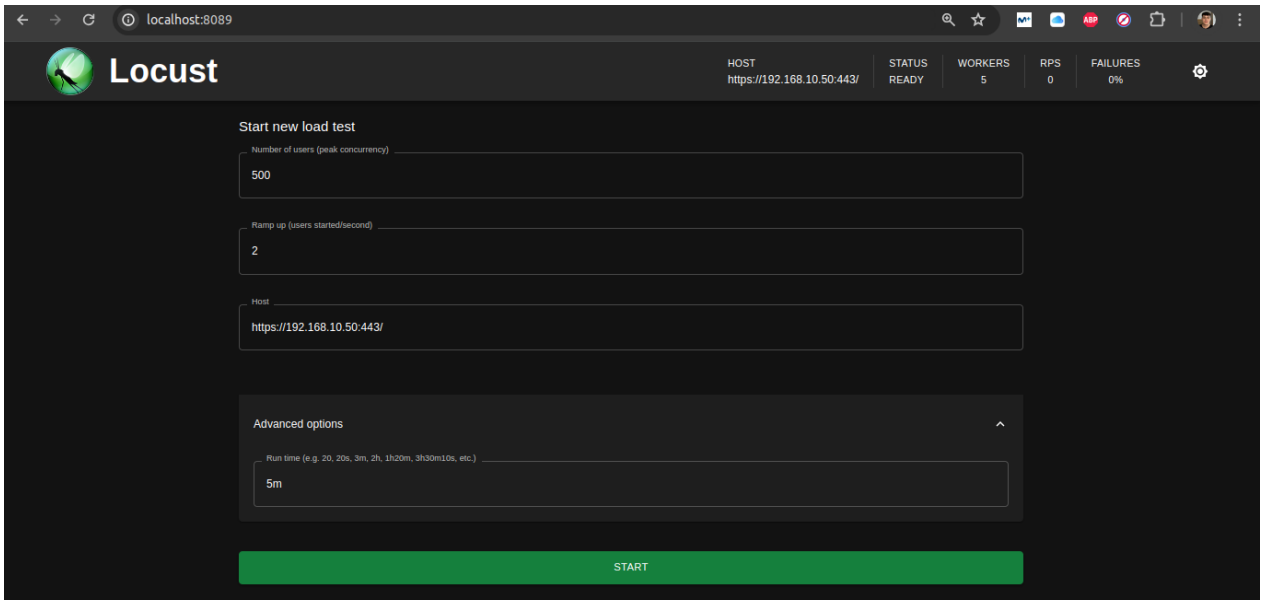

Figura 10: Interfaz para lanzar Locust.

5. Tareas Básicas - B5: Análisis de resultados

Vamos a poner a prueba la granja web y vamos a aumentar el número de solicitudes y la concurrencia de las mismas con Apache Benchmark. Por ejemplo, vamos a modificar el docker-compose.yml configurado para AB para que lance 100.000 peticiones con concurrencia de 100 peticiones.

```
command:
      [
        "ab",
        "-n",
        "100000",
        "-c",
        "100",
        "https://192.168.10.50:443/"
      ]
```

Listing 7: Modificación del comando que se lanzará.

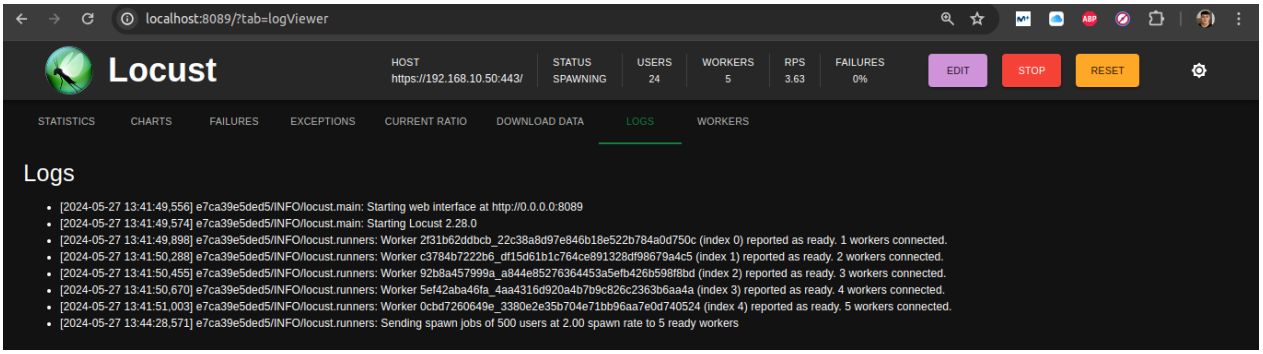

Figura 11: Logs de Locust ejecutado correctamente.

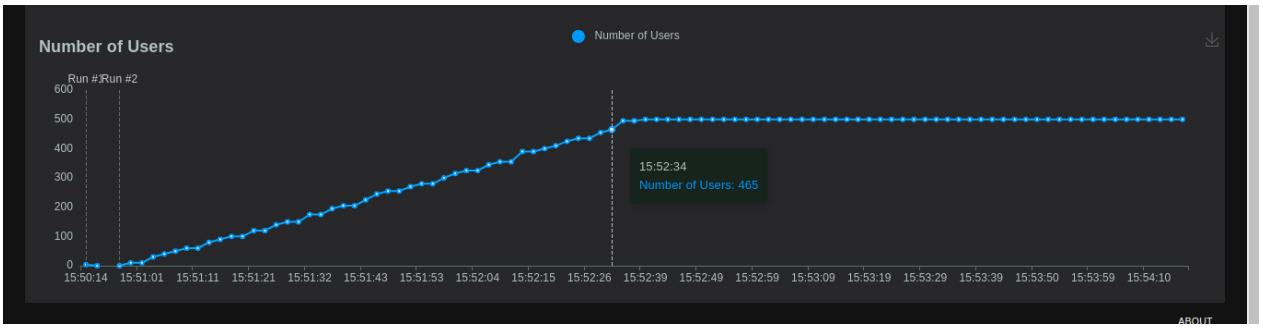

Figura 12: Número de usuarios en concurrencia.

Después de unos minutos, el log que se nos muestra desde la terminal muestra información básica como tiempo total empleado, número de peticiones fallidas, media de peticiones por segundo, media de tiempo por petición, máximo tiempo de petición, percentiles de tiempo de petición... etc (figura 16).

El benchmarking ha durado alrededor de 3 minutos, la media de peticiones por segundo ha sido de 459,73, el tiempo medio por petición es de 2,175 ms sin contar concurrencia y de 217,5 ms contando concurrencia, lo cual es obvio porque teníamos 100 peticiones de concurrencia. El tiempo de conexión máximo ha sido de 427 ms.

Ahora, vamos a usar la herramienta Locust para realizar pruebas, con 10.000 usuarios en concurrencia usando el mismo número de workers (cinco). Indicamos mediante la interfaz que el tiempo de ejecución sea de 5 minutos. En la siguiente figura (17) notamos cierta inestabilidad en el total de peticiones por segundo, con dos picos de 130 peticiones/segundo en los primeros segundos de ejecución, pero luego este número disminuye a una media de 60 peticiones/segundo. Nótese que las peticiones fallidas aumentan cuando se producen los picos, con una media de 10 peticiones fallidas por segundo durante varios segundos, lo cual se corresponde con aproximadamente un 7,7 % de peticiones fallidas como pico, un porcentaje relativamente alto.

La gráfica que informa de los tiempos medios de respuesta (amarillo) está dentro de lo normal, es decir, creciente, pues conforme aumenta el número de usuarios en concurrencia, aumenta la sobrecarga de la granja y por tanto aumentan los tiempos medios de respuesta. Una cosa a destacar es que, una vez estabilizado el número de usuarios en concurrencia (azul), el tiempo medio de respuesta no se estabiliza, debido al alto número de usuarios en concurrencia.

Locust también proporciona una tabla resumen con métricas (figura 18), la cual podemos comparar con la de AB. Se han realizado 13.000 peticiones de las cuales 316 han fallado. En AB se realizaron 459,73 peticiones

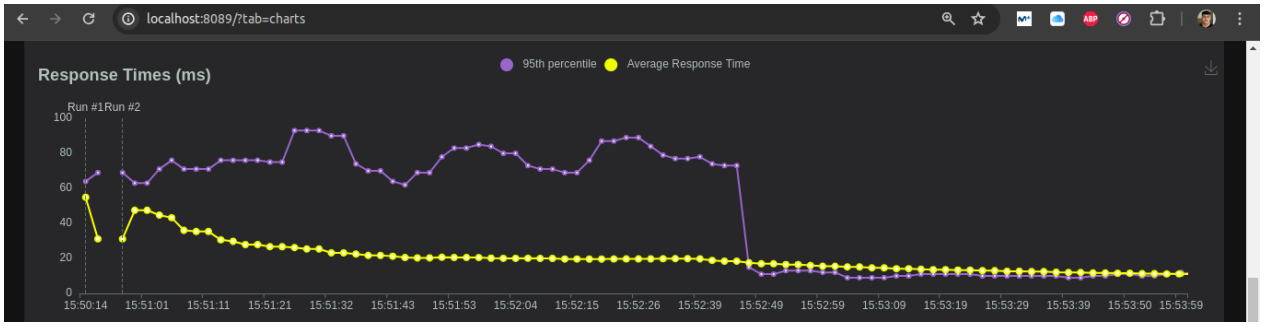

Figura 13: Tiempos de respuesta.

Figura 14: Gráfico con el número de peticiones por segundo.

por segundo, mientras que en Locust unas 67, es decir casi 7 veces menos.

Figura 15: Gráficos de estadísticas de Locust sobre HTTP.

```
Concurrency Level:      100
Time taken for tests:   217.517 seconds
Complete requests:      100000
Failed requests:        0
Total transferred:      48000000 bytes
HTML transferred:       29900000 bytes
Requests per second:    459.73 [#/sec] (mean)
Time per request:       217.517 [ms] (mean)
Time per request:       2.175 [ms] (mean, across all concurrent requests)
Transfer rate:          215.50 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        3  206  73.8    158     407
Processing:     5   11   4.7      9     111
Waiting:        1   11   4.7      8     110
Total:         12  217  77.5    167     427

Percentage of the requests served within a certain time (ms)
  50%    167
  66%    293
  75%    302
  80%    307
  90%    326
  95%    340
  98%    358
  99%    371
 100%    427 (longest request)
```

Figura 16: Información del benchmarking realizado.

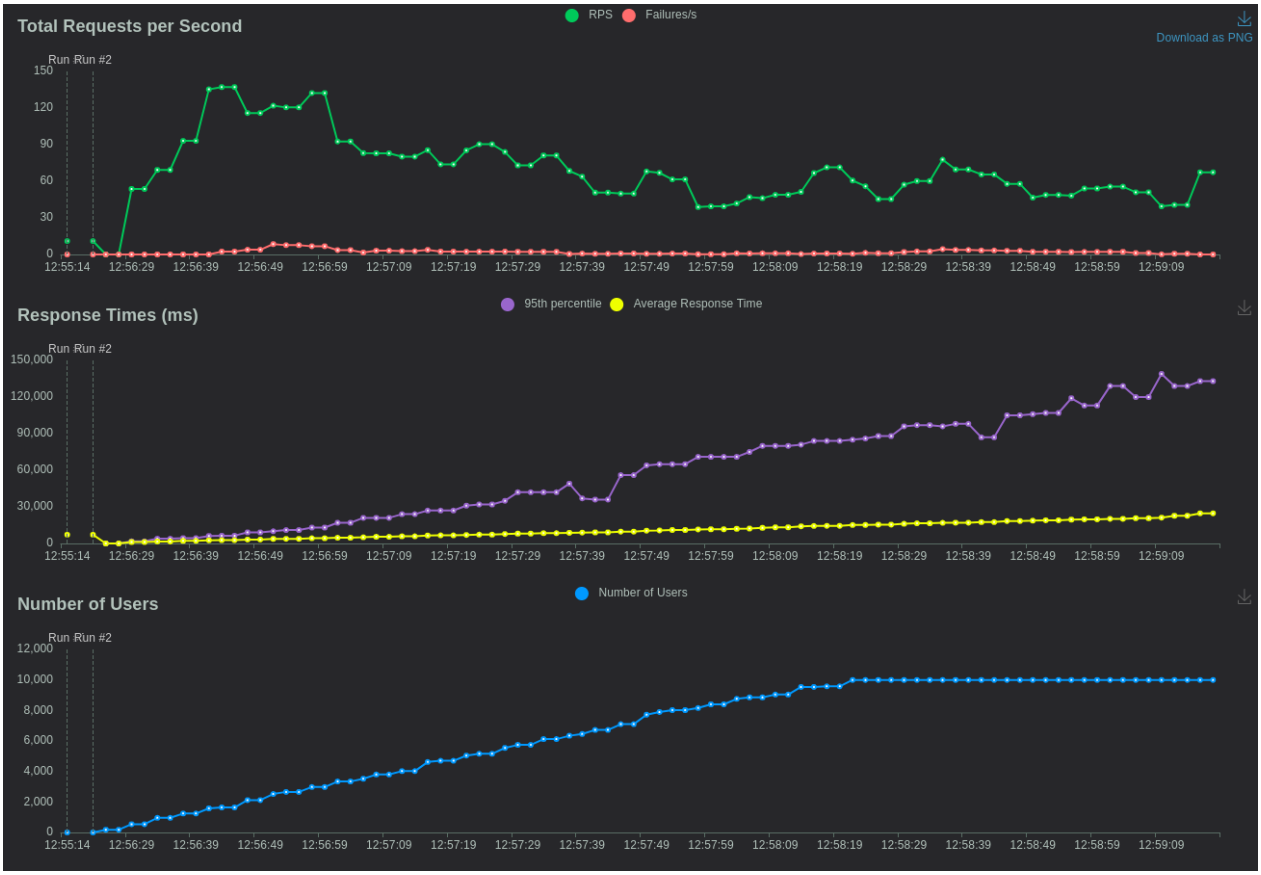

Figura 17: Información del benchmarking realizado con Locust.

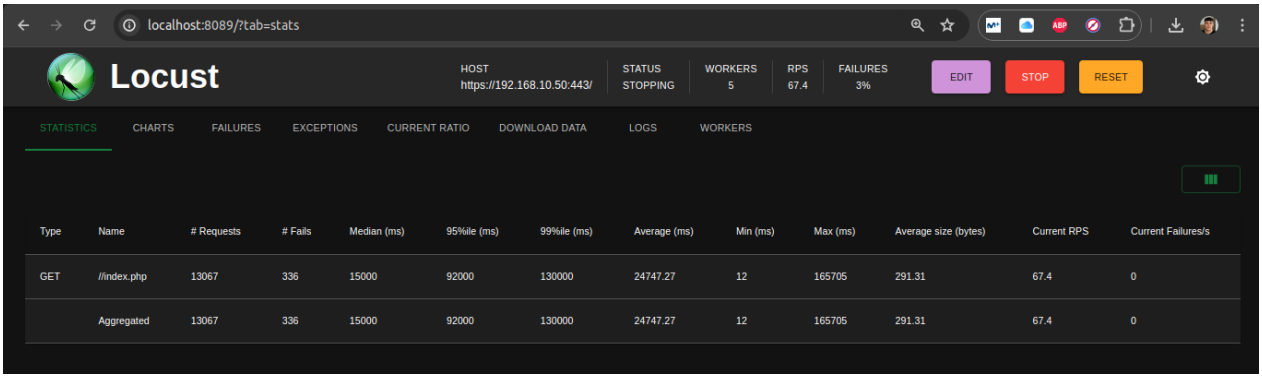

Figura 18: Tabla estadísticas Locust.