



UNIVERSIDAD DE GRANADA

P5 - Benchmarking

Servidores Web de Altas Prestaciones

UGR - ETSIIT

Torres Ramos, Juan Luis

29 mayo de 2024

Índice

Índice	1
Tareas Básicas - B1: Configuración del entorno de benchmarking	1
Tareas Básicas - B2: Implementación con Apache Benchmark	3
Tareas Básicas - B3: Implementación con Locust	5
Tareas Básicas - B4: Ejecución de pruebas de carga	7
Tareas Básicas - B5: Análisis de resultados	14
Tareas Avanzada - A1: Desarrollar tareas avanzadas en Locustfile.py	15
Tareas Avanzada - A2. Crear escenario multicontenedor con algún CMS	17
Tareas Avanzada - A3. Ejecución y Análisis de cargas de prueba avanzadas sobre CMS	23
Análisis IA	28
Nota	33

Tareas Básicas - B1: Configuración del entorno de benchmarking

Vamos a comenzar por configurar el entorno de trabajo. Creamos dos carpetas nuevas P5-ab y P5-locust donde implementaremos en cada uno un docker-compose.yml donde se van a desplegar los benchmarkings. Copiamos la granja web de la práctica 4 en otra nueva carpeta llamada P5-granjaweb. He tenido que borrar las reglas iptables de la práctica 4 por que daban conflicto con ab.

Con un tree visualizamos el entorno de trabajo

```
tree
.
├── P5-ab
├── P5-granjaweb
│   ├── docker-compose.yml
│   ├── P4-juanluis-apache
│   │   ├── DockerFileApacheP4
│   │   ├── juanluis-apache-ssl.conf
│   │   └── P4-juanluis-iptables-web
│   │       ├── entrypoint.sh
│   │       └── juanluis-iptables-web.sh
│   ├── P4-juanluis-certificados
│   │   ├── certificado_juanluis.crt
│   │   └── certificado_juanluis.key
│   ├── P4-juanluis-nginx
│   │   ├── DockerFileNginxP4
│   │   ├── juanluis-nginx-ssl.conf
│   │   └── P4-juanluis-iptables-balanceador
│   │       ├── entrypoint.sh
│   │       └── juanluis-iptables-balanceador.sh
│   └── web_juanluis
│       └── index.php
└── P5-locust
```

13 directories, 25 files

Tareas Básicas - B2: Implementación con Apache Benchmark

Comenzamos con ab (Apache benchmark) para lanzar pruebas de rendimiento contra el balanceador de carga de la granja web. Simulamos una carga de tráfico HTTP y HTTPS y evaluamos su interacción con el balanceador.

Primero creamos el Dockerfile(DockerFileAB) e implementamos lo siguiente

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

Es una imagen base de debian donde se instala el apache benchmark (apache2-utils)

Ahora pasamos a crear el docker-compose.yml

- Tenemos que definir el servicio apache-benchmark-P5 para el contenedor de AB a partir de la imagen anterior, tiene la dirección ip 192.168.10.60.
- Apache benchmark tiene que apuntar a la dirección ip del balanceador 192.168.10.50. Por problemas que tengo con el firewall, lo he acabado implementando dicha dirección al <https://localhost:4000/> (no me conecta a la ip que tiene asignada, solo me centro en hacer el benchmarking)

docker-compose.yml

```
services:
  apache-benchmark-P5:
    build:
      context: .
      dockerfile: DockerFileAB
    image: juanluis-ab-image:p5
    container_name: apache_benchmark-P5
    #command: ["ab", "-n", "10000", "-c", "100", "https://192.168.10.50:443/"]
    command: ["ab", "-n", "10000", "-c", "100", "https://localhost:4000/"]
    networks:
      red_web:
        ipv4_address: 192.168.10.60

networks:
  red_web:
    external: true
```

Para asegurar que el contenedor AB esté en la misma red (red_web) que el balanceador, hemos definido también red_web en el docker-compose.yml.

Por petición del guión, ahora las redes las hemos configurado desde fuera. En la granja web, en el docker compose, configuramos las redes para que sean también externas:

docker-compose.yml

```
...
networks:
  red_web:
    external: True
    #   driver: bridge
    #   ipam:
    #     driver: default
    #     config:
    #       - subnet: 192.168.10.0/24

  red_servicios:
    external: True
    #driver: bridge
    #ipam:
    #  driver: default
    #  config:
    #    - subnet: 192.168.20.0/24
```

y para crear las redes, desde la terminal:

```
docker network create \
  --driver=bridge \
  --subnet=192.168.10.0/24 \
  red_web

docker network create \
  --driver=bridge \
  --subnet=192.168.20.0/24 \
  red_servicios
```

Tareas Básicas - B3: Implementación con Locust

Pasamos ahora a configurar el benchmark de Locust. Vamos a implementar un nodo master y múltiples trabajadores. En la carpeta P5-locust creamos un `dockers-compose.yml` y un `locustfile.py`

Vamos a preparar el archivo `locustfile.py`. Vamos a implementar tareas definidas para realizar peticiones HTTP y HTTPS a la granja web y tiempo aleatorio entre peticiones.

En `self.client.get` implementos `verify=False` para permitir peticiones HTTPS a mi balanceador con certificado autofirmado

```
# Defino la clase tarea especifica para hacer peticiones GET a index.php  
# La clase simula a los usuarios que realizan las pruebas, con una espera  
aleatoria entre  
# 1 y 5 segundos entre cada accion para emular el comportamiento humano más  
realista  
  
from locust import HttpUser, TaskSet, task, between  
  
class P5JuanLuis(TaskSet):  
    @task  
    def access_balancer(self):  
        self.client.get("https://host.docker.internal:4000/", verify=False)  
  
class P5Usuarios(HttpUser):  
    tasks = [P5JuanLuis]  
    wait_time = between(1, 5)
```

Pasamos a definir el docker compose para Locust, implementamos el servicio máster y el servicio workers de la siguiente manera:

Docker-compose.yml

```
version: '3'
services:
  master-juanluis:
    image: locustio/locust
    ports:
      - "8089:8089"
    volumes:
      - ./:/mnt/locust
    command: -f /mnt/locust/locustfile.py --master -H https://localhost:4000/
    networks:
      red_web:
        ipv4_address: 192.168.10.70

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

networks:
  red_web:
    external: true
```

Tareas Básicas - B4: Ejecución de pruebas de carga

Para cada benchmark vamos a solicitar 10000 peticiones y 100 usuarios

1º creamos las redes red_web y red_servicios desde la terminal

```
docker network create \  
  --driver=bridge \  
  --subnet=192.168.10.0/24 \  
  red_web
```

```
docker network create \  
  --driver=bridge \  
  --subnet=192.168.20.0/24 \  
  red_servicios
```

2º desplegamos la grana web, desde la carpeta *P5-granjaweb*

```
docker compose build  
docker compose up -d
```

3º ejecutamos una prueba con apache benchmark

Primero hacemos el ab desde la terminal:

```
ab -n 10000 -c 100 https://localhost:4000/
```

```
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 localhost (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
Server Hostname:      localhost
Server Port:          4000
SSL/TLS Protocol:     TLSv1.3,TLS_AES_256_GCM_SHA384,2048,256
Server Temp Key:      X25519 253 bits
TLS Server Name:      localhost

Document Path:        /
Document Length:       325 bytes

Concurrency Level:     100
Time taken for tests:   11.723 seconds
Complete requests:     10000
Failed requests:        0
Total transferred:     7200000 bytes
HTML transferred:      3250000 bytes
Requests per second:   853.06 [#/sec] (mean)
Time per request:      117.226 [ms] (mean)
Time per request:      1.172 [ms] (mean, across all concurrent requests)
Transfer rate:         599.80 [Kbytes/sec] received

```

Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	7	51 17.2	50	133
Processing: 12	65	30.6	58	296
Waiting:	6	55 32.5	48	294
Total:	19	116 32.2	108	372

Percentage of the requests served within a certain time (ms)

```

50% 108
66% 120
75% 132
80% 140
90% 160
95% 177
98% 200
99% 223
100% 372 (longest request)

```

Ahora ejecutamos el docker-compose.yml del ab y nos metemos en su terminal a ver los resultados :

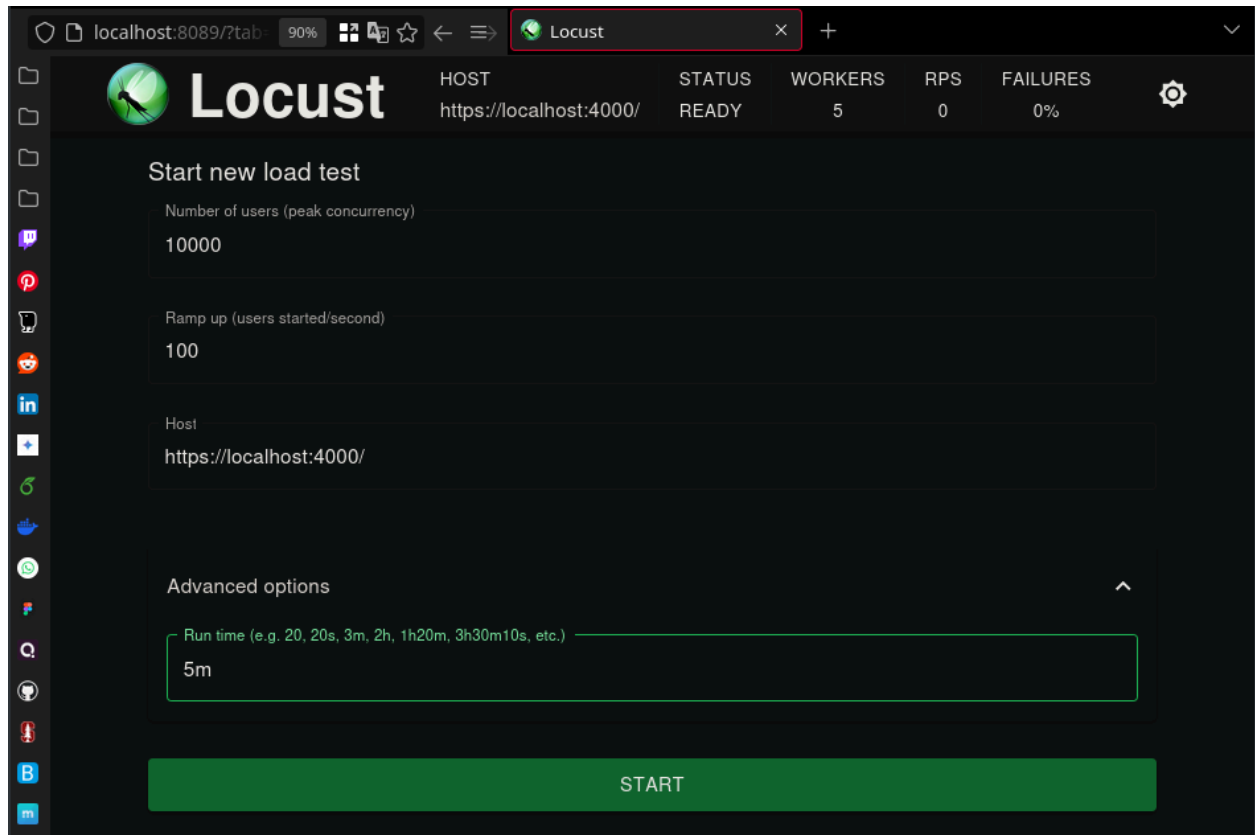
```
2024-05-26 19:35:03 This is ApacheBench, Version 2.3 <$Revision: 1913912 $>
2024-05-26 19:35:03 Copyright 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
2024-05-26 19:35:03 Licensed to The Apache Software Foundation,
http://www.apache.org/
2024-05-26 19:35:03
2024-05-26 19:35:03 Benchmarking 192.168.10.50 (be patient)
2024-05-26 19:35:04 Completed 1000 requests
2024-05-26 19:35:05 Completed 2000 requests
2024-05-26 19:35:06 Completed 3000 requests
2024-05-26 19:35:07 Completed 4000 requests
2024-05-26 19:35:08 Completed 5000 requests
2024-05-26 19:35:09 Completed 6000 requests
2024-05-26 19:35:10 Completed 7000 requests
2024-05-26 19:35:11 Completed 8000 requests
2024-05-26 19:35:12 Completed 9000 requests
2024-05-26 19:35:13 Completed 10000 requests
2024-05-26 19:35:13 Finished 10000 requests
2024-05-26 19:35:13
2024-05-26 19:35:13
2024-05-26 19:35:13 Server Software:      nginx
2024-05-26 19:35:13 Server Hostname:      192.168.10.50
2024-05-26 19:35:13 Server Port:          443
2024-05-26 19:35:13 SSL/TLS Protocol:
TLSv1.3,TLS_AES_256_GCM_SHA384,2048,256
2024-05-26 19:35:13 Server Temp Key:      X25519 253 bits
2024-05-26 19:35:13
2024-05-26 19:35:13 Document Path:         /
2024-05-26 19:35:13 Document Length:      325 bytes
2024-05-26 19:35:13
2024-05-26 19:35:13 Concurrency Level:     100
2024-05-26 19:35:13 Time taken for tests:   9.835 seconds
2024-05-26 19:35:13 Complete requests:     10000
2024-05-26 19:35:13 Failed requests:        0
2024-05-26 19:35:13 Total transferred:     7200000 bytes
2024-05-26 19:35:13 HTML transferred:      3250000 bytes
2024-05-26 19:35:13 Requests per second:   1016.76 [#/sec] (mean)
2024-05-26 19:35:13 Time per request:       98.351 [ms] (mean)
2024-05-26 19:35:13 Time per request:       0.984 [ms] (mean, across all
concurrent requests)
```

```

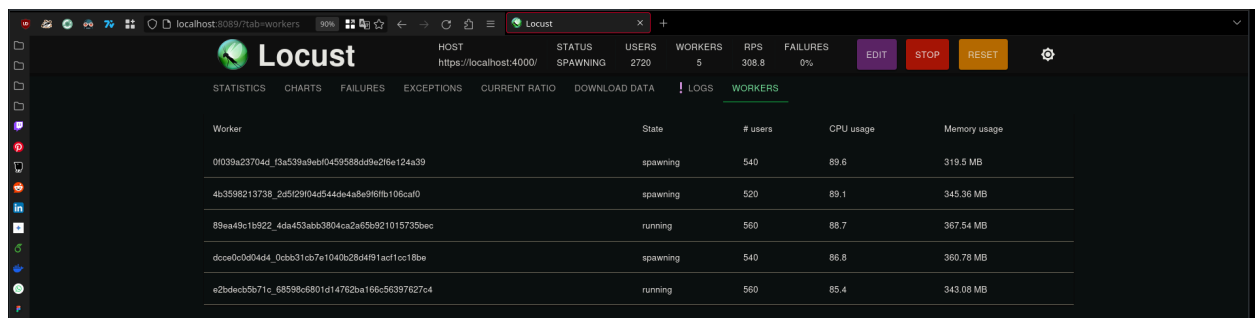
2024-05-26 19:35:13 Transfer rate:          714.91 [Kbytes/sec] received
2024-05-26 19:35:13
2024-05-26 19:35:13 Connection Times (ms)
2024-05-26 19:35:13          min  mean[+/-sd] median   max
2024-05-26 19:35:13 Connect:        5    56  16.8      55    107
2024-05-26 19:35:13 Processing:     10    42  16.3      41    165
2024-05-26 19:35:13 Waiting:        2    24  14.4      21    152
2024-05-26 19:35:13 Total:         16    98  10.6      97    196
2024-05-26 19:35:13
2024-05-26 19:35:13 Percentage of the requests served within a certain time
(ms)
2024-05-26 19:35:13   50%      97
2024-05-26 19:35:13   66%     101
2024-05-26 19:35:13   75%     103
2024-05-26 19:35:13   80%     104
2024-05-26 19:35:13   90%     108
2024-05-26 19:35:13   95%     113
2024-05-26 19:35:13   98%     121
2024-05-26 19:35:13   99%     133
2024-05-26 19:35:13  100%     196 (longest request)

```

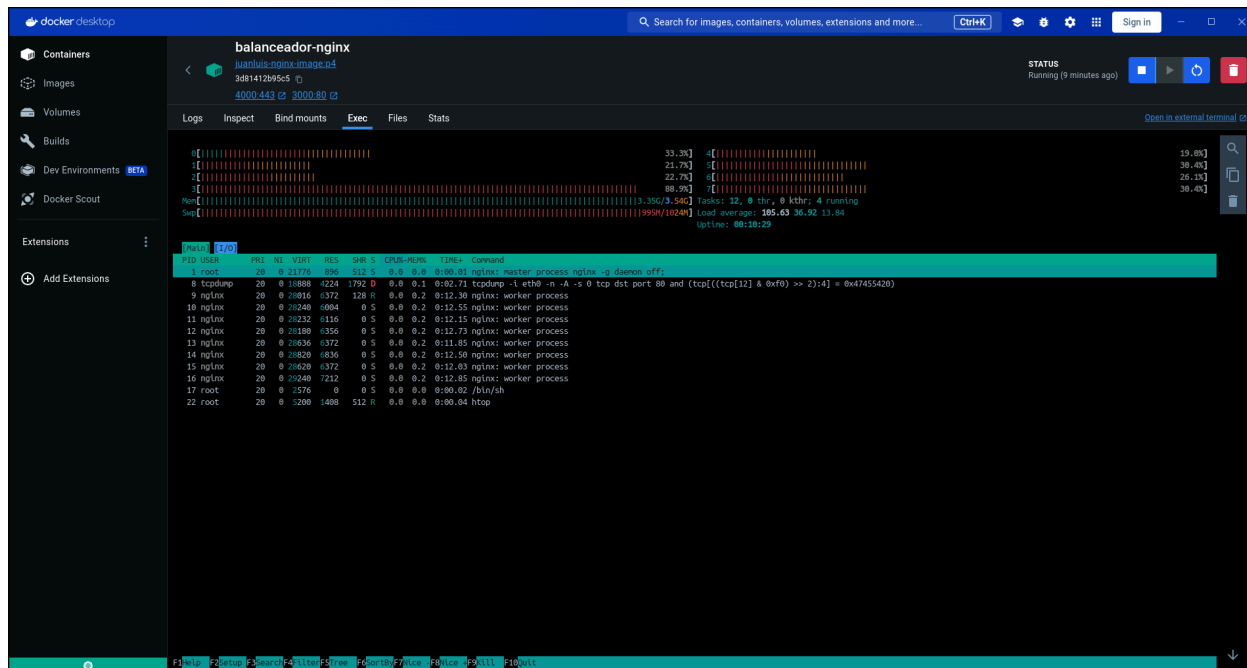
Pasamos a ejecutar el benchmark de Locust



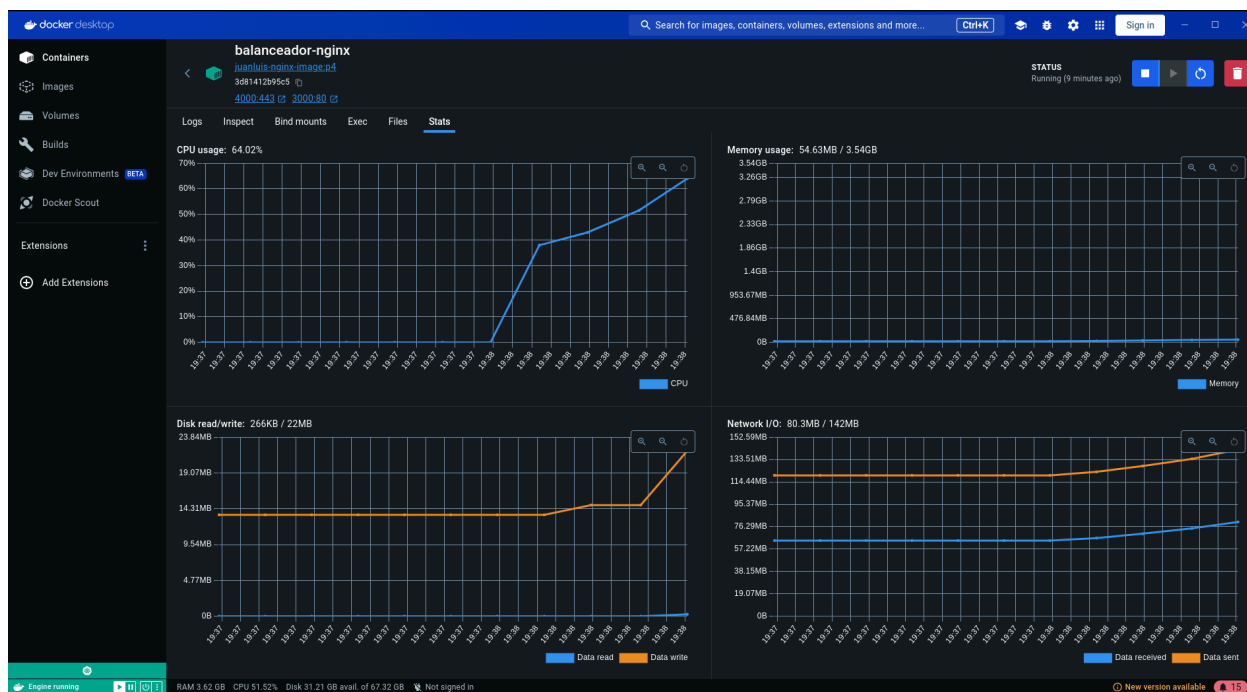
aquí están los workers activos



mientras se ejecuta el benchmark con htop podemos ver que se están usando recursos del contenedor, lo podemos ver con htop (hay que descargarlo primero en la imagen)

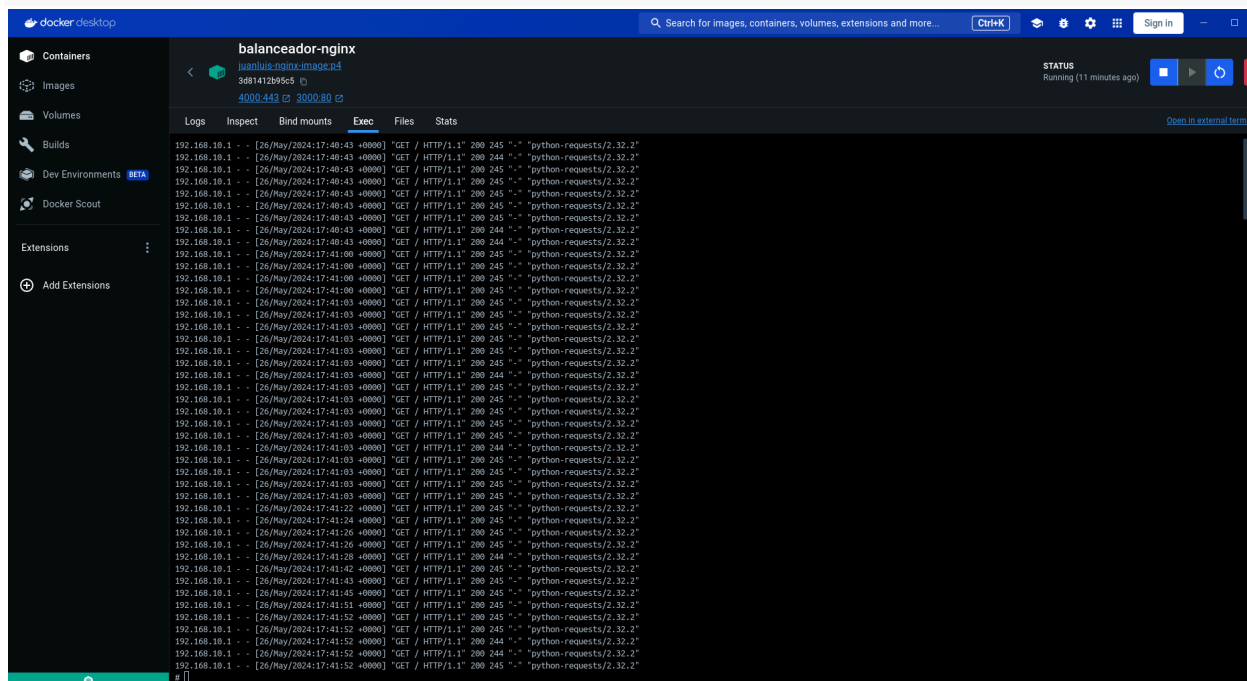


los stats desde el docker desktop

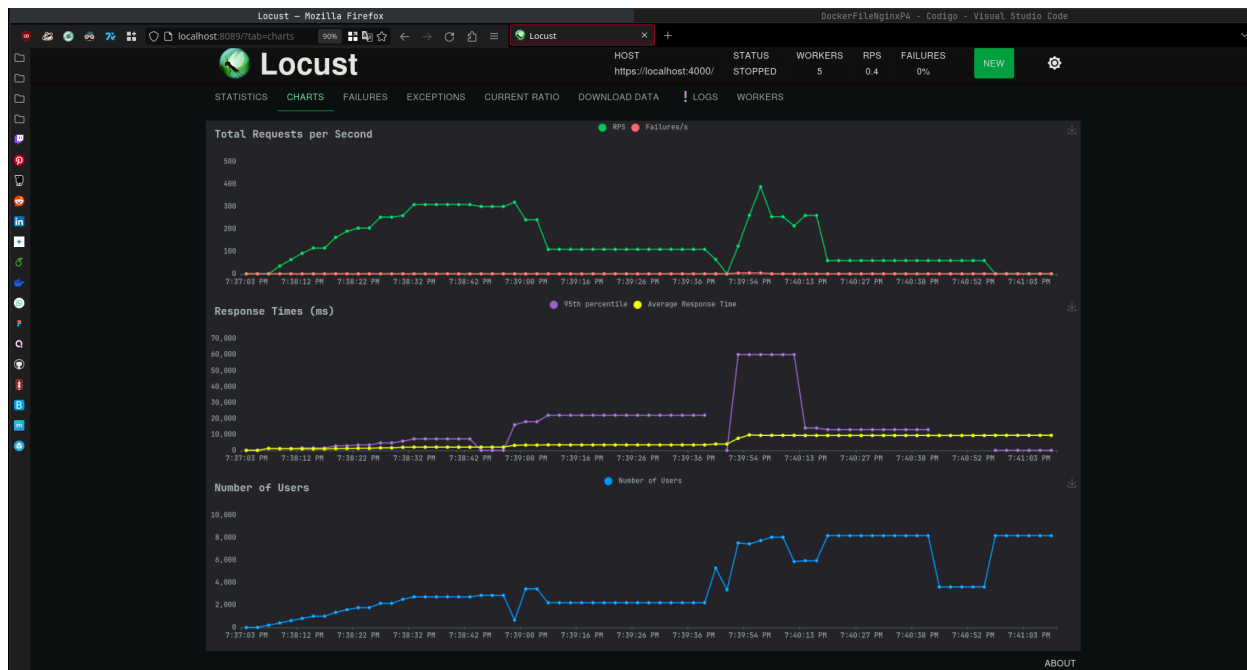


podemos también ver todos los logs que le llegan al balanceador desde el archivo configurado en el nginx.conf: log/var/nginx/nginx_juanluis.access.log, podemos hacer un

```
tail -f log/var/nginx/nginx_juanluis.access.log
```



los resultados luego de esperar 5 minutos:



Tareas Básicas - B5: Análisis de resultados

La documentación de los resultados la he puesto en el apartado anterior. Ahora voy a hacer un análisis sobre los datos que hemos obtenido.

Los resultados del benchmarking con ab y locust muestran que el servidor nginx está funcionando eficientemente bajo carga ,viendo desde htop oscila de 60-80% el uso de cpu y casi un 100% de memoria RAM). Para ambos benchmarks se completaron todas las solicitudes sin ningún fallo, por lo que podemos decir que el servidor es estable. La tasa de solicitudes por segundo es alta, con un promedio de 1016.76 en el test ab, por lo que deducimos que el tiempo de respuesta es rápido en el balanceador. Los tiempos de conexión y procesamiento también fueron bastante bajos, con tiempos promedio por solicitud de 98.351 milisegundos. También la distribución de los tiempos de respuesta muestra que la gran mayoría de las solicitudes que llegan se completan en un tiempo razonable, con un 99% de las solicitudes atendidas en menos de 133 milisegundos. En general, los resultados obtenidos indican un buen rendimiento del servidor bajo 10000 peticiones

Tareas Avanzada - A1: Desarrollar tareas avanzadas en Locustfile.py

Antes de ver este apartado, vamos a ver el apartado A2. Vamos a implementar tareas que van a reflejar operaciones típicas del CMS. Para ello vamos a implementar varias funciones dentro del locustfile.py.

Vamos a actualizar el locusfile.py de la siguiente manera :

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

class P5JuanLuis(TaskSet):
    @task(1)
    def access_homepage(self):
        self.client.get("http://host.docker.internal:8010/", verify=False)

    @task(2)
    def login(self):
        # Simulate Logging in
        response =
self.client.post("http://host.docker.internal:8010/user/login", {
    "name": "your_username",
    "pass": "your_password"
}, verify=False)
        if response.status_code == 200:
            print("Logged in successfully")

    @task(3)
    def navigate_pages(self):
        pages = ["http://host.docker.internal:8010/node/1",
"http://host.docker.internal:8010/node/2",
"http://host.docker.internal:8010/node/3"]
        for page in pages:
            self.client.get(page, verify=False)

    @task(4)
    def create_content(self):
        # Simula la creación de un nuevo contenido
        response =
self.client.post("http://host.docker.internal:8010/node/add/article", {
    "title": "Test Article",
    "body[und][0][value]": "This is a test article created by Locust.",
    "body[und][0][format]": "filtered_html"
}, verify=False)
```



```

        if response.status_code == 403:
            print("Forbidden: You don't have permission to create content")
        elif response.status_code == 200:
            print("Content created successfully")
        else:
            print(f"Failed to create content: {response.status_code}")

@task(5)
def post_comment(self):
    # Simula la publicación de un comentario
    response =
self.client.post("http://host.docker.internal:8010/node/4#comment-form", { #
Cambia 4 por el ID del nodo correcto
        "comment_body[0][value]": "This is a test comment.",
        "comment_body[0][format]": "plain_text"
    }, verify=False)
    if response.status_code == 403:
        print("Forbidden: You don't have permission to post a comment")
    elif response.status_code == 200:
        print("Comment posted successfully")
    else:
        print(f"Failed to post comment: {response.status_code}")

@task(6)
def logout(self):
    # Simula el cierre de sesión
    self.client.get("http://host.docker.internal:8010/user/logout",
verify=False)

class P5Usuarios(HttpUser):
    tasks = [P5JuanLuis]
    wait_time = between(1, 5)

```

¿Que estamos comprobando en nuestro CMS con este locustfile.py?

- acceder al homepage
- hacer un login del user admin con contraseña admin
- navegar entre las páginas node/1, node/2, node/3
- creamos artículo en node/add/article
- creamos comentarios en el artículo node/4
- logout

Tareas Avanzada - A2. Crear escenario multicontenedor con algún CMS

Voy a crear un CMS en un contenedor a parte. He intentado usar la versión 10 de drupal, pero tengo un error con mostrar el CSS de la página web luego de crearlo, como lo importante es hacer un benchmarking a un CMS, he decidido usar la versión 9.3.9, aunque esté desactualizado..

He implementado el siguiente Drupal CMS segun este repositorio github en el localhost en un contenedor a parte:

<https://github.com/consciousm1n9/drupal-docker-simplest>

<https://www.youtube.com/watch?v=IhUl1pylJ-U> (este video es del set up del cms anterior)

creamos el docker-compose para el cms:

```
version: '3'
services:
  master-juanluis:
    image: locustio/locust
    ports:
      - "8089:8089"
    volumes:
      - ./:/mnt/locust
    #command: -f /mnt/locust/locustfile.py --master -H https://localhost:4000/
    command: -f /mnt/locust/locustfile.py --master -H http://localhost:8010/
    networks:
      red_web:
        ipv4_address: 192.168.10.70

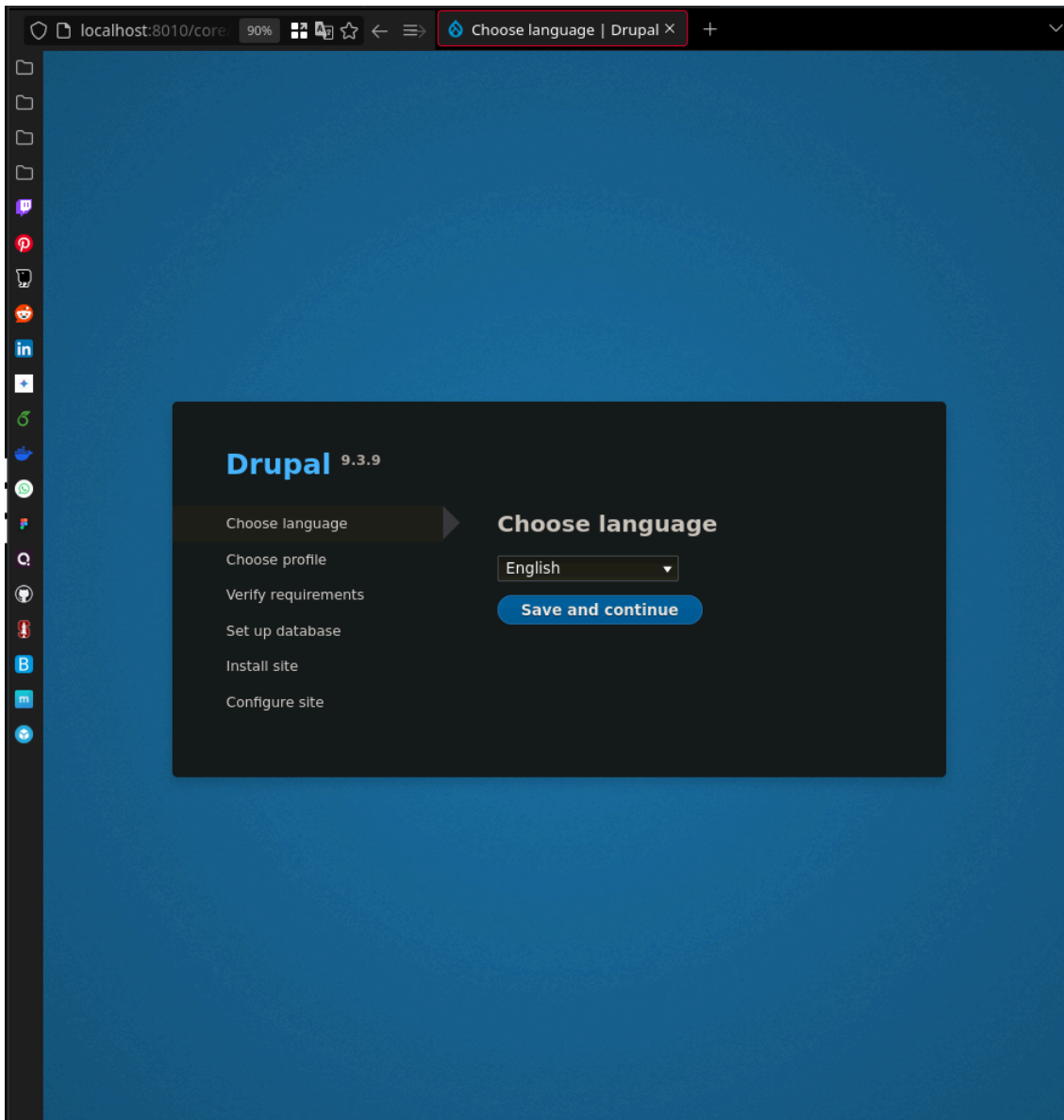
  worker-juanluis:
    image: locustio/locust
    volumes:
      - ./:/mnt/locust
    command: -f /mnt/locust/locustfile.py --worker --master-host master-juanluis
    depends_on:
      - master-juanluis
    deploy:
      replicas: 5
    networks:
      red_web:

networks:
  red_web:
    external: true
```

Creamos una nueva carpeta P5-drupal y clonamos el proyecto anterior, luego hacemos un

```
docker compose up -d
```

Accedemos a <http://localhost:8010/> para configurar el CMS



Para la base de datos

- usamos postgres
- database name: postgres
- database username: postgres
- database password: postgres
- vamos a advanced option
 - host: postgres
 - puerto: 5432

Configure site

- Site name: p5swap-drupal
- Email: example@gmail.com
- username: admin
- password: admin
- default country: cual sea
- time zone: Madrid

Drupal 9.3.9

Choose language

Choose profile

Verify requirements

Set up database

Install site

Configure site

Configure site

SITE INFORMATION

Site name *

p5swap-drupal

Site email address *

example@gmail.com

Automated emails, such as registration information, will be sent from this address. Use an address ending in your site's domain to help prevent these emails from being flagged as spam.

SITE MAINTENANCE ACCOUNT

Username *

admin

Several special characters are allowed, including space, period (.), hyphen (-), apostrophe ('), underscore (_), and the @ sign.

Password *

Password strength: Weak

Confirm password *

Passwords match: **yes**

Recommendations to make your password stronger:

- Make it at least 12 characters
- Add uppercase letters
- Add numbers
- Add punctuation
- Make it different from your username

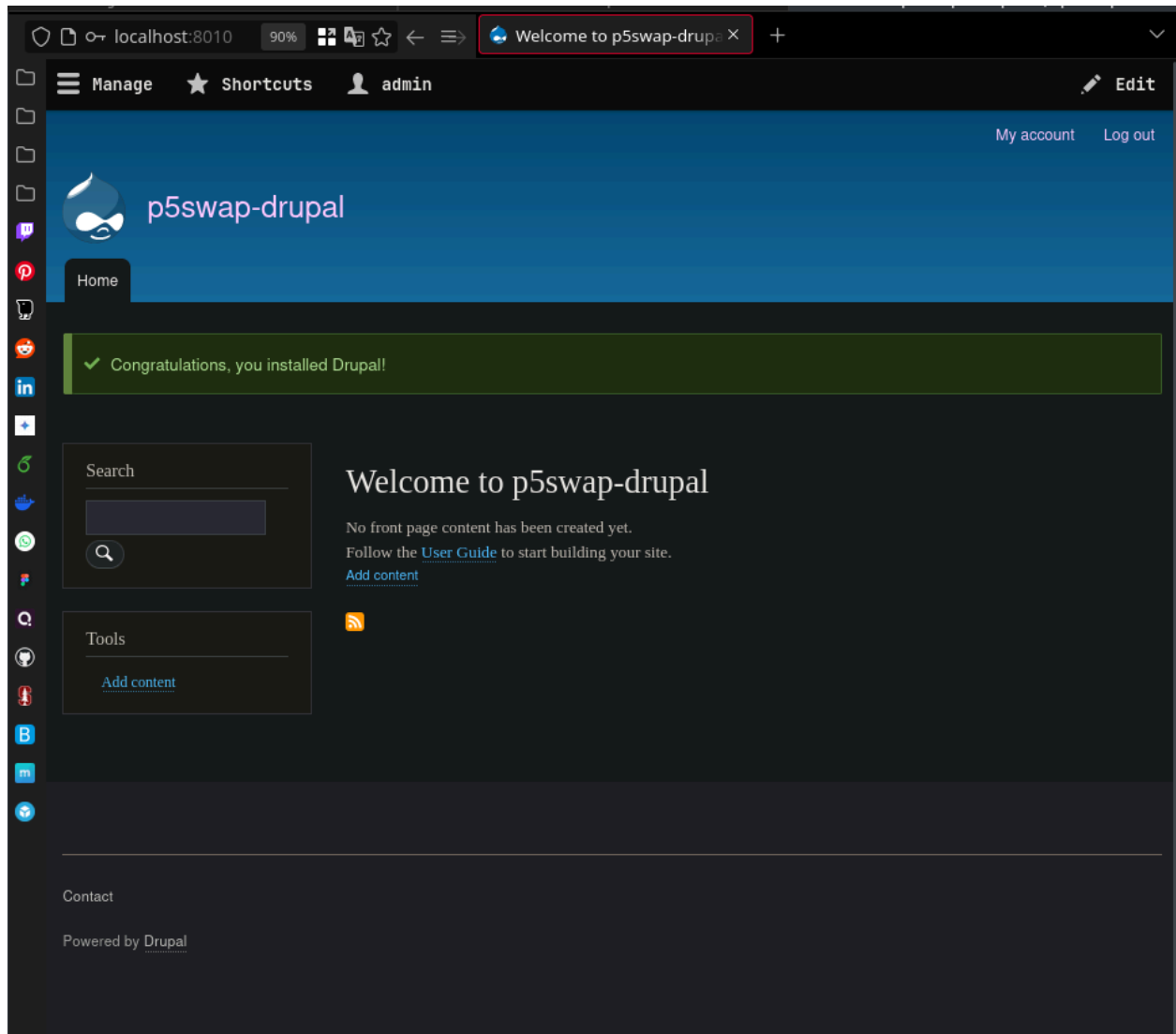
Email address *

example@gmail.com

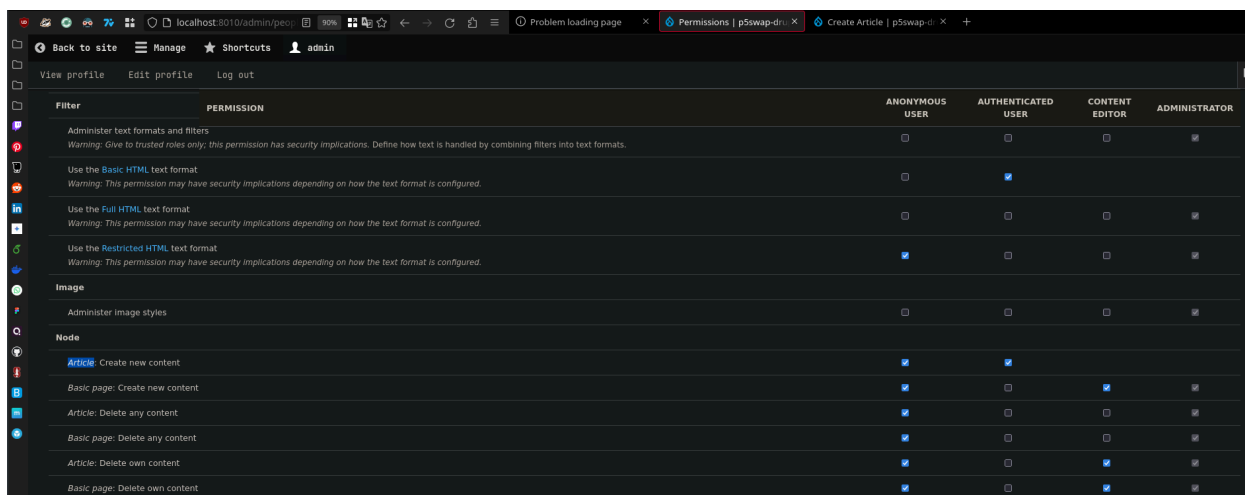
REGIONAL SETTINGS

Default country

Ahora nos llevará a la página principal



damos permisos para crear artículos y más (People -> Permissions) Para poder hacer post de articulos

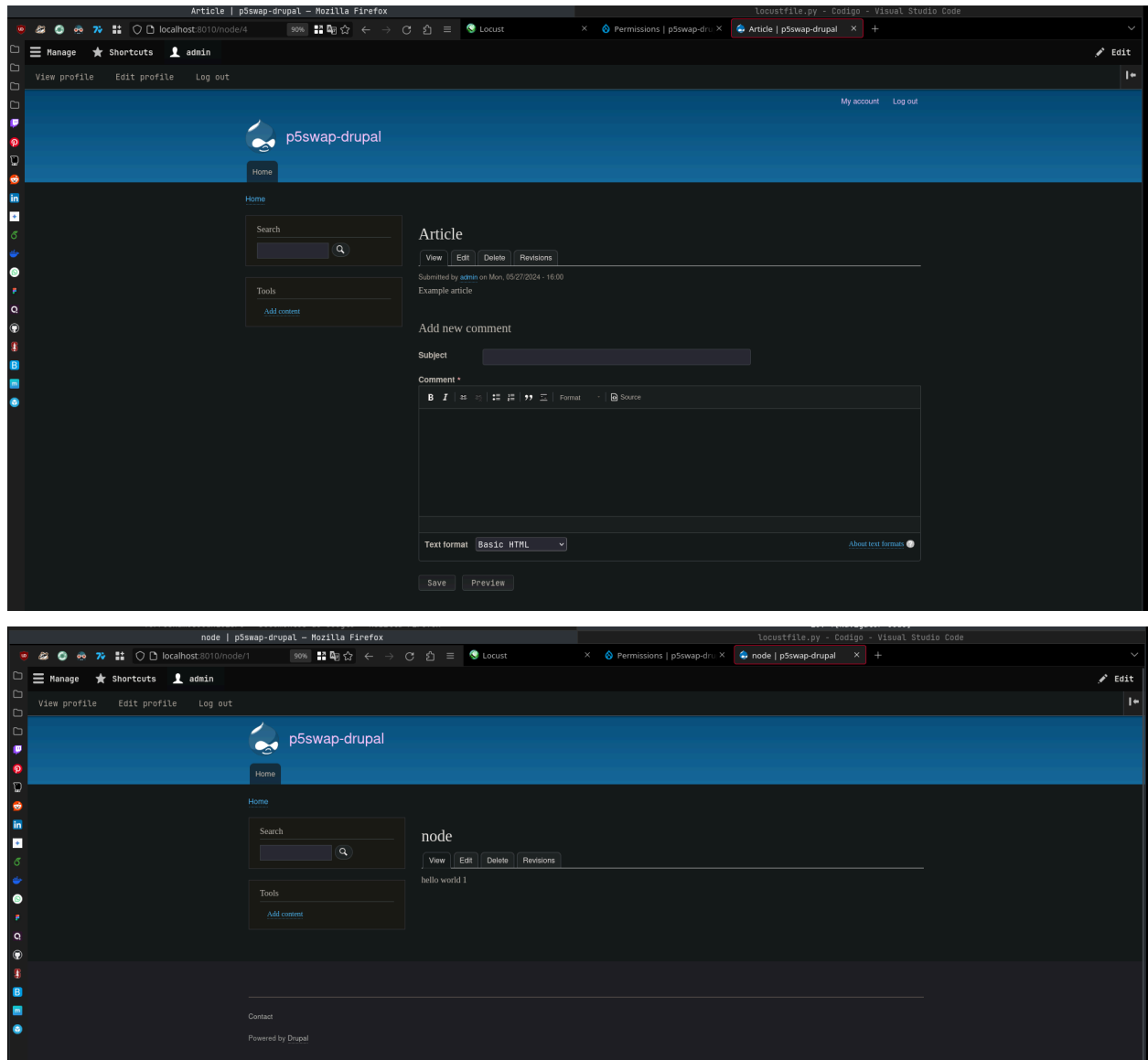


Comment				
Administer comment types and settings				
Warning: Give to trusted roles only; this permission has security implications.				
Administer comments and comment settings				
Edit own comments	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Post comments	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
Skip comment approval	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
View comments	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		

creamos 3 páginas, node/1, node/2, node/3 y un artículo en el node/4, para crear contenido en drupal:

The top screenshot shows the Drupal 9 admin interface. The browser address bar is `localhost:8010/node`. The user is logged in as `admin`. The sidebar menu includes `Manage`, `Shortcuts`, and `admin`. The main content area is titled `Add content` and features a security update warning. Below the warning, there are two options: `Article` (for time-sensitive content like news, press releases or blog posts) and `Basic page` (for static content, such as an 'About us' page).

The bottom screenshot shows the `Create Basic page` form. The browser address bar is `localhost:8010/node`. The user is logged in as `admin`. The sidebar menu is the same. The main content area is titled `Create Basic page` and features a security update warning. Below the warning, there is a `Title *` field with the value `node`. The `Body (Edit summary)` field contains the text `Hello World 1`.

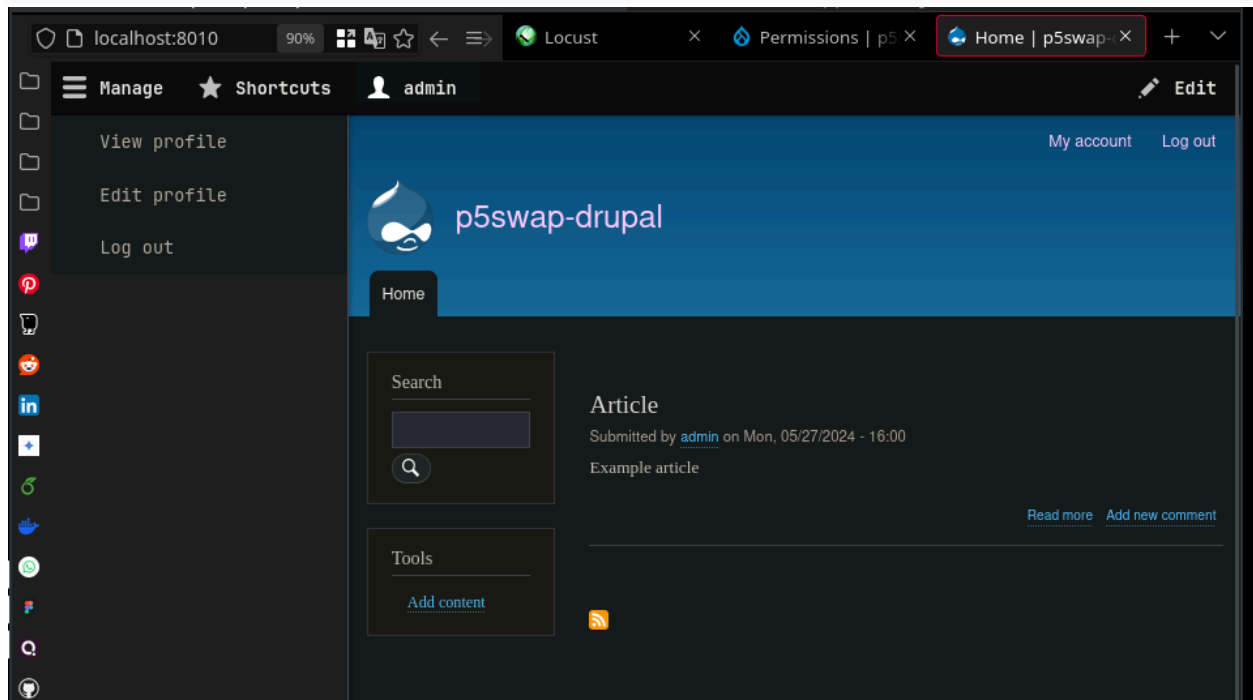


Tras tener totalmente configurado el drupal , podemos proceder con el apartado A1.

Tareas Avanzada - A3. Ejecución y Análisis de cargas de prueba avanzadas sobre CMS

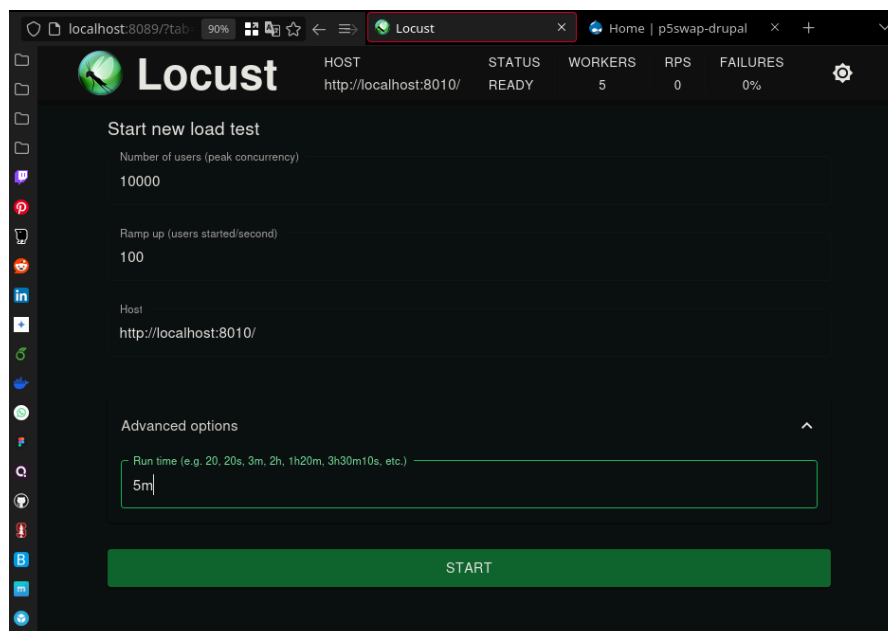
Vamos primero a desplegar nuestro contenedor donde se encuentra nuestro CMS

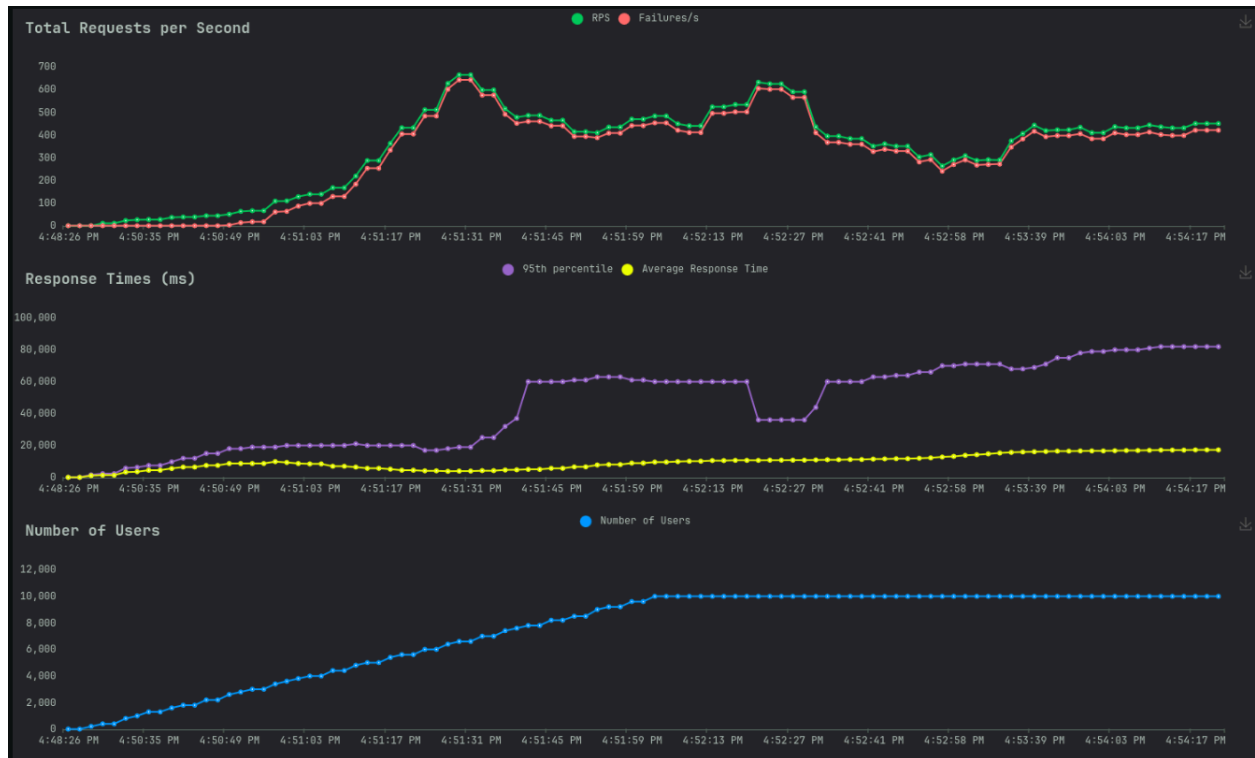
```
docker compose up -d
```



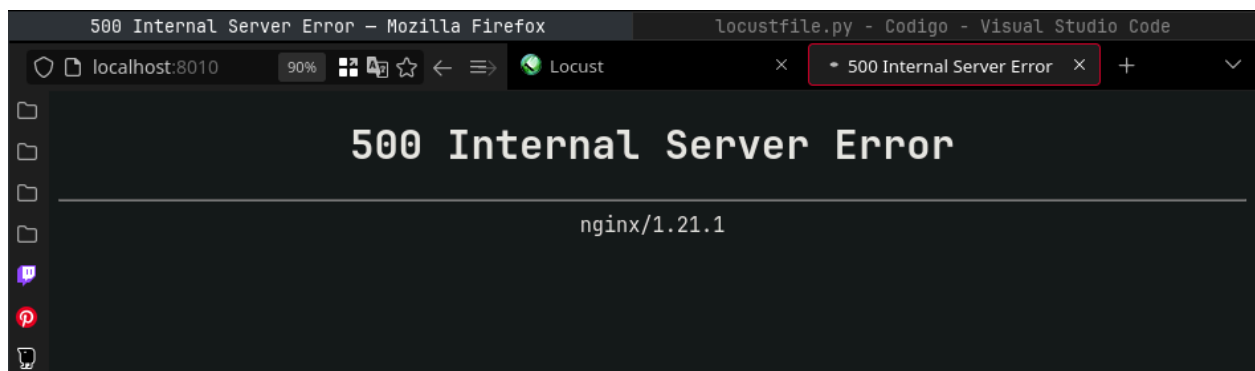
Luego procedemos a desplegar locust. Implementamos 10000 user con una ram up de 100.

```
docker compose up -d
```





Podemos ver que se producen muchas fallas para 10000 usuarios. La tasa de solicitudes aumenta rápidamente al inicio y alcanza un pico alrededor de 600 solicitudes por segundo. Las fallas aumentan a medida que aumenta la carga, coincidiendo con el pico de RPS. Los tiempos de respuesta promedio se mantienen relativamente estables y bajos, alrededor de 20,000 ms. Los tiempos de respuesta en el percentil 95 son mucho más altos, alcanzando hasta 100,000 más cuando hay mucha carga. Podemos deducir que hay un umbral de carga en el cual el sistema no es capaz de manejar y que los tiempos de respuesta en el percentil 95 son malas señales, por lo que no rinde bien a carga muy alta. Si carga la página web durante el benchmarking obtengo un error 500 al acceder al homepage.

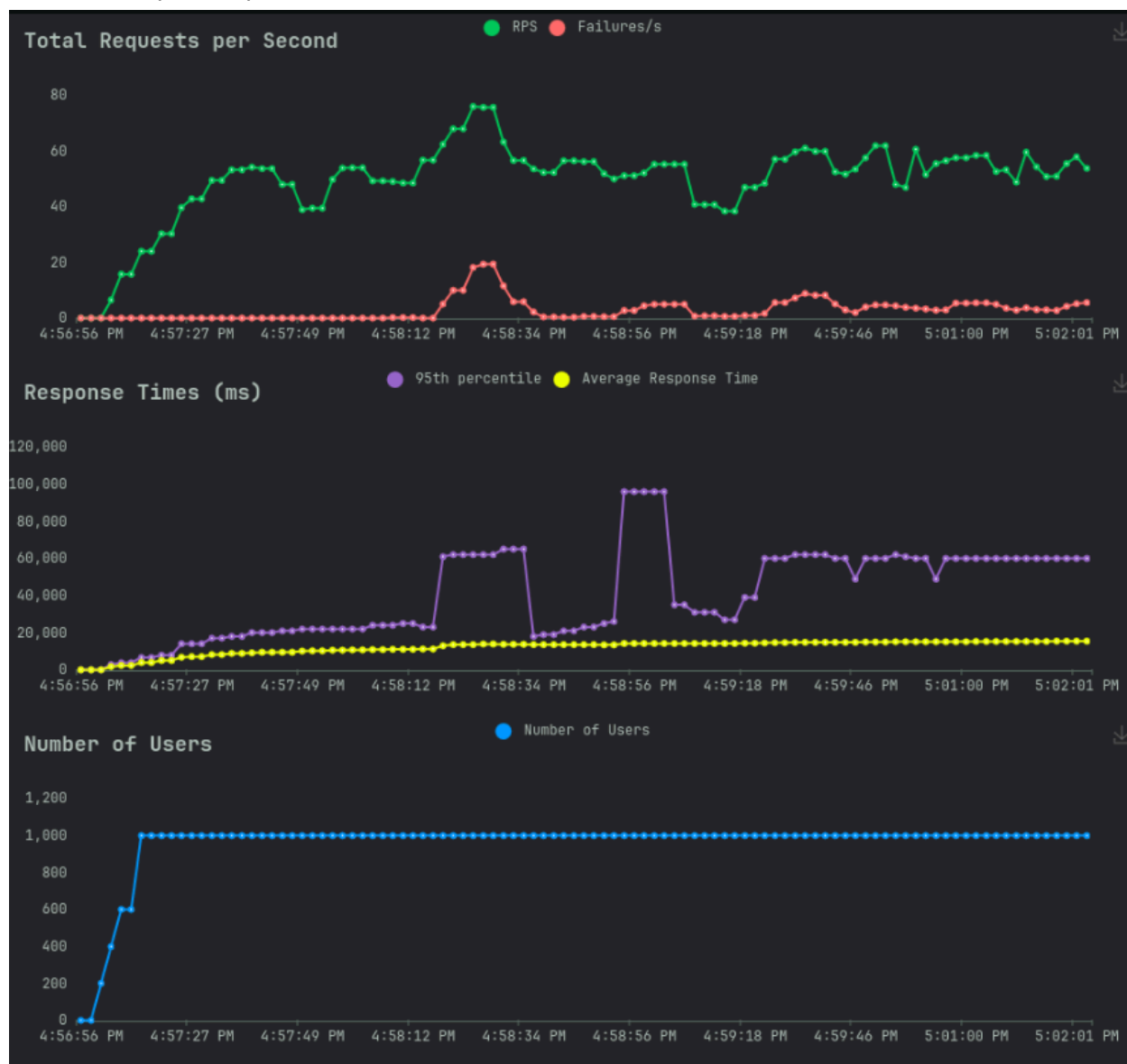


Puedo ver aquí que los errores que se generan durante la carga, son generalmente problemas de Gateway protocol.

The screenshot shows the Locust web interface in a browser window. The browser's address bar shows 'localhost:8089/?tab=90%'. The Locust interface has a top navigation bar with the following tabs: STATISTICS, CHARTS, **FAILURES**, EXCEPTIONS, CURRENT RATIO, DOWNLOAD DATA, LOGS, and WORKERS. The 'FAILURES' tab is selected and highlighted in green. Below the tabs, there is a table of failures. The table has four columns: '# Failures', 'Method', 'Name', and 'Message'. The data rows show various failures, including 504 Gateway Time-out and 502 Bad Gateway errors. The 'Locust' browser tab is highlighted with a red box.

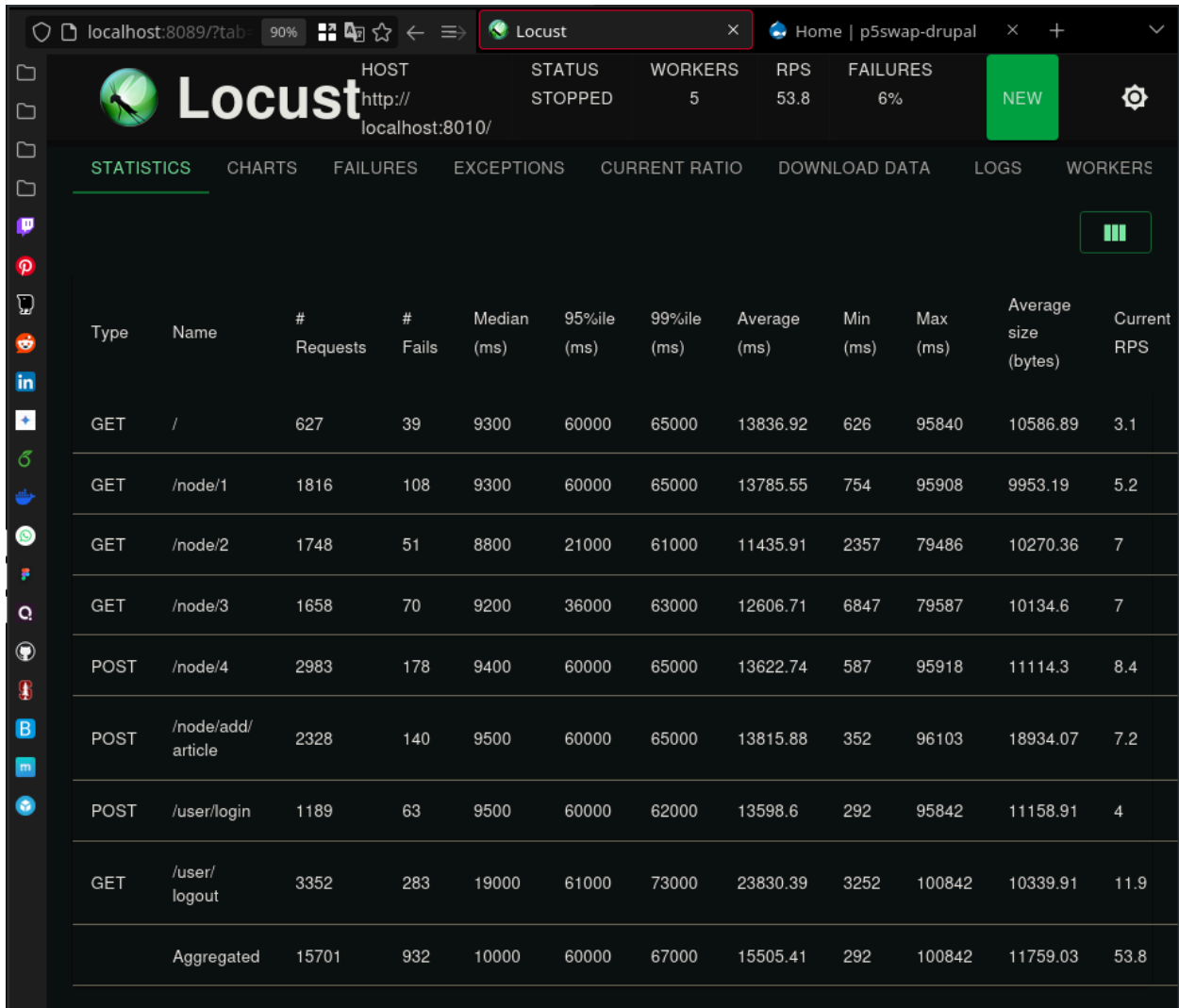
# Failures	Method	Name	Message
6	GET	/	HTTPError('504 Server Error: Gateway Time-out for url: /')
1	GET	/node/1	HTTPError('502 Server Error: Bad Gateway for url: /node/1')
30	GET	/node/1	HTTPError('504 Server Error: Gateway Time-out for url: /node/1')
1	GET	/node/2	HTTPError('504 Server Error: Gateway Time-out for url: /node/2')
1	GET	/node/3	HTTPError('504 Server Error: Gateway Time-out for url: /node/3')
54	POST	/node/4	HTTPError('504 Server Error: Gateway Time-out for url: /node/4')
42	POST	/node/add/article	HTTPError('504 Server Error: Gateway Time-out for url: /node/add/article')
20	POST	/user/login	HTTPError('504 Server Error: Gateway Time-out for url: /user/login')
1	GET	/user/logout	HTTPError('502 Server Error: Bad Gateway for url: /user/logout')
66	GET	/user/logout	HTTPError('504 Server Error: Gateway Time-out for url: /user/logout')

He vuelto a probar pero reduciendo el número de users a 1000, obtengo la siguiente gráfica:



La tasa de solicitudes aumenta al inicio y alcanza un pico alrededor de 70 solicitudes por segundo. Luego se estabiliza entre 40 y 60 solicitudes por segundo. Los tiempos de respuesta promedio se mantienen constantes, alrededor de 20,000 ms. Hay un periodo de varios segundos donde tarda en responder, cuando el percentil es de 95. Deducimos que drupal en su versión base puede manejar más o menos unas 70 solicitudes por segundo, antes que los fallos comiencen, además, en momentos de mayor carga, algunos usuarios experimentan tiempos altos de respuesta (donde se da un percentil de 95)

Aquí dejo las estadísticas que realiza mi locustfile.py. Si se diesen siempre fallos por cada GET o POST significa que la implementación en el locustfile.py estaria mal.



Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS
GET	/	627	39	9300	60000	65000	13836.92	626	95840	10586.89	3.1
GET	/node/1	1816	108	9300	60000	65000	13785.55	754	95908	9953.19	5.2
GET	/node/2	1748	51	8800	21000	61000	11435.91	2357	79486	10270.36	7
GET	/node/3	1658	70	9200	36000	63000	12606.71	6847	79587	10134.6	7
POST	/node/4	2983	178	9400	60000	65000	13622.74	587	95918	11114.3	8.4
POST	/node/add/article	2328	140	9500	60000	65000	13815.88	352	96103	18934.07	7.2
POST	/user/login	1189	63	9500	60000	62000	13598.6	292	95842	11158.91	4
GET	/user/logout	3352	283	19000	61000	73000	23830.39	3252	100842	10339.91	11.9
Aggregated		15701	932	10000	60000	67000	15505.41	292	100842	11759.03	53.8

Como conclusión, debo buscar maneras para optimizar mi CMS grupal. Podría darle más recursos a mis contenedores para que pudiesen soportar más carga o puedo optimizar mi docker-compose. Aquí dejo algunas paginas de interes.

<https://alexanderallen.medium.com/series-part-ii-optimizing-my-alpine-docker-image-for-use-with-drupal-8-x-and-php-7-3-13cfbd3800eb>

<https://alexanderallen.medium.com/series-part-ii-optimizing-my-alpine-docker-image-for-use-with-drupal-8-x-and-php-7-3-13cfbd3800eb>

Analisis IA

un ejemplo de get a mi balanceador: 192.168.10.1 - - [26/May/2024:15:30:00 +0000] "GET / HTTP/1.1" 400 248 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:126.0) Gecko/20100101 Firefox/126.0", pero luego con locust: 819641 GET /index.php ConnectionRefusedError(111, 'Connection refused'), puedo acceder a mi balanceador desde el <https://localhost:4000/>

El problema podría deberse a que Locust, que se está ejecutando dentro de un contenedor Docker, está intentando acceder a localhost, pero dentro del contenedor localhost se refiere al propio contenedor, no a la máquina anfitriona donde está ejecutándose Docker.

Puedes intentar cambiar la dirección de destino a la dirección IP del host. Puedes usar el nombre de host especial host.docker.internal para acceder a los servicios que se ejecutan en el host desde dentro de un contenedor Docker.

Aquí está la modificación de tu script de Locust para utilizar host.docker.internal:

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

class P5JuanLuis(TaskSet):
    @task
    def access_balancer(self):
        self.client.get("https://host.docker.internal:4000/")

class P5Usuarios(HttpUser):
    tasks = [P5JuanLuis]
    wait_time = between(1, 5)
```

Con esta modificación, cada usuario simulado por Locust intentará acceder al balanceador en <https://host.docker.internal:4000/>, que debería dirigir las solicitudes al balanceador que se ejecuta en el host de Docker. Asegúrate de que el balanceador esté configurado para aceptar conexiones desde host.docker.internal.

Aquí tuve un problema, en mi implementación de la granja no puedo acceder a la ip 192.168.10.1 desde fuera a pesar de tenerla creada y asociada a mis servicios. Busque una alternativa para implementar locust para que accediese a la pagina localhost, para ello tiene que acceder a <https://host.docker.internal:4000/> , si pruebo con <http://localhost:4000> no funciona.

```
create this network external networks: red_web: external: True driver:
bridge ipam: driver: default config: - subnet: 192.168.10.0/24
red_servicios: external: True driver: bridge ipam: driver: default config:
- subnet: 192.168.20.0/24
```

Given that the networks are defined as external in your docker-compose.yml, Docker expects these networks to already exist. Since the red_servicios network is defined as external, you need to create it manually if it doesn't already exist.

You can create these networks manually using the docker network create command with the necessary configurations. Here's how you can do it:

Step-by-Step Instructions

Create the red_web Network:

```
bash
docker network create \
  --driver=bridge \
  --subnet=192.168.10.0/24 \
  red_web
```

Create the red_servicios Network:

```
bash
docker network create \
  --driver=bridge \
  --subnet=192.168.20.0/24 \
  red_servicios
```

Verifying the Networks

After creating the networks, you can verify that they exist using the following command:

```
bash
docker network ls
```

You should see both red_web and red_servicios listed.

Example docker-compose.yml

Here's your docker-compose.yml with the external network definitions:

```
yaml
version: '3'
services:
  web:
    image: nginx
    ports:
      - "80:80"
```

```
networks:
  - red_web
  - red_servicios
```

```
networks:
  red_web:
    external: True
```

```
red_servicios:
  external: True
```

Running docker-compose

After creating the networks manually, you can run:

```
bash
```

```
docker-compose up -d
```

Summary

Create the red_web network:

```
bash
```

```
docker network create --driver=bridge --subnet=192.168.10.0/24 red_web
```

Create the red_servicios network:

```
bash
```

```
docker network create --driver=bridge --subnet=192.168.20.0/24
red_servicios
```

Run docker-compose up -d to start your services.

Suggestions

a. Verify the networks' configurations by using docker network inspect red_web and docker network inspect red_servicios.

b. If there are further issues, check the logs with docker-compose logs to identify any specific errors.

Para el paso de hacer mis redes, necesito crearlas desde la terminal, chatgpt me da el comando sin problemas.

<https://www.blazemeter.com/blog/locust-python>

Genera un archivo locustfile que permita realizar pruebas de carga para mi cms drupal. Completa el siguiente locustfile que te he pasado.

Para crear un archivo locustfile más completo que permita realizar pruebas de carga y rendimiento sobre tu CMS de Drupal, puedes definir varias tareas que simulen la navegación por diferentes páginas, la creación e interacción con contenido, y las interacciones con la base de datos, como insertar comentarios o publicaciones. Aquí tienes un ejemplo de cómo puedes hacerlo:

python

locustfile.py

from locust import HttpUser, TaskSet, task, between

class P5JuanLuis(TaskSet):

@task(1)

def access_homepage(self):

self.client.get("/", verify=False)

@task(2)

def navigate_pages(self):

Simulate navigating to a few pages

pages = [

"/node/1", # Assuming this is a valid page in your Drupal CMS

"/node/2",

"/node/3"

]

for page in pages:

self.client.get(page, verify=False)

@task(3)

def create_content(self):

Simulate creating a new content

self.client.post("/node/add/article", {

"title": "Test Article",

"body": "This is a test article created by Locust."

}, verify=False)

@task(4)

def post_comment(self):

Simulate posting a comment

self.client.post("/comment/reply/1", {

"comment_body[0][value]": "This is a test comment.",

"comment_body[0][format]": "plain_text"

}, verify=False)

@task(5)

def login(self):


```

        # Simulate Logging in
        self.client.post("/user/login", {
            "name": "your_username",
            "pass": "your_password",
            "form_id": "user_login_form"
        }, verify=False)

    @task(6)
    def logout(self):
        # Simulate Logging out
        self.client.get("/user/logout", verify=False)

class P5Usuarios(HttpUser):
    tasks = [P5JuanLuis]
    wait_time = between(1, 5)

# Iniciar la prueba con el siguiente comando en el directorio donde está tu
locustfile.py
# locust -f locustfile.py --host=http://host.docker.internal:8010

```

Descripción de las tareas:

access_homepage: Accede a la página principal del CMS.

navigate_pages: Navega a través de algunas páginas de contenido predefinido.

create_content: Simula la creación de un nuevo artículo de contenido.

post_comment: Simula la publicación de un comentario en un contenido existente.

login: Simula el inicio de sesión de un usuario.

logout: Simula el cierre de sesión de un usuario.

Cómo ejecutar:

Guarda este código en un archivo llamado locustfile.py.

En el directorio donde está guardado el archivo, ejecuta el siguiente comando para iniciar las pruebas con Locust:

```
sh
```

```
locust -f locustfile.py --host=http://host.docker.internal:8010
```

Sugerencias para la siguiente iteración:

a. Añadir más tareas específicas según las funcionalidades de tu CMS de Drupal, como la actualización de contenido existente. b. Implementar pruebas con usuarios autenticados y no autenticados para cubrir más casos de uso.

Para una primera aproximación a implementar el locustfile.py para testear mi CMS, he decidido partir de un ejemplo que me da chat gpt. He tenido que hacer varias modificaciones pero chat gpt me da una base sólida donde partir para implementar mi locustfile

Por lo demás, también he usado chatgpt para comentar código y a la hora de implementar el CMS ya que lo he intentado usando Wordpress o Moodle pero he tenido varios fallos a la hora de que la página se conectará al database. Al final opte por usar un repositorio github de que usaba drupal.

Nota

Lo hago en localhost por que con la ip no puedo acceder, tengo, por ejemplo si uso el apache benchmark obtengo el siguiente error:

```
> docker compose up
[*] Running 1/0
✓ Container apache_benchmark-P5 Created 0.1s
Attaching to apache_benchmark-P5
apache_benchmark-P5 | This is ApacheBench, Version 2.3 <$Revision: 1913912 $>
apache_benchmark-P5 | Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
apache_benchmark-P5 | Licensed to The Apache Software Foundation, http://www.apache.org/
apache_benchmark-P5 |
apache_benchmark-P5 | Benchmarking 192.168.10.50 (be patient)
apache_benchmark-P5 | SSL read failed (5) - closing connection
apache_benchmark-P5 | SSL read failed (5) - closing connection
apache_benchmark-P5 | Completed 1000 requests
apache_benchmark-P5 | SSL read failed (5) - closing connection
apache_benchmark-P5 | SSL read failed (5) - closing connection
apache_benchmark-P5 | Completed 2000 requests
apache_benchmark-P5 | SSL read failed (5) - closing connection
apache_benchmark-P5 | Completed 3000 requests
```

tuve una tutoría con Juan Luis , comprobé que la asociación a los servicios sobre la ip lo hacía correctamente. Recuerdo que en tutoría (28-05-2024) con cambiar localhost por <https://192.168.10.50:443/> funcionaba correctamente, pero a la hora de entregarte esto me vuelve a dar fallo. Como lo importante de la práctica es hacer benchmarking al final lo he dejado con localhost con la parte comentada del uso de la ip