

8. Optimización de Configuraciones

Para mejorar el rendimiento, la organización y la seguridad de los contenedores y servicios usando buenas prácticas,.

EJERCICIO 3

Vamos a modificar el archivo docker-compose.yml:

```
Sprint-1 > Ejercicio-3 > docker-compose.yml

  ▶ Run All Services
2  services:
  ▶ Run Service
3    nginx:
4      deploy:
5          resources:
6              limits:
7                  cpus: '1.0'
8                  memory: 512M
9
10     image: nginx:alpine
11     container_name: nginx
12     ports:
13         - "8080:80"
14     depends_on:
15         - php
16     networks:
17         - network
18
  ▶ Run Service
19    php:
20        deploy:
21            resources:
22                limits:
23                    cpus: '0.50'
24                    memory: 512M
25        image: php:8.1-apache
26        container_name: php
27        networks:
28            - network
29
  ▶ Run Service
30    mysql:
31        deploy:
32            resources:
33                limits:
34                    cpus: '0.50'
35                    memory: 512M
36        image: mysql:8.0
37        container_name: mysql
38        volumes:
39            - mysql-data:/var/lib/mysql
40        environment:
41            - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
42            - MYSQL_DATABASE=${MYSQL_DATABASE}
43            - MYSQL_USER=${MYSQL_USER}
44            - MYSQL_PASSWORD=${MYSQL_PASSWORD}
45        networks:
46            - network
47
```

1. Limitar recursos de CPU y memoria de los contenedores

Para mejorar el rendimiento y evitar que los contenedores usen más recursos de los necesarios, añadimos límites de CPU y memoria con:

```
deploy:
  resources:
    limits:
      cpus: '1.0'
      memory: 512M
```

Este bloque se añadirá a los contenedores nginx, php y mysql.

2. Utilizar imagen ligera alpine

Usamos la imagen alpine en el contenedor nginx para reducir el tamaño del contenedor y mejorar su rendimiento.

```
image: nginx:alpine
container_name: nginx
```

3. Crear archivo .env

Creamos un archivo .env para el contenedor mysql dónde definiremos las variables de entornos sensibles, y evitamos no exponerlas en el archivo docker-compose.yml.

```
Sprint-1 > Ejercicio-3 > .env
1  MYSQL_ROOT_PASSWORD: rootpass
2  MYSQL_DATABASE: mydb
3  MYSQL_USER: myuser
4  MYSQL_PASSWORD: password
5
```

4. Vamos a crear un archivo README.md para documentar los servicios y configuraciones

```
Sprint-1 > Ejercicio-3 > README.md > # Ejercicio 3: Docker Compose Avanzado > ## Configuración > ### .env
1  # Ejercicio 3: Docker Compose Avanzado
2
3
4  ## Descripción
5  En este ejercicio se ha creado un archivo `docker-compose.yml` que define varios servicios relacionados.
6
7  El objetivo es estructurar una aplicación multi-servicio que funcione de forma coordinada.
8
9  ## Servicios
10
11 - **PHP + Apache**: Servidor web con PHP 8.1 con Apache
12 - **MySQL 8.0**: Base de datos
13 - **Nginx**: Servidor proxy inverso
14
15 ## Configuración
16
17 ### `.env`
18 Archivo con las variables de entorno sensibles utilizadas por el contenedor MySQL:
19
20 ```env
21 MYSQL_ROOT_PASSWORD=rootpass
22 MYSQL_DATABASE=mydb
23 MYSQL_USER=myuser
24 MYSQL_PASSWORD=password
25 ```
```

EJERCICIO 4

Vamos a modificar el archivo docker-compose.yml:

```
Sprint-1 > Ejercicio-4 > docker-compose.yml
1  version: "3.8"
   ▶ Run All Services
2  services:
   ▶ Run Service
3    nginx:
4      deploy:
5        resources:
6          limits:
7            cpus: '1.0'
8            memory: 512M
9      image: nginx:alpine
10     container_name: nginx
11     ports:
12       - "8080:80"
13     volumes:
14       - ./www:/var/www/html
15       - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
16     depends_on:
17       - php
18     networks:
19       - network
20
   ▶ Run Service
21   php:
22     deploy:
23       resources:
24         limits:
25           cpus: '1.0'
26           memory: 512M
27     image: php:8.1-fpm-alpine
28     container_name: php
29     volumes:
30       - ./www:/var/www/html
31     networks:
32       - network
33
34   networks:
35     network:
36       name: network
37
```

1. Limitar recursos de CPU y memoria de los contenedores

Para mejorar el rendimiento y evitar que los contenedores usen más recursos de los necesarios, añadimos límites de CPU y memoria con:

```

deploy:
  resources:
    limits:
      cpus: '1.0'
      memory: 512M

```

Este bloque se añadirá a los contenedores nginx y php.

2. Utilizar imagen ligera alpine

Usamos la imagen alpine en los contenedores nginx y php para reducir el tamaño del contenedor y mejorar su rendimiento.

```

image: nginx:alpine
container_name: nginx

```

```

image: php:8.1-fpm-alpine
container_name: php

```

3. Creamos un archivo README.md para documentar los servicios y configuraciones

```

Sprint-1 > Ejercicio-4 > README.md > # Ejercicio 4: Configurar un contenedor para servir una aplicación PHP + Nginx > ## Servicios
1  # Ejercicio 4: Configurar un contenedor para servir una aplicación PHP + Nginx
2
3
4  ## Descripción
5  En este ejercicio se ha creado un archivo `docker-compose.yml` que define varios servicios relacionados.
6
7  El objetivo es crear una aplicación web con PHP utilizando Nginx como servidor web.
8
9  ## Estructura
10 Nuestro ejercicio va a tener la siguiente estructura de archivos y carpetas:
11 ```
12 Ejercicio-4/
13 |__ docker-compose.yml
14 |__ README.md
15 |__ www/
16 |   |__ index.php
17 |   |__ nginx/
18 |   |__ default.conf
19 |   ```
20
21 ## Servicios
22 - **Nginx**: Servidor proxy inverso
23 - **PHP-FPM**: Servidor web con PHP-FPM 8.1
24
25 ## Configuración
26 s
27 ### `default.conf`
28 Archivo de configuración de Nginx para que funcione con PHP-FPM:
29
30 ```default.conf
31 server {
32     listen 80;
33     server_name localhost;
34     root /var/www/html;
35     index index.php index.html;
36
37     location / {
38         try_files $uri $uri/ =404;
39     }
40
41     location ~ \.php$ {
42         include fastcgi_params;
43         fastcgi_pass php:9000;
44         fastcgi_param SCRIPT_FILENAME /var/www/html$fastcgi_script_name;
45     }
46 }
47 ```

```

```

48
49  ### `index.php`
50  Archivo de prueba para verificar que Nginx y PHP-FPM están funcionando de manera correcta:
51
52  ``index.php
53  <?php
54  phpinfo();
55  ?>
56  ``
57

```

EJERCICIO 6

Vamos a modificar el archivo docker-compose.yml:

```

Sprint-1 > Ejercicio-6 > docker-compose.yml
1  version: "3.8"
   ▶ Run All Services
2  services:
   ▶ Run Service
3    prometheus:
4      deploy:
5        resources:
6          limits:
7            cpus: '1.0'
8            memory: 512M
9      image: prom/prometheus:latest
10     container_name: prometheus
11     ports:
12       - "9090:9090"
13     volumes:
14       - ./prometheus.yml:/etc/prometheus/prometheus.yml
15     networks:
16       - monitoring
17
   ▶ Run Service
18   cadvisor:
19     deploy:
20       resources:
21         limits:
22           cpus: '1.0'
23           memory: 512M
24     image: gcr.io/cadvisor/cadvisor:latest
25     container_name: cadvisor
26     ports:
27       - "8080:8080"
28     volumes:
29       - /:/rootfs:ro
30       - /var/run:/var/run:rw
31       - /sys:/sys:ro
32       - /dev/disk/":"/dev/disk:ro
33     networks:
34       - monitoring
35
36   networks:
37     monitoring:
38       driver: bridge
39

```

1. Limitar recursos de CPU y memoria de los contenedores

Para mejorar el rendimiento y evitar que los contenedores usen más recursos de los necesarios, añadimos límites de CPU y memoria con:

```
deploy:
  resources:
    limits:
      cpus: '1.0'
      memory: 512M
```

Este bloque se añadirá a los contenedores prometheus y cadvisor.

2. Creamos un archivo README.md para documentar los servicios y configuraciones

```
Sprint-1 > Ejercicio-6 > README.md > Ejercicio 6: Monitorización Básica con Prometheus > Configuración > prometheus.yml
1 | # Ejercicio 6: Monitorización Básica con Prometheus
2 |
3 | ## Descripción
4 | En este ejercicio se ha creado un archivo 'docker-compose.yml' que define varios servicios relacionados.
5 |
6 | El objetivo es configurar Prometheus para recolectar métricas de tus servicios en contenedores.
7 |
8 | ## Servicios
9 |
10 | - **Prometheus**: Herramienta de monitorización
11 | - **cAdvisor**: Herramienta que se utiliza para monitorear el uso de recursos y las características de rendimiento de contenedores en ejecución.
12 |
13 | ## Configuración
14 |
15 | ### 'prometheus.yml'
16 | Archivo de configuración para que Prometheus monitorice a cAdvisor y a sí mismo:
17 |
18 | ```prometheus.yml
19 | global:
20 |   scrape_interval: 15s
21 |
22 | scrape_configs:
23 |   - job_name: 'prometheus'
24 |     static_configs:
25 |       - targets:
26 |         - 'prometheus:9090'
27 |
28 |   - job_name: 'cadvisor'
29 |     static_configs:
30 |       - targets:
31 |         - 'cadvisor:8080'
32 | ```
```


EJERCICIO 7

Vamos a modificar el archivo docker-compose.yml:

```
Sprint-1 > Ejercicio-7 > 🐳 docker-compose.yml
1  version: "3.8"
   ▶ Run All Services
2  services:
   ▶ Run Service
3    prometheus:
4      deploy:
5        resources:
6          limits:
7            cpus: '1.0'
8            memory: 512M
9      image: prom/prometheus:latest
10     container_name: prometheus
11     ports:
12       - "9090:9090"
13     volumes:
14       - ./prometheus.yml:/etc/prometheus/prometheus.yml
15     networks:
16       - monitoring
17
18   ▶ Run Service
19   cadvisor:
20     deploy:
21       resources:
22         limits:
23           cpus: '1.0'
24           memory: 512M
25     image: gcr.io/cadvisor/cadvisor:latest
26     container_name: cadvisor
27     ports:
28       - "8080:8080"
29     volumes:
30       - /:/rootfs:ro
31       - /var/run:/var/run:rw
32       - /sys:/sys:ro
33       - /dev/disk/:/dev/disk:ro
34     networks:
35       - monitoring
36
37   ▶ Run Service
38   grafana:
39     deploy:
40       resources:
41         limits:
42           cpus: '1.0'
43           memory: 512M
44     image: grafana/grafana:latest
45     container_name: grafana
46     ports:
47       - "3000:3000"
48     volumes:
49       - grafana_data:/var/lib/grafana
50     networks:
51       - monitoring
52
53   volumes:
54     grafana_data:
55
56   networks:
57     monitoring:
58       driver: bridge
```


1. Limitar recursos de CPU y memoria de los contenedores

Para mejorar el rendimiento y evitar que los contenedores usen más recursos de los necesarios, añadimos límites de CPU y memoria con:

```
deploy:
  resources:
    limits:
      cpus: '1.0'
      memory: 512M
```

Este bloque se añadirá a los contenedores prometheus, cadvisor y grafana.

2. Creamos un archivo README.md para documentar los servicios y configuraciones

```
Sprint-1 > Ejercicio-7 > ① README.md > ② # Ejercicio 7: Instalación de Grafana > ③ ## Configuración > ④ ### prometheus.yml
1 | # Ejercicio 7: Instalación de Grafana
2 |
3 | ## Descripción
4 | En este ejercicio se ha creado un archivo `docker-compose.yml` que define varios servicios relacionados.
5 |
6 | El objetivo es configurar Grafana para construir una vista visual del rendimiento de tu entorno en tiempo real.
7 |
8 | ## Servicios
9 |
10 | - **Prometheus**: Herramienta de monitorización
11 | - **cAdvisor**: Herramienta que se utiliza para monitorear el uso de recursos y las características de rendimiento de contenedores en ejecución.
12 | - **Grafana**: Plataforma que se utiliza para la visualización de datos, análisis y monitoreo de infraestructuras de TI.
13 |
14 | ## Configuración
15 |
16 | ### `prometheus.yml`
17 | Archivo de configuración para que Prometheus monitorice a cAdvisor y a sí mismo:
18 |
19 | ```prometheus.yml
20 | global:
21 |   scrape_interval: 15s
22 |
23 | scrape_configs:
24 |   - job_name: 'prometheus'
25 |     static_configs:
26 |       - targets:
27 |         - 'prometheus:9090'
28 |
29 |   - job_name: 'cadvisor'
30 |     static_configs:
31 |       - targets:
32 |         - 'cadvisor:8080'
33 | ```
```