



# HERRAMIENTAS INFORMATICAS AVANZADAS

*Trabajo practico Final Integrador*

Alumnos:  
Velazco, Nicolás  
Ortega, Cristian  
Valero, Fernando

# PRACTICA INTEGRADORA HERRAMIENTAS INFORMÁTICAS AVANZADAS

**OBJETIVO:** Este trabajo práctico integrador está diseñado para evaluar el manejo de herramientas avanzadas en la implementación, despliegue y mantenimiento de aplicaciones web.

## 1. Selección de la aplicación web

**Tarea:** Seleccionar una aplicación web previamente desarrollada (por ejemplo, un proyecto académico). Identificar el stack tecnológico utilizado:

- Backend: Node.js, Spring, Django, etc.
- Frontend: Angular, React, etc.
- Base de datos: PostgreSQL, MySQL, MongoDB, etc.
- Si la aplicación no está en un repositorio remoto, utiliza Git para inicializar y subirla (GitHub, GitLab, etc.)

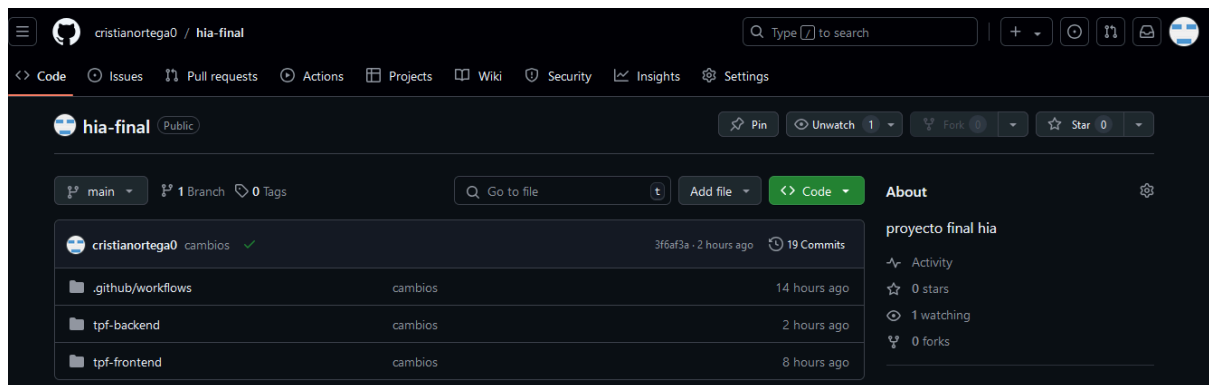
La aplicación elegida es la realizada en Programación y Servicios Web: Sistema de gestión de alquiler de locales comerciales.

Las características de la aplicación son:

Backend: Node.js

Frontend: Angular

Base de datos: MySQL



Dirección del repositorio: <https://github.com/cristianortega0/hia-final.git>

## 2. Actividad 1: Contenedores para servicios y base de datos

**Objetivo:** Dockerizar la aplicación. Para esto:

### 1. Base de datos:

- Crear un contenedor para la base de datos (p.ej., PostgreSQL o MySQL).
- Configurar un cliente dockerizado (p.ej., pgAdmin para PostgreSQL o phpMyAdmin para MySQL).
- Migrar los datos existentes al contenedor.
- Backups automatizados: Configura un contenedor adicional para ejecutar backups automáticos de la base de datos. Usa herramientas como pg\_dump (PostgreSQL) o mysqldump (MySQL)

A continuación se muestran los contenedores creados mediante el docker-compose.yml

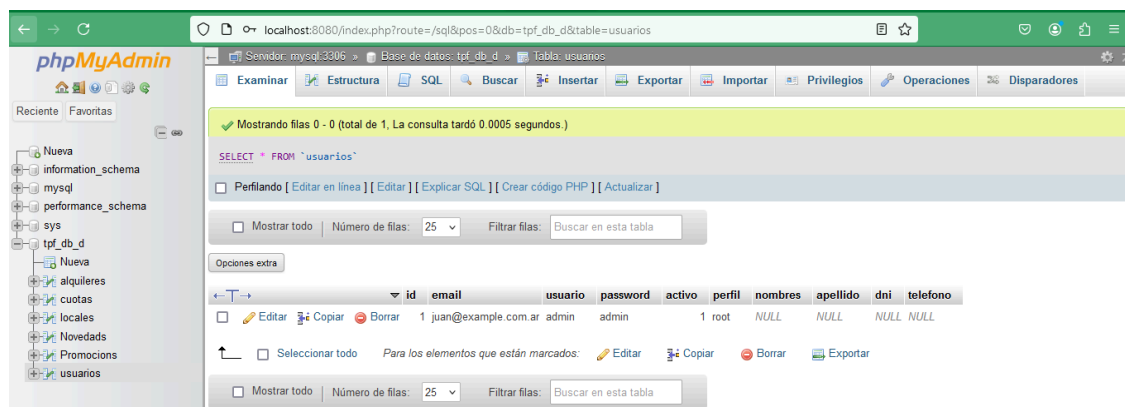
Contenedor para la base de datos

```
mysql:
  image: mysql:latest # Imagen oficial de MySQL
  environment:
    MYSQL_ROOT_PASSWORD: root # Contraseña del root
    MYSQL_DATABASE: tpf_db_d # Nombre de la base de datos
  volumes:
    - mysql-data:/var/lib/mysql # Volumen persistente para los datos
    - ./backup.sql:/docker-entrypoint-initdb.d/backup.sql # Montar
el archivo SQL para restaurar
  networks:
    - my-network
  ports:
    - "3307:3306" # Exponer el puerto 3306 para conexiones externas
```

## Contenedor cliente phpMyAdmin

```
phpmyadmin:
  image: phpmyadmin/phpmyadmin:latest # Imagen oficial de phpMyAdmin
  container_name: phpmyadmin
  environment:
    PMA_HOST: mysql # Nombre del servicio MySQL definido arriba
    PMA_PORT: 3306 # Puerto del servicio MySQL
    MYSQL_ROOT_PASSWORD: root # Contraseña del root
  depends_on:
    - mysql
  networks:
    - my-network
  ports:
    - "8080:80" # Puerto para acceder a phpMyAdmin desde el
navegador
```

En la siguiente imagen se muestra el acceso desde phpmyadmin



Para migrar los datos de una base existente al contenedor, se lo puede realizar con la linea

```
- ./backup.sql:/docker-entrypoint-initdb.d/backup.sql
```

dentro de la creación del contenedor en docker-compose.yml

En caso de hacerlo en forma manual se seguirán los siguientes pasos:

Usar el comando **docker cp** para copiar el archivo **backup.sql** desde la máquina local al contenedor de MySQL:

```
docker cp C:/Users/Cristian/Downloads/tpf-hia/backup.sql tpf-hia-mysql-1:/backup.sql
```

Se accede al contenedor

```
docker exec -it tpf-hia-mysql-1 bash
```

Dentro del contenedor se ejecuta el comando para restaurar la base de datos

```
mysql -u root -proot tpf_db_d < /backup.sql
```

Contenedor para el backup

```
backup:
  image: mysql:latest # Usa la misma imagen de MySQL
  container_name: mysql-backup
  volumes:
    - mysql-backups:/backups # Volumen Docker para almacenar backups
    - ./backup.sh:/usr/local/bin/backup.sh # Monta el script en la
ruta correcta
  depends_on:
    - mysql
  entrypoint: ["/bin/bash", "-c", "chmod +x /usr/local/bin/backup.sh
&& while true; do /usr/local/bin/backup.sh; sleep 84600; done"]
  networks:
    - my-network
```

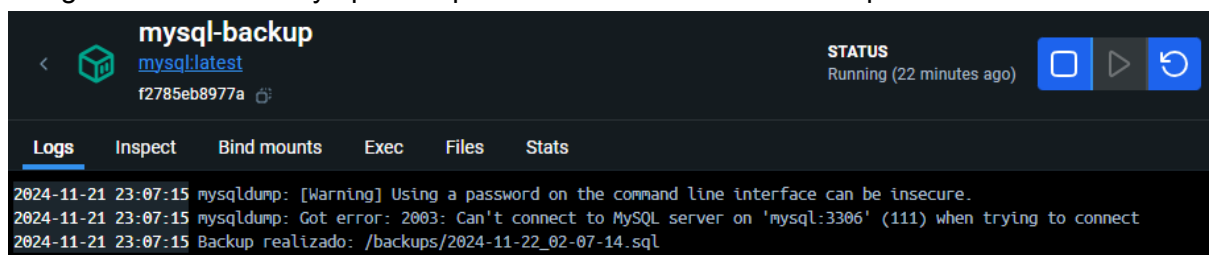
Script para configurar el backup automáticamente

```
#!/bin/bash
# Directorio de destino para los backups
BACKUP_DIR=/backups
# Formato del nombre del archivo de backup (YYYY-MM-DD_HH-MM-SS.sql)
FILE_NAME=$(date +"%Y-%m-%d_%H-%M-%S").sql

# Comando de backup con mysqldump
mysqldump -h mysql -u root -pPVisual23/9- tpf_db_d > $BACKUP_DIR/$FILE_NAME

echo "Backup realizado: $BACKUP_DIR/$FILE_NAME"
```

El log del contenedor mysql-backup muestra la creación del backup



**mysql-backup**  
mysql:latest  
f2785eb8977a

**STATUS**  
Running (22 minutes ago)


**Logs** | Inspect | Bind mounts | Exec | Files | Stats

```
2024-11-21 23:07:15 mysqldump: [Warning] Using a password on the command line interface can be insecure.
2024-11-21 23:07:15 mysqldump: Got error: 2003: Can't connect to MySQL server on 'mysql:3306' (111) when trying to connect
2024-11-21 23:07:15 Backup realizado: /backups/2024-11-22_02-07-14.sql
```

El mensaje Backup realizado /backups/2024-11-22\_02-07-14.sql indica que se realiza el backup

Entrando al contenedor se ven los archivos sql creados de backup

```
C:\Users\Cristian\Downloads\tpf-hia>docker exec -it mysql-backup bash
bash-5.1# cd backups/
bash-5.1# ls
2024-11-22_02-38-30.sql  2024-11-22_02-40-45.sql  2024-11-22_02-43-01.sql  2024-11-22_02-45-19.sql
2024-11-22_02-38-45.sql  2024-11-22_02-41-00.sql  2024-11-22_02-43-16.sql  2024-11-22_02-45-34.sql
2024-11-22_02-39-00.sql  2024-11-22_02-41-15.sql  2024-11-22_02-43-31.sql  2024-11-22_02-45-50.sql
2024-11-22_02-39-15.sql  2024-11-22_02-41-30.sql  2024-11-22_02-43-46.sql  2024-11-22_02-46-05.sql
2024-11-22_02-39-30.sql  2024-11-22_02-41-45.sql  2024-11-22_02-44-03.sql  2024-11-22_02-46-20.sql
2024-11-22_02-39-45.sql  2024-11-22_02-42-01.sql  2024-11-22_02-44-18.sql  2024-11-22_02-46-35.sql
2024-11-22_02-40-00.sql  2024-11-22_02-42-16.sql  2024-11-22_02-44-34.sql  2024-11-22_02-46-51.sql
2024-11-22_02-40-15.sql  2024-11-22_02-42-31.sql  2024-11-22_02-44-49.sql  2024-11-22_02-47-06.sql
2024-11-22_02-40-30.sql  2024-11-22_02-42-46.sql  2024-11-22_02-45-04.sql
bash-5.1#
```

 **mysql-restore**  
mysql:latest  
589994a719fe

STATUS  
Exited (0) (31 seconds ago)

Logs   Inspect   Bind mounts   Exec   Files   Stats

2024-11-24 04:30:16 Restauración completada  
2024-11-24 04:30:04 mysql: [Warning] Using a password on the command line interface can be insecure.

### 3. Actividad 2: Contenedores para servicios web

**Objetivo:** Dockerizar la aplicación principal (backend, frontend) y exponerla para que sea accesible desde el navegador.

- Configura contenedores para cada componente:
  - Frontend: Angular/React.
  - Backend: Node.js/Spring.
- Despliegue: Configura los puertos y dependencias en el docker-compose.yml.
- Pruebas: Asegúrate de que la aplicación funcione correctamente con datos mínimos cargados.
- Integre herramientas como Prometheus y Grafana para monitorear métricas en tiempo real (CPU, memoria, etc.).

Tanto el frontend como el backend se dockerizaron mediante el archivo docker-compose.yml

Para la creación se usa la imagen de cada uno que se encuentra en una cuenta docker hub. El frontend se expone a través del puerto 4200 escuchando el puerto 80. El puerto 80 es el puerto de escucha de Nginx donde se alojó el frontend

```
frontend:
  image: cristian241/frontend-app:latest # Imagen en Docker Hub
  ports:
    - "4200:80" # Mapea el puerto 4200
  networks:
    - my-network
```

El backend se expone a través del puerto 3000 escuchando el puerto 3306 que es el de mysql

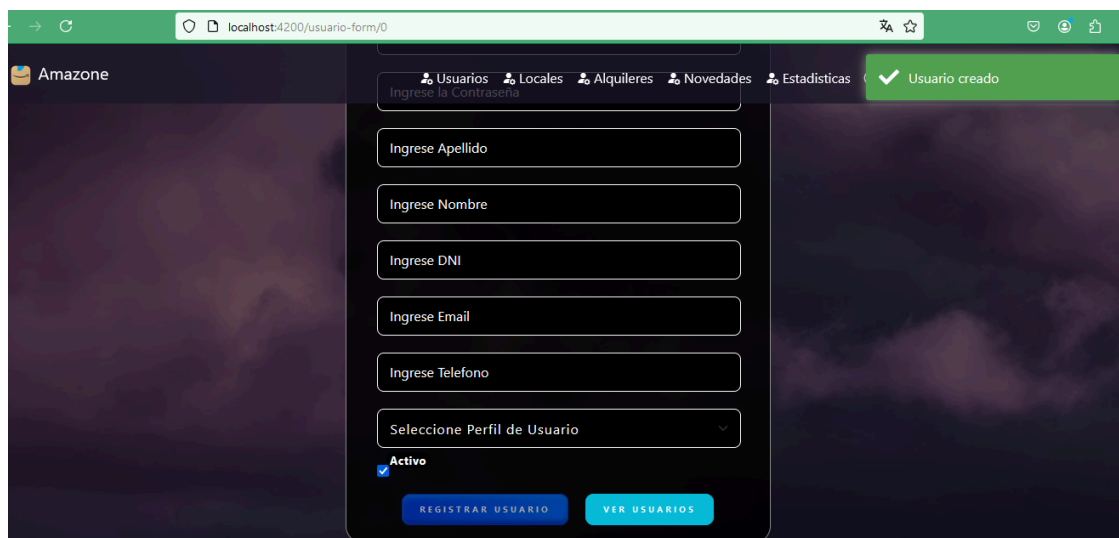
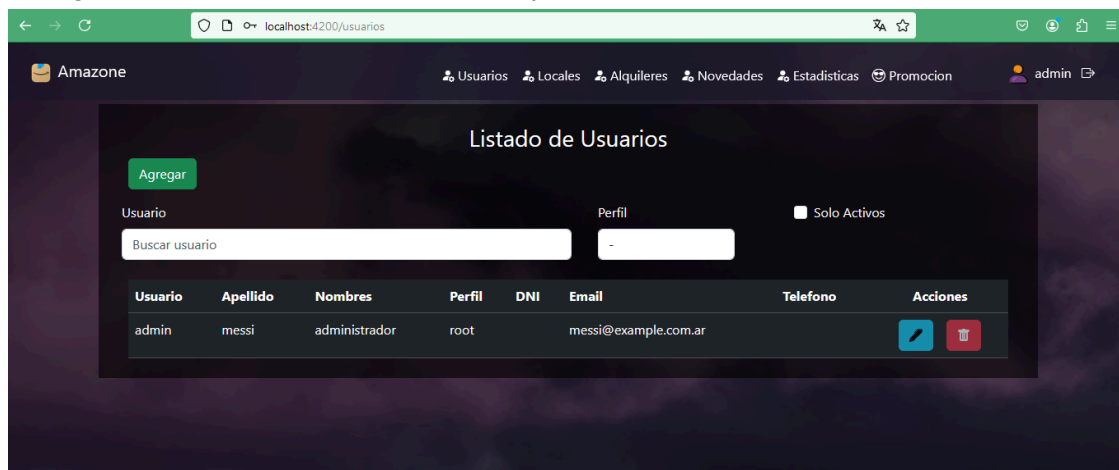
```

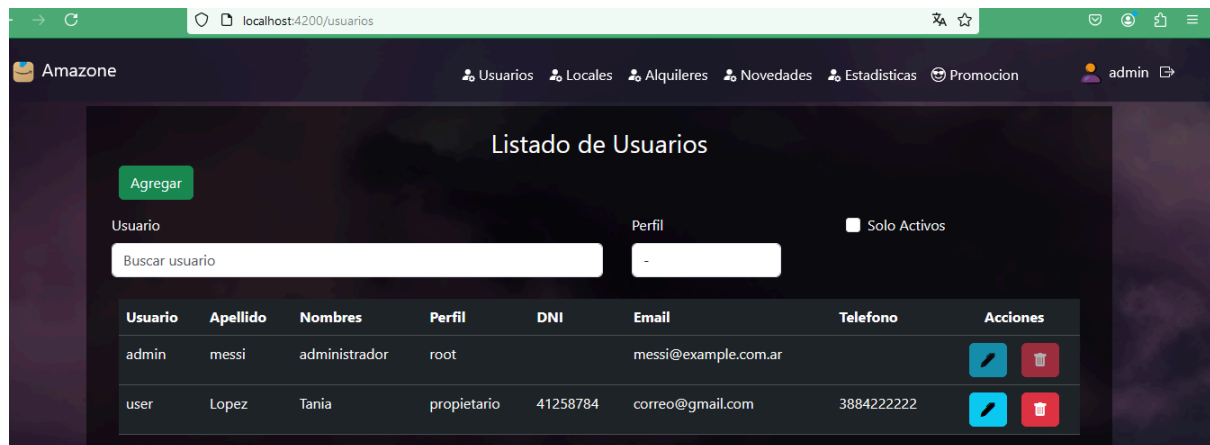
backend:
  image: cristian241/backend-app:latest # Imagen en Docker Hub
  environment:
    - NODE_ENV=production
    - DB_HOST=mysql
    - DB_PORT=3306
    - DB_USER=root
    - DB_PASSWORD=root
    - DB_NAME=tpf_db_d
  depends_on:
    - mysql
  networks:
    - my-network
  ports:
    - "3000:3000"

```

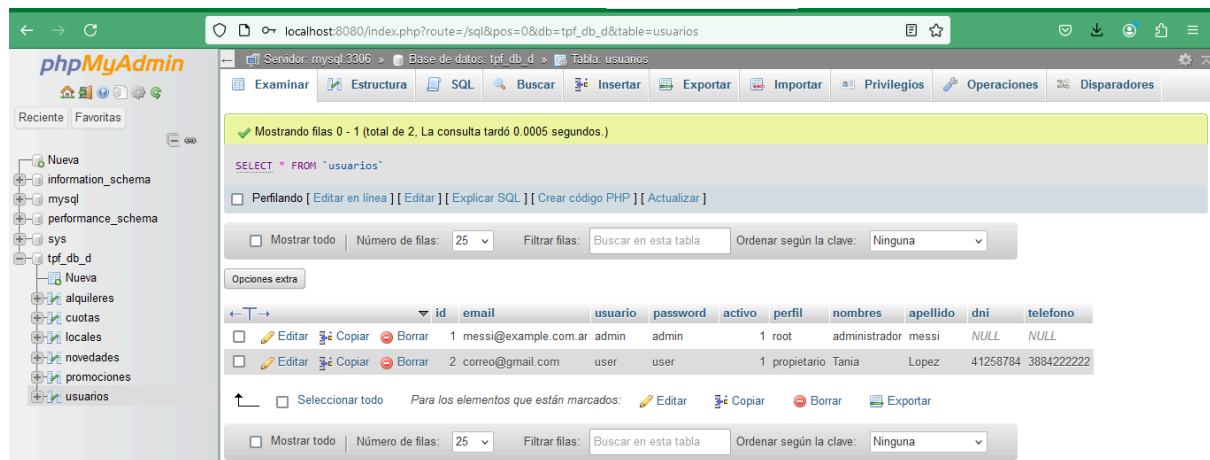
Prueba de funcionamiento.

Se ingresa con cuenta de administrador y se crea un usuario propietario





Estos datos también se reflejan en la base de datos



Para la monitorización de las métricas también se crearon contenedores de Prometheus, Grafana en el docker-compose.yml. Además de los contenedores Cadvisor y MysqlExporter

```
prometheus:
  image: prom/prometheus:latest
  container_name: prometheus
  volumes:
    - ./prometheus.yml:/etc/prometheus/prometheus.yml #
Configuración de Prometheus
  command:
    - '--config.file=/etc/prometheus/prometheus.yml'
  ports:
    - "9090:9090" # Puerto de Prometheus
  networks:
    - my-network
  restart: unless-stopped

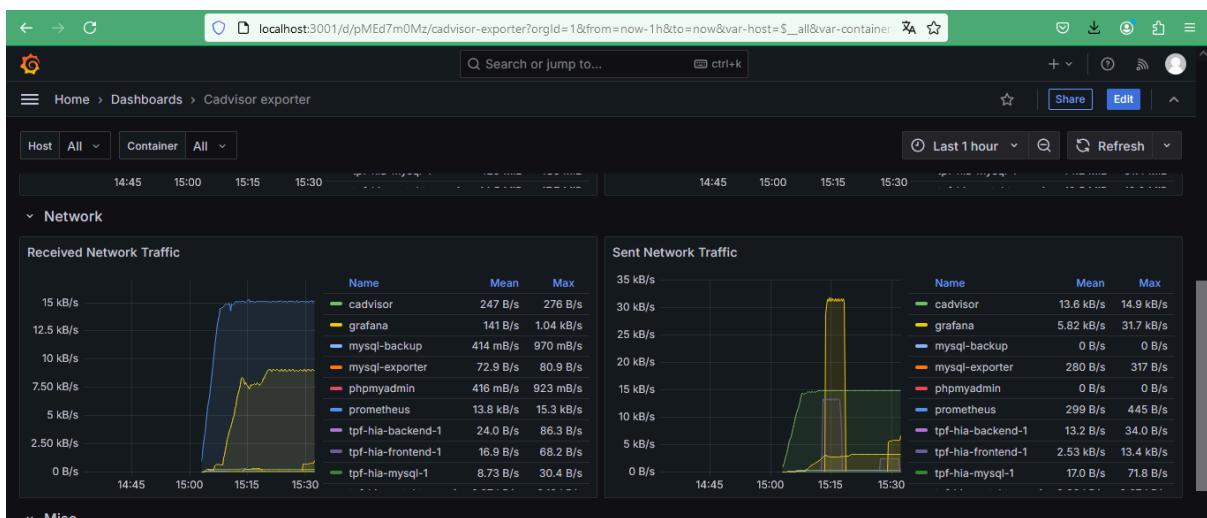
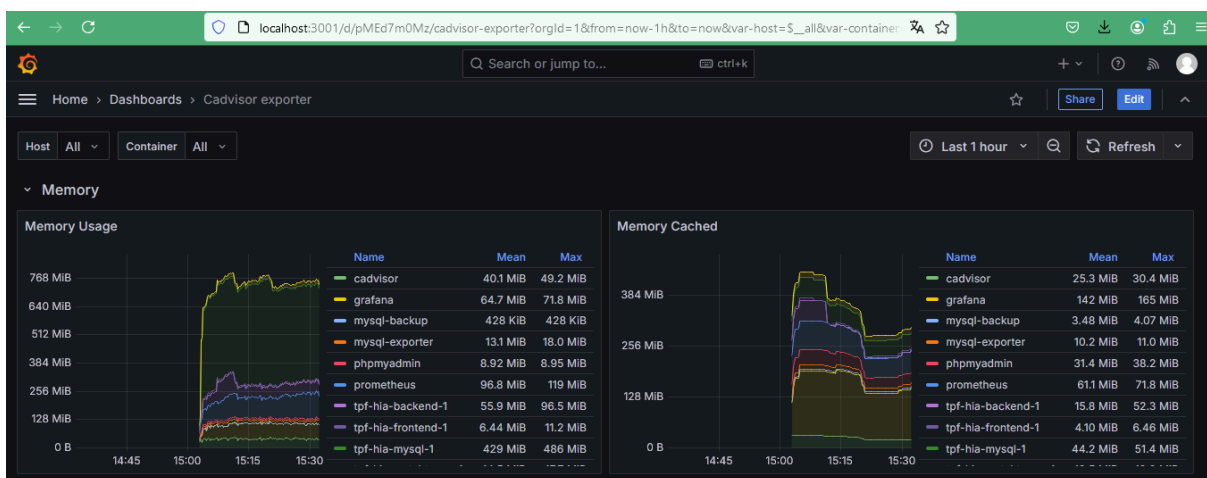
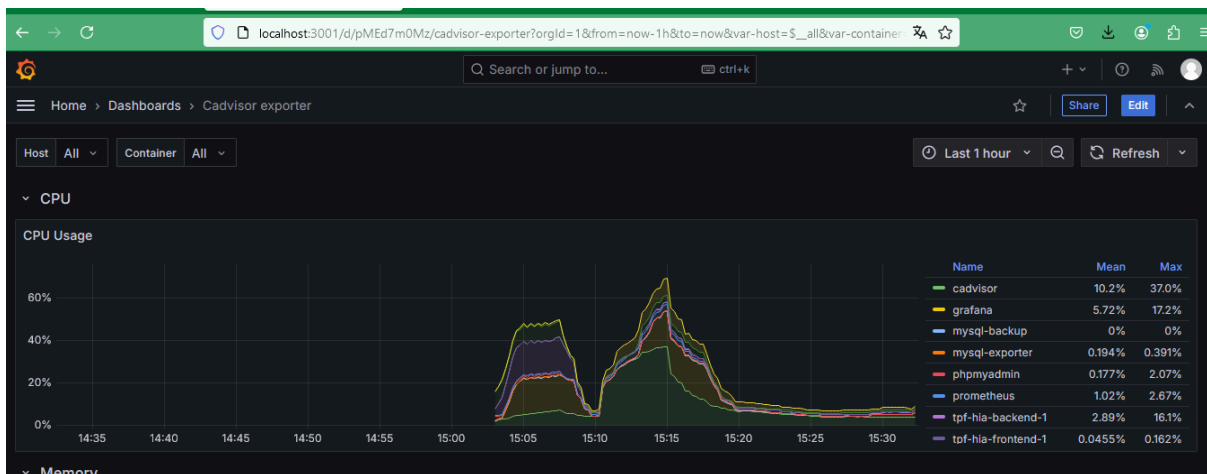
cadvisor:
  image: gcr.io/cadvisor/cadvisor:latest
  container_name: cadvisor
  ports:
```

```
- "8081:8080" # Puerto de cAdvisor
volumes:
  - /:/rootfs:ro
  - /var/run:/var/run:rw
  - /sys:/sys:ro
  - /var/lib/docker:/var/lib/docker:ro
networks:
  - my-network
restart: unless-stopped

grafana:
  image: grafana/grafana:latest
  container_name: grafana
  ports:
    - "3001:3000" # Puerto de Grafana
  volumes:
    - grafana-data:/var/lib/grafana
  networks:
    - my-network
  restart: unless-stopped

mysql-exporter:
  image: bitnami/mysqld-exporter
  container_name: mysql-exporter
  networks:
    - my-network
  command:
    - --config.my-cnf=/cfg/.my.cnf
    - --mysqld.address=mysql:3306
  volumes:
    - "./.my.cnf:/cfg/.my.cnf"
  environment:
    - DATA_SOURCE_NAME=root:root@tcp(mysql:3306)/"
  ports:
    - 9104:9104
```





#### 4. Actividad 3: Despliegue continuo

**Objetivo:** Automatizar el flujo desde el repositorio hasta la producción:

1. Configura un pipeline de integración y despliegue continuo (CI/CD): Herramientas recomendadas: GitHub Actions, GitLab CI/CD o Jenkins. Define un workflow que detecte cambios en el repositorio (p.ej., un push) y reconstruya la imagen del contenedor.
2. Modifica el header/footer de la aplicación para incluir los nombres de los integrantes.

**Pruebas:** Realiza un commit con los cambios y verifica que el despliegue sea automático.

Para el despliegue continuo se configura un archivo `deploy.yml` en la carpeta `./github/workflow` del repositorio.

Este archivo contiene el flujo de trabajo y al detectar un push en el repositorio, inicia una serie de pruebas y de resultar exitosa genera nuevas imágenes de frontend y backend, las cuales son subidas a docker hub automáticamente.

```
version: '3.8'

services:
  frontend:
    image: cristian241/frontend-app:latest # Imagen en Docker Hub
    ports:
      - "4200:80" # Mapea el puerto 4200
    networks:
      - my-network

  backend:
    image: cristian241/backend-app:latest # Imagen en Docker Hub
    environment:
      - NODE_ENV=production
      - DB_HOST=mysql
      - DB_PORT=3307
      - DB_USER=root
      - DB_PASSWORD=root
      - DB_NAME=tpf_db_d
    depends_on:
      - haproxy
    networks:
      - my-network
    ports:
      - "3000:3000"

# Nodo maestro
mysql-maestro:
  image: mysql:8.0
  container_name: mysql-maestro
  restart: always
  environment:
    MYSQL_ROOT_PASSWORD: root
    MYSQL_DATABASE: tpf_db_d
    MYSQL_REPLICATION_USER: root
    MYSQL_REPLICATION_PASSWORD: root
  volumes:
    - maestro-dato:/var/lib/mysql
```

```

ports:
  - "3310:3306"  # Exponer el puerto 3310
command: >
  --server-id=1
  --log-bin=mysql-bin
  --binlog-format=ROW
  --gtid_mode=ON
  --enforce_gtid_consistency=TRUE
  --master_info_repository=TABLE
  --relay_log_info_repository=TABLE
  --log_slave_updates=TRUE
  --read_only=FALSE
networks:
  - my-network

# Nodo esclavo
mysql-esclavo:
  image: mysql:8.0
  container_name: mysql-esclavo
  restart: always
  environment:
    MYSQL_ROOT_PASSWORD: root
    MYSQL_DATABASE: tpf_db_d  # Base de datos adaptada
    MYSQL_REPLICATION_USER: root
    MYSQL_REPLICATION_PASSWORD: root
  volumes:
    - esclavo-dato:/var/lib/mysql
    -

./setup-replication.sh:/docker-entrypoint-initdb.d/setup-replication.sh
ports:
  - "3311:3306"  # Exponer el puerto 3311
command: >
  --server-id=2
  --log-bin=mysql-bin
  --relay-log=relay-bin
  --read-only=1
  --binlog-format=ROW
  --gtid_mode=ON
  --enforce_gtid_consistency=TRUE
  --master_info_repository=TABLE
  --relay_log_info_repository=TABLE
  --log_slave_updates=TRUE
  --skip-host-cache

```

```

    --skip-name-resolve
depends_on:
  - mysql-maestro
networks:
  - my-network

haproxy:
  image: haproxy:latest
  container_name: mysql
  restart: always
  ports:
    - '3307:3306' # Acceder al HAProxy a través del puerto 3307
    - '3308:3306'
  volumes:
    - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg
  depends_on:
    - mysql-maestro
    - mysql-esclavo
  networks:
    - my-network

phpmyadmin:
  image: phpmyadmin/phpmyadmin:latest # Imagen oficial de phpMyAdmin
  container_name: phpmyadmin
  environment:
    PMA_HOST: mysql-maestro # Nombre del servicio MySQL definido
arriba
    PMA_PORT: 3306 # Puerto del servicio MySQL
    MYSQL_ROOT_PASSWORD: root # Contraseña del root
  depends_on:
    - mysql-maestro
  networks:
    - my-network
  ports:
    - "8080:80" # Puerto para acceder a phpMyAdmin desde el
navegador
  labels:
    com.centurylinklabs.watchtower.enable: "false"

backup:
  image: mysql:latest # Usa la misma imagen de MySQL
  container_name: mysql-backup
  volumes:

```

```

    - mysql-backups:/backups # Volumen Docker para almacenar backups
    - ./backup.sh:/usr/local/bin/backup.sh # Monta el script en la
ruta correcta
  depends_on:
    - mysql-maestro
  entrypoint: ["/bin/bash", "-c", "chmod +x /usr/local/bin/backup.sh
&& while true; do /usr/local/bin/backup.sh; sleep 84600; done"]
  networks:
    - my-network

restore:
  image: mysql:latest
  container_name: mysql-restore
  environment:
    MYSQL_ROOT_PASSWORD: root
  volumes:
    - mysql-data:/var/lib/mysql
    - ./backup.sql:/backup.sql
  command: bash -c "sleep 55 && mysql -h mysql -u root -proot
tpf_db_d < /backup.sql && echo 'Restauración completada'"
  networks:
    - my-network
  depends_on:
    - mysql-maestro

watchtower:
  image: containrrr/watchtower
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
  environment:
    WATCHTOWER_CLEANUP: "true"
  command: --interval 200 # Configurar el intervalo a 200 segundos
  networks:
    - my-network

prometheus:
  image: prom/prometheus:latest
  container_name: prometheus
  volumes:
    - ./prometheus.yml:/etc/prometheus/prometheus.yml #
Configuración de Prometheus
  command:
    - '--config.file=/etc/prometheus/prometheus.yml'

```

```
ports:
  - "9090:9090" # Puerto de Prometheus
networks:
  - my-network
restart: unless-stopped

cadvisor:
  image: gcr.io/cadvisor/cadvisor:latest
  container_name: cadvisor
  ports:
    - "8081:8080" # Puerto de cAdvisor
  volumes:
    - /:/rootfs:ro
    - /var/run:/var/run:rw
    - /sys:/sys:ro
    - /var/lib/docker:/var/lib/docker:ro
  networks:
    - my-network
  restart: unless-stopped

grafana:
  image: grafana/grafana:latest
  container_name: grafana
  ports:
    - "3001:3000" # Puerto de Grafana
  volumes:
    - grafana-data:/var/lib/grafana
  networks:
    - my-network
  restart: unless-stopped

mysql-exporter:
  image: bitnami/mysqld-exporter
  container_name: mysql-exporter
  networks:
    - my-network
  command:
    - --config.my-cnf=/cfg/.my.cnf
    - --mysqld.address=mysql:3306
  volumes:
    - "./.my.cnf:/cfg/.my.cnf"
  environment:
    - DATA_SOURCE_NAME=root:root@tcp(mysql:3306) /"
```

```
ports:
  - 9104:9104

networks:
  my-network:

volumes:
  mysql-data:
  mysql-backups: # Volumen específico para backups
  grafana-data:
  maestro-dato:
    name: maestro-dato
  esclavo-dato:
    name: esclavo-dato
```

Para las pruebas realizamos el siguiente test en el backend, esto sirve como **prueba mínima de control** para verificar que el flujo de integración y despliegue continuo está correctamente configurado.

```
tpf-backend > test > JS control.test.js > ...
Tabnine | Edit | Test | Explain | Document | Ask
1 describe("Prueba controlada para verificar el flujo de despliegue", () => {
2   const resultadoEsperado = true;
3
4   test("Test de prueba", () => {
5     expect(resultadoEsperado).toBe(true);
6   });
7 });
8
```

Los resultados del workflow pueden verse en Actions de Github

The screenshot shows the GitHub Actions interface for a workflow named 'cambio #16'. The workflow is triggered by a push to the 'main' branch. The status is 'Success' with a total duration of '1m 8s'. The workflow consists of three jobs: 'Run Backend Tests', 'Build and Deploy Frontend', and 'Build and Deploy Backend'. The 'Run Backend Tests' job is shown with a warning: 'Run Backend Tests: The following actions use a deprecated Node.js version and will be forced to run on node20: actions/checkout@v3. For more info: https://github.blog/changelog/2024-03-07-github-actions-all-actions-will-run-on-node20-inst...'. The 'Build and Deploy Frontend' and 'Build and Deploy Backend' jobs are also shown with their respective durations.

y en la cuenta de docker hub tambien se ven las actualizaciones de las imagenes

The screenshot shows the Docker Hub repository page for 'cristian241/frontend-app'. The repository is public and has 8 tags. The tags are listed in a table:

Tag	OS	Type	Pulled	Pushed
latest	linux	Image	an hour ago	2 hours ago
3e62197fde2d2a8b1...	linux	Image	an hour ago	2 hours ago
ec4b00e03bdbc7560...	linux	Image	an hour ago	2 hours ago
3f6af3a62fd9b371f7...	linux	Image	3 hours ago	4 hours ago

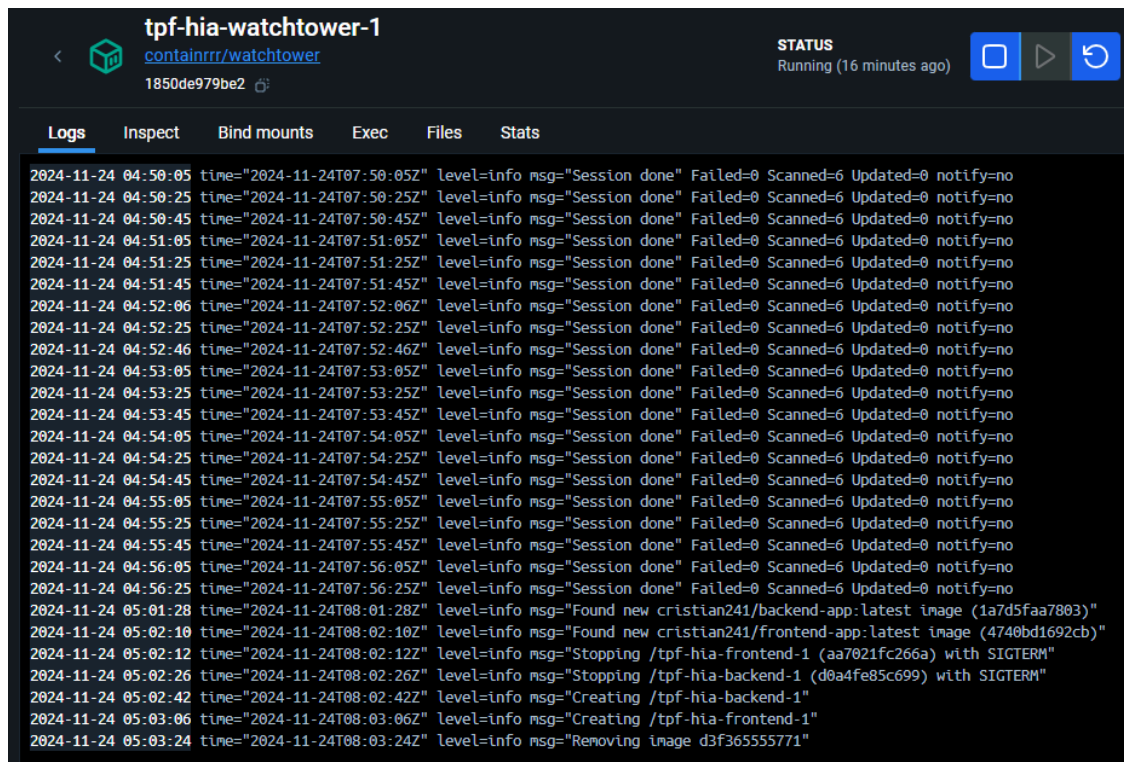
The page also shows Docker commands for pushing a new tag: `docker push cristian241/frontend-app:tagname`. There is also a section for 'Automated builds' with a link to 'Read more about automated builds'.

Para que se actualice la imagen es necesario agregar el contenedor watchtower el cual detecta las actualizaciones de las imágenes y actualiza el contenedor correspondiente

```
watchtower:  
  image: containrrr/watchtower  
  volumes:  
    - /var/run/docker.sock:/var/run/docker.sock  
  environment:  
    WATCHTOWER_CLEANUP: "true"  
  command: --interval 200 # Configurar el intervalo a 200 segundos  
  networks:  
    - my-network
```

La imagen muestra los logs de watchtower. Se ve que al detectar un cambio en una imagen, detiene el contenedor correspondiente, lo actualiza y luego elimina la imagen antigua.





## 5. Actividad 4: Clúster de replicación

**Objetivo:** Configurar un clúster de replicación para la base de datos:

1. Implementa un clúster de replicación para la base de datos (p.ej., PostgreSQL).

Herramientas: Patroni, Pgpool-II, o servicios internos de MySQL/MongoDB.

### Implementación de Clúster de Replicación

Utilizamos los servicios internos de replicación de MySQL y configuramos un clúster compuesto por:

- Un nodo maestro (mysql-maestro): Este nodo gestiona todas las operaciones de escritura y sincroniza los datos hacia el esclavo.

```
mysql-maestro:
  image: mysql:8.0
  container_name: mysql-maestro
  restart: always
  environment:
    MYSQL_ROOT_PASSWORD: root
    MYSQL_DATABASE: tpf_db_d
    MYSQL_REPLICATION_USER: root
    MYSQL_REPLICATION_PASSWORD: root
  volumes:
    - maestro-dato:/var/lib/mysql
  ports:
    - "3310:3306" # Exponer el puerto 3310
  command: >
    --server-id=1
    --log-bin=mysql-bin
    --binlog-format=ROW
    --gtid_mode=ON
    --enforce_gtid_consistency=TRUE
    --master_info_repository=TABLE
    --relay_log_info_repository=TABLE
    --log_slave_updates=TRUE
    --read_only=FALSE
  networks:
    - my-network
```

- Un nodo esclavo (mysql-esclavo): Recibe y replica automáticamente los datos del maestro.

```
mysql-esclavo:
  image: mysql:8.0
  container_name: mysql-esclavo
  restart: always
  environment:
    MYSQL_ROOT_PASSWORD: root
    MYSQL_DATABASE: tpf_db_d # Base de datos adaptada
    MYSQL_REPLICATION_USER: root
    MYSQL_REPLICATION_PASSWORD: root
  volumes:
    - esclavo-dato:/var/lib/mysql
    - ./setup-replication.sh:/docker-entrypoint-initdb.d/setup-replication.sh
  ports:
    - "3311:3306" # Exponer el puerto 3311
  command: >
    --server-id=2
    --log-bin=mysql-bin
    --relay-log=relay-bin
    --read-only=1
    --binlog-format=ROW
    --gtid_mode=ON
    --enforce_gtid_consistency=TRUE
    --master_info_repository=TABLE
    --relay_log_info_repository=TABLE
    --log_slave_updates=TRUE
    --skip-host-cache
    --skip-name-resolve
  depends_on:
    - mysql-maestro
  networks:
    - my-network
```

- HAProxy (haproxy): Se implementó como balanceador de carga para gestionar las consultas de la aplicación y dirigir las dinámicamente al maestro (lectura/escritura) o al esclavo (solo lectura).

```
haproxy:
  image: haproxy:latest
  container_name: mysql
  restart: always
  ports:
    - '3307:3306' # Acceder al HAProxy a través del puerto 3307
    - '3308:3306'
  volumes:
    - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg
  depends_on:
    - mysql-maestro
    - mysql-esclavo
  networks:
    - my-network
```

## Configuración de Replicación

- El nodo maestro tiene habilitados los binlogs (log-bin) y configuraciones de replicación global (gtid\_mode=ON).
- El nodo esclavo está configurado como "read-only" y se conecta al maestro utilizando la instrucción CHANGE REPLICATION SOURCE TO para sincronizar datos automáticamente.

El archivo docker-compose.yml incluye un script (setup-replication.sh) que automatiza la configuración de la replicación en el nodo esclavo.

```
#!/bin/bash

# Esperar a que mysql-maestro esté disponible
while ! mysqladmin ping -h mysql-maestro --silent; do
    echo "Esperando a que mysql-maestro esté listo..."
    sleep 2
done

# Configurar el usuario root en mysql-maestro para usar mysql_native_password
mysql -u root -proot -h mysql-maestro -e "ALTER USER 'root'@ '%' IDENTIFIED WITH mysql_native_password BY 'root';"

# Configurar replicación en el esclavo
mysql -u root -proot -e "
CHANGE REPLICATION SOURCE TO
    SOURCE_HOST='mysql-maestro',
    SOURCE_PORT=3306,
    SOURCE_USER='root',
    SOURCE_PASSWORD='root',
    SOURCE_AUTO_POSITION=1;"
```

Además con el siguiente comando verificamos que la replica se levantó de forma exitosa:

```
PS D:\nuevo\setup\h1a-final> docker exec -it mysql-esclavo mysql -u root -proot -e "SHOW SLAVE STATUS\G"
mysql: [Warning] Using a password on the command line interface can be insecure.
***** 1. row *****
Slave_IO_State: Waiting for source to send event
Master_Host: mysql-maestro
Master_User: root
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.000003
Read_Master_Log_Pos: 3727
Relay_Log_File: relay-bin.000005
Relay_Log_Pos: 3943
Relay_Master_Log_File: mysql-bin.000003
```

2. Cambia la configuración de tu aplicación para que apunte al clúster en lugar de a la instancia aislada.

Para apuntar al clúster, la aplicación fue configurada para conectarse a HAProxy en lugar de al nodo maestro directamente. Esto asegura que la aplicación interactúe con el clúster de forma transparente y permita la gestión de alta disponibilidad.

Cambios clave en el Backend de la aplicación:

```
backend:
  image: cristian241/backend-app:latest
  environment:
    - NODE_ENV=production
    - DB_HOST=mysql
    - DB_PORT=3307
    - DB_USER=root
    - DB_PASSWORD=root
    - DB_NAME=tpf_db_d
  depends_on:
    - haproxy
  networks:
    - my-network
  ports:
    - "3000:3000"
```

HAProxy fue configurado para:

- Dirigir operaciones de lectura/escritura al maestro (mysql-maestro).
- Redirigir operaciones de solo lectura al esclavo (mysql-esclavo), como respaldo en caso de fallas del maestro.

3. Realiza pruebas para verificar la alta disponibilidad y la replicación de datos.

#### a) Prueba de Recopilación de Datos

1. Accedemos al nodo maestro:

```
PS D:\nuevo\setup\hia-final> docker exec -it mysql-maestro bash;
bash-5.1#
```

2. Accedemos al MySQL dentro del contenedor maestro:

```
bash-5.1# mysql -u root -proot
mysql: [Warning] Using a password on the command line int
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 8.0.40 MySQL Community Server - GPL
```

3. Seleccionamos nuestra Base de Datos:

```
mysql> USE tpf_db_d;
Reading table information for completion of table and column names
```

4. Ingresamos un registro en el maestro:

```
mysql> INSERT INTO usuarios (email, usuario, password, activo, perfil, nombres, apellido, dni, telefono) VALUES ('sara@sara.com', 'sara', 'sara', 1, 'administrativo', 'Sara', 'Mercado', 12345678, 1234567890);
Query OK, 1 row affected (0.01 sec)
```

5. Consultamos al esclavo para verificar que el registro fue replicado:

```
PS D:\nuevo\setup\hia-final> docker exec -it mysql-esclavo mysql -u root -proot -D tpf_db_d -e "SELECT * FROM usuarios;"
mysql: [Warning] Using a password on the command line interface can be insecure.
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | email      | usuario | password | activo | perfil      | nombres | apellido | dni      | telefono |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1  | sara@sara.com | sara   | sara     | 1      | administrativo | Sara   | Mercado  | 12345678 | 1234567890 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

## b) Prueba de Alta Disponibilidad (Falla del Maestro)

Detenemos al maestro:

```
PS D:\nuevo\setup\hia-final> docker stop mysql-maestro
mysql-maestro
```

Verificamos que las consultas de solo lectura de la aplicación sigue funcionando

```
PS D:\nuevo\setup\hia-final> docker exec -it mysql-esclavo mysql -u root -proot -D tpf_db_d -e "SELECT * FROM usuarios;"
mysql: [Warning] Using a password on the command line interface can be insecure.
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | email      | usuario | password | activo | perfil      | nombres | apellido | dni      | telefono |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1  | sara@sara.com | sara   | sara     | 1      | administrativo | Sara   | Mercado  | 12345678 | 1234567890 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

El clúster implementado garantiza alta disponibilidad y replicación confiable de datos gracias a HAProxy y los servicios internos de MySQL. La aplicación ahora se beneficia de una infraestructura robusta que permite tolerancia a fallos y escalabilidad para operaciones críticas.