MÓDULO 4(ejercicio):

Ejercicio SQL(gemini)

Aquí tienes un ejemplo de un archivo `docker-stack.yml` utilizando una red **overlay** en Docker Swarm, con comentarios explicativos:

```yaml
# docker-stack.yml
version: '3.8'

# Servicios a desplegar
services:
  # Servicio web (Ejemplo: Nginx)
  web:
    image: nginx:alpine
    ports:
      - "80:80"
    networks:
      - my-overlay-net
    deploy:
      replicas: 3
      placement:
        constraints:
          - node.role == worker
      resources:
        limits:
          cpus: '0.5'
          memory: 256M
      restart_policy:
        condition: on-failure
      update_config:
        parallelism: 1
        delay: 10s

  # Servicio de base de datos (Ejemplo: PostgreSQL)
  db:
    image: postgres:13
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: admin123
    networks:
      - my-overlay-net
    volumes:
      - postgres-data:/var/lib/postgresql/data
    deploy:
      replicas: 1
      placement:
        constraints:
          - node.role == manager
      restart_policy:
        condition: on-failure
```

```
# Redes definidas
networks:
  my-overlay-net:
    driver: overlay
    attachable: true  # Permite que otros servicios/contenedores se unan a la
red

# Volúmenes para persistencia de datos
volumes:
  postgres-data:
    driver: local  # En producción usar un driver adecuado (nfs, cloud, etc.)
```

Explicación de elementos clave:

1. **Red Overlay (`my-overlay-net`)**:
   - `driver: overlay`: Permite comunicación entre contenedores en nodos diferentes del Swarm.
   - `attachable: true`: Permite que otros servicios/contenedores se conecten manualmente.
2. **Servicios**:
   - **Web**:
     - 3 réplicas distribuidas en nodos `worker`.
     - Expone el puerto 80 del host mapeado al puerto 80 del contenedor.
   - **DB**:
     - 1 réplica en un nodo `manager`.
     - Volumen para persistencia de datos (almacenamiento local en este ejemplo).
3. **Volúmenes**:
   - `postgres-data`: Almacena datos de PostgreSQL. En entornos reales, usar drivers como `nfs` o soluciones cloud.
4. **Políticas de despliegue**:
   - `restart_policy`: Reinicia el servicio si falla.
   - `update_config`: Controla cómo se actualizan los servicios (paralelismo y tiempo de espera).

Desplegar el Stack:

```
docker stack deploy -c docker-stack.yml mi-stack
```

Consideraciones importantes:

1. **Red Overlay**:
   - Requiere que el Swarm esté inicializado (`docker swarm init`).
   - Los nodos deben tener los puertos necesarios abiertos (puerto `2377`, `7946`, `4789`).
2. **Persistencia de datos**:
   - En producción, usar volúmenes externos (AWS EBS, NFS, etc.).
3. **Seguridad**:
   - Usar `secrets` para contraseñas en lugar de variables de entorno planas.
   - Ejemplo:

```
environment:
  POSTGRES_PASSWORD_FILE: /run/secrets/db_password
secrets:
  - db_password
```

4. **Escalabilidad**:
   - Ajustar `replicas` según la carga esperada.
   - Usar `docker service scale mi-stack_web=5` para escalar manualmente.
5. **Healthchecks**:
   - Añadir comprobaciones de salud a los servicios:

```
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost"]
  interval: 30s
  timeout: 10s
  retries: 3
```

Este ejemplo cubre un escenario básico de despliegue en Swarm con una red overlay. Adapta los servicios, configuraciones y recursos según tus necesidades.