Practica 5

1. Utilizando sus palabras describa, ¿qué es docker compose?

   Es una herramienta que facilita el despliegue de aplicaciones que están compuestas por muchos contenedores que se comunican entre sí para el funcionamiento de las mismas.

   Es entonces, el conjunto de la herramienta (el binario) y los archivos de configuración llamados “archivo compose” o “compose file” que definen los recursos deseados.

2. ¿Qué es el archivo compose y cuál es su función? ¿Cuál es el “lenguaje” del archivo?

   Es un archivo escrito en el lenguaje de marcado YAML que describe las características de los contenedores a desplegar. Es usualmente llamado docker-compose.yml.

   Toda la configuración se define en el archivo compose. A cada contenedor definido en este archivo se lo denomina “service”

3. ¿Cuáles son las versiones existentes del archivo docker-compose.yaml existentes y qué características aporta cada una? ¿Son compatibles entre sí?¿Por qué?

   Existen tres versiones para el formato de archivo de docker-compose.yml:

   - Versión 1:
     - Está obsoleta.
     - No requería especificar una versión en el archivo docker-compose.yml.
     - Características básicas para definir servicios, volúmenes y redes.
     - Todos los servicios se declaran en la raíz del documento

    - No permite declarar volúmenes nombrados, redes o argumentos de construcción
    - Todos los contenedores se conectan a la misma red predeterminada, de tipo Bridge
- Versión 2.x
    - Introducción de la sección version.
    - Mejoras en la configuración de redes y volúmenes.
    - Todos los servicios se declaran dentro de la clave services
    - Permite declarar volúmenes nombrados, dentro de la clave volumes
    - Permite declarar redes, dentro de la clave networks
    - Por default, los contenedores se conectan a una misma red, y se usa como nombre de host, el nombre de servicio
    - Las versiones 2.1 a 2.4 agregan otras claves y características
    - Soporte para dependencias de servicios mediante depends_on.
    - Añadido soporte para configuraciones más avanzadas como healthcheck, deploy, y secrets.
    - Permite el uso de comandos de construcción (build) más avanzados.
- Versión 3.x:
    - Ultima versión y la recomendada por Docker.
    - Enfocada en el uso con Docker Swarm, permitiendo la configuración de despliegues en un entorno de clúster.
    - Introducción de la sección deploy para especificar políticas de despliegue (número de réplicas, restricciones de recursos, etc.).
    - Soporte mejorado para secretos y configuraciones (configs).
    - Opciones avanzadas para redes y volúmenes, como configuraciones de driver y driver_opts.
    - healthcheck avanzado para verificar el estado de los servicios.
    - Soporte para configs y secrets para gestionar configuraciones sensibles y secretas.
    - Se remueven las claves:
        - volume_driver
        - volumes_from
        - cpu_shares
        - cpu_quota
        - cpuset

        - mem_limit
        - memswap_limit
        - extends
        - group_add
    - Las versiones 3.1 a 3.8 agregan otras claves y características

Las versiones 2.x y 3.x comparten estructura pero no algunas opciones. Esto hace que haya una retrocompatibilidad hasta cierto punto dado que no todas las características entre versiones son compatibles.

4. Investigue y describa la estructura de un archivo compose.

```yaml
version: "3.9"
services:
  web:
    build: .
    ports:
      - "8000:5000"
    volumes:
      - .:/code
    environment:
      FLASK_ENV: development
  redis:
    image: "redis:alpine"
```

Desarrolle al menos sobre los siguientes bloques indicando para qué se usan:

a. services: define los servicios que componen la aplicación. Cada uno corresponde a un contenedor.

b. build: especifica cómo construir una imagen de Docker a partir de un Dockerfile y un contexto de construcción. Puede contener la ruta a un directorio con un archivo Dockerfile, o puede ser un objeto que incluya el contexto de construcción y el Dockerfile.

c. image: especifica qué imagen de Docker usar como base para el contenedor.

d. volumes: permite definir volúmenes para conservar datos o compartir datos entre contenedores. Se pueden especificar configuraciones de volumen, como rutas de origen, rutas de destino dentro de contenedores y configuraciones de solo lectura.

e. restart: permite especificar políticas de reinicio para los servicios, definiendo cómo deben comportarse en caso de fallas o al reiniciar. Las opciones incluyen no, always, on-failurey unless-stopped.

f. depends_on: permite definir dependencias entre servicios. Esto garantiza que un servicio se inicie sólo después de que los servicios de los cuales depende estén en funcionamiento.

g. environment: permite definir variables de entorno para cada servicio. Sirve para pasar ajustes de configuración a sus contenedores.

h. ports: se utiliza para mapear puertos del contenedor a puertos en el host. También se puede definir el protocolo (TCP o UDP) para cada puerto.

i. expose: expone puertos del contenedor solo a otros contenedores conectados en la misma red, sin exponerlos al host.

j. networks: permite definir redes personalizadas para los servicios. Estas permiten que los contenedores se comuniquen entre sí a través de redes aisladas. Se pueden especificar configuraciones de red como alias, direcciones IP y conectividad externa.
   Hay un networks raíz y un networks en cada servicio particular. En el networks raíz se definen todas las redes que se van a crear.

k. healthcheck: permite definir configuraciones de comprobación de estado para los servicios. Los health checks monitorean el estado de los contenedores y pueden usarse para determinar si un servicio está en buen estado o no.

5. Conceptualmente: ¿Cómo se podrían usar los bloques “healthcheck” y “depends_on” para ejecutar una aplicación Web dónde el backend debería ejecutarse si y sólo si la base de datos ya está ejecutándose y lista?

   El backend para asegurarse de que la base de datos se inicie primero usaría *depends_on* con la condición *service_healthy* que asegura que el backend solo se inicie cuando el healthcheck de la base de datos sea exitoso. El healthcheck en la base de datos se usaría para verificar que la misma esté lista para recibir conexiones antes de que el backend comience.

6. Indique qué hacen y cuáles son las diferencias entre los siguientes comandos:
   a. docker compose create y docker compose up

      El primero crear todos los contenedores. El segundo lo mismo y además los inicia.

   b. docker compose stop y docker compose down

      El primero frena todos los contenedores. El segundo lo mismo y además los elimina. También elimina las redes y volúmenes asociados pero no las imágenes.

   c. docker compose run y docker compose exec

      El primero crea e inicia un nuevo contenedor para ejecutar un comando específico y termina. El segundo ejecuta un comando dentro de un contenedor en ejecución. El contenedor sigue en ejecución después de que se complete el comando.

   d. docker compose ps

      Lista los contenedores gestionados por Docker Compose, mostrando el estado, nombres y puertos mapeados.

   e. docker compose logs

Muestra los logs de salida de los contenedores .Muestra el output de los contenedores.

7. ¿Qué tipo de volúmenes puede utilizar con docker compose? ¿Cómo se declara cada tipo en el archivo compose?

   Anonymous volumes: no tienen un nombre definido por el usuario. En su lugar, Docker los crea automáticamente cuando se crea un contenedor y asigna un ID único al volumen. Generalmente, es más difícil gestionar y almacenar volúmenes debido a la falta de un identificador legible. Dado que Docker los crea automáticamente, es común usar volúmenes anónimos para almacenamiento temporal.

```
services:
      example_service:
            volumes:
                  - /container/path
```

   Docker host-mounted volumes: permite montar un directorio o archivo del host en el contenedor. Puede ser útil para compartir datos entre el host y el contenedor.

```
services:
      example_service:
             volumes:
                  - /host/path:/container/path
```

Named Volumes: tienen un nombre definido por el usuario, lo que los hace fáciles de identificar, gestionar y compartir entre múltiples contenedores. Docker crea y gestiona volúmenes nombrados y almacena sus datos en una ubicación específica en el sistema host. Se pueden definir como internal (predeterminados) o external.

```
services:
      example_service:
            volumes:
                  - named_volume_name:/container/path
```

Internal: tienen el alcance de un único archivo Docker-compose y Docker los crea si no existen.

```
volumes:
      named_volume:
services:
      example_service:
            volumes:
                  - named_volume:/container/path
```

External: se pueden usar en toda la instalación de Docker y deben ser creados por el usuario (de lo contrario, falla) usando el comando Docker Volume Create .

Definir volumen:

```
docker volume create --driver local \
    --opt type=none \
    --opt device=/var/opt/my_website/dist \
    --opt o=bind named_volume
```

```
volumes:
      named_volume:
            external:true
services:
      example_service:
            volumes:
                  - named_volume:/container/path
```

8. ¿Qué sucede si en lugar de usar el comando “docker compose down” utilizo “Docker compose down -v/--volumes”?

docker compose down detiene y elimina los servicios y redes de la aplicación. De forma predeterminada, no elimina los volúmenes con nombre. Para eliminar volúmenes con nombre, se utiliza la marca --volumes o -v .

Ejercicio guiado - Instanciando un Wordpress y una Base de Datos.

Dado el siguiente código de archivo compose:

```
version: "3.9"

services:
  db:
    image: mysql:5.7
    networks:
      - wordpress
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    networks:
      - wordpress
    volumes:
      - ${PWD}:/data
      - wordpress_data:/var/www/html
    ports:
      - "127.0.0.1:8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress


    volumes:
      db_data: {}
      wordpress_data: {}
    networks:
      wordpress:
```

Preguntas

Intente analizar el código ANTES de correrlo y responda:

- ¿Cuántos contenedores se instancian?

  Se están instanciando dos contenedores: db y wordpress

- ¿Por qué no se necesitan Dockerfiles?

  Esto se debe a que se están utilizando imágenes de los contenedores predefinidas disponibles en Docker Hub (con Dockerfiles ya definidos), haciendo que no sea necesaria la definición de Dockerfiles personalizados.

- ¿Por qué el servicio identificado como “wordpress” tiene la siguiente línea?
  depends_on:
  - db

  Porque depende del servicio db, necesita que este se inicie antes de poder iniciarse.

- ¿Qué volúmenes y de qué tipo tendrá asociado cada contenedor?

  db tiene un named volume y wordpress tiene uno de tipo host-mounted y otro de tipo named volume

- ¿Por qué uso el volumen nombrado
  volumes:
  - db_data:/var/lib/mysql
  para el servicio db en lugar de dejar que se instancie un volumen anónimo con el contenedor?

  Para persistir los datos (también se lograría con el anónimo o host-mounted) pero también para que sea más fácil de gestionar. A su vez permite usar un mismo volumen en distintos servicios, en lugar de duplicar la ruta del host a usar.

- ¿Qué genera la línea

volumes:

- ${PWD}:/data

en la definición de wordpress?

Permite el mapeo de los archivos que están dentro del directorio de trabajo actual (PWD) a la ruta /data dentro del contenedor.

- ¿Qué representa la información que estoy definiendo en el bloque environment de cada servicio?¿Cómo se “mapean” al instanciar los contenedores?

  Representa variables de entorno que se configuran dentro de los contenedores durante su ejecución (en este caso la información necesaria para configurar la BD). Cada clave y valor se mapea a una variable de ese nombre y ese valor en el entorno del contenedor.

- ¿Qué sucede si cambio los valores de alguna de las variables definidas en bloque “environment” en solo uno de los contenedores y hago que sean diferentes? (Por ej: cambio SOLO en la definición de wordpress la variable WORDPRESS_DB_NAME)

  Wordpress no podría conectarse a la BD.

- ¿Cómo sabe comunicarse el contenedor “wordpress” con el contenedor “db” si nunca doy información de direccionamiento?

  Wordpress se puede comunicarse con el contenedor bd usando el nombre del servicio como un alias de host. Docker Compose se encarga de la resolución de nombres y crea una red interna donde los contenedores pueden comunicarse entre sí utilizando los nombres de servicio como alias de host. Básicamente, se

usa el nombre del servicio como nombre de dominio y Docker Compose se encarga de la resolución de nombres.

- ¿Qué puertos expone cada contenedor según su Dockerfile? (pista: navegue el sitio https://hub.docker.com/_/wordpress y https://hub.docker.com/_/mysql para acceder a los Dockerfiles que generaron esas imágenes y responder esta pregunta.)

  Wordpress
  80

  MySQL
  3306 33060

- ¿Qué servicio se "publica" para ser accedido desde el exterior y en qué puerto?¿Es necesario publicar el otro servicio? ¿Por qué?

  Se publica el servicio wordpress para ser accedido desde el exterior en el puerto 8000 del host que se mapea al 80 de wordpress. No es necesario publicar el otro servicio porque no debe ser accedido directamente desde fuera del entorno de Docker, ya que es usado internamente por wordPress.

Fuentes

https://docs.docker.com/compose/migrate/

https://medium.com/@meghasharmaa704/file-structure-of-docker-compose-yml-file-e955a2f99a1f#:~:text=The%20structure%20of%20a%20docker,file%20format%20you're%20using.

https://kinsta.com/blog/docker-compose-volumes/#:~:text=specific%20use%20case.-,Types%20of%20Docker%20Volumes,manage%20data%20in%20your%20applications.

https://devopscell.com/docker/docker-compose/volumes/2018/01/16/volumes-in-docker-compose.html

https://devopscell.com/docker/docker-compose/volumes/2018/01/16/volumes-in-docker-compose.html