UNIVERSIDAD TECNOLOGICA DE SANTIAGO
(UTESA)

PRESENTADO POR:

ERIK CRUZ
1-18-0759

PRESENTADO A:

IVAN MENDOZA

ASIGNATURA:

ALGORITMOS PARALELOS

ASIGNACION:

ACTIVIDAD SEMANA 4

Santiago de los Caballeros
República Dominicana
Febrero, 2024

https://github.com/kelok3rik/TAREA-SEMANA-4-PARALELOS

1. ¿Qué es?

Docker es una plataforma de software que permite el desarrollo, el envío y la ejecución de aplicaciones dentro de contenedores. Estos contenedores encapsulan todo lo necesario para que una aplicación se ejecute de forma independiente en cualquier entorno, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución.

Los contenedores son una forma de virtualización a nivel de sistema operativo que es más ligera que la virtualización tradicional, ya que comparten el núcleo del sistema operativo subyacente y solo virtualizan los recursos del sistema necesarios para ejecutar la aplicación.

2. ¿Para qué nos sirve en desarrollo?

- **Entornos de Desarrollo Reproducibles:** Docker permite crear entornos de desarrollo reproducibles y consistentes. Los desarrolladores pueden definir todas las dependencias y configuraciones necesarias para ejecutar una aplicación dentro de un contenedor Docker, lo que garantiza que todos los miembros del equipo trabajen en el mismo entorno, independientemente de las diferencias en sus sistemas locales.
- **Aislamiento de Aplicaciones:** Docker utiliza contenedores para aislar aplicaciones y sus dependencias del sistema operativo subyacente y de otras aplicaciones en ejecución. Esto evita conflictos entre dependencias y simplifica la gestión de versiones de software.
- **Facilidad de Distribución:** Docker facilita la distribución de aplicaciones al empaquetarlas junto con todas sus dependencias en una imagen Docker. Estas imágenes pueden compartirse a través de repositorios públicos o privados, lo que simplifica el proceso de implementación y colaboración entre equipos.
- **Despliegue Eficiente:** Docker agiliza el proceso de despliegue de aplicaciones al permitir la creación y ejecución rápida de contenedores. Los contenedores son más ligeros que las máquinas virtuales

tradicionales, lo que reduce el tiempo de inicio y los recursos necesarios para ejecutar una aplicación.

3. ¿Cómo se utiliza?

- **Instalación de Docker:**
  Docker es una plataforma de software que te permite crear, desplegar y ejecutar aplicaciones dentro de contenedores. Lo primero que necesitas hacer es instalar Docker en tu sistema. Puedes encontrar instrucciones detalladas de instalación en la documentación oficial de Docker, que varían según el sistema operativo que estés utilizando (Linux, macOS, Windows).

- **Creación de un Dockerfile:**
  Un Dockerfile es un archivo de texto que contiene las instrucciones sobre cómo construir una imagen de Docker. Puedes crear un Dockerfile en el directorio de tu proyecto. Este archivo especifica qué sistema operativo base utilizar, qué paquetes instalar, qué comandos ejecutar, etc. Por ejemplo, un Dockerfile básico podría tener este contenido:

  ```
  FROM ubuntu:latest
  RUN apt-get update && apt-get install -y apache2
  CMD ["apache2ctl", "-D", "FOREGROUND"]
  ```

- **Construcción de la imagen:**
  Una vez que tienes un Dockerfile, puedes construir una imagen de Docker utilizando el comando `docker build`. Por ejemplo:

  ```
  docker build -t mi-aplicacion .
  ```

  Esto construirá una imagen llamada "mi-aplicacion" utilizando el Dockerfile presente en el directorio actual (`.`).

- **Ejecución de contenedores:**
  Una vez que tienes una imagen de Docker, puedes ejecutar contenedores basados en esa imagen utilizando el comando `docker run`. Por ejemplo:

  ```
  docker run -d -p 8080:80 mi-aplicacion
  ```

Esto ejecutará un contenedor basado en la imagen "mi-aplicacion" en segundo plano (`-d`), exponiendo el puerto 80 del contenedor al puerto 8080 del host (`-p 8080:80`).

4. Cluster o Balanceo de carga

En Docker, puedes implementar un cluster o balanceo de carga utilizando herramientas como Docker Swarm o Kubernetes.

- **Docker Swarm:** Es una herramienta de orquestación nativa de Docker que permite crear y administrar clusters de nodos Docker. Ofrece funcionalidades básicas de balanceo de carga y es más simple de configurar que Kubernetes.
- **Kubernetes:** Ofrece funcionalidades más avanzadas y es una opción popular para entornos de producción a gran escala. Proporciona balanceo de carga avanzado, escalado automático y características completas de orquestación.
- **Balanceadores de carga externos:** Puedes usar balanceadores de carga externos como NGINX o HAProxy para distribuir el tráfico entre los contenedores Docker. Esto ofrece flexibilidad y la posibilidad de utilizar tecnologías existentes de balanceo de carga.

5. Comando

- **docker images:** Lista todas las imágenes disponibles localmente en tu sistema.
- **docker pull <nombre_de_la_imagen>:** Descarga una imagen de Docker desde un registro público o privado.
- **docker rmi <nombre_de_la_imagen>:** Elimina una imagen de Docker del sistema.
- **docker ps**: Muestra todos los contenedores en ejecución.
- **docker ps -a:** Muestra todos los contenedores, incluidos los detenidos.
- **docker run <nombre_de_la_imagen>:** Crea y ejecuta un nuevo contenedor a partir de una imagen.
- **docker start <ID_del_contenedor>:** Inicia un contenedor detenido.
- **docker stop <ID_del_contenedor>:** Detiene un contenedor en ejecución.
- **docker rm <ID_del_contenedor>:** Elimina un contenedor.

- **docker exec -it <ID_del_contenedor> <comando>:** Ejecuta un comando dentro de un contenedor en ejecución.
- **docker volume ls:** Lista todos los volúmenes disponibles en el sistema.
- **docker volume create <nombre_del_volumen>:** Crea un nuevo volumen de Docker.
- **docker volume rm <nombre_del_volumen>:** Elimina un volumen de Docker.
- **docker network ls:** Lista todas las redes disponibles en el sistema.
- **docker network create <nombre_de_la_red>:** Crea una nueva red de Docker.
- **docker network rm <nombre_de_la_red>:** Elimina una red de Docker.
- **docker info:** Muestra información detallada sobre la configuración de Docker en el sistema.
- **docker version:** Muestra la versión de Docker instalada en el sistema.
- **docker-compose:** Herramienta para definir y ejecutar aplicaciones Docker multi-contenedor utilizando un archivo YAML.

6. Ejemplos de docker-compose levantando varias aplicaciones en distintas tecnologias: PHP, JAVA y NODEJS)

Php:

```yaml
version: '3'
services:
  php_app:
    image: php:latest
    volumes:
      - ./php_app:/var/www/html
    ports:
      - "8080:80"
```

Este archivo docker-compose.yml levanta un servicio de aplicación PHP utilizando la imagen oficial de PHP. Monta el directorio local ./php_app en el directorio /var/www/html dentro del contenedor para proporcionar el código de la aplicación PHP. Expone el puerto 8080 del host al puerto 80 del contenedor para acceder a la aplicación.

Java:

```yaml
version: '3'
services:
  java_app:
    image: openjdk:latest
    volumes:
      - ./java_app:/usr/src/app
    working_dir: /usr/src/app
    command: "java -jar your_java_app.jar"
```

Este archivo docker-compose.yml levanta un servicio de aplicación Java utilizando la imagen oficial de OpenJDK. Monta el directorio local ./java_app en el directorio /usr/src/app dentro del contenedor. Luego, ejecuta el comando java -jar your_java_app.jar para iniciar la aplicación.

Node.Js

```yaml
version: '3'
services:
  nodejs_app:
    image: node:latest
    volumes:
      - ./nodejs_app:/usr/src/app
    working_dir: /usr/src/app
    command: "npm start"
    ports:
      - "3000:3000"
```

Este archivo docker-compose.yml levanta un servicio de aplicación Node.js utilizando la imagen oficial de Node.js. Monta el directorio local ./nodejs_app en el directorio /usr/src/app dentro del contenedor. Luego, ejecuta el comando npm start para iniciar la aplicación. Expone el puerto 3000 del host al puerto 3000 del contenedor para acceder a la aplicación.