Docker

1. ¿Qué es Docker?

Docker es una plataforma que permite a desarrolladores y administradores de sistemas **crear, desplegar y ejecutar aplicaciones dentro de contenedores**. Estos contenedores son unidades ligeras y portátiles que incluyen:

- Código.
- Entorno de ejecución.
- Bibliotecas.
- Dependencias necesarias para ejecutar una aplicación.

2. Ventajas de Docker

1. **Consistencia**:
    a. Garantiza que las aplicaciones funcionen de la misma forma en diferentes entornos (local, servidor o nube).
    b. Elimina problemas como "funciona en mi máquina, pero no en la tuya".
2. **Velocidad y eficiencia**:
    a. Los contenedores se inician casi instantáneamente.
    b. Usan menos recursos que las máquinas virtuales tradicionales porque comparten el núcleo del sistema operativo.
3. **Portabilidad**:
    a. Ejecuta aplicaciones en cualquier sistema con Docker instalado (Windows, Linux, Mac, etc.).
4. **Gestión simplificada de dependencias**:
    a. Incluye todas las dependencias necesarias dentro del contenedor.
    b. Evita buscar versiones específicas de librerías o módulos.

3. Conceptos Clave de Docker

1. **Imágenes**:
    a. Plantillas de solo lectura con las instrucciones para crear contenedores.
    b. Ejemplo: Una imagen con Ubuntu y Python instalado para ejecutar un programa.
2. **Contenedores**:
    a. Instancia en ejecución de una imagen.
    b. Aislados, ligeros, y se pueden detener, reiniciar o eliminar sin afectar el sistema host.
3. **Dockerfile**:

    a. Script con instrucciones para construir una imagen de Docker.
    b. Incluye pasos como instalar software, copiar código, configurar, y ejecutar aplicaciones.
4. **Volúmenes**:
    a. Almacenamiento persistente para datos generados o utilizados por un contenedor.
    b. Permiten que los datos sobrevivan incluso si el contenedor se elimina.

4. Ejemplo Práctico

- **Problema**: Compartir una aplicación de Python que utiliza una base de datos específica (por ejemplo, InfluxDB) con requisitos de versiones concretas.
- **Solución con Docker**:
    - Crear un Dockerfile especificando las versiones y dependencias necesarias.
    - Generar una imagen de Docker.
    - Compartir la imagen con otra persona.
    - La persona puede crear un contenedor y ejecutar la aplicación sin problemas de configuración.

Instalación de Docker

1. Preparación del Sistema Operativo

Antes de instalar Docker, asegúrate de actualizar completamente tu sistema operativo.

1. **Actualizar paquetes**:

```
sudo apt update
```

2. **Actualizar el sistema**:

```
sudo apt upgrade
```

2. Descarga e Instalación de Docker

1. **Descargar el script de instalación de Docker**: Utiliza curl para obtener el instalador oficial:

```
curl -fsSL https://get.docker.com -o docker-install.sh
```

2. **Ejecutar el script**: Ejecuta el archivo descargado para instalar

```
Docker:sh docker-install.sh
```

3. Configuración de Permisos

1. **Añadir usuario al grupo Docker**: Esto permite ejecutar comandos de Docker sin usar sudo cada vez:

```
sudo usermod -aG docker <nombre_usuario>
```

Reemplaza <nombre_usuario> con tu nombre de usuario.

4. Verificación de la Instalación

1. **Probar un contenedor básico**: Ejecuta el contenedor de prueba "Hello World":

```
sudo docker run hello-world
```

   a. Si Docker no encuentra la imagen localmente, la descargará automáticamente desde el repositorio oficial.
   b. Verás el mensaje "Hello from Docker" si todo está instalado correctamente.

Uso Básico de Docker

1. Verificar la Instalación

1. Comprueba que Docker está instalado ejecutando:

```
docker --version
```

Esto mostrará la versión instalada.

2. Comandos Básicos de Docker

1. **Ejecutar un contenedor de prueba**:
   a. Usa el contenedor "Hello World" para verificar el funcionamiento:

```
sudo docker run hello-world
```

2. **Descargar imágenes desde Docker Hub**:
   a. Buscar y descargar una imagen con:

```
docker pull <nombre_imagen>
```

   b. Por ejemplo, para descargar **Nginx**:

```
docker pull nginx
```

   c. Descargar una versión específica:

```
docker pull nginx:1.19
```

3. **Listar imágenes locales**:

```
docker images
```

4. **Ejecutar una imagen en un contenedor**:
    a. Ejecutar en segundo plano (modo "detach") con puerto mapeado:

```
docker run -d -p 8080:80 nginx
```

5. **Ver contenedores en ejecución**:

```
docker ps
```

6. **Detener un contenedor**:

```
docker stop <id_contenedor>
```

7. **Eliminar un contenedor**:

```
docker rm <id_contenedor>
```

3. Crear una Imagen Personalizada

1. **Crear una aplicación de ejemplo**:
    a. Archivo app.py:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
   return "Hola desde Docker"

if __name__ == '__main__':
   app.run(host='0.0.0.0', port=5000)
```

    b. Archivo requirements.txt:

```
Flask==1.1.2
```

2. **Escribir un Dockerfile**:

```
FROM python
WORKDIR /app
COPY . .
RUN pip install --no-cache-dir -r requirements.txt
CMD ["python", "app.py"]
```

3. **Construir la imagen**:

```
docker build -t app
```

4. **Ejecutar el contenedor**:

```
docker run -d -p 5000:5000 --name app app
```

5. **Verificar en el navegador**:
    a. Accede a http://localhost:5000 para ver la aplicación en ejecución.

4. Gestión de Contenedores e Imágenes

1. **Listar todos los contenedores (incluidos los detenidos)**:

```
docker ps -a
```

2. **Eliminar contenedores inactivos**:

```
docker rm <id_contenedor>
```

3. **Eliminar imágenes no utilizadas**:

```
docker rmi <id_imagen>
```

4. **Limpiar recursos innecesarios**:

```
docker system prune
```

Uso de Docker Compose

1. ¿Qué es Docker Compose?

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones compuestas por múltiples contenedores Docker, mediante un archivo de configuración (docker-compose.yml). Es útil para proyectos complejos que necesitan varios servicios, como servidores web, APIs y bases de datos, trabajando de forma integrada.

2. Instalación de Docker Compose

1. **Actualizar el sistema**:

```
sudo apt update
```

2. **Instalar Docker Compose**:

```
sudo apt install docker-compose
```

3. **Verificar la instalación**:

```
docker-compose --version
```

3. Crear y Configurar un Proyecto con Docker Compose

1. **Crear un directorio para el proyecto**:

```
mkdir docker-compose
cd docker-compose
```

2. **Crear el archivo docker-compose.yml**:
   a. Ejemplo básico para un servidor web con **Nginx** y una base de datos

```
Redis:version: '3'

services:
  web:
    image: nginx
    ports:
      - "80:80"
    networks:
      - app_network

  redis:
    image: redis:alpine
    ports:
      - "6379:6379"
    networks:
      - app_network
```

```
networks:
  app_network:
    driver: bridge
```

b. Este archivo define:
   i. **Servicios**:
      1. web: Usa la imagen nginx y mapea el puerto 80.
      2. redis: Usa la imagen redis:alpine y mapea el puerto 6379.
   ii. **Red**: Conecta los contenedores mediante una red personalizada app_network.

4. Ejecutar y Administrar los Contenedores

1. **Iniciar los servicios**:
   a. Ejecución estándar:

```
docker-compose up
```

   b. Ejecución en segundo plano:

```
docker-compose up -d
```

2. **Verificar los servicios**:
   a. Accede a http://localhost para comprobar el servidor web (puerto 80).
   b. Usa herramientas específicas para interactuar con Redis en el puerto 6379.
3. **Ver contenedores en ejecución**:

```
docker ps
```

4. **Detener y limpiar los servicios**:
   a. Detener los servicios y borrar la configuración:

```
docker-compose down
```

5. Ventajas de Docker Compose

1. **Centralización**: Todos los servicios necesarios están definidos en un solo archivo.
2. **Escalabilidad**: Fácil de expandir para incluir más servicios.

3. **Portabilidad**: El archivo docker-compose.yml puede ser compartido para replicar configuraciones.
4. **Simplicidad**: Los comandos up, down, y ps facilitan la gestión de múltiples contenedores.