Práctica de Docker

MATI Fabricio A Suárez Domínguez

Primeros Comandos Básicos:

1. **Correr un Contenedor Docker**:
   - Usa el siguiente comando para ejecutar un contenedor Docker simple con la imagen `hello-world`:

```sh
docker run hello-world
```

   - Deberías ver un mensaje de bienvenida que confirma que Docker está funcionando correctamente.

2. **Listar Contenedores Activos**:
   - Usa el comando `docker ps` para listar los contenedores activos.

```sh
docker ps
```

3. **Detener un Contenedor**:
   - Encuentra el ID del contenedor que deseas detener (puedes usar `docker ps` para obtenerlo) y usa el siguiente comando para detenerlo:

```sh
docker stop <CONTAINER_ID>
```

Pasos para Crear una Aplicación Node.js

1. Configurar el Entorno de Trabajo

Primero, asegúrate de tener Node.js y npm instalados en tu sistema.

2. Crear el Directorio del Proyecto

Crea un nuevo directorio para tu proyecto y navega hasta él:

```sh
mkdir mi-app
cd mi-app
```

3. Inicializar un Proyecto Node.js

Dentro del directorio del proyecto, inicializa un nuevo proyecto Node.js. Esto creará un archivo `package.json`.

```sh
npm init -y
```

4. Instalar Dependencias

Instala `express`, un framework minimalista para Node.js que se utilizará en la aplicación.

```sh
npm install express
```

5. Crear la Estructura de Archivos

Crea los archivos necesarios para la aplicación:

```sh
touch Dockerfile package.json app.js
```

6. Editar `package.json`

Asegúrate de que tu `package.json` tenga el siguiente contenido:

```json
{
  "name": "mi-app",
  "version": "1.0.0",
  "description": "Una aplicación simple",
  "main": "app.js",
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

6. Editar `package.json`

Asegúrate de que tu `package.json` tenga el siguiente contenido:

Json

```json
{
  "name": "mi-app",
  "version": "1.0.0",
  "description": "Una aplicación simple",
  "main": "app.js",
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

7. Crear `app.js`

Agrega el siguiente contenido a `app.js`:

Javascript

```javascript
const express = require('express');
const app = express();
const port = 8080;

app.get('/', (req, res) => {
  res.send('¡Hola, mundo desde Docker!');
});

app.listen(port, () => {
  console.log(`La aplicación está escuchando en http://localhost:${port}`);
});
```

8. Crear el Dockerfile

Añade el siguiente contenido a tu `Dockerfile`:

Dockerfile

```
# Usa una imagen base de Node.js
FROM node:14

# Crea un directorio de trabajo
WORKDIR /usr/src/app

# Copia los archivos de package.json y package-lock.json
COPY package*.json ./

# Instala las dependencias
RUN npm install

# Copia el resto de los archivos de la aplicación
COPY . .

# Expone el puerto 8080
EXPOSE 8080

# Comando para ejecutar la aplicación
CMD [ "node", "app.js" ]
```

9. Construir la Imagen Docker

Construye la imagen Docker usando el Dockerfile:

```sh
docker build -t mi-app .
```

10. Correr el Contenedor Docker

Finalmente, ejecuta el contenedor basado en la imagen `mi-app`:

```sh
docker run -p 8080:8080 mi-app
```

Tu aplicación debería estar funcionando ahora y accesible en `http://localhost:8080`.

Creación de Contenedores a Partir de Imágenes:

1. **Correr un Contenedor**:
   - Usa el siguiente comando para correr un contenedor basado en la imagen `mi-app-node`:

```sh
docker run -p 8080:8080 mi-app-node
```

   - Esto ejecutará la aplicación Node.js en el puerto 8080.

2. **Listar Contenedores Activos**:
   - Usa `docker ps` para listar todos los contenedores activos:

```sh
docker ps
```

3. **Detener y Eliminar Contenedores**:
   - Detén un contenedor usando su ID:

```sh
docker stop <CONTAINER_ID>
```

   - Elimina un contenedor detenido:

```sh
docker rm <CONTAINER_ID>
```

Uso de Volúmenes para Persistencia de Datos:

1. **Crear un Volumen Docker**:
   - Usa el siguiente comando para crear un volumen:

```sh
docker volume create mi-volumen
```

2. **Montar un Volumen en un Contenedor**:
   - Corre un contenedor y monta el volumen creado:

```sh
docker run -d -v mi-volumen:/data --name mi-contenedor busybox
```

   - En este ejemplo, el volumen `mi-volumen` está montado en el directorio `/data` dentro del contenedor.

3. **Verificar el Volumen**:
   - Usa `docker volume ls` para listar todos los volúmenes:

```sh
docker volume ls
```

Montar un volumen en Docker sirve para gestionar el almacenamiento persistente de datos. Aquí te explico los beneficios clave:

Beneficios de Montar un Volumen en Docker

1. **Persistencia de Datos**: Los datos almacenados en un contenedor se eliminan cuando el contenedor se detiene o se elimina. Al usar volúmenes, los datos persisten más allá del ciclo de vida del contenedor.
2. **Separación de Datos y Aplicación**: Permite mantener los datos independientes del contenedor de la aplicación, facilitando la actualización y mantenimiento de la aplicación sin riesgo de pérdida de datos.
3. **Compartir Datos entre Contenedores**: Puedes montar un volumen en varios contenedores, permitiendo que compartan y accedan a los mismos datos.
4. **Backup y Restore**: Facilita la realización de copias de seguridad y la restauración de datos. Puedes montar un volumen en un contenedor temporal para hacer una copia de seguridad de los datos y luego desmontarlo cuando termines.

Crear y Ejecutar un Contenedor con el Volumen Montado

Usa el comando `docker run` para crear y ejecutar un contenedor con el volumen montado. Por ejemplo, vamos a usar una imagen de BusyBox para demostrarlo:

```sh
docker run -it --name mi-contenedor -v mi-volumen:/data busybox
```

En este comando:

- `-it` permite interactuar con el contenedor.
- `--name mi-contenedor` le da un nombre al contenedor.
- `-v mi-volumen:/data` monta el volumen `mi-volumen` en el directorio `/data` dentro del contenedor.

Guardar Información en el Volumen

Ahora que el contenedor está en ejecución y el volumen montado, puedes guardar información en el volumen. Dentro del contenedor, navega al directorio `/data` y crea un archivo:

```sh
cd /data
echo "Hola, mundo desde Docker!" > archivo.txt
```

Este comando crea un archivo llamado `archivo.txt` en el directorio `/data` y escribe "Hola, mundo desde Docker!" en él.