Contenedores Docker

Los contenedores Docker son una tecnología de virtualización ligera que permite empaquetar aplicaciones y sus dependencias en unidades estandarizadas llamadas contenedores. A diferencia de las máquinas virtuales tradicionales, los contenedores comparten el kernel del sistema operativo host, lo que los hace más eficientes.

Conceptos clave

- **Imagen Docker**: Es una plantilla de solo lectura que contiene el código, bibliotecas, dependencias y herramientas necesarias para ejecutar una aplicación.

- **Contenedor**: Es una instancia en ejecución de una imagen Docker. Puedes crear múltiples contenedores a partir de la misma imagen.

- **Dockerfile**: Archivo de texto que contiene instrucciones para construir una imagen Docker.

- **Docker Hub**: Repositorio público donde se almacenan imágenes Docker.

Ventajas principales

1. **Portabilidad**: La aplicación funciona igual en cualquier entorno compatible con Docker.
2. **Aislamiento**: Cada contenedor opera de forma independiente.
3. **Eficiencia**: Consumen menos recursos que las máquinas virtuales.
4. **Escalabilidad**: Fácil de escalar horizontalmente.
5. **Despliegue rápido**: Los contenedores se inician en segundos.

Comandos básicos

```
# Descargar una imagen
docker pull nombre_imagen:tag

# Crear y ejecutar un contenedor
```

```
docker run nombre_imagen

# Listar contenedores en ejecución
docker ps

# Detener un contenedor
docker stop id_contenedor

# Eliminar un contenedor
docker rm id_contenedor
```

Los archivos que creas dentro de un contenedor Docker no se guardan automáticamente fuera del mismo, y cuando el contenedor se elimina, esos archivos se pierden. Para casos como un proyecto de data science con cuadernos Jupyter, hay varias formas de hacer que tus datos persistan:

1. Volúmenes de Docker

Los volúmenes son la manera recomendada para persistir datos en Docker:

```
# Crear un volumen
docker volume create mi_volumen_jupyter

# Ejecutar un contenedor con el volumen montado
docker run -p 8000:8888 -v mi_volumen_jupyter:/home/jovyan/work
jupyter/datascience-notebook
```

Aquí, cualquier archivo que guardes en la carpeta `/home/jovyan/work` dentro del contenedor se almacenará en el volumen `mi_volumen_jupyter` y persistirá incluso si eliminas el contenedor.

2. Bind mounts (montajes de directorio)

Esta opción vincula un directorio de tu máquina local con uno dentro del contenedor:

```
# Ejecutar contenedor con directorio local montado
docker run -p 8000:8888 -v /ruta/local/en/tu/pc:/home/jovyan/work
jupyter/datascience-notebook
```

Con esto, todos los notebooks y archivos que crees en `/home/jovyan/work` dentro del contenedor aparecerán en tu directorio local `/ruta/local/en/tu/pc`, y viceversa.

3. Usando Docker Compose

Para proyectos más complejos, puedes usar Docker Compose para definir toda la configuración:

```
version: '3'
services:
  jupyter:
    image: jupyter/datascience-notebook
    ports:
      - "8000:8888"
    volumes:
      - ./notebooks:/home/jovyan/work
```

Consideraciones para proyectos de Data Science

- Monta directorios separados para datos, notebooks y modelos
- Para datasets grandes, considera usar volúmenes específicos
- Establece permisos adecuados en los directorios montados

Vamos a crear un ejemplo práctico de un proyecto de data science usando Docker con NumPy, asegurándonos de que los datos persistan correctamente.

Ejemplo: Proyecto de Data Science con NumPy y persistencia de datos

1. Estructura de directorios local

Primero, crea esta estructura en tu máquina local:

```
mi_proyecto_ds/
├── notebooks/      # Para guardar tus Jupyter notebooks
├── data/           # Para guardar datasets
├── results/        # Para guardar modelos y resultados
└── docker-compose.yml
```

2. Archivo docker-compose.yml

Crea el archivo `docker-compose.yml` con el siguiente contenido:

```yaml
version: '3'
services:
  jupyter:
    image: jupyter/scipy-notebook
    ports:
      - "8000:8888"
    volumes:
      - ./notebooks:/home/jovyan/notebooks
      - ./data:/home/jovyan/data
      - ./results:/home/jovyan/results
    environment:
      - JUPYTER_ENABLE_LAB=yes
    command: start-notebook.sh --NotebookApp.token='' --NotebookApp.password=''
```

3. Iniciar el contenedor

En la terminal, desde el directorio `mi_proyecto_ds`, ejecuta:

```
docker-compose up
```

Esto iniciará el contenedor Jupyter. Podrás acceder a Jupyter Lab en http://localhost:8888.

4. Ejemplo de notebook con NumPy

Crea un nuevo notebook en Jupyter y guárdalo en la carpeta /notebooks. Aquí hay un ejemplo de código que:

1. Usa NumPy para crear un array
2. Lo guarda persistentemente
3. Luego lo vuelve a cargar

```
import numpy as np
import os

# Crear un directorio para resultados si no existe
os.makedirs('/home/jovyan/results', exist_ok=True)

# Crear datos de ejemplo con NumPy
datos = np.random.randn(1000, 5)  # 1000 filas, 5 columnas de números aleatorios
print("Array creado:", datos.shape)

# Guardar el array en el volumen persistente
ruta_archivo = '/home/jovyan/results/mi_array.npy'
np.save(ruta_archivo, datos)
print(f"Datos guardados en: {ruta_archivo}")

# Verificar que se guardó correctamente cargándolo de nuevo
datos_cargados = np.load(ruta_archivo)
print("Datos cargados correctamente:", np.array_equal(datos, datos_cargados))

# Mostrar algunas estadísticas básicas
print("\nEstadísticas del dataset:")
print("Media por columna:", datos.mean(axis=0))
print("Desviación estándar:", datos.std(axis=0))
```

5. Características clave de esta configuración

- **Persistencia**: Todos los notebooks que crees en Jupyter se guardarán en la carpeta `notebooks` de tu máquina local.
- **Datos**: Puedes colocar tus datasets en la carpeta `data` local, y estarán disponibles en `/home/jovyan/data` dentro del contenedor.
- **Resultados**: Los arrays de NumPy, modelos y otros resultados guardados en `/home/jovyan/results` dentro del contenedor, persistirán en la carpeta `results` local.

6. Para detener el contenedor

Cuando termines, puedes detener el contenedor con:

```
docker-compose down
```

Tus datos seguirán intactos en las carpetas locales incluso después de detener o eliminar el contenedor.

No, un contenedor Docker no puede tener varias imágenes. La relación entre imágenes y contenedores en Docker funciona de la siguiente manera:

1. **Un contenedor se crea a partir de una única imagen**: Cada contenedor es una instancia en ejecución de exactamente una imagen Docker.

2. **Lo que sí puede tener múltiples capas**: Una imagen Docker está compuesta por múltiples capas (layers) que representan cada instrucción en el Dockerfile. Estas capas se apilan para formar la imagen final.

3. **Relación entre imágenes y contenedores**:

    - Una imagen puede generar múltiples contenedores (muchos contenedores pueden basarse en la misma imagen)
    - Un contenedor solo puede basarse en una única imagen

Lo que sí puedes hacer:

1. Imágenes con múltiples servicios

Puedes crear una imagen que contenga múltiples aplicaciones o servicios, por ejemplo, una imagen que incluya Python, NumPy, Pandas y Jupyter en un solo paquete.

2. Composición con Docker Compose

Para ejecutar aplicaciones que requieren múltiples imágenes, usamos Docker Compose:

```
version: '3'
services:
  jupyter:
    image: jupyter/scipy-notebook
    ports:
      - "8000:8888"

  postgres:
    image: postgres:13
    environment:
      - POSTGRES_PASSWORD=secret
    ports:
      - "5432:5432"
```

Este archivo define una aplicación compuesta por dos contenedores separados, cada uno basado en su propia imagen.

3. Imagen multi-etapa (multi-stage builds)

En un Dockerfile puedes usar múltiples imágenes para construir tu imagen final, pero el contenedor resultante sigue basándose en una sola imagen:

```
# Primera etapa - compilación
FROM python:3.9 AS builder
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt

# Segunda etapa - imagen final
FROM python:3.9-slim
COPY --from=builder /usr/local/lib/python3.9/site-packages
/usr/local/lib/python3.9/site-packages
WORKDIR /app
```

```
COPY . .
CMD ["python", "app.py"]
```

Sí, puedes crear imágenes nuevas a partir de un contenedor existente. Esta es una práctica común cuando has realizado cambios o configuraciones en un contenedor y quieres preservarlos en una nueva imagen. El proceso se llama "commit" de un contenedor.

Creando una imagen a partir de un contenedor

1. Usando **docker commit**

```
# Sintaxis básica
docker commit [opciones] CONTENEDOR [REPOSITORIO[:ETIQUETA]]

# Ejemplo práctico
docker commit mi_contenedor_jupyter mi_imagen_jupyter:v1
```

2. Ejemplo paso a paso

Imagina que tienes un contenedor de Jupyter donde has:

- Instalado nuevas bibliotecas (scikit-learn, matplotlib)
- Configurado preferencias personales
- Añadido datos o scripts útiles

```
# 1. Lista tus contenedores en ejecución
docker ps

# 2. Realiza el commit del contenedor para crear una imagen
docker commit 12345abcde mi_jupyter_personalizado:v1

# 3. Verifica que la imagen se creó correctamente
docker images

# 4. Crea un nuevo contenedor a partir de tu imagen personalizada
docker run -p 8888:8888 mi_jupyter_personalizado:v1
```

3. Añadiendo metadatos durante el commit

Puedes incluir información adicional durante el commit:

```
docker commit \
  --author "Tu Nombre <tu@email.com>" \
  --message "Imagen Jupyter con scikit-learn y matplotlib preinstalados" \
  mi_contenedor_jupyter \
  mi_jupyter_ds:v1
```

Buenas prácticas

1. **Usa Dockerfiles cuando sea posible**: Aunque el comando `commit` es útil, es mejor práctica definir tus imágenes con Dockerfiles para que sean reproducibles.

2. **Documenta los cambios**: Añade metadata o documentación sobre qué cambios realizaste.

3. **Limpia antes de commit**: Elimina archivos temporales, caches y datos innecesarios antes de crear la imagen.

4. **Guarda en un registro**: Puedes subir tu imagen personalizada a Docker Hub u otro registro:

```
docker tag mi_jupyter_personalizado:v1 usuario/mi_jupyter:v1
docker push usuario/mi_jupyter:v1
```

Esta capacidad es especialmente útil en entornos de data science donde puedes crear una imagen con todas tus bibliotecas, configuraciones y herramientas favoritas preinstaladas, facilitando así la reproducibilidad de tu entorno de trabajo.