Taller Docker

□□ Integrantes:

- □□ Daniel Sarmiento
- □□ Redactor: Jorge Garcia

□□ Material de apoyo:

- □□ **Diapositivas:** Ver presentaciones
- Repositorio

Primera imagen Segunda imagen

Taller docker

Uno de los problemas actuales es como compartir modelos, con sus configuraciones y dependencias, a otras personas. Una manera de hacer esto es con docker.

Docker es una plataforma de contenedorización que permite ejecutar aplicaciones dentro de entornos aislados denominados *containers*.
Cada contenedor incluye el código, las dependencias y configuraciones necesarias, asegurando que el comportamiento de la aplicación sea idéntico sin importar el sistema donde se ejecute.

Aunque a menudo se compara con una máquina virtual (VM), Docker no virtualiza hardware. En cambio, utiliza el *kernel* del sistema operativo anfitrión junto con tecnologías como *namespaces* y *cgroups* para aislar procesos, lo que lo hace mucho más liviano y rápido.

Comparativa de Container vs Virtual Machine

Comparativa de container vs virtual machine

En la ilustración se observa que las VMs requieren un sistema operativo completo dentro del host, mientras que los contenedores comparten el mismo kernel. Esto permite que un contenedor arranque en segundos y consuma una fracción de los recursos.

Instalación

Para la instalación, en linux se pueden utilizar sus paqueterías oficiales.

Linux

Arch Linux / Manjaro

```
sudo pacman -S docker
sudo systemctl enable --now docker
```

Ubuntu / Debian

```
sudo apt update
sudo apt install docker.io -y
sudo systemctl enable --now docker
```

Fedora

```
sudo dnf install docker -y
sudo systemctl enable --now docker
```

Una vez instalado, puedes verificar su funcionamiento con:

```
docker run hello-world
```

Windows y macOS

Docker ofrece una aplicación oficial llamada **Docker Desktop**, que integra todas las herramientas necesarias. Puede descargarse desde el sitio oficial: https://www.docker.com/products/docker-desktop

---

Comandos básicos

Comando	Descripción
`docker ps -a`	Muestra todos los contenedores (incluidos los detenidos)
`docker pull <imagen>`	Descarga una imagen desde un registro (por ejemplo, Docker Hub)
`docker run <imagen>`	Ejecuta una imagen en un nuevo contenedor
`docker images`	Lista las imágenes disponibles en el sistema
`docker rm <id>`	Elimina un contenedor
`docker rmi <id>`	Elimina una imagen
`docker exec -it <nombre> bash`	Abre una sesión interactiva dentro de un contenedor

Dockerfile

Para construir una imagen personalizada, se utiliza un archivo llamado `Dockerfile`. Este define los pasos necesarios para preparar el entorno de ejecución.

Ejemplo simple:

```
FROM docker.io/postgres:latest  # Imagen base
RUN apt update && apt upgrade -y
ENV POSTGRES_USER=user \
    POSTGRES_PASSWORD=password \
    POSTGRES_DB=exampledb
```

Cada instrucción genera una **capa (layer)**, lo que permite que las reconstrucciones sean más rápidas y eficientes.

Contenerizando un Modelo de IA con PyTorch

Imaginemos que queremos crear un contenedor para entrenar un modelo convolucional sencillo en el dataset CIFAR-10. El modelo en PyTorch podría ser el siguiente:

```
import torch.nn as nn
import torch.nn.functional as F
import torch

class Net(nn.Module):
    def __init__(self, dropout_rate=0.3):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
        self.dropout = nn.Dropout(dropout_rate)
        self._initialize_weights()

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, (nn.Conv2d, nn.Linear)):
                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
                if m.bias is not None:
                    nn.init.constant_(m.bias, 0)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = self.dropout(F.relu(self.fc2(x)))
        x = self.fc3(x)
        return x
```

Para contenerizarlo, creamos un `Dockerfile` como este:

```
# Imagen base ligera con Python
FROM python:3.12-slim

# Directorio de trabajo
WORKDIR /app
```

```
# Copiar el código fuente
COPY . /app

# Instalar dependencias (CPU-only)
RUN pip install torch torchvision --index-url https://download.pytorch.org/whl/cpu
RUN pip install matplotlib scikit-learn albumentations tqdm tensorboard

# Instalar dependencias adicionales (si existen)
RUN if [ -f requirements.txt ]; then pip install -r requirements.txt; fi

# Variables de entorno
ENV PYTHONUNBUFFERED=1
ENV PYTHONDONTWRITEBYTECODE=1

# Comando por defecto
CMD ["python", "train.py"]
```

> □□ Si cuentas con GPU y drivers CUDA, puedes usar una imagen base con soporte GPU, como:
>
> ```
> FROM pytorch/pytorch:2.2.0-cuda12.1-cudnn8-runtime
> ```
>
> e instalando las dependencias con la versión de cuda, es decir, la predeterminada
>
> ```
> RUN pip install torch torchvision
> ```

Orquestación y Docker Compose

En proyectos de IA es habitual tener múltiples tareas: entrenamiento, evaluación y monitoreo. Con **Docker Compose** podemos definir todos los servicios en un solo archivo `docker-compose.yml`:

```
services:
  train:
```

```
  build:
    context: .
    dockerfile: Dockerfile
  image: pytorch-uv-app:latest
  container_name: pytorch-train
  command: python train.py
  volumes:
    - ./data:/app/data
    - ./model:/app/model
    - ./runs:/app/runs
    - ./checkpoints:/app/checkpoints
  restart: "no"

test:
  image: pytorch-uv-app:latest
  container_name: pytorch-test
  command: python test.py
  volumes:
    - ./data:/app/data
    - ./model:/app/model
    - ./runs:/app/runs
    - ./checkpoints:/app/checkpoints
  restart: "no"

tensorboard:
  image: python:3.12-slim
  container_name: pytorch-tensorboard
  command: >
    sh -c "pip install --quiet tensorboard>=2.20.0 &&
           tensorboard --logdir=/app/runs --host=0.0.0.0 --port=6006"
  ports:
    - "6006:6006"
  volumes:
    - ./runs:/app/runs
  restart: unless-stopped
  profiles:
    - monitoring
```

Ejecución

Ejecutar el entrenamiento:

```
docker compose up train
```

Ejecutar las pruebas:

```
docker compose up test
```

Iniciar TensorBoard para monitoreo:

```
docker compose --profile monitoring up tensorboard
```

Luego, accede a http://localhost:6006 para visualizar las métricas.

— Jorge García

> **Recursos adicionales:**
>
> - Documentación oficial de Docker
> - Imágenes oficiales de PyTorch
> - Guía de Docker Compose

Revision #12
Created 24 October 2025 01:05:00 by Jorge Garcia
Updated 24 October 2025 21:23:47 by Jorge Garcia