02 - Dockerfile Básico

Crear tus propias imágenes Docker

Objetivo

Aprender a crear tus propias imágenes Docker usando Dockerfile.

¿Qué aprenderás?

- Sintaxis básica de Dockerfile
- Comandos: FROM, RUN, COPY, CMD, EXPOSE
- Cómo construir y ejecutar imágenes personalizadas

¿Qué es un Dockerfile?

Archivo de texto con **instrucciones paso a paso** para construir una imagen Docker.

Analogía: Como una receta de cocina que dice qué ingredientes usar y cómo prepararlos.

¿Por qué usar Dockerfile?

✅ **Reproducibilidad**: Mismo resultado siempre
✅ **Automatización**: No instalar manualmente
✅ **Versionado**: Control de cambios
✅ **Colaboración**: Compartir fácilmente

Estructura del Proyecto

```
02-dockerfile-basico/
├── README.md
├── Dockerfile
├── app.py
└── requirements.txt
```

Dockerfile Básico

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py .
EXPOSE 5000
CMD ["python", "app.py"]
```

Explicación Línea por Línea

Comando	Descripción
`FROM`	Imagen base (Python 3.9)
`WORKDIR`	Establece directorio de trabajo
`COPY`	Copia archivos al contenedor
`RUN`	Ejecuta comandos durante build
`EXPOSE`	Documenta el puerto usado
`CMD`	Comando por defecto al ejecutar

Construir la Imagen

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

Explicación:

- `-t mi-app-python`: Nombre (tag) de la imagen
- `.`: Contexto de construcción (directorio actual)

Ejecutar el Contenedor

```
docker run -p 5000:5000 mi-app-python
```

Visita `http://localhost:5000` en tu navegador

Optimizar el Dockerfile

❌ Orden incorrecto

```
COPY . .
RUN pip install -r requirements.txt
```

✅ Orden correcto

```
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY app.py .
```

¿Por qué? Mejor uso de caché de Docker

Variables de Entorno

```
ENV FLASK_ENV=production
```

Ejecutar con variable personalizada:

```
docker run -e FLASK_ENV=development mi-app
```

Comandos Útiles

```
# Construir sin caché
docker build --no-cache -t mi-app .

# Ver historial de capas
docker history mi-app-python

# Construir con tag específico
docker build -t mi-app:v1.0 .
```

Capas en Docker

Cada comando en Dockerfile crea una **capa**:

- Las capas se cachean
- Si una capa cambia, las siguientes se reconstruyen
- **Orden importante**: Cambios frecuentes al final

Ejemplo:

```
Capa 1: FROM python:3.9-slim (casi nunca cambia)
Capa 2: RUN pip install (cambia poco)
Capa 3: COPY app.py (cambia frecuentemente)
```

Buenas Prácticas

✅ Usa imágenes base oficiales y específicas
✅ Ordena comandos de menos a más frecuentes cambios
✅ Usa `.dockerignore` para excluir archivos
✅ Combina comandos RUN cuando sea posible
✅ Usa usuarios no-root cuando sea posible

Práctica

1. Crea un Dockerfile para tu aplicación
2. Construye la imagen
3. Ejecuta el contenedor
4. Optimiza el orden de los comandos
5. Agrega variables de entorno

Siguiente Paso

Aprende sobre volúmenes para persistir datos.

Módulo 03: Volúmenes

Preguntas?

¡Tiempo para preguntas y práctica!