06 - Multi-Stage Builds

Crear imágenes más pequeñas y optimizadas

Objetivo

Aprender a usar multi-stage builds para crear imágenes Docker más pequeñas y optimizadas.

¿Qué aprenderás?

- Cómo usar múltiples etapas en un Dockerfile
- Reducir el tamaño de las imágenes finales
- Separar dependencias de build de las de runtime

¿Por qué Multi-Stage Builds?

Problema: Una imagen tradicional incluye:

- Código fuente
- Herramientas de compilación
- Dependencias de desarrollo
- Todo junto = imagen grande

Solución: Separar en etapas

- **Etapa 1**: Compilar (con herramientas)
- **Etapa 2**: Runtime (solo lo necesario)

Ventajas de Multi-Stage

- **Imágenes más pequeñas**: Solo incluyes lo necesario
- **Seguridad**: No incluyes herramientas de build
- **Optimización**: Diferentes etapas para diferentes propósitos
- **Rapidez**: Descargas más rápidas

Ejemplo: De 900MB a 150MB (83% más pequeño)

Problema: Dockerfile Tradicional

```
FROM node:18
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
RUN npm run build
CMD ["node", "dist/index.js"]
```

Problema: La imagen incluye:

- Node.js completo
- npm y dependencias de desarrollo
- Código fuente
- Herramientas de build

Tamaño: ~900MB

Solución: Multi-Stage Build

```
# Etapa 1: Build
FROM node:18 AS builder
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
RUN npm run build

# Etapa 2: Producción
FROM node:18-alpine
WORKDIR /app
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/node_modules ./node_modules
COPY package.json .
CMD ["node", "dist/index.js"]
```

Tamaño: ~150MB

Ventajas del Multi-Stage

✅ Solo incluye lo necesario en la etapa final
✅ Usa `alpine` (imagen más pequeña)
✅ No incluye código fuente ni herramientas de build
✅ Más seguro (menos superficie de ataque)

Ejemplo: Python

```
# Etapa 1: Build
FROM python:3.9-slim AS builder
WORKDIR /app
COPY requirements.txt .
RUN pip install --user --no-cache-dir -r requirements.txt

# Etapa 2: Runtime
FROM python:3.9-slim
WORKDIR /app
COPY --from=builder /root/.local /root/.local
COPY app.py .
ENV PATH=/root/.local/bin:$PATH
CMD ["python", "app.py"]
```

Ejemplo: Go

```
# Etapa 1: Compilar
FROM golang:1.21 AS builder
WORKDIR /app
COPY go.mod go.sum .
RUN go mod download
COPY . .
RUN CGO_ENABLED=0 GOOS=linux go build -o /app/server

# Etapa 2: Runtime mínimo
FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /app/server .
CMD ["./server"]
```

Resultado: Imagen de solo ~10-20MB

Construir Etapa Específica

```
# Construir solo la etapa "builder"
docker build --target builder -t mi-app:builder .

# Construir etapa de producción
docker build --target production -t mi-app:prod .
```

Comparación de Tamaños

Enfoque	Tamaño
Single-stage (node:18)	~900MB
Multi-stage (node:18-alpine)	~150MB
Multi-stage optimizado	~50MB

Mejores Prácticas

1. ✅ Usa imágenes base pequeñas en la etapa final (alpine, slim)
2. ✅ Copia solo archivos necesarios con `--from`
3. ✅ Nombra las etapas con `AS` para claridad
4. ✅ Separa dependencias de desarrollo de producción
5. ✅ Usa `.dockerignore` para excluir archivos innecesarios

Docker Compose con Múltiples Dockerfiles

Idea: Comparar diferentes enfoques en un solo archivo

```yaml
services:
  app-optimizado:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "3000:3000"

  app-tradicional:
    build:
      context: .
      dockerfile: Dockerfile.tradicional
    ports:
      - "3001:3000"

  app-python:
    build:
      context: .
      dockerfile: Dockerfile.python
    ports:
      - "5000:5000"
```

Usar Docker Compose

```
# Construir todos los servicios
docker compose build

# Iniciar todos los servicios
docker compose up -d

# Comparar tamaños de imágenes
docker images | grep multi-stage

# Ver logs
docker compose logs app-optimizado
```

Ventaja: Puedes comparar todos los enfoques fácilmente

Práctica

1. Crea un Dockerfile tradicional
2. Conviértelo a multi-stage
3. Compara los tamaños de las imágenes
4. Optimiza aún más

Siguiente Paso

Aprende Docker Compose avanzado con múltiples servicios.

Módulo 07: Docker Compose Avanzado

Preguntas?

¡Tiempo para preguntas y práctica!