08 - Optimización de Imágenes

Técnicas avanzadas para optimizar tamaño y rendimiento

Objetivo

Aprender técnicas avanzadas para optimizar el tamaño y rendimiento de las imágenes Docker.

¿Por qué optimizar?

- ⚡ Descargas más rápidas
- 💾 Menos espacio en disco
- 🚀 Despliegues más rápidos
- 💰 Menor costo en la nube

Principio Fundamental

Menos capas = Imagen más pequeña

Cada comando en Dockerfile crea una capa. Menos comandos = menos capas = imagen más pequeña.

¿Qué aprenderás?

- Reducir el tamaño de imágenes
- Mejorar tiempos de build
- Optimizar capas y caché
- Usar .dockerignore eficientemente

Estrategia 1: Imágenes Base Pequeñas

❌ Mal

```
FROM ubuntu:latest
RUN apt-get update && apt-get install -y python3
```

✅ Bien

```
FROM python:3.9-alpine
# Alpine es ~5MB vs Ubuntu ~70MB
```

Estrategia 2: Combinar Comandos RUN

❌ Mal (múltiples capas)

```
RUN apt-get update
RUN apt-get install -y curl
RUN apt-get install -y git
RUN apt-get clean
```

✅ Bien (una sola capa)

```
RUN apt-get update && \
    apt-get install -y curl git && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
```

Estrategia 3: .dockerignore

Crear .dockerignore

```
node_modules
npm-debug.log
.git
.gitignore
README.md
.env
*.md
.DS_Store
```

Beneficios:

- Builds más rápidos
- Imágenes más pequeñas
- No incluir archivos sensibles

Estrategia 4: Orden de Capas

❌ Orden incorrecto

```
FROM node:18
WORKDIR /app
COPY . .              # Cambia frecuentemente
RUN npm install       # Dependencias cambian menos
```

Problema: Cada cambio en código invalida la caché de `npm install`

Estrategia 4: Orden de Capas

✅ Orden correcto

```
FROM node:18
WORKDIR /app
COPY package.json package-lock.json .  # Cambia menos
RUN npm ci                              # Se cachea mejor
COPY . .                                # Cambia frecuentemente
```

Ventaja: Mejor uso de caché de Docker

Estrategia 5: Multi-Stage Optimizado

```
# Etapa 1: Dependencias (caché separado)
FROM node:18-alpine AS deps
WORKDIR /app
COPY package.json package-lock.json .
RUN npm ci --only=production && \
    npm cache clean --force

# Etapa 2: Build
FROM node:18-alpine AS builder
WORKDIR /app
COPY package.json package-lock.json .
RUN npm ci
COPY . .
RUN npm run build

# Etapa 3: Producción mínima
FROM node:18-alpine
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
```

Estrategia 6: Limpiar en la Misma Capa

❌ Mal

```
RUN apt-get update
RUN apt-get install -y build-essential
RUN apt-get clean
RUN rm -rf /var/lib/apt/lists/*
```

✅ Bien

```
RUN apt-get update && \
    apt-get install -y build-essential && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
```

Analizar Imágenes

```
# Ver historial y tamaños
docker history mi-imagen

# Ver tamaño de cada capa
docker history --human --format "{{.CreatedBy}}: {{.Size}}" mi-imagen

# Ver espacio usado
docker system df
```

Checklist de Optimización

- [ ] Usar imágenes base pequeñas (alpine, slim)
- [ ] Combinar comandos RUN
- [ ] Ordenar capas de menos a más cambios
- [ ] Usar .dockerignore
- [ ] Multi-stage builds
- [ ] Limpiar en la misma capa
- [ ] Usar cache mounts (BuildKit)
- [ ] Eliminar dependencias de desarrollo
- [ ] Usar tags específicos

Docker Compose para Comparar

Idea: Comparar versión optimizada vs no optimizada

```yaml
services:
  app-no-optimizado:
    build:
      context: .
      dockerfile: Dockerfile.no-optimizado
    ports:
      - "5001:5000"

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

Comparar con Docker Compose

```
# Construir ambas versiones
docker compose build

# Comparar tamaños
docker images | grep optimizacion

# Iniciar ambas
docker compose up -d

# Ver diferencias
docker compose ps
```

Ventaja: Comparación directa de tamaños y rendimiento

Comparación de Tamaños

Estrategia	Tamaño
Sin optimizar	~900MB
Alpine base	~300MB
Multi-stage	~150MB
Multi-stage + optimizado	~50MB
Distroless	~20MB

Práctica

1. Crea una imagen sin optimizar
2. Aplica técnicas de optimización
3. Compara los tamaños
4. Analiza las capas

Siguiente Paso

Aprende sobre healthchecks y logging para aplicaciones en producción.

Módulo 09: Healthchecks y Logging

Preguntas?

¡Tiempo para preguntas y práctica!