MÓDULO 10: DOCKER AVANZADO

Multi-stage Builds, Distroless, Seguridad y Optimización

Guía MLOps v5.0: Senior Edition | DuqueOM | Noviembre 2025

MÓDULO 10: Docker Avanzado

De 1.2GB a 150MB: El Arte de las Imágenes Optimizadas

"Una imagen Docker de 1GB es una confesión de pereza."

Duración	Teoría	Práctica
5-6 horas	25%	75%

Los 3 Niveles de Madurez Docker

10.1 Nivel 1: El Dockerfile Básico (Anti-patrón)

```dockerfile
# × NIVEL 1: Funcional pero MALO
FROM python:3.11

WORKDIR /app

# Copia TODO (incluyendo .git, __pycache__, etc.)
COPY . .

# Instala como root
RUN pip install -r requirements.txt

# Expone puerto sin documentar
EXPOSE 8000

# Corre como root
CMD ["python", "app/fastapi_app.py"]

# PROBLEMAS:
# • Imagen ~1.2GB
# • Corre como root (inseguro)
# • Incluye archivos innecesarios
# • Cache de layers ineficiente
# • Sin health check
```

10.2 Nivel 2: Multi-stage Optimizado

```dockerfile
#  NIVEL 2: Multi-stage build
# Dockerfile

# ═══════════════════════════════════════════════════════════
# STAGE 1: Builder (instala dependencias)
# ═══════════════════════════════════════════════════════════
FROM python:3.11-slim AS builder

WORKDIR /app

# Instalar dependencias de compilación
RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential \
    && rm -rf /var/lib/apt/lists/*

# Copiar solo requirements primero (mejor cache)
COPY requirements.txt .

# Crear virtualenv e instalar
RUN python -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"
RUN pip install --no-cache-dir --upgrade pip \
    && pip install --no-cache-dir -r requirements.txt

# ═══════════════════════════════════════════════════════════
# STAGE 2: Runtime (imagen final limpia)
# ═══════════════════════════════════════════════════════════
FROM python:3.11-slim AS runtime

WORKDIR /app

# Crear usuario no-root
RUN groupadd --gid 1000 appgroup \
    && useradd --uid 1000 --gid appgroup --shell /bin/bash appuser

# Copiar virtualenv del builder
COPY --from=builder /opt/venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

# Copiar solo código necesario
COPY --chown=appuser:appgroup src/ ./src/
COPY --chown=appuser:appgroup app/ ./app/
COPY --chown=appuser:appgroup configs/ ./configs/
COPY --chown=appuser:appgroup models/ ./models/

# Cambiar a usuario no-root
USER appuser

# Variables de entorno
ENV PYTHONUNBUFFERED=1 \
    PYTHONDONTWRITEBYTECODE=1 \
    PORT=8000

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:${PORT}/health || exit 1

EXPOSE ${PORT}

# Comando de inicio
CMD ["uvicorn", "app.fastapi_app:app", "--host", "0.0.0.0", "--port", "8000"]
```

10.3 Nivel 3: Production-Ready con Distroless

```dockerfile
# 🔒 NIVEL 3: Production con Distroless
# Dockerfile.production

# ═══════════════════════════════════════════════════════════
# STAGE 1: Builder
# ═══════════════════════════════════════════════════════════
FROM python:3.11-slim AS builder

WORKDIR /app

# Dependencias de sistema para compilación
RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential \
    curl \
    && rm -rf /var/lib/apt/lists/*

# Instalar pip-tools
RUN pip install pip-tools

# Copiar requirements y generar lockfile
COPY requirements.in .
RUN pip-compile requirements.in -o requirements.txt

# Crear venv e instalar
RUN python -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"
RUN pip install --no-cache-dir -r requirements.txt

# Copiar código
COPY src/ ./src/
COPY pyproject.toml .

# Instalar paquete
RUN pip install --no-cache-dir .

# ═══════════════════════════════════════════════════════════
# STAGE 2: Runtime con Distroless
# ═══════════════════════════════════════════════════════════
FROM gcr.io/distroless/python3-debian12 AS runtime

WORKDIR /app

# Copiar venv y código desde builder
COPY --from=builder /opt/venv /opt/venv
COPY --from=builder /app/src ./src
COPY --from=builder /app/app ./app
COPY --from=builder /app/configs ./configs
COPY --from=builder /app/models ./models

# Distroless no tiene shell, así que PATH se configura diferente
ENV PATH="/opt/venv/bin:$PATH" \
    PYTHONUNBUFFERED=1

EXPOSE 8000

# Distroless no soporta shell form, usar exec form
ENTRYPOINT ["/opt/venv/bin/python", "-m", "uvicorn"]
CMD ["app.fastapi_app:app", "--host", "0.0.0.0", "--port", "8000"]
```

10.4 .dockerignore Profesional

```
# .dockerignore

# Git
.git
.gitignore
.gitattributes

# Python
__pycache__
*.py[cod]
*$py.class
*.so
.Python
.venv
venv
ENV
.eggs
*.egg-info
*.egg

# Testing
.pytest_cache
.coverage
htmlcov
.tox
.nox

# IDE
.idea
.vscode
*.swp
*.swo

# Notebooks
*.ipynb
.ipynb_checkpoints

# Docs
docs/
*.md
!README.md

# CI/CD
.github
.gitlab-ci.yml
Jenkinsfile

# DVC (datos no deben ir en imagen)
*.dvc
.dvc
data/

# MLflow (runs locales)
mlruns/
mlartifacts/

# Otros
*.log
*.tmp
.env*
!.env.example
Makefile
docker-compose*.yml
```

10.5 Seguridad: Escaneo de Vulnerabilidades

Trivy (Recomendado)

```
        # Instalar Trivy
        brew install trivy  # macOS
        # o
        sudo apt-get install trivy  # Ubuntu

        # Escanear imagen
        trivy image bankchurn:latest

        # Escanear con severidad mínima
        trivy image --severity HIGH,CRITICAL bankchurn:latest

        # Escanear Dockerfile
        trivy config Dockerfile

        # Output en JSON para CI
        trivy image -f json -o trivy-results.json bankchurn:latest
```

Integrar en CI

```yaml
# .github/workflows/docker-security.yml
name: Docker Security Scan

on:
  push:
    paths:
      - 'Dockerfile'
      - 'requirements*.txt'

jobs:
  scan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Build image
        run: docker build -t bankchurn:scan .

      - name: Run Trivy vulnerability scanner
        uses: aquasecurity/trivy-action@master
        with:
          image-ref: 'bankchurn:scan'
          format: 'sarif'
          output: 'trivy-results.sarif'
          severity: 'CRITICAL,HIGH'

      - name: Upload Trivy scan results
        uses: github/codeql-action/upload-sarif@v2
        with:
          sarif_file: 'trivy-results.sarif'
```

10.6 Docker Compose para Desarrollo

```yaml
# docker-compose.yml
version: '3.8'

services:
  # ═══════════════════════════════════════════════════════════
  # API Principal
  # ═══════════════════════════════════════════════════════════
  api:
    build:
      context: .
      dockerfile: Dockerfile
      target: runtime  # Multi-stage: usar stage específico
    ports:
      - "8000:8000"
    environment:
      - MLFLOW_TRACKING_URI=http://mlflow:5000
      - LOG_LEVEL=INFO
    volumes:
      - ./models:/app/models:ro  # Read-only para producción
    depends_on:
      mlflow:
        condition: service_healthy
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8000/health"]
      interval: 30s
      timeout: 10s
      retries: 3
    restart: unless-stopped

  # ═══════════════════════════════════════════════════════════
  # MLflow Server
  # ═══════════════════════════════════════════════════════════
  mlflow:
    image: ghcr.io/mlflow/mlflow:v2.8.0
    ports:
      - "5000:5000"
    environment:
      - MLFLOW_TRACKING_URI=sqlite:///mlflow/mlflow.db
    volumes:
      - mlflow-data:/mlflow
    command: >
      mlflow server
      --host 0.0.0.0
      --port 5000
      --backend-store-uri sqlite:///mlflow/mlflow.db
      --default-artifact-root /mlflow/artifacts
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5000/health"]
      interval: 30s
      timeout: 10s
      retries: 3

  # ═══════════════════════════════════════════════════════════
  # Prometheus (Métricas)
  # ═══════════════════════════════════════════════════════════
  prometheus:
    image: prom/prometheus:v2.47.0
    ports:
      - "9090:9090"
    volumes:
      - ./infra/prometheus-config.yaml:/etc/prometheus/prometheus.yml:ro
      - prometheus-data:/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
    restart: unless-stopped

  # ═══════════════════════════════════════════════════════════
  # Grafana (Dashboards)
  # ═══════════════════════════════════════════════════════════
  grafana:
    image: grafana/grafana:10.2.0
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
      - GF_USERS_ALLOW_SIGN_UP=false
    volumes:
      - grafana-data:/var/lib/grafana
    depends_on:
      - prometheus
    restart: unless-stopped

volumes:
  mlflow-data:
  prometheus-data:
  grafana-data:
```

10.7 Comparativa de Tamaños

```
╔═══════════════════════════════════════════════════════════════════════╗
║                  COMPARATIVA DE TAMAÑOS DE IMAGEN                     ║
╠═══════════════════════════════════════════════════════════════════════╣
║                                                                       ║
║   Base Image                    | Tamaño Aprox | Seguridad | Uso      ║
║   ──────────────────────────────┼──────────────┼───────────┼──────────  ║
║   python:3.11                   | ~1.0 GB      | ⚠         | Desarrollo ║
║   python:3.11-slim              | ~150 MB      | ✓         | Producción ║
║   python:3.11-alpine            | ~50 MB       | ✓         | Minimalista* ║
║   gcr.io/distroless/python3     | ~50 MB       | ✓         | Producción ║
║                                                                       ║
║   * Alpine puede tener problemas con algunas librerías de ML (musl vs glibc) ║
║                                                                       ║
║   IMAGEN BANKCHURN:                                                   ║
║   ├── Nivel 1 (básico):        ~1.2 GB                                ║
║   ├── Nivel 2 (multi-stage):   ~400 MB                                ║
║   └── Nivel 3 (distroless):    ~150 MB                                ║
║                                                                       ║
╚═══════════════════════════════════════════════════════════════════════╝
```

10.8 Ejercicio Integrador

Crear Dockerfile Production-Ready

1. **Multi-stage build** con builder y runtime
2. **Usuario non-root**
3. **Health check**
4. **Escaneo de vulnerabilidades**

Checklist

```
OPTIMIZACIÓN:
[ ] Multi-stage build implementado
[ ] .dockerignore completo
[ ] Layer caching optimizado
[ ] Imagen < 500MB

SEGURIDAD:
[ ] Non-root user
[ ] No secrets hardcodeados
[ ] Base image slim/distroless
[ ] Trivy scan sin CRITICAL

OPERACIONES:
[ ] Health check definido
[ ] docker-compose funcional
[ ] Logs configurados
```

Siguiente Paso

Con Docker dominado, es hora de crear **APIs profesionales con FastAPI**.

Ir a Módulo 11: FastAPI Profesional →

Módulo 10 completado. Tus imágenes Docker ahora son production-ready.