10 - Producción y Mejores Prácticas

Ejecutar Docker en producción de forma segura

Objetivo

Aprender las mejores prácticas para ejecutar Docker en producción de forma segura y eficiente.

Diferencia clave:

- **Desarrollo**: Funciona = suficiente
- **Producción**: Seguro, estable, monitoreado, escalable

Principios de Producción

🔒 **Seguridad**: Usuario no-root, imágenes escaneadas
📊 **Monitoreo**: Healthchecks, logs estructurados
🔄 **Confiabilidad**: Restart policies, límites de recursos
📦 **Versionado**: Tags específicos, no "latest"
💾 **Backup**: Estrategia de respaldo definida

¿Qué aprenderás?

- Seguridad en contenedores
- Gestión de secrets
- Estrategias de despliegue
- Monitoreo y observabilidad
- Backup y recuperación

Seguridad: Usuario No-Root

Dockerfile seguro

```dockerfile
FROM python:3.9-slim

# Crear usuario no-root
RUN groupadd -r appuser && useradd -r -g appuser appuser

WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py .

# Cambiar ownership
RUN chown -R appuser:appuser /app

# Cambiar a usuario no-root
USER appuser

EXPOSE 5000
CMD ["python", "app.py"]
```

Escanear Imágenes

```
# Docker Scout (nativo)
docker scout quickview mi-imagen

# Trivy
docker run --rm \
  -v /var/run/docker.sock:/var/run/docker.sock \
  aquasec/trivy image mi-imagen
```

Secrets en Producción

❌ Nunca hacer

```yaml
services:
  app:
    environment:
      - DATABASE_PASSWORD=password123  # ❌
```

✅ Usar Docker Secrets

```yaml
services:
  app:
    environment:
      - DATABASE_PASSWORD_FILE=/run/secrets/db_password
    secrets:
      - db_password

secrets:
  db_password:
    external: true
```

Variables de Entorno

docker compose.prod.yml

```
services:
  app:
    image: mi-app:${VERSION:-latest}
    environment:
      - NODE_ENV=production
      - DATABASE_URL=${DATABASE_URL}
    env_file:
      - .env.production
    restart: always
```

Resource Limits

```yaml
services:
  app:
    deploy:
      resources:
        limits:
          cpus: '2.0'
          memory: 1G
        reservations:
          cpus: '1.0'
          memory: 512M
    restart: always
    restart_policy:
      condition: on-failure
      max_attempts: 3
```

Healthchecks en Producción

```yaml
services:
  app:
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5000/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 60s
    restart: always
```

Logging en Producción

```yaml
services:
  app:
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
        labels: "production"
```

Backup de Volúmenes

```
#!/bin/bash
# backup.sh

DATE=$(date +%Y%m%d_%H%M%S)

# Backup de volumen de PostgreSQL
docker run --rm \
  -v postgres-data:/data \
  -v $BACKUP_DIR:/backup \
  alpine tar czf /backup/postgres-$DATE.tar.gz -C /data .
```

Checklist de Producción

Seguridad

- [ ] Usar usuario no-root
- [ ] Escanear imágenes por vulnerabilidades
- [ ] No hardcodear secrets
- [ ] Usar imágenes oficiales y actualizadas
- [ ] Limitar recursos (CPU, memoria)

Configuración

- [ ] Variables de entorno para configuración
- [ ] Healthchecks implementados
- [ ] Logging configurado y rotado
- [ ] Restart policies configuradas
- [ ] Tags de versión específicos (no latest)

Checklist de Producción

Operaciones

- [ ] Estrategia de backup definida
- [ ] Plan de recuperación documentado
- [ ] Monitoreo y alertas configurados
- [ ] Documentación actualizada
- [ ] Proceso de despliegue automatizado

Dockerfile de Producción

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

# Production stage
FROM python:3.9-slim
RUN groupadd -r appuser && useradd -r -g appuser appuser
WORKDIR /app
COPY --from=builder /root/.local /root/.local
COPY --chown=appuser:appuser app.py .
USER appuser
ENV PATH=/root/.local/bin:$PATH
CMD ["python", "app.py"]
```

Comandos de Producción

```
# Verificar configuración
docker compose -f docker compose.prod.yml config

# Iniciar en producción
docker compose -f docker compose.prod.yml up -d

# Ver logs
docker compose -f docker compose.prod.yml logs -f

# Actualizar servicio
docker compose -f docker compose.prod.yml pull
docker compose -f docker compose.prod.yml up -d
```

Práctica

1. Crea un Dockerfile seguro
2. Configura healthchecks
3. Establece límites de recursos
4. Configura logging
5. Implementa backup

¡Felicidades!

Has completado el curso de Docker. Ahora tienes las habilidades para:

- ✅ Crear y gestionar contenedores
- ✅ Construir imágenes optimizadas
- ✅ Orquestar aplicaciones con Docker Compose
- ✅ Desplegar aplicaciones en producción
- ✅ Implementar mejores prácticas de seguridad

Recursos Adicionales

- Docker Security Best Practices
- OWASP Docker Security
- Docker Production Guide

¡Sigue Practicando!

¡Gracias por completar el curso! 🎓