07 - Docker Compose Avanzado

Configuraciones avanzadas para aplicaciones complejas

Objetivo

Aprender configuraciones avanzadas de Docker Compose para aplicaciones complejas.

Cuándo usar:

- Múltiples entornos (dev/staging/prod)
- Servicios con dependencias complejas
- Necesitas escalar servicios
- Gestión de secrets y configs

Concepto: Override Files

Idea: Un archivo base + archivos de override

Ejemplo:

- `docker-compose.yml` → Configuración base
- `docker-compose.dev.yml` → Cambios para desarrollo
- `docker-compose.prod.yml` → Cambios para producción

Ventaja: Reutilizas la base, solo cambias lo necesario

¿Qué aprenderás?

- Override files y entornos múltiples
- Healthchecks y dependencias
- Escalado de servicios
- Secrets y configs
- Profiles

Override Files

docker-compose.yml (base)

```yaml
services:
  web:
    image: nginx:alpine
    ports:
      - "80:80"
```

docker-compose.override.yml (desarrollo)

```yaml
services:
  web:
    volumes:
      - ./html:/usr/share/nginx/html
      - ./nginx-dev.conf:/etc/nginx/nginx.conf
    environment:
      - DEBUG=true
```

Uso: Docker Compose automáticamente usa override.yml

Healthchecks y Dependencias

```yaml
services:
  db:
    image: postgres:15
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 30s

  web:
    build: ./app
    depends_on:
      db:
        condition: service_healthy  # Espera a que db esté healthy
```

Escalado de Servicios

docker-compose.yml escalable

```yaml
services:
  web:
    build: ./app
    ports:
      - "5000-5002:5000"  # Múltiples puertos
```

Escalar servicios

```bash
# Escalar web a 3 instancias
docker compose up -d --scale web=3

# Ver servicios escalados
docker compose ps
```

Secrets (Docker Swarm)

Crear secrets

```
echo "mi_password_secreto" | docker secret create db_password -
```

Usar en docker compose.yml

```
services:
  db:
    secrets:
      - db_password
    environment:
      POSTGRES_PASSWORD_FILE: /run/secrets/db_password

secrets:
  db_password:
    external: true
```

Nota: Secrets requieren Docker Swarm mode

Profiles

docker-compose.yml con profiles

```yaml
services:
  web:
    build: ./app

  redis:
    image: redis:7-alpine
    profiles:
      - cache  # Solo se inicia con --profile cache

  monitoring:
    image: prom/prometheus
    profiles:
      - monitoring
```

Usar profiles

```bash
# Iniciar solo servicios base
docker compose up -d

# Iniciar con redis
docker compose --profile cache up -d

# Iniciar con múltiples profiles
docker compose --profile cache --profile monitoring up -d
```

Entornos Múltiples

docker-compose.prod.yml

```yaml
services:
  web:
    image: mi-app:latest
    restart: always
    environment:
      - FLASK_ENV=production
    deploy:
      replicas: 3
      resources:
        limits:
          cpus: '0.5'
          memory: 512M
```

Usar archivos específicos

```
# Producción
docker compose -f docker-compose.yml -f docker-compose.prod.yml up
```

Resource Limits

```yaml
services:
  web:
    deploy:
      resources:
        limits:
          cpus: '1.0'
          memory: 512M
        reservations:
          cpus: '0.5'
          memory: 256M
    restart: unless-stopped
```

Comandos Avanzados

```
# Validar configuración
docker compose config

# Pausar servicios
docker compose pause

# Recrear servicios
docker compose up -d --force-recreate
```

Mejores Prácticas

✅ Usa override files para diferentes entornos
✅ Define healthchecks para servicios críticos
✅ Usa profiles para servicios opcionales
✅ Establece límites de recursos
✅ Separa configuraciones por entorno

Práctica

1. Crea override files para dev/prod
2. Implementa healthchecks
3. Configura profiles
4. Establece límites de recursos
5. Prueba escalado de servicios

Siguiente Paso

Aprende a optimizar imágenes Docker para producción.

Módulo 08: Optimización de Imágenes

Preguntas?

¡Tiempo para preguntas y práctica!