IACADEMY

DE FUNDAMENTOS A ARQUITECTURA DE IA

MÓDULO 17

DevOps con IA

Avanzado

iacedemy.com — 2026

MÓDULO 17

DevOps con IA

Nivel: Avanzado

Autor: Ricardo Gutierrez

Publicación: Mayo 2026

Plataforma: iacedemy.com

Docker fundamentals

Docker resuelve un problema que todo desarrollador conoce: "en mi maquina funciona". Empaqueta tu aplicacion con todas sus dependencias (sistema operativo, librerias, configuraciones) en un contenedor que funciona igual en cualquier sitio.

Tres conceptos que necesitas dominar antes de tocar nada:

1. **Dockerfile**: la receta para construir tu contenedor. Cada instruccion es un paso: instalar dependencias, copiar codigo, configurar el entorno.
2. **Imagen**: el resultado de ejecutar esa receta. Es inmutable. Una vez construida, no cambia.
3. **Contenedor**: una instancia en ejecucion de una imagen. Puedes tener 10 contenedores corriendo la misma imagen.

Tu primer Dockerfile

Vamos a crear un Dockerfile para una API FastAPI con multi-stage build. Esto significa que usamos una etapa para instalar dependencias y otra para la imagen final. El resultado es una imagen mas ligera y segura.

```
 # Etapa 1: Builder (instala dependencias)
FROM python:3.11-slim AS builder
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Etapa 2: Runtime (solo lo necesario)
FROM python:3.11-slim
WORKDIR /app
COPY --from=builder /usr/local/lib/python3.11/site-packages /usr/local/lib/
python3.11/site-packages
COPY --from=builder /usr/local/bin /usr/local/bin
COPY . .
EXPOSE 8000
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

La etapa builder instala las dependencias de Python. La etapa runtime copia solo los paquetes instalados y el codigo fuente. No incluye pip, compiladores ni archivos temporales. Resultado: imagen 50-70% mas pequeña.

Volumes: persistir datos

Cuando un contenedor se destruye, sus datos desaparecen. Los volumes solucionan esto: almacenan datos fuera del contenedor, en el sistema de archivos del host.

```
 # Crear un volume
docker volume create postgres_data

# Usar el volume al ejecutar el contenedor
docker run -v postgres_data:/var/lib/postgresql/data postgres:16
```

Para bases de datos, logs y archivos subidos por usuarios, los volumes son obligatorios. Sin ellos, pierdes todo al reiniciar el contenedor.

Networks: comunicacion entre contenedores

Por defecto, los contenedores estan aislados. No se ven entre si. Creas una network para que se comuniquen internamente sin exponer puertos al exterior.

```
 # Crear network
docker network create app_network

# Conectar contenedores a la red
docker run --network app_network --name fastapi mi-api
docker run --network app_network --name postgres postgres:16
```

Dentro de la network, los contenedores se encuentran por nombre. FastAPI conecta a PostgreSQL usando `postgres:5432` como host, no `localhost`.

Docker Compose para tu stack

Docker Compose te permite definir multiples contenedores en un solo archivo YAML. En lugar de ejecutar 5 comandos `docker run`, defines todo en `docker-compose.yml` y levantas con un comando.

Este es el stack real que vamos a desplegar:

```
 version: "3.9"
services:
  postgres:
    image: postgres:16-alpine
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      POSTGRES_DB: ${DB_NAME}
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${DB_USER}"]
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - internal
    restart: unless-stopped

  redis:
    image: redis:7-alpine
    volumes:
      - redis_data:/data
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - internal
    restart: unless-stopped

  fastapi:
    build:
      context: ./backend
      dockerfile: Dockerfile
    environment:
      DATABASE_URL: postgresql://${DB_USER}:${DB_PASSWORD}@postgres:5432/$
{DB_NAME}
      REDIS_URL: redis://redis:6379
    depends_on:
      postgres:
        condition: service_healthy
```

```
      redis:
        condition: service_healthy
    networks:
      - internal
    restart: unless-stopped

  n8n:
    image: n8nio/n8n:latest
    volumes:
      - n8n_data:/home/node/.n8n
    environment:
      N8N_BASIC_AUTH_ACTIVE: "true"
      N8N_BASIC_AUTH_USER: ${N8N_USER}
      N8N_BASIC_AUTH_PASSWORD: ${N8N_PASSWORD}
    networks:
      - internal
    restart: unless-stopped

  caddy:
    image: caddy:2-alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./Caddyfile:/etc/caddy/Caddyfile
      - caddy_data:/data
      - caddy_config:/config
    networks:
      - internal
    restart: unless-stopped

volumes:
  postgres_data:
  redis_data:
  n8n_data:
  caddy_data:
  caddy_config:

networks:
  internal:
    driver: bridge
```

Detalles criticos de este archivo:

- **Variables de entorno**: todas las credenciales vienen de un archivo `.env` que NO se commitea al repositorio. Creas un `.env.example` sin valores reales como referencia.
- **Health checks**: PostgreSQL y Redis tienen health checks. FastAPI no arranca hasta que ambos esten listos (`condition: service_healthy`).
- **Restart policy**: `unless-stopped` significa que si el servidor se reinicia, los contenedores vuelven automaticamente. Solo se detienen si tu los paras manualmente.
- **Network interna**: todos los servicios comparten una red privada. Solo Caddy expone puertos (80 y 443) al exterior.

> **Secretos en produccion**
>
> Nunca pongas contraseñas reales en el `docker-compose.yml` que commiteas. Usa archivos `.env` (excluidos del repo con `.gitignore`) o Docker secrets para produccion. En GitHub Actions, usa Secrets del repositorio.

Caddy como reverse proxy

Caddy es el reverse proxy mas simple que existe. Una configuracion de 4 lineas te da SSL automatico con Let's Encrypt, HTTP/2, compresion, y headers de seguridad. Todo lo que nginx necesita 50 lineas y plugins adicionales, Caddy lo hace por defecto.

```
 api.tudominio.com {
    reverse_proxy fastapi:8000
}

n8n.tudominio.com {
    reverse_proxy n8n:5678
}

tudominio.com {
    root * /srv/frontend
    file_server
```

```
    try_files {path} /index.html
}
```

Eso es todo el Caddyfile. Caddy automaticamente:

- Solicita certificados SSL de Let's Encrypt para cada dominio
- Los renueva cada 60 dias (antes de que expiren a los 90)
- Redirige HTTP a HTTPS
- Activa HSTS (HTTP Strict Transport Security)
- Configura headers de seguridad basicos

Comparado con la alternativa nginx, que requiere: instalar certbot como servicio separado, configurar cron para renovacion, escribir bloques `server` con `ssl_certificate` y `ssl_certificate_key`, configurar redirects manuales. Caddy elimina toda esa complejidad.

Zero-downtime deploys: cuando actualizas tu API, Caddy mantiene las conexiones existentes mientras el nuevo contenedor arranca. No hay interrupcion de servicio.

Hetzner: tu servidor en 10 minutos

Hetzner es hosting europeo (Alemania/Finlandia), con data centers en la UE. Cumplimiento RGPD nativo. Un servidor CPX31 (4 vCPU, 8 GB RAM, 160 GB SSD) cuesta unos 15 EUR/mes. Suficiente para desplegar varios proyectos simultaneamente.

Paso 1: Crear el servidor

1. Crea una cuenta en Hetzner Cloud
2. Nuevo proyecto, nuevo servidor
3. Ubicacion: Falkenstein (fsn1) o Helsinki (hel1)
4. Imagen: Ubuntu 22.04
5. Tipo: CPX31 (o CPX21 si tu presupuesto es ajustado)
6. SSH Key: sube tu clave publica (`~/.ssh/id_ed25519.pub`)

Paso 2: Seguridad basica

```
 # Conectar por SSH
ssh root@TU_IP_SERVIDOR

# Actualizar sistema
apt update && apt upgrade -y

# Crear usuario no-root
adduser deploy
usermod -aG sudo deploy

# Copiar SSH key al nuevo usuario
mkdir -p /home/deploy/.ssh
cp ~/.ssh/authorized_keys /home/deploy/.ssh/
chown -R deploy:deploy /home/deploy/.ssh

# Desactivar login por password (solo SSH key)
sed -i 's/PasswordAuthentication yes/PasswordAuthentication no/' /etc/ssh/
sshd_config
systemctl restart sshd

# Firewall basico
ufw default deny incoming
ufw default allow outgoing
ufw allow 22/tcp    # SSH
ufw allow 80/tcp    # HTTP
ufw allow 443/tcp   # HTTPS
ufw enable
```

Paso 3: Instalar Docker

```
 # Instalar Docker
curl -fsSL https://get.docker.com | sh
usermod -aG docker deploy

# Instalar Docker Compose plugin
apt install docker-compose-plugin -y

# Verificar
```

```
docker --version
docker compose version
```

Paso 4: Primer despliegue

```
 # Como usuario deploy
su - deploy

# Clonar tu repositorio
git clone https://github.com/tu-usuario/tu-proyecto.git
cd tu-proyecto

# Crear archivo .env con credenciales reales
cp .env.example .env
nano .env  # Editar con valores reales

# Levantar todo
docker compose up -d

# Verificar que todo esta corriendo
docker compose ps
docker compose logs --tail 50
```

En este punto tienes tu stack corriendo. Si configuraste DNS apuntando a la IP del servidor, Caddy ya esta generando certificados SSL. Tu API es accesible por HTTPS.

Cloudflare Pages y DNS

Para frontends estaticos (Next.js en modo export, landing pages, documentacion), Cloudflare Pages es gratuito y tiene CDN global. Deploy automatico desde GitHub, previews por branch, custom domains incluidos.

Deploy de un frontend

1. Ve a Cloudflare Pages
2. Conecta tu repositorio de GitHub
3. Configura el build:
   - Framework: Next.js (o "None" para HTML estatico)
   - Build command: `npm run build`

    - Output directory: `out/` (static export) o `.next/`
4. Deploy. Cada push a main redespliega automaticamente.

Configuracion DNS

Si tu dominio esta en Cloudflare (recomendado), configuras los registros DNS asi:

- `tudominio.com` CNAME a `tu-proyecto.pages.dev` (frontend en CF Pages)
- `api.tudominio.com` A record a la IP de Hetzner (proxy off, DNS only)
- `n8n.tudominio.com` A record a la IP de Hetzner (proxy off, DNS only)

> **Proxy on vs proxy off**
>
> Para subdominios que apuntan a Hetzner con Caddy, usa proxy off (DNS only, nube gris). Si usas proxy on (nube naranja), Cloudflare gestiona el SSL y puede interferir con los certificados de Caddy. Elige uno u otro, no ambos.

Workers y Functions

Cloudflare Workers te permite ejecutar logica en el edge (servidor mas cercano al usuario). Para funciones simples (redireccion, A/B testing, headers), es gratuito y muy rapido. Para logica de negocio compleja, mantente con FastAPI en Hetzner.

CI/CD con GitHub Actions

CI/CD (Continuous Integration / Continuous Deployment) significa que cuando haces push a main, el codigo se testea, se construye y se despliega automaticamente. Sin entrar al servidor manualmente.

El pipeline completo

```
 name: Deploy
on:
  push:
    branches: [main]
```

```
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: "3.11"
      - run: pip install -r requirements.txt
      - run: pip install ruff pytest
      - run: ruff check .            # Lint
      - run: pytest tests/ -v        # Tests

  build-and-push:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}
      - uses: docker/build-push-action@v5
        with:
          push: true
          tags: ghcr.io/${{ github.repository }}:latest

  deploy:
    needs: build-and-push
    runs-on: ubuntu-latest
    steps:
      - name: Deploy to Hetzner
        uses: appleboy/ssh-action@v1
        with:
          host: ${{ secrets.SERVER_HOST }}
          username: deploy
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          script: |
            cd /home/deploy/tu-proyecto
            docker compose pull
            docker compose up -d --remove-orphans
            docker image prune -f
```

El pipeline tiene tres fases:

1. **Test**: ejecuta linter (ruff) y tests (pytest). Si falla, no continua.
2. **Build and push**: construye la imagen Docker y la sube a GitHub Container Registry.
3. **Deploy**: conecta por SSH al servidor Hetzner, descarga la nueva imagen y reinicia los contenedores.

Secrets: las credenciales (`SERVER_HOST` , `SSH_PRIVATE_KEY`) se guardan en Settings > Secrets > Actions del repositorio de GitHub. Nunca en el codigo.

Monitoring, backups y secretos

UptimeRobot (gratis)

Monitoriza tus endpoints cada 5 minutos. Si algo cae, te avisa por email o Telegram. Plan gratuito: 50 monitores.

1. Crea cuenta en UptimeRobot
2. Añade monitor HTTP para cada endpoint critico: API health, frontend, n8n
3. Configura alertas por Telegram (Bot API + chat ID)

Backups de PostgreSQL

```
 # Script de backup: /home/deploy/scripts/backup-db.sh
#!/bin/bash
BACKUP_DIR=/home/deploy/backups
DATE=$(date +%Y%m%d_%H%M%S)
FILENAME="${BACKUP_DIR}/db-${DATE}.sql.gz"

# Crear backup comprimido
docker exec postgres pg_dump -U $DB_USER $DB_NAME | gzip > $FILENAME

# Eliminar backups de mas de 30 dias
find $BACKUP_DIR -name "*.sql.gz" -mtime +30 -delete

echo "Backup completado: $FILENAME"
```

```
 # Cron: ejecutar cada dia a las 3AM
crontab -e
```

```
0 3 * * * /home/deploy/scripts/backup-db.sh >> /home/deploy/logs/backup.log
2>&1
```

Restore: para restaurar un backup:

```
 gunzip -c /home/deploy/backups/db-20260512.sql.gz | docker exec -i postgres
psql -U $DB_USER $DB_NAME
```

Gestion de logs

```
 # Ver logs de un servicio especifico
docker compose logs fastapi --tail 100 -f

# Ver logs de todos los servicios
docker compose logs --tail 50

# Rotacion de logs (evitar que llenen el disco)
# Docker ya rota logs por defecto, pero puedes configurar limites:
# En /etc/docker/daemon.json:
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  }
}
```

SSL/TLS con Caddy

Caddy gestiona todo el ciclo de vida de los certificados SSL:

- Solicita certificados de Let's Encrypt automaticamente al detectar un dominio en el Caddyfile
- Los renueva 30 dias antes de la expiracion
- Activa HSTS por defecto (fuerza HTTPS)
- Soporta HTTP/2 y HTTP/3 (QUIC) sin configuracion adicional

Si necesitas un certificado wildcard (`*.tudominio.com`), Caddy lo soporta via DNS challenge con plugins para Cloudflare, Route53, etc.

IA en DevOps

Claude Code es especialmente bueno generando archivos de configuracion DevOps. En lugar de buscar en documentacion durante horas, le describes tu stack y te genera la configuracion.

Generar Dockerfile con IA

```
Prompt: "Genera un Dockerfile multi-stage para una API FastAPI
con Python 3.11. Usa slim como imagen base. Instala dependencias
desde requirements.txt. El entrypoint es uvicorn main:app en el
puerto 8000. Incluye un usuario no-root para seguridad."
```

Generar docker-compose con IA

```
Prompt: "Genera un docker-compose.yml para: PostgreSQL 16,
Redis 7, mi API FastAPI (build desde ./backend), y Caddy como
reverse proxy. Incluye health checks, volumes persistentes,
network interna, y restart unless-stopped. Variables de entorno
desde archivo .env."
```

Generar GitHub Actions con IA

```
Prompt: "Genera un workflow de GitHub Actions que: 1) ejecute
ruff y pytest, 2) construya imagen Docker y la suba a GHCR,
3) despliegue via SSH a un servidor. Solo en push a main.
Usa secrets para credenciales."
```

La IA te genera el 80% de la configuracion. Tu ajustas el 20% restante: nombres de servicios, puertos especificos, variables de tu proyecto. Esto transforma horas de configuracion en minutos.

> **Siempre revisa la configuracion generada**
>
> La IA puede generar configuraciones con versiones desactualizadas, puertos incorrectos, o practicas no recomendadas. Siempre verifica que las imagenes

Docker son las versiones que quieres, que los puertos no colisionan, y que los secretos no estan hardcodeados.

Ejercicio practico

Ejercicio M17: Despliega un stack completo desde cero

1. **Dockeriza tu API**: crea un Dockerfile multi-stage para una API FastAPI basica (o usa la del ejemplo). Construye la imagen y verifica que funciona localmente con `docker run`.
2. **Docker Compose**: crea un `docker-compose.yml` con PostgreSQL + Redis + tu API + Caddy. Verifica que todo levanta con `docker compose up`.
3. **Servidor Hetzner**: crea un CPX21 (el mas barato), configura SSH, firewall y Docker. Sube tu stack y despliega.
4. **DNS y SSL**: apunta un subdominio a tu servidor. Verifica que Caddy genera el certificado SSL.
5. **CI/CD**: crea un workflow de GitHub Actions que ejecute tests y despliegue automaticamente al hacer push a main.
6. **Monitoring**: configura UptimeRobot para monitorizar tu API.
7. **Backup**: crea el script de backup de PostgreSQL y configuralo con cron.

Bonus: pide a Claude Code que te genere toda la configuracion (Dockerfile, docker-compose.yml, Caddyfile, workflow). Compara con lo que escribiste manualmente.

Conclusiones clave

Key takeaways del M17

1. Docker empaqueta tu aplicacion con sus dependencias. Multi-stage builds producen imagenes ligeras y seguras.

2. Docker Compose define tu stack completo en un archivo. Health checks, volumes persistentes y restart policies son obligatorios en produccion.
3. Caddy simplifica SSL/TLS a una linea de configuracion. Sin certbot, sin cron, sin renovaciones manuales.
4. Hetzner ofrece servidores EU a 15 EUR/mes. Configuracion de seguridad basica: usuario no-root, firewall UFW, solo SSH key.
5. Cloudflare Pages despliega frontends gratis con CDN global. DNS en Cloudflare simplifica la gestion.
6. GitHub Actions automatiza test, build y deploy. Secretos en el repositorio, nunca en el codigo.
7. Monitoring con UptimeRobot (gratis) + backups diarios con cron + rotacion de logs: el minimo viable de operaciones.
8. La IA genera configuraciones DevOps en minutos. Siempre revisa antes de desplegar.

Siguiente: M18 - SEO y GEO con IA

Tu stack esta desplegado. Ahora necesitas que la gente te encuentre. SEO clasico, GEO para motores generativos, y como la IA multiplica tu produccion de contenido.

Ir al Modulo 18

IACADEMY

iacedemy.com

De fundamentos a arquitectura de IA.
12 módulos prácticos. 24 recursos descargables.
Quizzes con certificado. Vídeos profesionales.

Empieza gratis en iacedemy.com/free