Docker

Jonathan Hechenleitner

22 de septiembre– 27 de septiembre

Introducción a Docker

- Docker es una **plataforma de contenedores** que permite empaquetar aplicaciones y sus dependencias en un solo artefacto: el contenedor.
- La ventaja principal es que la aplicación se ejecuta igual en cualquier lugar: tu laptop, un servidor en la nube, o un clúster Kubernetes.
- **Problema que resuelve**: “En mi máquina funciona, en producción no” → con Docker, las dependencias se definen y ejecutan siempre de la misma forma.
- **Comparación con máquinas virtuales**
- **Máquina Virtual (VM)**:
  - Requiere un hipervisor.
  - Emula hardware completo.
  - Cada VM necesita su propio sistema operativo.
  - Más consumo de recursos.
- **Docker**:
  - Comparte el kernel del host.
  - Ejecuta contenedores ligeros.
  - Arranca en segundos.
  - Ideal para microservicios.

```
1  docker run hello-world
```

Instalación y configuración

- Instalación:
  - Seguir guía de la documentación oficial: https://docs.docker.com/engine/install/
- Configuración inicial
  - Verificar versión:

```
1  docker --version
```

- Probar información del daemon:

```
docker info
```

- Evitar sudo en Linux:

```
1  sudo usermod -aG docker $USER
```

Conceptos básicos de Docker

- **Elementos principales**
  - **Imagen**: plantilla inmutable que contiene el sistema base y dependencias.
  - **Contenedor**: instancia en ejecución de una imagen.
  - **Registro**: repositorio de imágenes (Docker Hub, GitHub Container Registry, AWS ECR, etc.).
- **Ejemplo práctico**
  - Ejecutar un servidor Nginx:

```
1  docker run -d -p 8080:80 nginx
```

  - -d: ejecuta en segundo plano.
  - -p: mapea el puerto 8080 del host al 80 del contenedor.
  - Luego puedes abrir http://localhost:8080 y ver Nginx funcionando.

Manejo de imágenes

- **Comandos clave**
  - Descargar imagen:
    - docker pull alpine
  - Listar imágenes locales:
    - docker images
  - Eliminar imagen:
  - docker rmi alpine
- **Crear imagen propia desde Dockerfile**
  - Ejemplo:

```
FROM alpine:3.18
CMD ["echo", "Hola desde mi imagen"]
```

```
docker build -t mi-imagen:1.0 .
docker run mi-imagen:1.0
```

Manejo de contenedores

- **Crear y ejecutar contenedor interactivo**
  - docker run -it ubuntu bash
- **Listado y gestión**
  - Contenedores activos:
    - docker ps
  - Todos los contenedores (incluye detenidos):
    - docker ps -a
  - Detener, iniciar y eliminar:

```
docker stop <ID>
docker start <ID>
docker rm <ID>
```

Dockerfile y construcción avanzada

- **Explicación**
  - Un **Dockerfile** describe cómo construir una imagen.
  - Incluye base (FROM), dependencias (RUN), archivos (COPY), variables (ENV), comandos (CMD o ENTRYPOINT).
- **Ejemplo Node.js (multi-stage build para optimizar)**

```
# Etapa 1: build
FROM node:18 as build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# Etapa 2: runtime
FROM node:18-slim
WORKDIR /app
COPY --from=build /app/dist ./dist
CMD ["node", "dist/index.js"]
```

- Multi-stage build reduce tamaño de la imagen final.

Dockerfile y construcción avanzada

- **Explicación**
  - Un **Dockerfile** describe cómo construir una imagen.
  - Incluye base (FROM), dependencias (RUN), archivos (COPY), variables (ENV), comandos (CMD o ENTRYPOINT).
- **Ejemplo Node.js (multi-stage build para optimizar)**

```
# Etapa 1: build
FROM node:18 as build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# Etapa 2: runtime
FROM node:18-slim
WORKDIR /app
COPY --from=build /app/dist ./dist
CMD ["node", "dist/index.js"]
```

- Multi-stage build reduce tamaño de la imagen final.

Persistencia con volúmenes

- Por defecto, los datos en contenedores desaparecen al borrarlos.
- Solución: usar volúmenes (-v).

```
docker run -d \
  --name mysql \
  -e MYSQL_ROOT_PASSWORD=secret \
  -v /mi_disco/mysql:/var/lib/mysql \
  mysql:8
```

- Ahora los datos de MySQL persisten en /mi_disco/mysql.

Redes en Docker

- **Redes disponibles**
  - **bridge** (por defecto).
  - **host** (usa red del host).
  - **none** (aislamiento total).
  - **user-defined networks**: permite DNS interno entre contenedores.
- **Ejemplo**

```
1  docker network create mi_red
2  docker run -d --name db --network mi_red mysql:8
3  docker run -d --name app --network mi_red nginx
```

- El contenedor app puede conectarse a db por nombre.

Docker Compose

- **Explicación**
  - Herramienta para **definir y orquestar múltiples contenedores** en un solo archivo YAML.
  - Facilita levantar aplicaciones completas con un comando.
- **Ejemplo**

```
docker-compose.yml
version: "3.8"
services:
  app:
    build: .
    ports:
      - "3000:3000"
    depends_on:
      - db
  db:
    image: mysql:8
    environment:
      MYSQL_ROOT_PASSWORD: secret
```

- Levantar servicios:
  - docker compose up -d

Buenas prácticas con Docker

- Usar imágenes oficiales o verificadas.
- Evitar imágenes demasiado grandes (usar alpine).
- Usar .dockerignore para no copiar archivos innecesarios.
- Mantener imágenes actualizadas.
- Nunca guardar credenciales en el Dockerfile → usar variables de entorno o secretos.

Buenas prácticas con Docker

- Usar imágenes oficiales o verificadas.
- Evitar imágenes demasiado grandes (usar alpine).
- Usar .dockerignore para no copiar archivos innecesarios.
- Mantener imágenes actualizadas.
- Nunca guardar credenciales en el Dockerfile → usar variables de entorno o secretos.

Bonus: Kubernetes

- **¿Qué es Kubernetes?**
  - Kubernetes (K8s) es una **plataforma de orquestación de contenedores**.
  - Permite **automatizar** el despliegue, la gestión, el escalado y la recuperación de aplicaciones que se ejecutan en contenedores (Docker u otros runtimes).
  - Fue creado por Google y ahora es mantenido por la **Cloud Native Computing Foundation (CNCF)**.

- **¿Por qué usar Kubernetes?**
- Cuando tienes pocos contenedores, con Docker basta. Pero en proyectos grandes aparecen problemas:
  - ¿Cómo reiniciar un contenedor caído automáticamente?
  - ¿Cómo escalar de 2 a 50 contenedores según la carga?
  - ¿Cómo distribuir tráfico entre réplicas?
  - ¿Cómo actualizar una aplicación sin downtime?
  - Kubernetes resuelve estos desafíos.
-

- **Conceptos clave**
  - **Cluster**: conjunto de nodos (máquinas físicas o virtuales).
  - **Nodo**: cada servidor dentro del cluster (puede ser master o worker).
  - **Pod**: la unidad más pequeña; es un contenedor (o grupo pequeño de contenedores).
  - **Deployment**: define cuántos pods debe haber y cómo actualizarlos.
  - **Service**: expone pods al exterior o a otros servicios dentro del cluster.
  - **Ingress**: administra el acceso externo (URLs, dominios, HTTPS).
  - **ConfigMap y Secret**: para configurar aplicaciones sin hardcodear datos.

JONATHAN HECHENLEITNER