Taller de Docker!

DC - FCEyN - UBA

October 24, 2025

Motivación

¿Por qué aprender Docker?

- Herramienta clave para la vida profesional: reproducir entornos de desarrollo y producción sin dolores de cabeza.

Motivación

¿Por qué aprender Docker?

- Herramienta clave para la vida profesional: reproducir entornos de desarrollo y producción sin dolores de cabeza.
- No siempre es obligatoria en la currícula (aunque se esta empezando a incluir), pero es altamente esperada en la industria y un poco en investigación.

Motivación

¿Por qué aprender Docker?

- Herramienta clave para la vida profesional: reproducir entornos de desarrollo y producción sin dolores de cabeza.
- No siempre es obligatoria en la currícula (aunque se esta empezando a incluir), pero es altamente esperada en la industria y un poco en investigación.
- Nos ahorra tiempo: tareas repetitivas como instalar dependencias o configurar entornos pueden automatizarse fácilmente.

Ejemplo de Aplicación: Análisis de Imágenes Médicas

Objetivo

Sistema web que permite a profesionales médicos subir imágenes radiográficas y recibir un diagnóstico automático basado en un modelo de Deep Learning:

Backend

- Python (Flask, PyTorch, OpenCV, psycopg2)
- API para subir imágenes
- Procesamiento con modelo preentrenado
- Almacenamiento de metadatos en PostgreSQL

Otros Componentes

- **Base de datos (PostgreSQL)**: usuarios y resultados
- **Frontend (React + Axios)**: interfaz web
- **Redis**: cache de predicciones

Ejemplo de aplicacion: Requisitos

Python y librerías

- Python 3.10+ (con entorno virtual configurado)
- Flask (versiones compatibles con Python 3.10)
- PyTorch (versión que soporte la GPU/CPU del sistema)
- OpenCV (compilación con soporte de imágenes y video)
- psycopg2 (requiere librerías de PostgreSQL instaladas en el sistema)

Dependencias del sistema

- `gcc`, `g++`, `make`
- `libpq-dev`, `libjpeg-dev`, `zlib1g-dev`
- Configuración de drivers CUDA/cuDNN si se usa GPU

Ejemplo de aplicacion: Requisitos

Base de Datos

- PostgreSQL 14+ (instalación, usuarios, roles, configuración de puertos)
- Configuración de backups y permisos

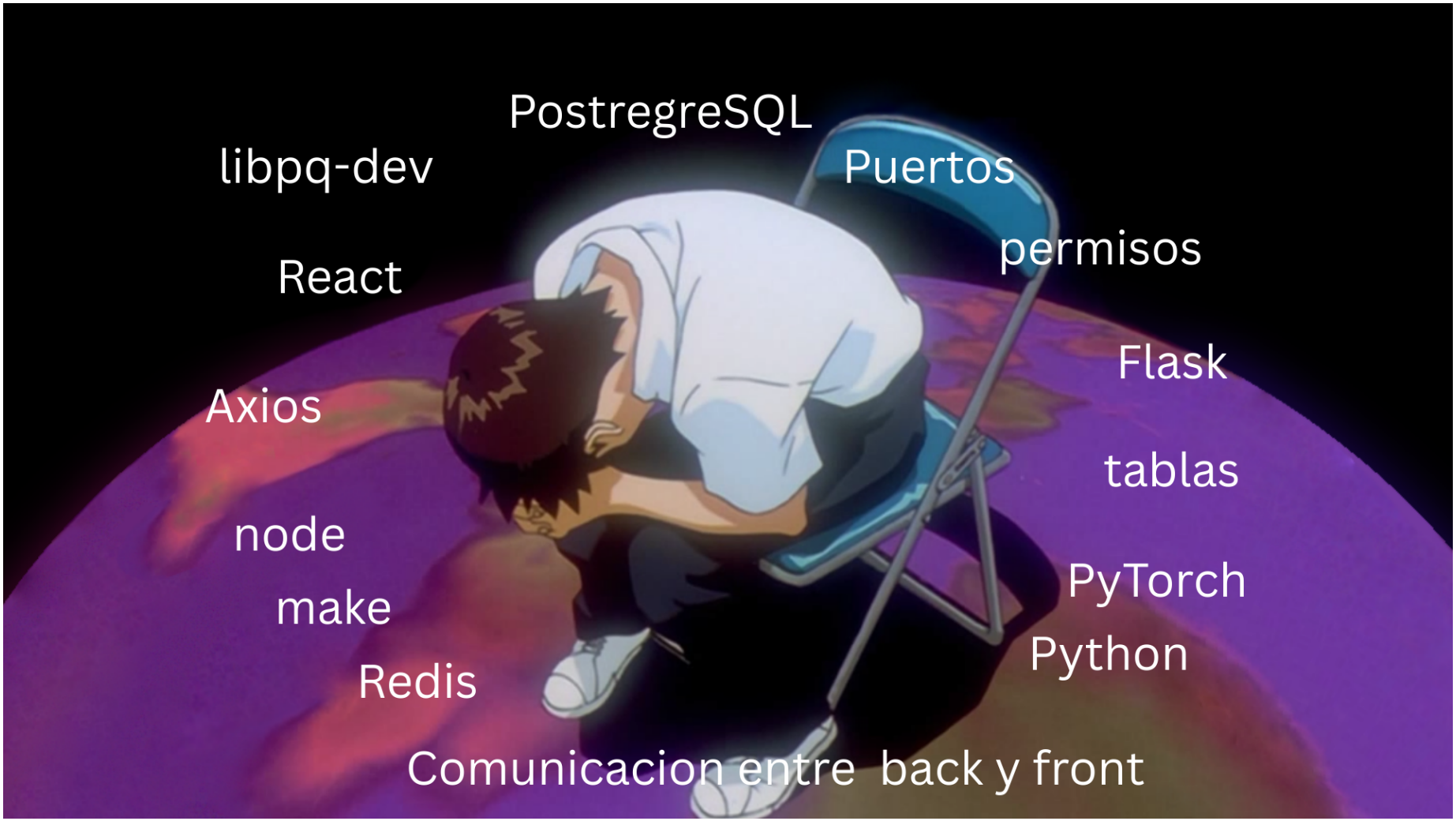
Frontend

- Node.js 18+ y npm
- Librerías de React y Axios (con dependencias propias)
- Posible conflicto de versiones entre proyectos

Cache

- Redis (instalación, configuración de persistencia y memoria)
- Gestión de puertos y compatibilidad con el backend

Ejemplo de aplicacion: Requisitos



Beneficios de Docker

Solución

- En lugar de instalar manualmente todas las dependencias en nuestra máquina, Docker permite levantar un entorno aislado con todo preconfigurado y listo para usar.
- Cada componente se ejecuta en su propio contenedor, aislado, y se comunica de manera controlada con los demás servicios.

Beneficios de Docker

Aislamiento

Cada componente corre en su propio contenedor, evitando conflictos de librerías y versiones.

Facilidad

Todo se inicia con un solo comando: sin instalaciones complejas.

Reproducibilidad

El entorno completo puede levantarse igual en cualquier máquina, garantizando que funcione siempre.

Escalabilidad

Contenedores separados permiten escalar y migrar servicios fácilmente.

Contenedores

Definición

- ¿Es un tipo de virtualización?

Contenedores

Definición

- ¿Es un tipo de virtualización? No, no son una virtualización completa.

Contenedores

Definición

- ¿Es un tipo de virtualización? No, no son una virtualización completa.
- ¿Cómo funciona?

Contenedores

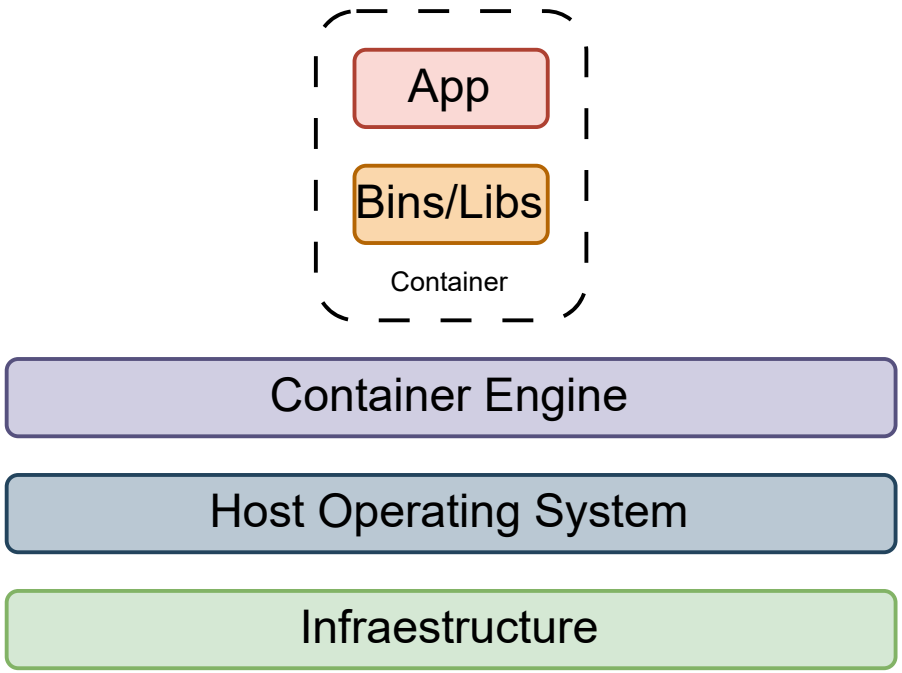
Definición

- ¿Es un tipo de virtualización? No, no son una virtualización completa.
- ¿Cómo funciona? Se apoya en funciones del kernel de linux (como Namespaces, Cgroups y OverlayFS).

Contenedores

Definición

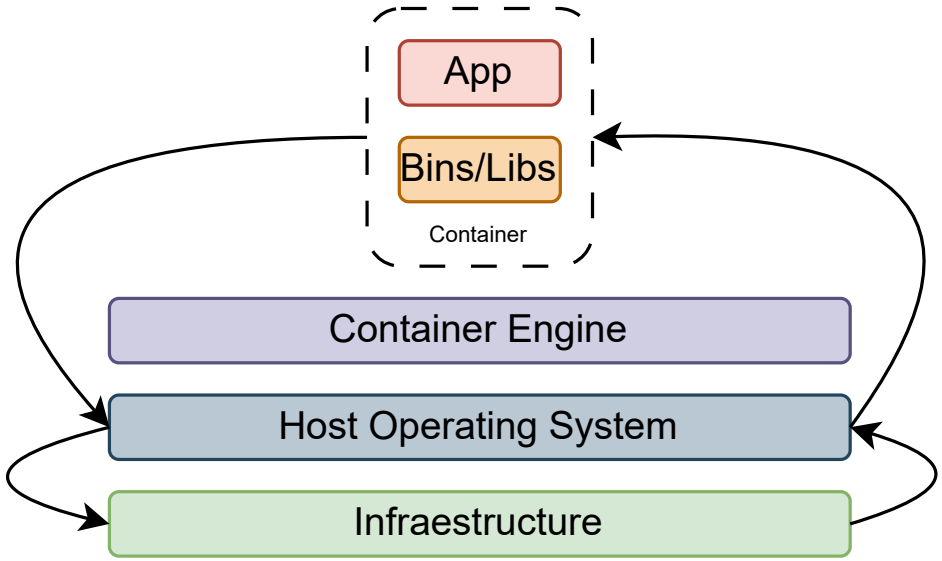
- ¿Es un tipo de virtualización? No, no son una virtualización completa.
- ¿Cómo funciona? Se apoya en funciones del kernel de linux (como Namespaces, Cgroups y OverlayFS).



Contenedores

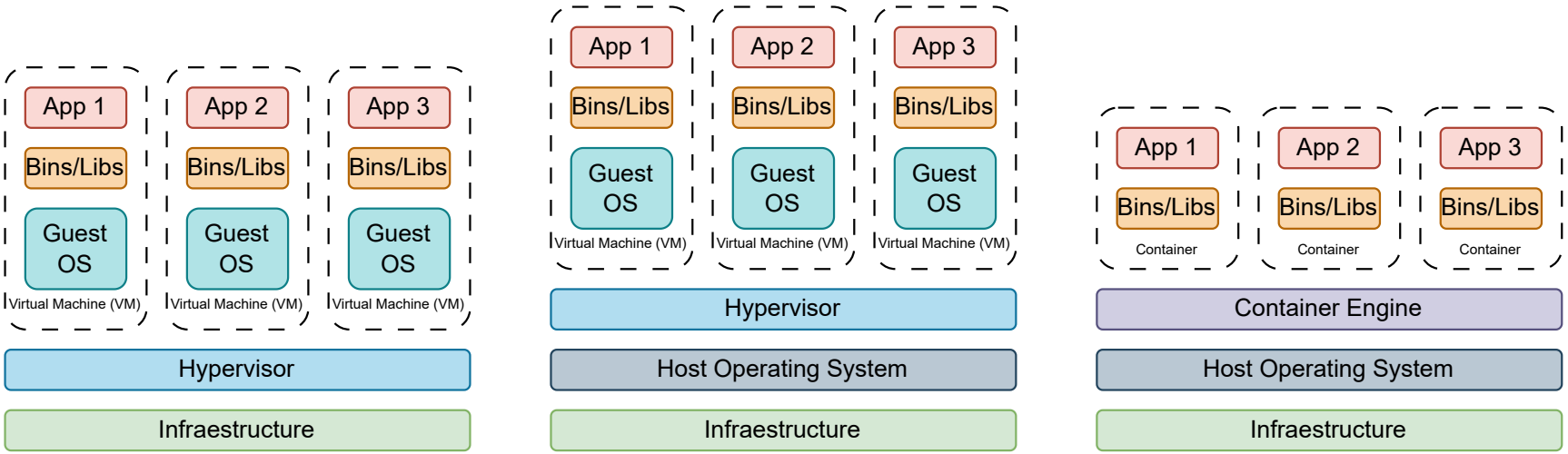
Definición

- ¿Es un tipo de virtualización? No, no son una virtualización completa
- ¿Cómo funciona? Se apoya en funciones del kernel de linux (como Namespaces, Cgroups y OverlayFS)



Contenedores

Definición



Imágenes

Definición

- Imágen de VM (VM Image):

Imágenes

Definición

- Imágen de VM (VM Image): Es un archivo que contiene un disco virtual que tiene instalado un sistema operativo, que el hipervisor la arranca como si fuese un servidor físico.

Imágenes

Definición

- Imágen de VM (VM Image): Es un archivo que contiene un disco virtual que tiene instalado un sistema operativo, que el hipervisor la arranca como si fuese un servidor físico.
- Imagen de container (Container Image):

Imágenes

Definición

- Imágen de VM (VM Image): Es un archivo que contiene un disco virtual que tiene instalado un sistema operativo, que el hipervisor la arranca como si fuese un servidor físico.
- Imagen de container (Container Image): Es es un archivo que incluye todos los archivos, binarios, bibliotecas y configuraciones, para poder ejecutar un contenedor a través de un container engine.

Imágenes

Definición

- Imágen de VM (VM Image): Es un archivo que contiene un disco virtual que tiene instalado un sistema operativo, que el hipervisor la arranca como si fuese un servidor físico.
- Imagen de container (Container Image): Es es un archivo que incluye todos los archivos, binarios, bibliotecas y configuraciones, para poder ejecutar un contenedor a través de un container engine.

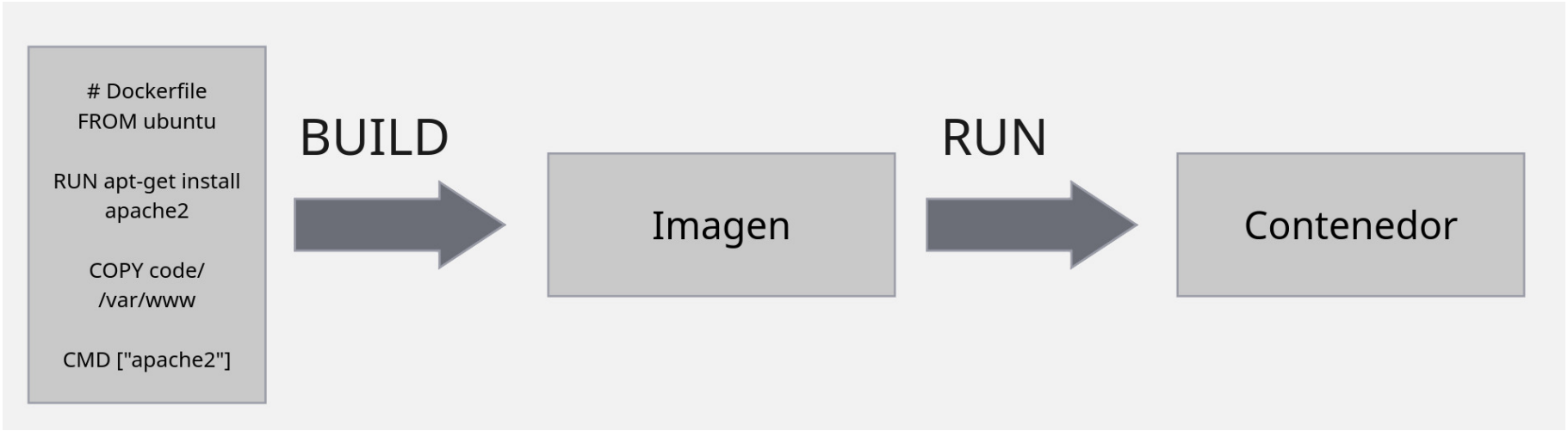
Imagen

Una imagen puede tener:

- SO
- Software
- App

Imágenes

Definición



Contenedores

- Escribimos un dockerfile que es un recetario para crear una imagen.
- Generamos un build para generar la imagen.
- Un contenedor es la ejecucion activa de una imagen.

¿Cómo podemos compartir nuestras imágenes?

Registry

Registry de contenedores

Un **registry** es un servicio donde podemos:

- Almacenar imágenes de contenedores.
- Compartirlas con otros usuarios o equipos.
- Versionarlas y administrarlas de forma centralizada.

Así como un repositorio de código guarda programas, un registry guarda las imágenes listas para ejecutar.

Ejemplos de registries de imágenes

El rol de Docker Hub

- **Docker Hub**: el más popular.

Ejemplos de registries de imágenes

El rol de Docker Hub

- **Docker Hub**: el más popular.
- GitHub Container Registry (GHCR)
- GitLab Container Registry
- Amazon Elastic Container Registry (ECR)
- Google Artifact/Container Registry (GAR/GCR)
- Azure Container Registry (ACR)
- Quay.io, Harbor, JFrog Artifactory

Enfoque

Aunque existen muchas opciones, **Docker Hub es el estándar de facto** y el que usaremos en este taller.

Motor de contenedores

Almacenamiento de imágenes

- docker run image_name *# Crea y ejecuta un contenedor a partir de una imagen.*
- docker start|stop container_name *# Inicia o para un container.*
- docker rm container_name *# Remueve el contenedor.*
- docker ps *# Lista los contenedores que están ejecutando.*
- docker container stats *# Muestra el estado del uso de los recursos.*
- docker exec container_name command *# Ejecuta el comando dentro del contenedor.*
- docker logs container_name *# Muestra los logs del contenedor.*

[0] **https://docs.docker.com/reference/cli/docker/**

Dockerfile

Comandos

- FROM: Crear una nueva etapa de compilación a partir de una imagen base.
- WORKDIR: Cambiar directorio de trabajo.
- RUN: Ejecutar comandos de compilación.
- COPY: Copiar archivos y directorios.
- EXPOSE: Describe en qué puertos está escuchando tu aplicación.
- CMD: Especificar comandos predeterminados.
- ...

[0] **https://docs.docker.com/reference/dockerfile/**

Motor de contenedores

Puertos

- Los contenedores tambien manejan su propia red interna.
- Si queremos comunicarnos con el contenedor, podemos exponer puertos hacia nuestro host
- Una de las maneras de hacerlo es con "-p" en el comando de docker run.

¿Cómo funcionan los volúmenes en Docker?

Persistencia de datos

Definición

Un **volumen** es un mecanismo de Docker para almacenar datos fuera del ciclo de vida de un contenedor. Sirven para que la información **no se pierda** al detener o eliminar un contenedor.

- Los datos se almacenan en el host (por defecto en `/var/lib/docker/volumes/`).
- Se pueden compartir entre varios contenedores.
- Separan la **aplicación** (contenedor) de sus **datos**.

¿Cómo funcionan los volúmenes en Docker?

Persistencia de datos

Definición

Un **volumen** es un mecanismo de Docker para almacenar datos fuera del ciclo de vida de un contenedor. Sirven para que la información **no se pierda** al detener o eliminar un contenedor.

- Los datos se almacenan en el host (por defecto en `/var/lib/docker/volumes/`).
- Se pueden compartir entre varios contenedores.
- Separan la **aplicación** (contenedor) de sus **datos**.

Ejemplo

```
docker run -v  /data:/app/data myapp
```

- `/data` → carpeta en el host.
- `/app/data` → carpeta dentro del contenedor.

Variables de entorno en Docker

Parametrización de contenedores

¿Qué es una variable de entorno?

Una **variable de entorno** es un par `NOMBRE=VALOR` que se almacena en el sistema operativo y que las aplicaciones pueden leer para modificar su comportamiento sin cambiar el código.

- Permiten parametrizar la configuración de programas.
- Ejemplo típico: base de datos, puerto, modo debug, usuario, contraseña.
- Se pueden definir al crear un contenedor y son accesibles desde dentro del mismo.

Variables de entorno en Docker

Parametrización de contenedores

¿Qué es una variable de entorno?

Una **variable de entorno** es un par `NOMBRE=VALOR` que se almacena en el sistema operativo y que las aplicaciones pueden leer para modificar su comportamiento sin cambiar el código.

- Permiten parametrizar la configuración de programas.
- Ejemplo típico: base de datos, puerto, modo debug, usuario, contraseña.
- Se pueden definir al crear un contenedor y son accesibles desde dentro del mismo.

Uso en Docker

`docker run -e NOMBRE_VAR=valor imagen`

Ejemplo: `docker run -e MYSQL_ROOT_PASSWORD=secret mysql`

- `MYSQL_ROOT_PASSWORD` es la variable de entorno.
- `secret` es el valor que se le asigna.

¿Para qué sirven las variables de entorno en Docker?

Parametrización de contenedores por el usuario

Idea principal

Las variables de entorno permiten al usuario **configurar un contenedor** sin necesidad de modificar la imagen ni el código de la aplicación.

- Configurar contraseñas, nombres de usuario, puertos o rutas de archivos.
- Activar o desactivar modos de operación (ej. modo debug).
- Permiten que la misma imagen funcione en distintos entornos (desarrollo, testing, producción) solo cambiando las variables.

¿Qué es Docker Compose?

Automatización de contenedores

Definición

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones multicontenedor mediante un archivo de configuración (`docker-compose.yml`).

- Permite describir contenedores, redes, volúmenes y variables de entorno de forma declarativa.
- Sustituye la necesidad de escribir múltiples comandos `docker run` a mano.
- Facilita levantar, detener y escalar aplicaciones completas con un solo comando: `docker-compose up` y `docker-compose down`.
- Mantiene toda la configuración de la aplicación en un solo archivo, fácil de versionar y compartir.

¿Cómo funcionan los networks en Docker?

Conexión entre contenedores

Definición

Un **network** en Docker es una red virtual que conecta contenedores entre sí y con el host, de manera aislada y controlada.

- **bridge** (por defecto): cada contenedor recibe una IP propia y puede comunicarse con otros usando **DNS interno de Docker**, es decir, por **nombre de contenedor** en lugar de IP.
- **host**: el contenedor comparte directamente la red del host (no tiene IP separada).
- **none**: sin acceso a red, solo `localhost`.

¿Cómo funcionan los networks en Docker?

Conexión entre contenedores

Definición

Un **network** en Docker es una red virtual que conecta contenedores entre sí y con el host, de manera aislada y controlada.

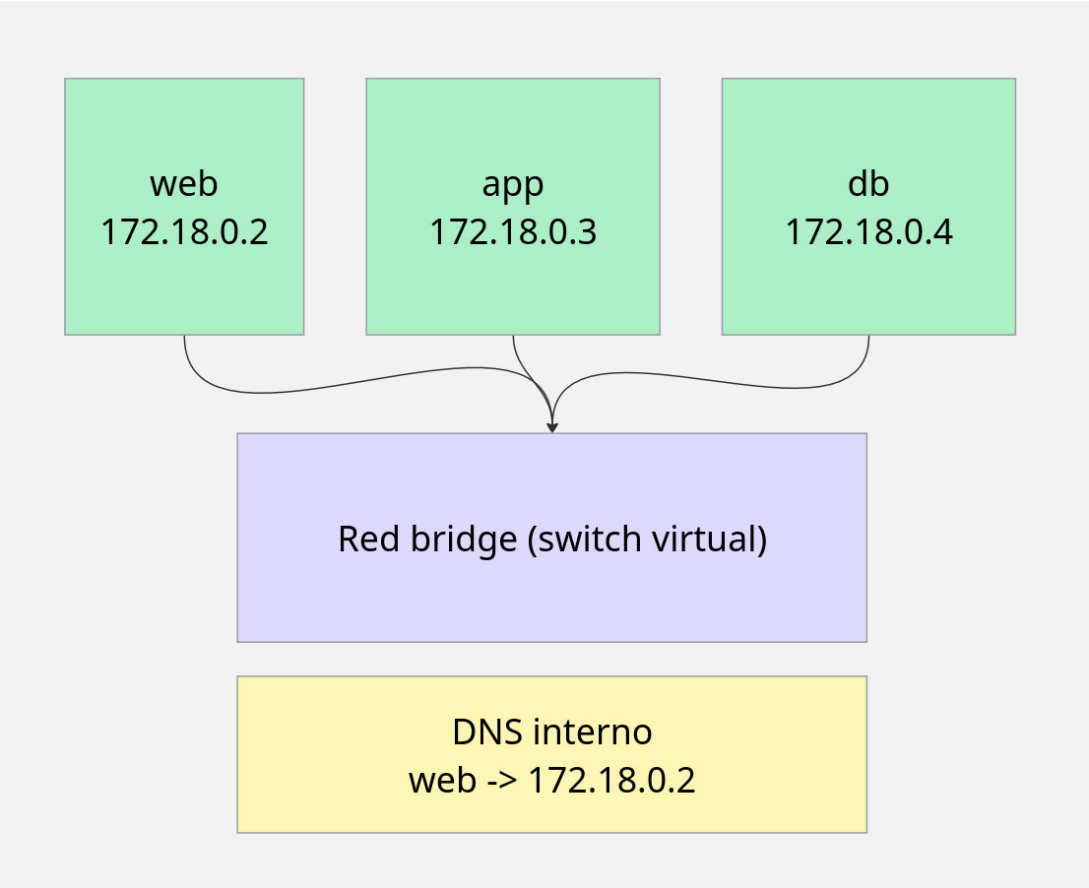
- **bridge** (por defecto): cada contenedor recibe una IP propia y puede comunicarse con otros usando **DNS interno de Docker**, es decir, por **nombre de contenedor** en lugar de IP.
- **host**: el contenedor comparte directamente la red del host (no tiene IP separada).
- **none**: sin acceso a red, solo `localhost`.

Ejemplo

```
docker network create mi_red
docker run -d --name web --network mi_red nginx
docker run -it --rm --network mi_red alpine ping web
```

¿Cómo funcionan los networks en Docker?

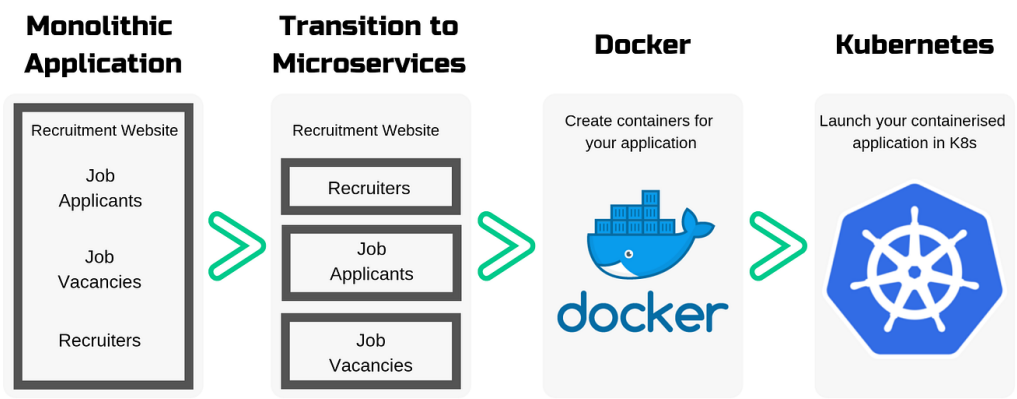
Conexión entre contenedores



Orquestación de contenedores: siguiente paso después de Docker Compose

- **¿Qué es la orquestación de contenedores?**
  - Gestión automatizada de múltiples contenedores en uno o varios servidores.
  - Asegura que los contenedores funcionen correctamente, se comuniquen y escalen según demanda.
- **Por qué es necesaria**
  - Docker Compose funciona localmente, pero tiene limitaciones al distribuir servicios en varios nodos.
  - Necesidad de recuperación automática, escalado y balanceo de carga.
  - Gestión centralizada de redes y almacenamiento persistente.
- **Qué hace la orquestación**
  - Automatiza la ejecución de contenedores.
  - Monitorea y reinicia contenedores que fallan.
  - Escala servicios según la demanda.
  - Balancea carga entre instancias.
  - Gestiona almacenamiento y redes de manera declarativa.

Orquestación de contenedores: siguiente paso después de Docker Compose



Bibliografía

- Documentación de Docker: https://docs.docker.com/reference/