



Módulo 5: Arquitectura y escalabilidad

Fundamentos de tecnologías de contenedores

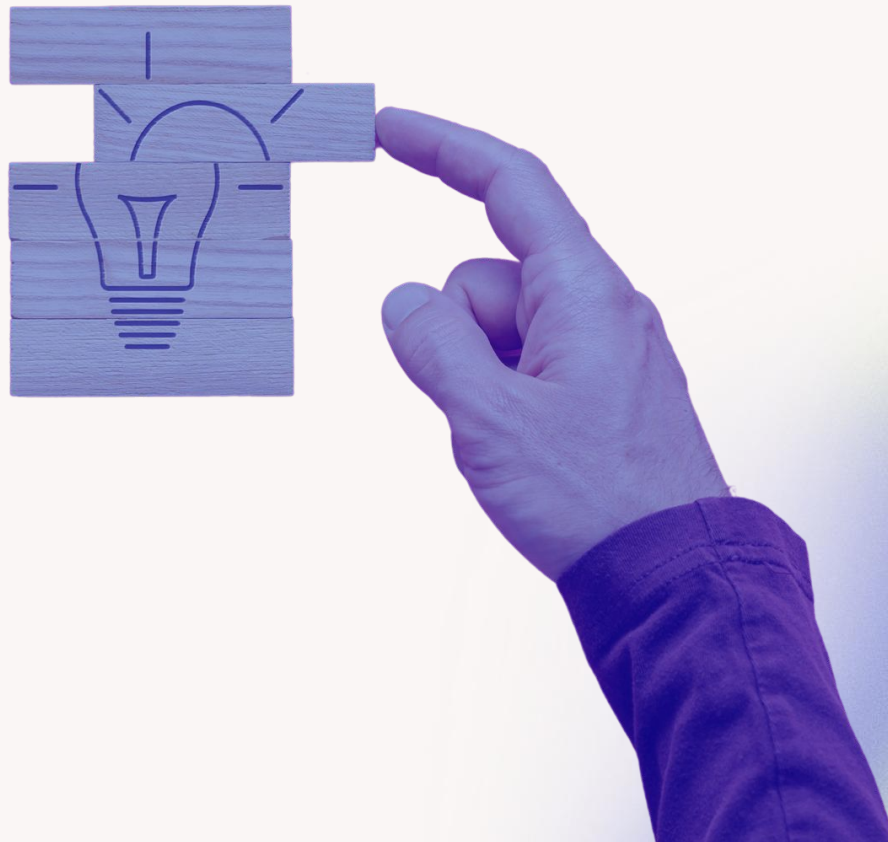


MÓDULO 5

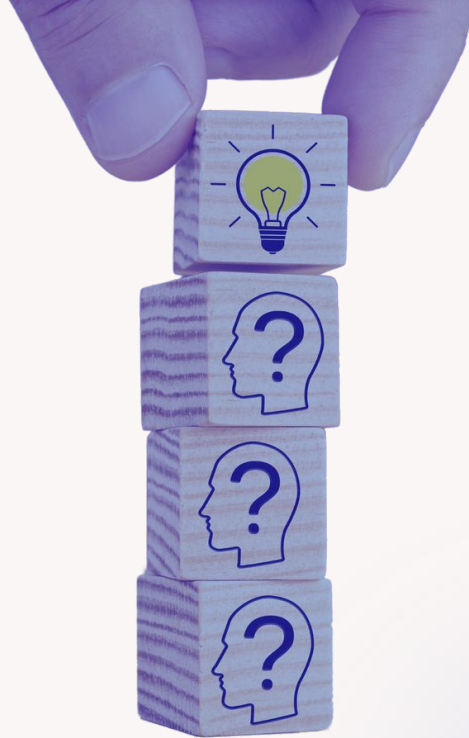
- Principios fundamentales de diseño de una arquitectura
- Arquitecturas monolíticas y microservicios
- **Fundamentos de tecnologías de contenedores**
- Orquestación de contenedores con Kubernetes
- Registro de contenedores
- Platform as a Service, Backend as a Service y Frontend as a Service

Objetivos

Comprender los fundamentos de las tecnologías de contenedores, su uso con Docker y su integración en flujos DevOps para el despliegue de servicios en entornos aislados y escalables.




¿Qué ventajas aporta
el uso de
contenedores frente a
las máquinas virtuales
en el desarrollo de
aplicaciones?



Conceptos generales



¿Qué es un contenedor?

 Un **contenedor** es una unidad estándar de software que empaqueta el código y todas sus dependencias para que la aplicación se ejecute de manera rápida y confiable en cualquier entorno.

✓ Características:


- Aislados del sistema anfitrión.
- Ligeros y portables.
- Inician en segundos.
- Basados en el kernel del sistema operativo.

Ejemplo práctico (usando Docker):

```
docker run hello-world
```

Este comando ejecuta un contenedor basado en una imagen ligera que imprime un mensaje de bienvenida.





Diferencias entre contenedores y máquinas virtuales

 Aunque ambos proporcionan entornos aislados, su arquitectura es diferente:

Característica	Contenedor	Máquina Virtual
Arranque	Segundos	Minutos
Peso	Ligero (MB)	Pesado (GB)
Virtualización	A nivel de sistema operativo	A nivel de hardware
Aislamiento	Parcial (procesos)	Completo (kernel y procesos)
Uso común	Microservicios, CI/CD	Infraestructura tradicional

Beneficios y casos de uso de los contenedores

✓ Beneficios clave:

-  Portabilidad: Ejecuta la misma aplicación en desarrollo, pruebas y producción.
-  Eficiencia: Reduce uso de recursos frente a las VM.
-  Reproducibilidad: Entornos consistentes en todo el ciclo DevOps.
-  Escalabilidad: Facilita despliegues con herramientas como Kubernetes.

Casos de uso comunes:

- Pruebas automatizadas en CI/CD.
- Aplicaciones distribuidas (microservicios).
- Aislamiento de entornos para desarrollo.
- Despliegue de entornos de datos temporales.

Seguridad en entornos de contenedores

📌 Los contenedores comparten el kernel del sistema operativo, por lo que **la seguridad depende de una correcta configuración.**


✅ Buenas prácticas:

- 🔒 Usar imágenes oficiales y firmadas.
- 🧪 Escanear vulnerabilidades en imágenes (docker scan).
- 🔒 Ejecutar procesos como usuarios no root.
- 🔄 Mantener actualizadas las imágenes.
- 📦 Limitar permisos con seccomp, AppArmor o SELinux.

📖 Ejemplo: escaneo de vulnerabilidades

```
docker scan nginx
```

Comparación entre diferentes tecnologías de contenedores

 Aunque Docker es el más conocido, existen otras herramientas y tecnologías de contenedores.

Tecnología	Enfoque	Compatible con Docker	Caso de uso típico
Docker	Contenedorización general	Sí	Aplicaciones web y microservicios
Podman	Contenedores sin daemon	Parcial	Sistemas rootless
LXC/LXD	Virtualización ligera	No	Hosting de múltiples sistemas
CRI-O	Compatible con Kubernetes	Sí	Sustituye a Docker en K8s
rkt (CoreOS)	Aislación con seguridad	No	Seguridad en producción (deprecated)

Ejemplo con Podman:

```
podman run nginx
```

*Funciona igual que **docker run**, pero sin requerir un demonio central (**dockerd**).*


¿Cuáles son los principales beneficios de utilizar contenedores en entornos de desarrollo?



Introducción a Docker



¿Qué es Docker y para qué se utiliza?

 Docker es una plataforma open-source diseñada para desarrollar, enviar y ejecutar aplicaciones dentro de contenedores. Simplifica la distribución de software al encapsular la aplicación con todas sus dependencias.

✓ Docker permite:


- Crear entornos consistentes en cualquier sistema operativo.
- Empaquetar aplicaciones en imágenes portables.
- Automatizar despliegues y pruebas.

Ejemplo de uso básico:

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

*Lanza un contenedor en segundo plano basado en la imagen de **nginx**, mapeando el puerto 8080 del host al 80 del contenedor.*

Arquitectura de un contenedor Docker






 La arquitectura Docker se basa en una jerarquía de componentes clave que permiten construir y ejecutar contenedores de forma aislada.


Componentes clave:

- **Docker CLI:** Interfaz para interactuar con el motor Docker.
- **Docker Daemon:** Gestiona contenedores e imágenes.
- **Imágenes:** Plantillas inmutables para crear contenedores.
- **Contenedores:** Instancias en ejecución de una imagen.
- **Registry:** Almacena y distribuye imágenes (e.g., Docker Hub).


Beneficios de utilizar un contenedor Docker

✓ Ventajas destacadas:




-  Empaquetado completo: Incluye código, dependencias, configuraciones.
-  Entornos reproducibles: Misma ejecución en desarrollo, test y producción.
-  Velocidad: Contenedores arrancan en segundos.
-  Portabilidad: Funcionan en cualquier host con Docker.
-  Eficiencia: Bajo uso de recursos frente a VMs.

 **Ejemplo práctico:** un backend Node.js se ejecuta de igual forma en cualquier entorno sin problemas de dependencias, gracias a su contenedor.

Rol del contenedor Docker dentro del ciclo DevOps

 Docker es un facilitador clave en la cultura DevOps por su capacidad de automatizar entornos y acelerar despliegues.

✓ Aplicación en DevOps:


-  Se integra con herramientas como Jenkins, GitHub Actions y GitLab CI.
-  Permite ejecutar pruebas automáticas en contenedores limpios.
-  Facilita entregas continuas con contenedores predefinidos.

Ejemplo de integración:

```
docker build -t myapp:v1 .  
docker run --rm myapp:v1 npm test
```

Construye la imagen y ejecuta las pruebas automáticamente.

Comparación de Docker con otras herramientas de contenedorización

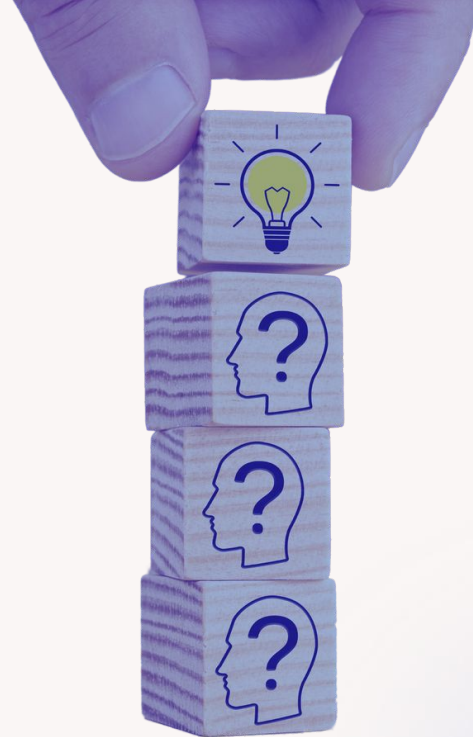
 Existen otras plataformas similares, pero Docker destaca por su ecosistema maduro y adopción masiva.

Herramienta	Característica destacada	Comparación con Docker
Podman	Rootless y sin daemon	Más seguro, pero menos popular
LXC/LXD	Virtualización ligera de sistema	Más cercano a VM que a contenedor app
CRI-O	Reemplazo de Docker en Kubernetes	No incluye CLI, usa OCI estándar
containerd	Runtime de contenedores ligero	Usado por Docker como motor interno

 **Ejemplo con Podman (equivalente a Docker):**

```
podman run -d httpd
```

¿Cuáles son los
componentes clave de
la arquitectura de
Docker y cómo
contribuyen al flujo de
trabajo?



Conceptos básicos de Docker



Instalación de Docker en distintos entornos

📌 Docker puede instalarse en Linux, macOS y Windows.

<https://www.docker.com/>



Instalación en Linux (Ubuntu/Debian)

```
# Eliminar versiones anteriores si las hay
sudo apt-get remove docker docker-engine docker.io
containerd runc

# Actualizar e instalar dependencias necesarias
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
lsb-release


# Agregar la clave GPG oficial de Docker
sudo mkdir -p /etc/apt/keyrings
curl -fsSL
https://download.docker.com/linux/ubuntu/gpg \
| sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

```
# Agregar el repositorio de Docker
echo \
  "deb [arch=$(dpkg --print-architecture) \
    signed-by=/etc/apt/keyrings/docker.gpg] \
    https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable" \
  | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null

# Instalar Docker Engine
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli
containerd.io


# Verificar instalación
docker --version
```

Instalación en macOS

1. Descarga Docker Desktop desde:
 <https://www.docker.com/products/docker-desktop/>
2. Ejecuta el instalador y sigue las instrucciones.
3. Al finalizar, abre Docker desde la barra de menú.
4. Verifica instalación con:

```
docker --version  
docker run hello-world
```

Instalación en Windows 10/11

1. Asegúrate de tener Windows 10/11 Pro, Enterprise o Education (Docker requiere WSL2).
2. Activa WSL y la virtualización desde el BIOS si aún no está activa.
3. Descarga Docker Desktop:
 <https://www.docker.com/products/docker-desktop/>
4. Ejecuta el instalador y selecciona la opción de usar WSL2.
5. Al finalizar, abre Docker desde el menú de inicio.
6. Verifica instalación:

```
docker --version  
docker run hello-world
```

Imágenes y contenedores en Docker

📌 En Docker, **una imagen** es como una receta congelada con instrucciones para ejecutar una app. **Un contenedor** es una instancia viva de esa imagen.

✅ **Ejemplo práctico 1:** Descargar y ejecutar una app simple:

```
docker run -d -p 3000:80 nginx
```

✅ Ver contenedores activos:

```
docker ps
```

✅ Ver imágenes descargadas:

```
docker images
```

📖 *Esto lanza NGINX desde una imagen oficial, corriendo en segundo plano (-d) y exponiendo el puerto 80 del contenedor al 3000 del host.*

Dockerfile y construcción de imágenes

 Un **Dockerfile** permite construir imágenes personalizadas usando instrucciones secuenciales.

✓ Ejemplo práctico 2:

Creamos una app web en Node.js.

Estructura de archivos:

```
/miapp
├── Dockerfile
├── package.json
└── index.js
```

package.json

```
{
  "name": "miapp",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

Dockerfile y construcción de imágenes

index.js

```
const express = require('express');
const app = express();
const port = process.env.PORT || 3000;

app.get('/', (req, res) => res.send('¡Hola desde un contenedor Docker!'));

app.listen(port, () => console.log(`Servidor en http://localhost:${port}`));
```


Dockerfile

```
FROM node:18
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

Construcción y ejecución:

```
docker build -t miapp .
docker run -p 3000:3000 miapp
```

Repositorios de imágenes y Docker Hub


 Docker Hub es un repositorio en la nube donde se alojan y comparten imágenes.

✓ **Publicar tu imagen:**

```
docker tag miapp usuariodocker/miapp:1.0
```


```
docker login
```

```
docker push usuariodocker/miapp:1.0
```

 Esto permite que tu equipo o tus pipelines descarguen la imagen desde cualquier lugar con:

```
docker pull usuariodocker/miapp:1.0
```

Variables de entorno y configuración en contenedores

 Las variables de entorno son ideales para personalizar el comportamiento sin modificar el código.

✓ **Modificar el puerto con una variable de entorno:**

 `index.js` ya usa:

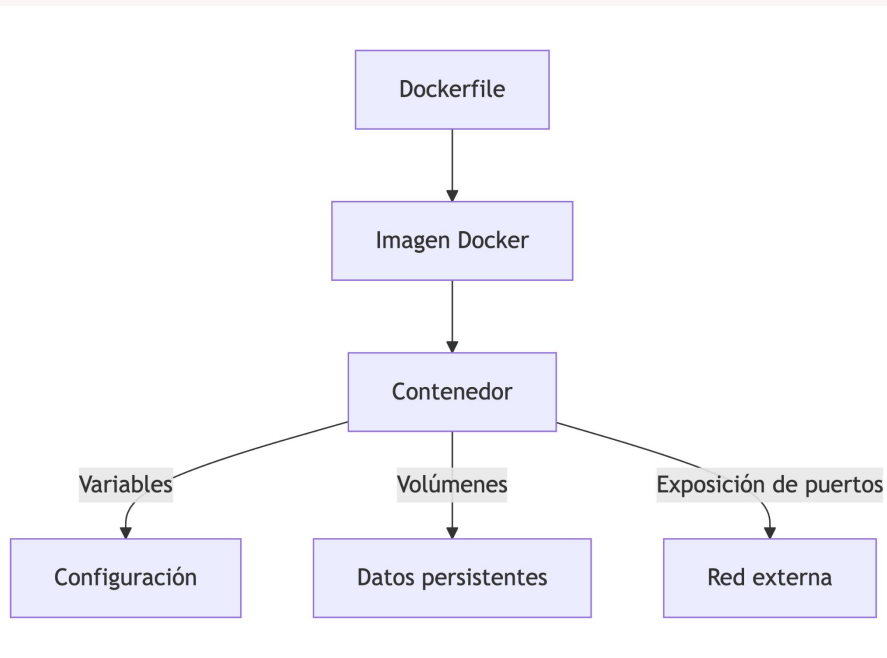
```
const port = process.env.PORT || 3000;
```

✓ **Ejecutar con variable de entorno personalizada:**

```
docker run -e PORT=4000 -p 4000:4000 miapp
```

Volúmenes y persistencia de datos en Docker


📌 Por defecto, los datos generados en un contenedor se pierden al detenerlo. Los volúmenes permiten persistencia.



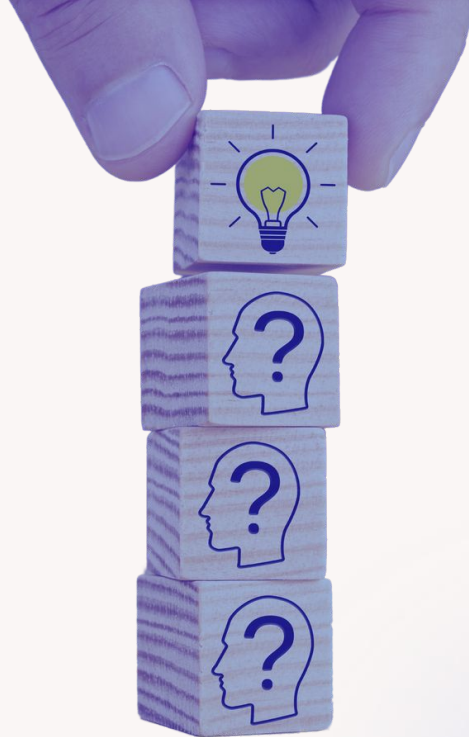
Volúmenes y persistencia de datos en Docker

Ejemplo práctico con un contenedor de PostgreSQL:

```
docker volume create pgdata  
docker run --name postgres-dev -e POSTGRES_PASSWORD=admin \  
-v pgdata:/var/lib/postgresql/data -p 5432:5432 -d postgres
```

 Ahora los datos se guardan en el volumen pgdata y no se pierden aunque reinicies el contenedor.


¿Cómo se utilizan los
Dockerfiles y los
volúmenes en Docker
para construir
imágenes
personalizadas?



Operación básica de Docker



Operación básica de Docker

 Una vez instalado Docker, es fundamental conocer los comandos básicos que permiten gestionar el ciclo de vida de contenedores, imágenes y redes.

```
# Ejecutar un contenedor (descarga si no existe)
docker run hello-world
```

```
# Descargar una imagen sin ejecutarla
docker pull nginx
```

```
# Ver imágenes disponibles localmente
docker images
```

```
# Crear y ejecutar un contenedor
docker run -d --name webserver -p 8080:80 nginx
```

```
# Listar contenedores en ejecución
docker ps
```

```
# Listar todos los contenedores (incluso detenidos)
docker ps -a
```

```
# Detener un contenedor
docker stop webserver
```

```
# Iniciar un contenedor detenido
docker start webserver
```

```
# Eliminar un contenedor detenido
docker rm webserver
```

```
# Eliminar una imagen
docker rmi nginx
```

```
# Construir una imagen personalizada (usando Dockerfile)
docker build -t mi-imagen .
```

```
# Subir una imagen a Docker Hub
docker push usuario/mi-imagen
```

```
# Descargar una imagen desde Docker Hub
docker pull usuario/mi-imagen
```

Creación y ejecución de contenedores

Vamos a iniciar un contenedor NGINX que estará disponible en <http://localhost:8080>

```
docker run -d --name webserver -p 8080:80 nginx
```

- **-d**: Modo “detached” (en segundo plano).
- **--name**: Nombre del contenedor.
- **-p**: Redireccionamiento de puertos.



Puedes acceder a la página de bienvenida de NGINX en el navegador: <http://localhost:8080>

Interacción con un contenedor en ejecución

- **exec**: Permite ejecutar comandos dentro del contenedor.
- **logs**: Visualiza la salida del proceso del contenedor.

```
# Abrir una terminal dentro del contenedor
```

```
docker exec -it webserver bash
```

```
# Ver Logs del contenedor
```

```
docker logs webserver
```

Gestión del ciclo de vida del contenedor

Reiniciar un contenedor

```
docker restart webserver
```

Ver estadísticas de uso de recursos

```
docker stats
```

Inspeccionar detalles técnicos

```
docker inspect webserver
```

Redes y comunicación entre contenedores

Docker crea una red bridge por defecto. Para que dos contenedores se comuniquen entre sí:

```
# Crear una red personalizada
```

```
docker network create red-interna
```

```
# Ejecutar dos contenedores conectados a esa red
```

```
docker run -d --name nginx1 --network red-interna nginx
```

```
docker run -d --name nginx2 --network red-interna nginx
```


Ahora, **nginx1** y **nginx2** pueden comunicarse usando sus nombres como hostname.

¿Cómo se gestionan y comunican entre sí los contenedores durante su ciclo de vida en entornos Docker?



Implementación de servicios comunes en Docker

Base de datos PostgreSQL en Docker

 PostgreSQL es una base de datos relacional muy usada. Docker nos permite levantar una instancia aislada y lista para conectar desde cualquier

```
docker run --name pgdb -e POSTGRES_PASSWORD=admin123 -p 5432:5432 -d postgres
```


✓ Explicación:

- `--name pgdb`: Nombre del contenedor.
- `-e POSTGRES_PASSWORD`: Variable de entorno con la contraseña.
- `-p 5432:5432`: Puerto de exposición.
- `-d`: Modo en segundo plano.

Verificar conexión:

```
docker exec -it pgdb psql -U postgres
```


Base de datos MongoDB en Docker

 MongoDB es una base de datos NoSQL orientada a documentos. También se ejecuta fácilmente en contenedores.

```
docker run --name mongodb -d -p 27017:27017 -e MONGO_INITDB_ROOT_USERNAME=admin -e  
MONGO_INITDB_ROOT_PASSWORD=admin123 mongo
```


Persistencia con volúmenes:

```
docker run --name mongodb \  
-v mongodata:/data/db \  
-d -p 27017:27017 \  
-e MONGO_INITDB_ROOT_USERNAME=admin \  
-e MONGO_INITDB_ROOT_PASSWORD=admin123 mongo
```

Conexión con cliente Mongo:

```
docker exec -it mongodb mongosh -u admin -p admin123
```

Servidor web Nginx en Docker

 Nginx es un servidor web ligero y potente. Es ideal para servir contenido estático o actuar como proxy.

```
docker run --name nginx-web -d -p 8080:80 nginx
```

 Servir un sitio web desde un volumen local:

1. Crear un archivo HTML:


```
mkdir -p ./html  
echo "<h1>Hola desde NGINX en Docker</h1>" >  
./html/index.html
```

2. Ejecutar Nginx montando el contenido:

```
docker run --name nginx-vol -d -p 8081:80 -v  
$(pwd)/html:/usr/share/nginx/html nginx
```

 Navegar a <http://localhost:8081> para ver el contenido.

Implementación de Redis en un contenedor

 Redis es una base de datos en memoria usada para caché y mensajería. Su despliegue es inmediato en Docker:

```
docker run --name redis-server -d -p 6379:6379 redis
```

Verificar conexión:


```
docker exec -it redis-server redis-cli
```

Comandos en Redis:

```
SET saludo "Hola Docker"
```

```
GET saludo
```

Configuración de entornos multi-contenedor con Docker Compose

 Cuando necesitas varios servicios funcionando en conjunto, Docker Compose te permite declararlos en un solo archivo y levantarlos con un solo comando.

Ejemplo de `docker-compose.yml`

```
version: "3.8"

services:
  db:
    image: postgres
    environment:
      POSTGRES_PASSWORD: admin123
    ports:
      - "5432:5432"

  redis:
    image: redis
    ports:
      - "6379:6379"

  web:
    image: nginx
    ports:
      - "8080:80"
    volumes:
      - ./html:/usr/share/nginx/html
```

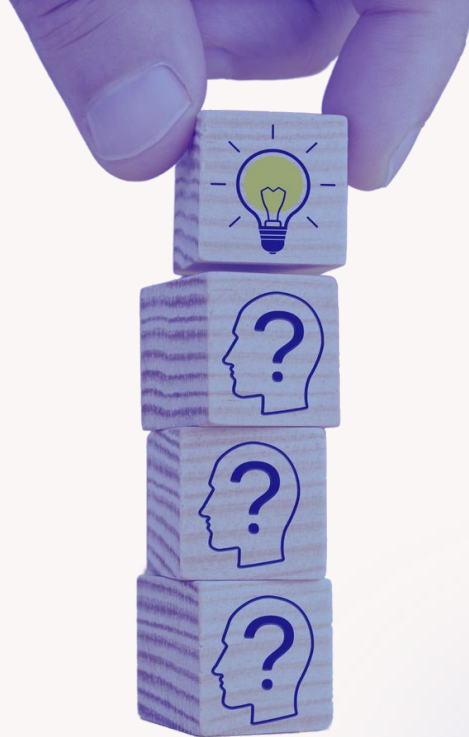
Comandos:

```
# Levantar todos los servicios
docker-compose up -d

# Ver estado
docker-compose ps

# Detener servicios
docker-compose down
```

¿Cuáles son las ventajas de utilizar Docker Compose para implementar entornos multi-contenedor?



aspasia

LA FORMACIÓN DE TU FUTURO



Ejercicio Guiado: Creación de un Entorno Multi-Contenedor



Ejercicio Guiado: Creación de un Entorno Multi-Contenedor

En este ejercicio vas a construir un entorno básico de contenedores para una aplicación web compuesta por:

- Un **frontend HTML estático servido con Nginx**
- Una **base de datos PostgreSQL**
- Una configuración unificada con **Docker Compose**

Objetivos

- Instalar y operar Docker en el entorno local.
- Construir una imagen personalizada para un servidor web.
- Levantar un contenedor de PostgreSQL con datos persistentes.
- Conectar servicios mediante redes Docker.
- Usar docker-compose.yml para automatizar la gestión del entorno.





Paso 1: Instalación de Docker

- 📌 ¿Qué haremos?
- Verificaremos que Docker esté instalado correctamente.

Instrucciones:

- Verifica si Docker está instalado:

```
docker --version
```

- Si no está instalado, accede a <https://www.docker.com/products/docker-desktop/>
- Verifica que puedes ejecutar un contenedor de prueba:

```
docker run hello-world
```

- ✅ Si ves un mensaje de éxito, estás listo para comenzar.



Paso 2: Crear la estructura del proyecto



¿Qué haremos?

- Prepararemos los archivos y carpetas necesarias.

```
mkdir docker_inmobiliaria && cd docker_inmobiliaria  
mkdir nginx html  
touch html/index.html nginx/Dockerfile docker-compose.yml
```



Paso 3: Crear el frontend con Nginx



¿Qué haremos?

- Serviremos una página estática con Nginx en un contenedor.

Contenido de `html/index.html`:

```
<!DOCTYPE html>
<html>
<head><title>Inmuebles</title></head>
<body>
  <h1>Bienvenido a InmoApp</h1>
  <p>Listado de propiedades pronto disponible...</p>
</body>
</html>
```

Contenido de `nginx/Dockerfile`:

```
FROM nginx:latest
COPY ../html /usr/share/nginx/html
```



Paso 4: Configurar la base de datos PostgreSQL



¿Qué haremos?

- Usaremos la imagen oficial de PostgreSQL en un contenedor con volumen persistente.
- No requiere Dockerfile, lo configuraremos vía Docker Compose.

Paso 5: Crear el archivo `docker-compose.yml`

¿Qué haremos?

- Definiremos ambos servicios y su configuración.

```
version: '3.8'
```

```
services:
```

```
  web:
```

```
    build: ./nginx
```

```
    ports:
```

```
      - "8080:80"
```

```
    depends_on:
```

```
      - db
```

```
  db:
```

```
    image: postgres:13
```

```
    environment:
```

```
      POSTGRES_USER: admin
```

```
      POSTGRES_PASSWORD: admin123
```

```
      POSTGRES_DB: inmobiliaria
```

```
    volumes:
```

```
      - pgdata:/var/lib/postgresql/data
```

```
volumes:
```

```
  pgdata:
```



Paso 6: Levantar los contenedores



¿Qué haremos?

- Ejecutaremos todo con un solo comando.

```
docker-compose up -d
```



Luego abre <http://localhost:8080> para ver el frontend.

Para inspeccionar los contenedores:

```
docker ps
```



Paso 7: Validar la persistencia de la base de datos



¿Qué haremos?

- Nos conectaremos al contenedor de PostgreSQL.

```
docker exec -it docker_inmobiliaria-db-1 psql -U admin -d inmobiliaria
```

Dentro del cliente PostgreSQL puedes ejecutar:

```
CREATE TABLE propiedades (id SERIAL PRIMARY KEY, direccion TEXT);  
INSERT INTO propiedades (direccion) VALUES ('Calle 123, Bogotá');  
SELECT * FROM propiedades;
```



Luego puedes detener los contenedores y al reiniciarlos (`docker-compose up -d`) la información persistirá gracias al volumen.

Preguntas finales

- ¿Cuál es la diferencia entre una imagen y un contenedor?
- ¿Qué beneficios aporta Docker Compose frente a ejecutar contenedores por separado?
- ¿Qué mecanismos de seguridad podrías aplicar en esta arquitectura?
- ¿Cómo podrías extender este entorno para simular una app completa?

Entregable:

Cada equipo debe entregar:

- El archivo docker-compose.yml.
- El Dockerfile de Nginx.
- Captura de pantalla del navegador accediendo al frontend.
- Captura del resultado de una consulta a la tabla propiedades.
- Una breve explicación de cómo se comunican los servicios entre sí.



Resumen de lo aprendido

- **Conceptos de Contenedores:** Aislamiento de aplicaciones sin necesidad de máquinas virtuales, mejorando portabilidad, eficiencia y tiempos de despliegue.
- **Introducción a Docker:** Herramienta líder en contenedorización, utilizada para crear, ejecutar y gestionar contenedores dentro del ciclo de vida DevOps.
- **Operación con Docker:** Uso de imágenes, Dockerfile, volúmenes y redes, además de comandos clave para gestionar contenedores y su ciclo de vida.
- **Servicios en Contenedores:** Implementación de servicios comunes como PostgreSQL, MongoDB, Redis y Nginx, así como entornos multi-contenedor con Docker Compose.

Próxima clase...

Orquestación de contenedores con Kubernetes

