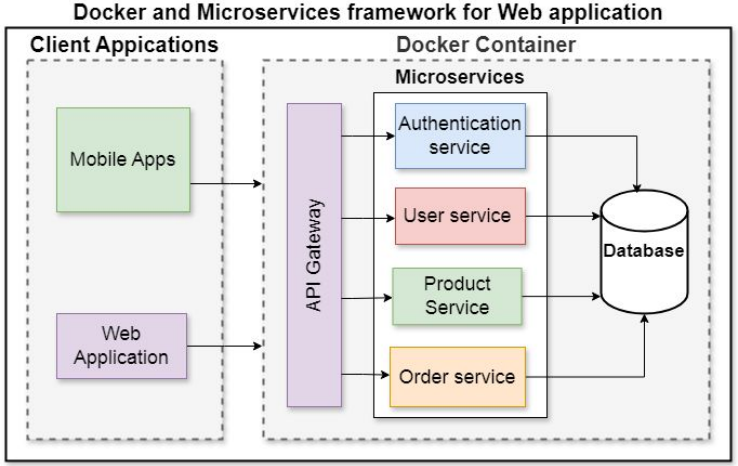
Introducción a Docker

Del monolito a microservicios

¿Qué es Docker?

- Plataforma de contenedores que permite empaquetar aplicaciones junto a sus dependencias.
- Se pueden ejecutar en cualquier entorno de manera consistente (como una máquina virtual) pero son mucho más ligeros.
- Permite separar servicios: front, back, login, gestión de DBs…
    - Más mantenibilidad, separación de responsabilidades y trabajo en paralelo.



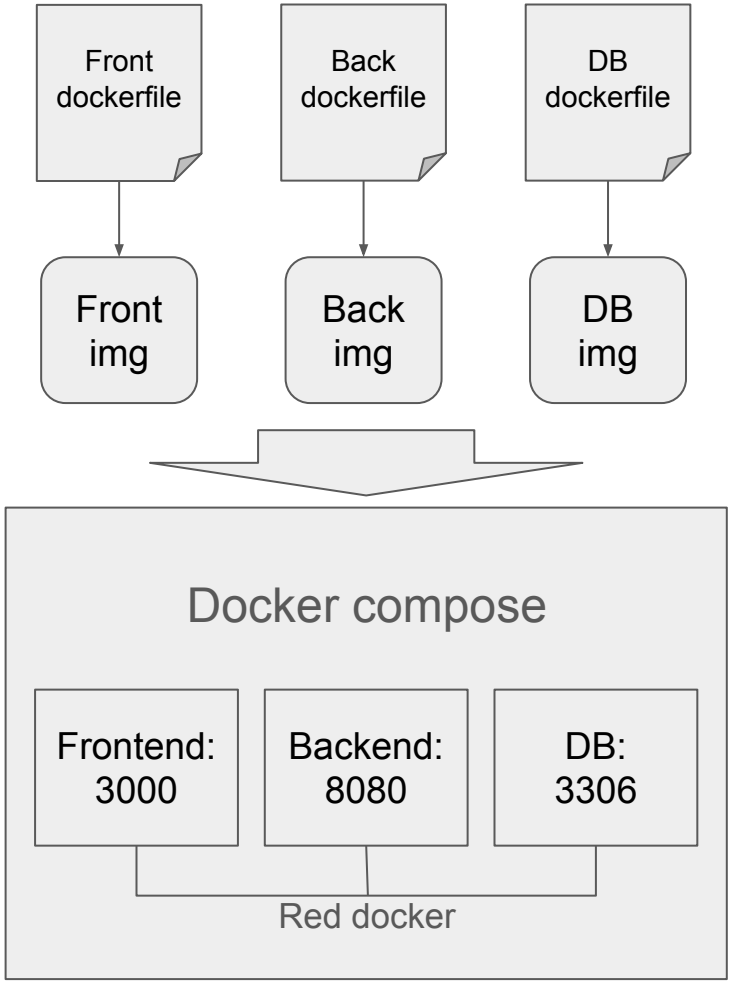
¿Qué es Docker?

- No están pensados para usuario, puesto que no tienen ‘escritorio’ y deberían contener lo básico.
- Son efímeros, pueden crearse y destruirse rápidamente.
- Se comportan como un proceso, así que comparten el kernel.
- Están aislados por defecto del host y de otros contenedores.
- Son muy portables. Se acabó el ‘en mi PC funciona’.

	Máquina Virtual	Contenedor Docker
Tamaño	GB (sistema completo)	MB (solo lo necesario)
Arranque	Minutos	Segundos
Recursos	Alto consumo	Muy eficiente
Aislamiento	Completo (hardware virtual)	A nivel de proceso
Portabilidad	Limitada	Excelente
Rendimiento	Overhead del hypervisor	Casi nativo

Conceptos fundamentales

- **Imagen**: el ‘blueprint’ del contenedor donde se especifica qué ejecuta, dependencias, configuración, etc. Se definen con un Dockerfile.
- **Contenedor**: una imagen en ejecución
- **Dockerhub**: registro público de imágenes Docker preconfiguradas, similar a Github.
  - Imágenes oficiales de Node, Python, MySQL…
- **DockerCompose**: herramienta para definir y ejecutar aplicaciones multi-contenedor



backend > Dockerfile

```dockerfile
# Imagen base de Node.js (versión Alpine p
FROM node:18-alpine

# Crear directorio de trabajo
WORKDIR /app

# Copiar archivos de dependencias primero
COPY package*.json ./

# Instalar dependencias de producción
RUN npm install --omit=dev

# Copiar el código fuente
COPY . .

# El puerto que expondrá el contenedor
EXPOSE 3000

# Usuario no-root por seguridad
RUN addgroup -g 1001 -S nodejs
RUN adduser -S nodeuser -u 1001
USER nodeuser

# Comando para iniciar la aplicación
CMD ["node", "server.js"]
```

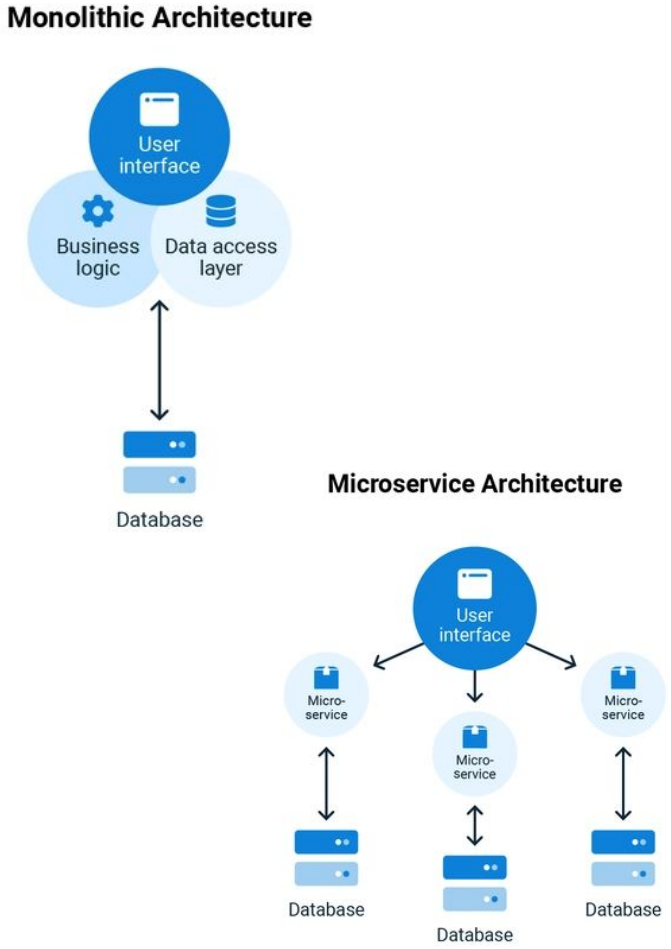
```dockerfile
# Imagen base de Nginx Alpine
FROM nginx:alpine

# Eliminar la configuración por defecto
RUN rm /etc/nginx/conf.d/default.conf

# Copiar nuestra configuración de Nginx
COPY nginx.conf /etc/nginx/conf.d/

# Copiar los archivos del frontend
COPY index.html /usr/share/nginx/html/
COPY styles.css /usr/share/nginx/html/
COPY app.js /usr/share/nginx/html/

# Puerto que expone el contenedor
EXPOSE 80

# Nginx se ejecuta en primer plano
CMD ["nginx", "-g", "daemon off;"]
```

```yaml
services:
  # ============================================
  # BASE DE DATOS - MySQL
  # ============================================
  database:
    image: mysql:8.0
    container_name: tienda-database
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: tienda
      MYSQL_USER: tienda_user
      MYSQL_PASSWORD: tienda_password
    ports:
      - "3306:3306"
    volumes:
      # Script de inicialización
      - ./database/init.sql:/docker-entrypoint-initdb.d/init.sql
      # Persistencia de datos
      - mysql_data:/var/lib/mysql
    networks:
      - app-network
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost", "-u", "root
      interval: 10s
      timeout: 5s
      retries: 5

  # ============================================
  # BACKEND - Node.js + Express
  # ============================================
  backend:
```

```yaml
  # ============================================
  # BACKEND - Node.js + Express
  # ============================================
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: tienda-backend
    restart: unless-stopped
    environment:
      PORT: 3000
      DB_HOST: database
      DB_USER: root
      DB_PASSWORD: rootpassword
      DB_NAME: tienda
    ports:
      - "3000:3000"
    depends_on:
      database:
        condition: service_healthy
    networks:
      - app-network

  # ============================================
  # FRONTEND - Nginx
  # ============================================
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    container_name: tienda-frontend
    restart: unless-stopped
```

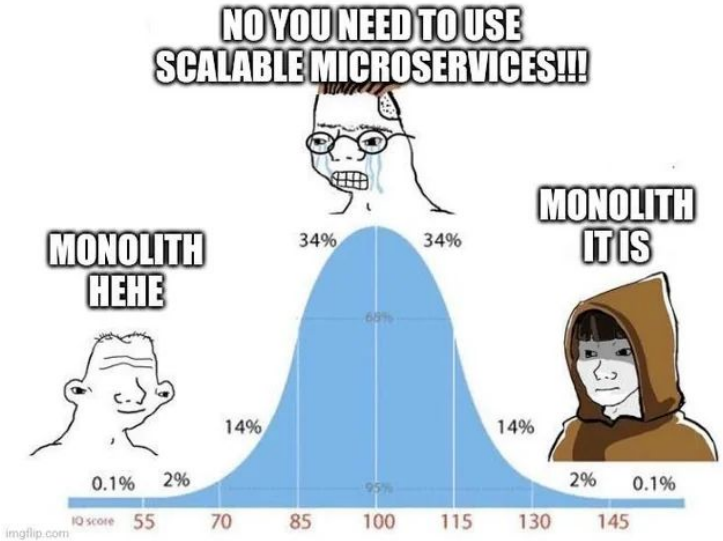
Ventajas

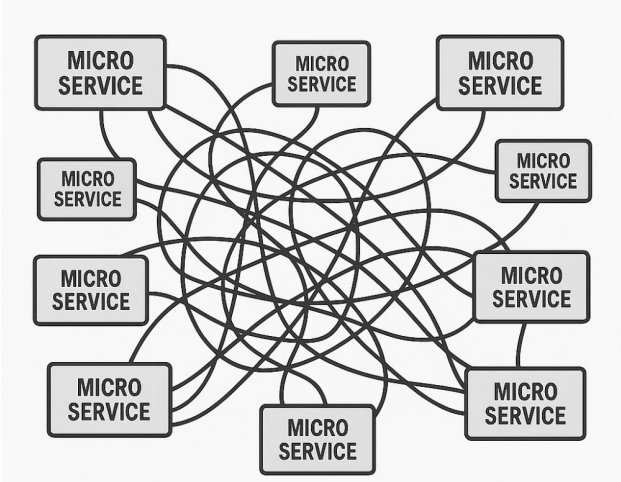
- **Consistencia entre entornos**: El contenedor funciona igual en desarrollo, testing y producción.
- **Aislamiento**: Cada contenedor es independiente. Si uno falla, los demás siguen funcionando.
- **Eficiencia de recursos**: Los contenedores comparten el kernel del SO, consumiendo menos recursos que las VMs.
- **Rapidez**: Arranque en segundos. Despliegues rápidos y rollbacks instantáneos.
- **Escalabilidad**: Fácil replicar contenedores para manejar más carga.
- **Versionado**: Las imágenes se versionan como el código. Puedes volver a versiones anteriores.
- **Ecosistema rico**: Miles de imágenes oficiales disponibles en Docker Hub.
- **Documentación como código**: El Dockerfile documenta exactamente cómo se construye el entorno.



Desventajas

- **Persistencia de datos:** Los contenedores son efímeros. Necesitas configurar volúmenes para datos persistentes.
- **Seguridad:** Comparten el kernel del host. Una vulnerabilidad puede afectar a todos los contenedores.
- **Overhead en Windows/Mac:** Docker necesita una VM Linux subyacente en estos sistemas.
- **No apto para todo:** Aplicaciones con interfaces gráficas o que requieren acceso directo al hardware.
- **Complejidad en orquestación:** Gestionar muchos contenedores requiere herramientas adicionales (Kubernetes).
- **Tamaño de imágenes:** Las imágenes mal optimizadas pueden ocupar mucho espacio.





Gracias!