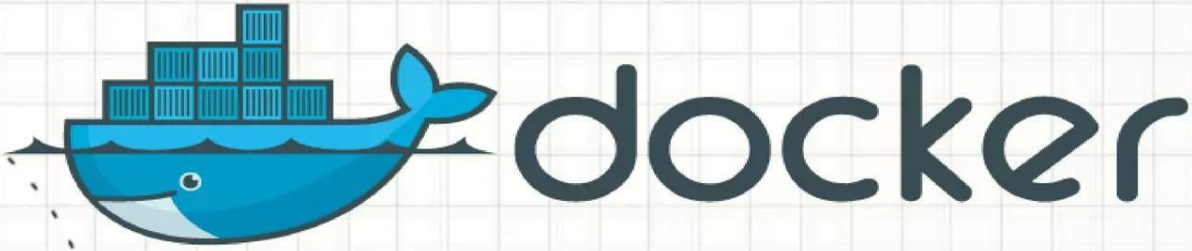
Cikume Software



Tópicos de la Presentación

- Por que Docker?
- Diferencia entre VM y Docker
- Que es Docker?
- Docker Arquitectura
- Docker Línea de comandos
- DockerFile
- Docker-compose
- Comunicación entre contenedores

Por qué Usar Docker

1. **Consistencia en Entornos**: Docker permite crear contenedores que contienen todo lo necesario para ejecutar una aplicación, asegurando que se comporte de la misma manera en cualquier entorno (desarrollo, pruebas, producción).

2. **Aislamiento de Aplicaciones**: Cada contenedor se ejecuta de forma aislada, lo que significa que múltiples aplicaciones pueden coexistir en la misma máquina sin interferir entre sí.

3. **Portabilidad**: Los contenedores Docker se pueden ejecutar en cualquier lugar que soporte Docker, ya sea en tu máquina local, en servidores en la nube o en entornos de producción.

4. **Escalabilidad**: Docker facilita el escalado de aplicaciones, permitiendo crear y destruir contenedores rápidamente para adaptarse a la demanda.

5. **Optimización de Recursos**: Los contenedores son más ligeros que las máquinas virtuales, lo que significa que utilizan menos recursos del sistema y permiten un mejor aprovechamiento del hardware.

6. **Desarrollo Rápido**: Docker acelera el ciclo de desarrollo al permitir a los desarrolladores crear, probar y desplegar aplicaciones rápidamente.

7. **Microservicios**: Facilita la arquitectura de microservicios, donde las aplicaciones se dividen en servicios pequeños y manejables que se pueden desplegar y escalar de forma independiente.

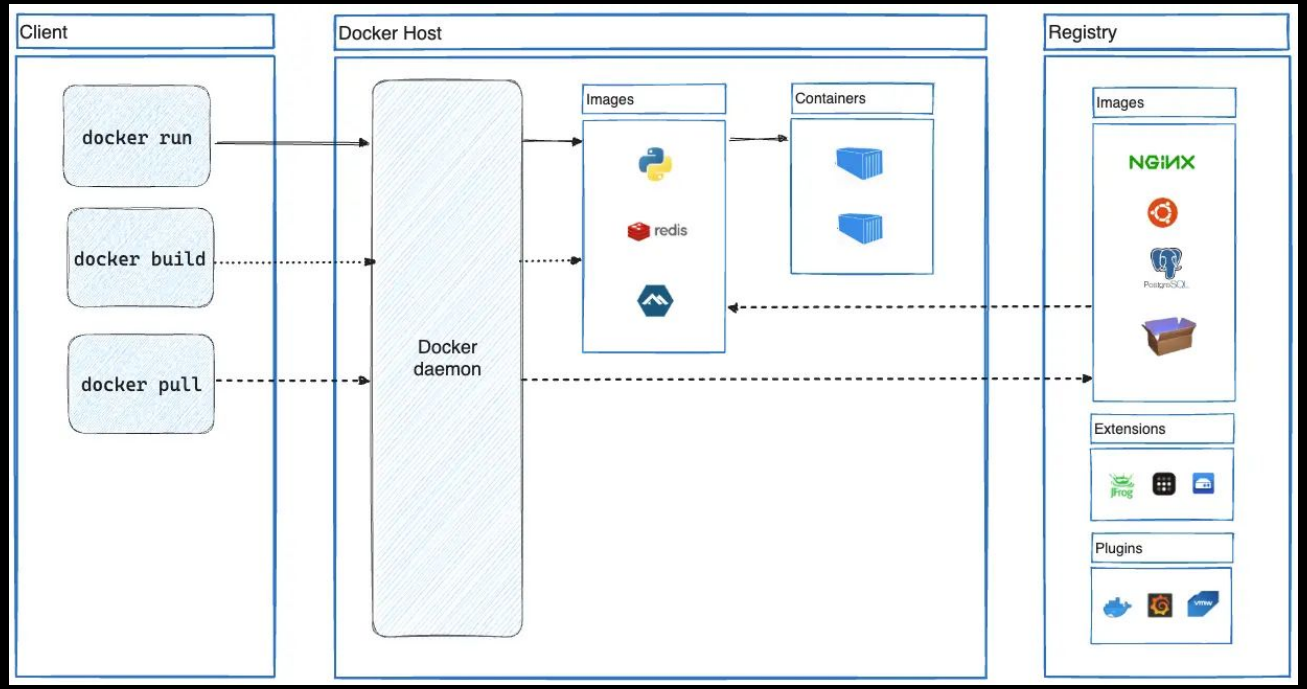
8. **Facilidad de Integración Continua/Despliegue Continuo (CI/CD)**: Docker se integra bien con herramientas de CI/CD, lo que permite automatizar el proceso de construcción, prueba y despliegue de aplicaciones.

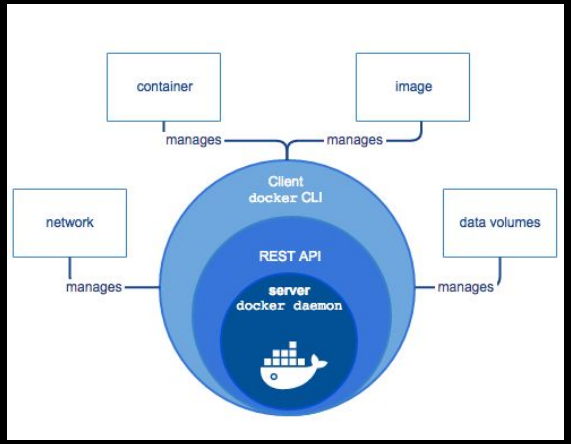
Diferencia entre Docker y VMs

Docker y las máquinas virtuales (VMs) son tecnologías de virtualización que permiten ejecutar aplicaciones en entornos aislados, pero tienen diferencias clave en su arquitectura, rendimiento y usos. Aquí te explico las principales diferencias entre ambas:

Característica	Docker	Máquinas Virtuales (VMs)
Estructura	Contenedor	Sistema operativo completo
Peso	Ligero	Pesado
Arranque	Rápido (segundos)	Lento (minutos)
Rendimiento	Cercano al sistema anfitrión	Menor debido a virtualización
Portabilidad	Alta	Menor
Escalabilidad	Muy fácil y rápido	Más complejo y lento
Aislamiento	Bajo a nivel de procesos	Alto con sistema operativo completo

Docker Arquitectura





Códigos básicos

```
# Imágenes
docker pull <nombre_imagen>                                  # Descargar una imagen
docker images                                                # Listar imágenes locales
docker rmi <nombre_imagen> o <ID_imagen>                     # Eliminar una imagen
docker build -t <nombre_imagen> <ruta_Dockerfile>            # Construir imagen desde Dockerfile

# Contenedores
docker run <nombre_imagen>                                   # Crear y ejecutar un contenedor
docker run -d <nombre_imagen>                                # Ejecutar contenedor en segundo plano
docker run --name <nombre_contenedor> <nombre_imagen>        # Asignar nombre al contenedor
docker run -it <nombre_imagen> /bin/bash                     # Ejecutar contenedor con acceso a la
terminal
docker ps                                                    # Listar contenedores en ejecución
docker ps -a                                                 # Listar todos los contenedores
docker stop <nombre_contenedor> o <ID_contenedor>            # Detener un contenedor
docker start <nombre_contenedor> o <ID_contenedor>           # Iniciar un contenedor detenido
docker rm <nombre_contenedor> o <ID_contenedor>              # Eliminar un contenedor

# Volúmenes
docker volume create <nombre_volumen>                        # Crear un volumen
docker volume ls                                             # Listar volúmenes
docker volume rm <nombre_volumen>                            # Eliminar un volumen

# Redes
docker network create <nombre_red>                           # Crear una red
docker network ls                                            # Listar redes
docker network rm <nombre_red>                               # Eliminar una red
```

DockerFile

```
# Usar una imagen base de Node.js
FROM node:14

# Establecer el directorio de trabajo en el contenedor
WORKDIR /usr/src/app

# Copiar el package.json y package-lock.json
COPY package*.json ./

# Instalar las dependencias
RUN npm install

# Copiar el resto de los archivos de la aplicación
COPY . .

# Exponer el puerto en el que la aplicación escuchará
EXPOSE 8080

# Comando por defecto para ejecutar la aplicación
CMD ["node", "app.js"]
```

Ejemplos Prácticos

Docker-compose, Qué es y Estructura

Docker Compose: es una herramienta que permite definir y ejecutar aplicaciones Docker de múltiples contenedores utilizando un archivo de configuración YAML. Facilita la gestión de aplicaciones complejas que requieren varios servicios (contenedores) que pueden comunicarse entre sí, como bases de datos, servidores web y otros componentes.

Ventajas de Docker Compose

- **Facilidad de uso**: Permite definir toda la configuración de una aplicación en un solo archivo.
- **Gestión de múltiples contenedores**: Facilita el inicio, la parada y la escalabilidad de aplicaciones que constan de varios servicios.
- **Configuración reproducible**: Se puede compartir fácilmente el archivo de configuración, lo que permite que otros usuarios ejecuten la misma aplicación con la misma configuración.

```
version: '3.8'  # Especifica la versión de Docker Compose

services:       # Define los servicios (contenedores) que componen la aplicación
  web:          # Nombre del servicio
    image: nginx:latest  # Imagen de Docker que se utilizará para el servicio web
    ports:       # Mapea los puertos del contenedor a los del host
      - "8080:80"  # Accede al puerto 80 del contenedor a través del puerto 8080 del host

  db:           # Nombre del servicio de la base de datos
    image: mysql:5.7  # Imagen de Docker para la base de datos MySQL
    environment:      # Configura las variables de entorno necesarias para el contenedor
      MYSQL_ROOT_PASSWORD: example  # Contraseña para el usuario root de MySQL
    volumes:        # Monta un volumen para la persistencia de datos
      - db_data:/var/lib/mysql  # Monta el volumen db_data en la ruta de datos de MySQL

volumes:        # Define volúmenes persistentes
  db_data:      # Nombre del volumen para la base de datos
```

Network

Característica	Red `bridge`	Red `external`	Red `host`	Red `none`
Creación	Se crea automáticamente por Docker.	Debe ser creada manualmente por el usuario.	Se crea automáticamente por Docker.	Se crea automáticamente por Docker.
Uso	Aislada a la máquina donde se ejecuta Docker.	Puede ser compartida entre múltiples proyectos o aplicaciones.	No hay aislamiento; comparte la red del host.	No se permite la comunicación.
Configuración	Se define en el archivo `docker-compose.yml`.	Se hace referencia en el archivo `docker-compose.yml` como `external: true`.	No se requiere configuración adicional.	Se utiliza en el archivo `docker-compose.yml` como `network_mode: "none"`.
Comunicación	Los contenedores pueden comunicarse entre sí usando nombres de contenedor.	Los contenedores pueden comunicarse con otros servicios en la misma red.	Los contenedores pueden acceder a la red del host directamente.	No pueden comunicarse con otros contenedores o el host.
Propósito	Ideal para aplicaciones que necesitan comunicarse internamente.	Útil para integrar contenedores con redes existentes.	Para aplicaciones que requieren máximo rendimiento y acceso directo a la red del host.	Para contenedores que no requieren conectividad.
Aislamiento	Proporciona aislamiento entre contenedores.	No proporciona aislamiento entre contenedores y redes externas.	No hay aislamiento; los contenedores comparten la red del host.	No hay conectividad; el contenedor es completamente aislado.

Network avanzado

```yaml
version: '3.8'

services:
  web:
    image: nginx
    networks:
      - my_overlay_network

  api:
    image: myapi
    networks:
      - my_overlay_network

networks:
  my_overlay_network:
    external: true
```

```yaml
version: '3.8'

services:
  web:
    image: nginx
    networks:
      app_net:
        ipv4_address: 172.28.1.2

  api:
    image: myapi
    networks:
      app_net:
        ipv4_address: 172.28.1.3

networks:
  app_net:
    driver: bridge
    ipam:
      config:
        - subnet: 172.28.1.0/24
```

```yaml
version: '3.8'

services:
  web:
    image: nginx
    networks:
      - frontend
      - backend

  api:
    image: myapi
    networks:
      - backend

  db:
    image: postgres
    networks:
      - backend

networks:
  frontend:
    driver: bridge
  backend:
    driver: bridge
```

```yaml
version: '3.8'

services:
  web:
    image: nginx
    networks:
      my_network:
        aliases:
          - frontend

  api:
    image: myapi
    networks:
      my_network:
        aliases:
          - backend

networks:
  my_network:
    driver: bridge
```

Docker build Desde línea de comandos

- `-t my_image:latest`: Asigna una etiqueta (`tag`) a la imagen construida. En este caso, la imagen se llamará `my_image` con la etiqueta `latest`.
- `.`: Indica que el contexto de construcción se encuentra en el directorio actual.

```bash
docker build -t my_image:latest .
```

Este otro ejemplo indica como debe de hacerse cuando se usa un docker registry como dockerhub o alguno externo Azure, AWS, etc

```bash
docker build -t <your_username>/<image_name>:<tag> .
```

Ejemplo cuando se tienen o múltiples dockerfile o no se está usando ese nombre

```bash
docker build -t myusername/myapp:dev --file Dockerfile.dev .
```

Compilacion Usando Docker compose

yaml

```yaml
version: '3.8'
services:
  app:
    build:
      context: .               # El contexto es el directorio actual
    image: myusername/myapp:latest  # Etiqueta para Docker Hub
    ports:
      - "8000:8000"            # Mapea el puerto 8000 del contenedor al puerto 8000
    environment:
      - NODE_ENV=production     # Variable de entorno para el contenedor
```

Para construir y ejecutar el servicio, simplemente usa:

bash

```bash
docker-compose up --build
```

Subida a Docker Hub

Una vez construido, para subir la imagen a Docker Hub, puedes usar:

bash

```bash
docker-compose push
```

yaml

```yaml
version: '3.8'
services:
  app_dev:
    build:
      context: .
      dockerfile: Dockerfile.dev  # Dockerfile para desarrollo
    image: myusername/myapp:dev  # Etiqueta para Docker Hub
    ports:
      - "8000:8000"

  app_prod:
    build:
      context: .
      dockerfile: Dockerfile.prod  # Dockerfile para producción
    image: myusername/myapp:prod  # Etiqueta para Docker Hub
    ports:
      - "8001:8000"  # Mapea a otro puerto
```