CURSO DEVOPS

CODE
SPACE
ACADEMY

Ramón Díaz Fernández

¿Quién soy?

Ramón Díaz Fernández
Ingeniero de Telecomunicaciones
4 años trabajando en el mundo de la informática

LinkedIn: https://www.linkedin.com/in/ramond-fdez
Correo: ramond_fdez@outlook.com

Actualmente trabajo como DevOps en Vodafone Málaga
Y en mi tiempo libre doy clases DevOps en CodeSpace

DevOps

Objetivos del Curso

- Comprender los conceptos fundamentales de DevOps.
- Aprender a utilizar Linux con soltura, scripting, manejo de permisos y procesos.
- Conocer los distintos tipos de Virtualización.
- Aprender a usar herramientas básicas de contenedores como Docker.
- Poder desplegar varios contenedores simultáneos usando Docker Compose.

Objetivos del Curso

- Conocer las distintas herramientas de orquestación de contenedores.
- Reconocer los conceptos básicos de CI/CD.
- Aprender a implementar una pipeline sencilla.
- Saber qué son las soluciones Cloud y conocer las principales opciones.

Requisitos Básicos

Requisitos para el curso:

- Linux / WSL / VM con Linux / Contenedor Docker con Linux
- Docker Desktop o Docker Engine + Docker CLI + Docker Compose
- Cuenta en Dockerhub.com
- Cuenta en Gitlab.com
- Recomendable usar Visual Studio Code
- Inglés muy básico

Material:

- Repositorio del Curso: : https://github.com/ramondfdez/CursoDevOps

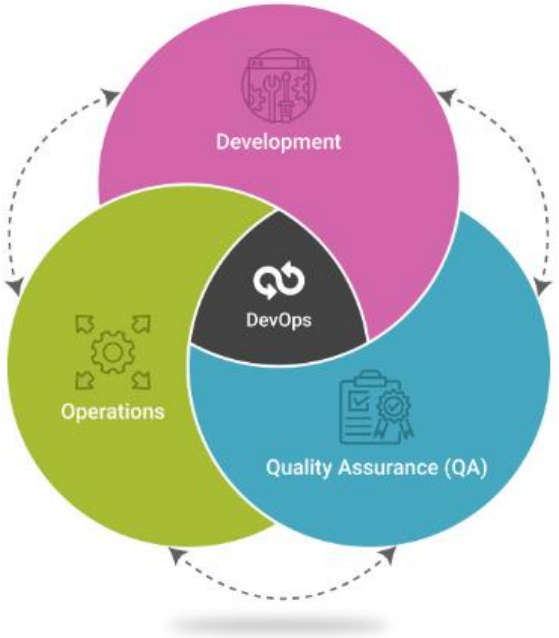
Índice

Tema 1

- ¿Qué es DevOps?
- Ciclo de Vida del Software
- Funciones de un DevOps
- Variaciones de DevOps

Tema 1: ¿Qué es DevOps?

DevOps proviene de la unión de las palabras Development + Operations. El objetivo de la metodología DevOps es hacer más rápido el **ciclo de vida del desarrollo de software** y proporcionar una entrega continua de alta calidad. El término apareció por primera vez en la conferencia Agile 2008 Toronto. *[Fuente: Wikipedia]*



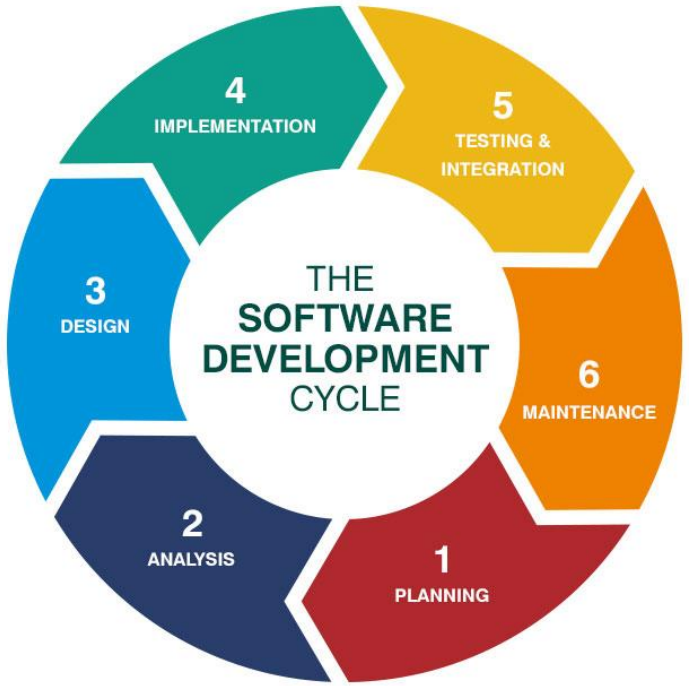
Tema 1: ¿Qué es DevOps?

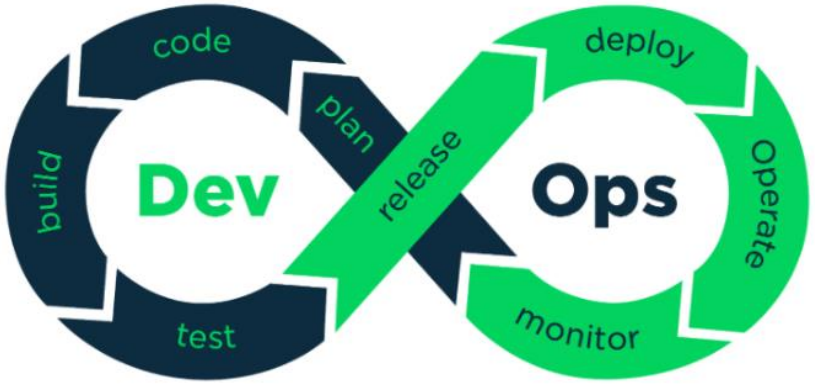
- Desarrollador: Un desarrollador de software es una persona involucrada en el proceso de creación de un software específico, este proceso incluye el estudio, diseño, programación, testeo y otras facetas relacionadas.

- Operaciones: Son las personas que se encargan de asegurar que los dispositivos y sistemas funcionan de la mejor forma posible y en caso contrario hacerlos funcionar de esa manera. Esto incluye soporte de hardware, software y sistemas informáticos internos y externos.

Tema 1: Ciclo de vida del Software

El ciclo de vida de Software corresponde a los pasos que atraviesa el código fuente para llegar a ser desplegado en un ambiente y convertirse en una aplicación. Los DevOps estamos presentes durante todo el proceso.

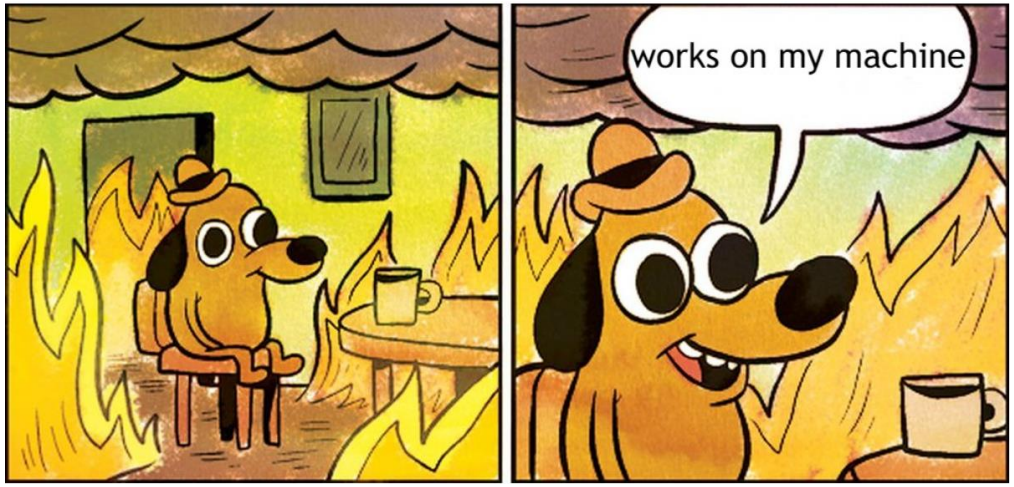


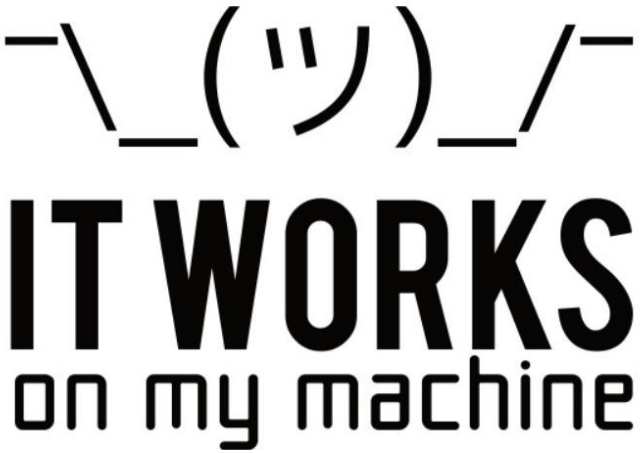


Tema 1: Funciones de un DevOps

Todas las funciones que hemos visto hasta ahora que realizan los desarrolladores y el equipo de operaciones. Pero además…

Evitamos que ocurra esto:

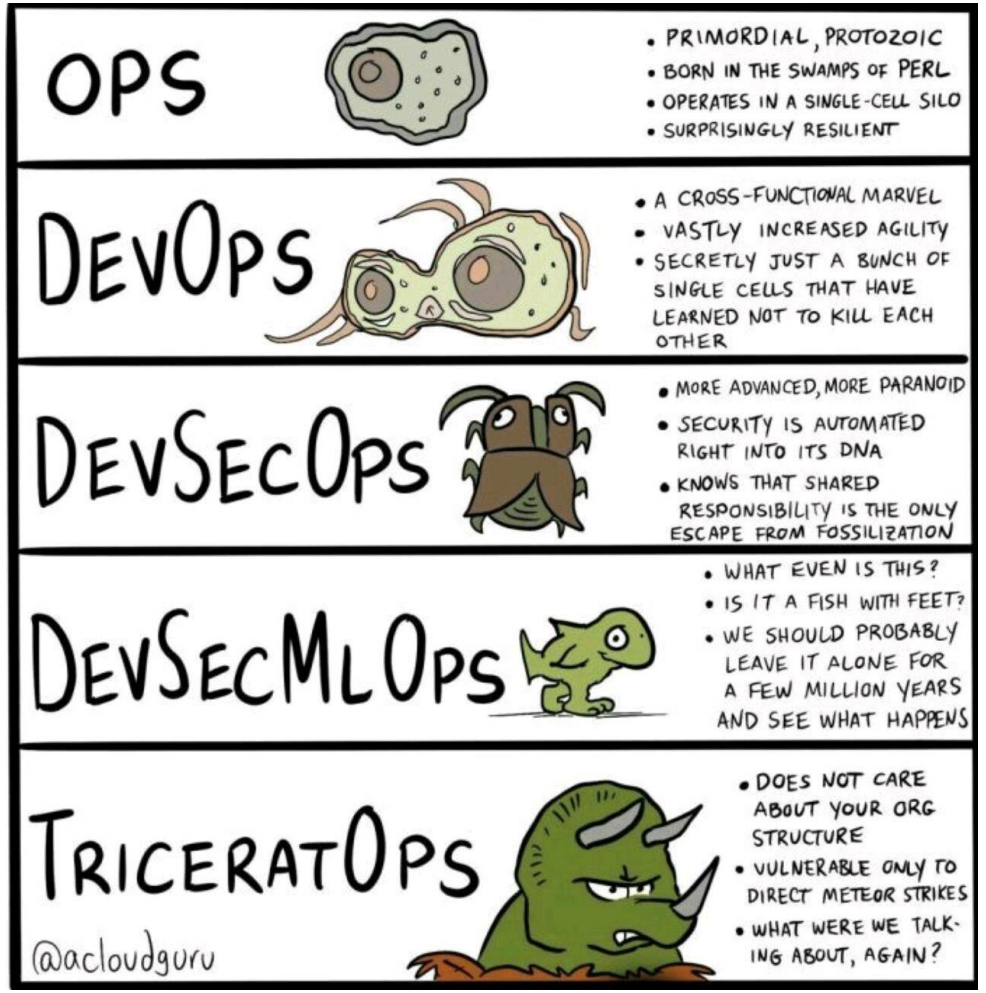




Tema 1: Variaciones de DevOps

Han surgido nuevas variaciones del puesto, en el cuál se enfatiza más en una faceta técnica u otra:

- DevOps
- DevSecOps
- SecOps
- MLOps
- DataOps
- AIOps



Tema 2 : Administración de Sistemas Linux

CODE SPACE ACADEMY

- ¿Qué es?
- Linux (Conceptos Básicos)
- Permisos
- Scripting

Tema 2: ¿Qué es?

Un administrador de sistemas es la persona que tiene la responsabilidad de implementar, configurar, mantener, monitorizar, documentar y asegurar el correcto funcionamiento de un sistema informático, o algún aspecto de este. [Fuente: Wikipedia]

Tema 2: ¿Por qué Linux?

¿Por qué Linux y no Windows para servidores y máquinas?

- Estabilidad
- Mucho más ligero y rápido
- Flexibilidad en cuanto optimización
- Seguridad
- Gratuito y fácil de actualizar

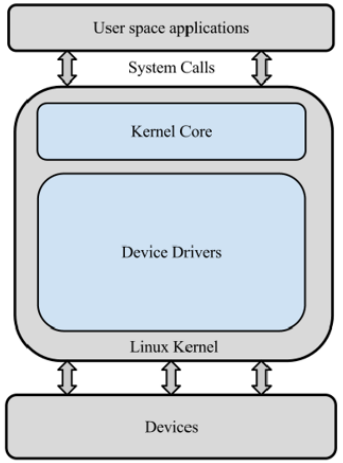
Tema 2: Distribuciones Linux

CODE SPACE ACADEMY

Tema 2: Kernel Linux

El Kernel de Linux es el componente principal del Sistema Operativo Linux y es la interfaz principal entre el hardware del ordenador y los procesos del sistema. También es el encargado de que estos 2 se comuniquen de la manera más optima y eficiente posible. Entre sus funciones principales están:

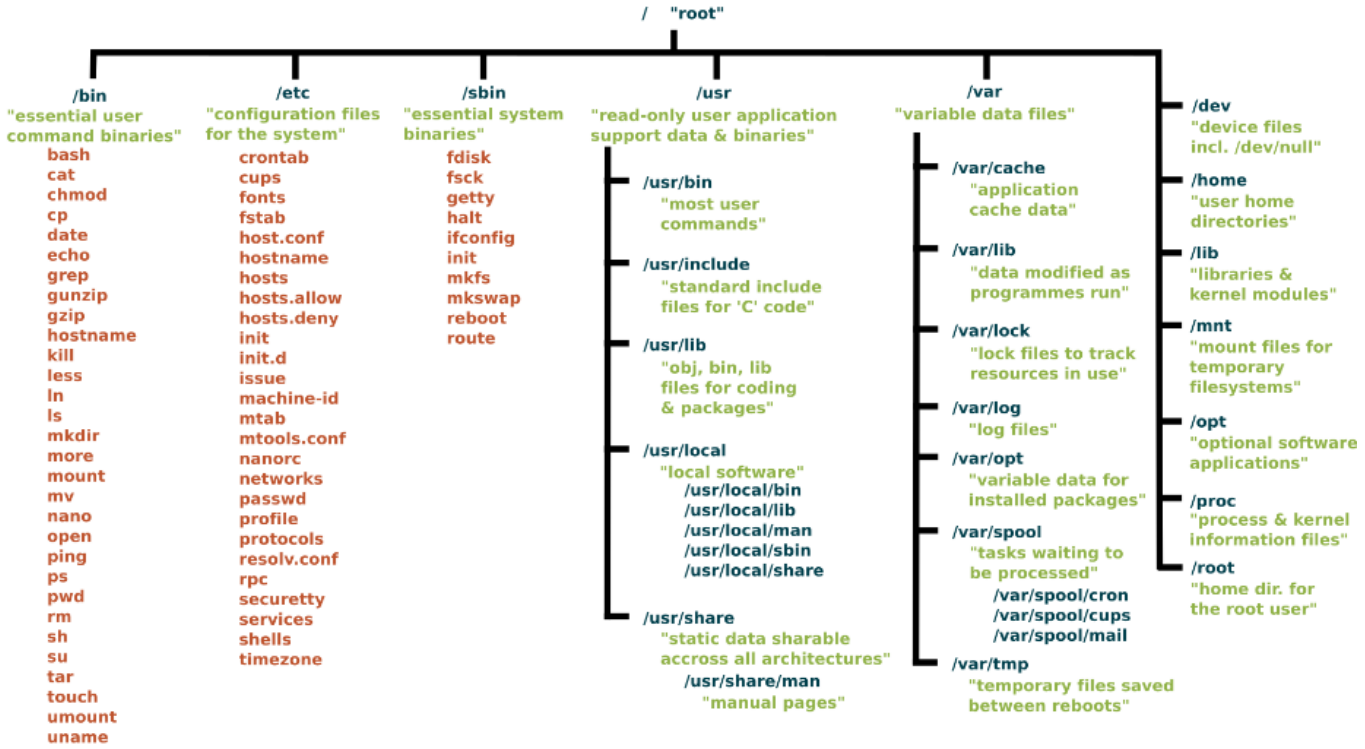
- Administración de memoria: Asigna cuanta memoria es usada para almacenar el qué y dónde.
- Administración de procesos: Determina qué procesos pueden usar la CPU, cuándo y por cuánto tiempo.
- Drivers de dispositivos: Actúa como mediador entre el hardware y procesos.
- Llamadas del sistema: Recibe peticiones de servicio de los procesos.



* Mapa del Kernel de Linux Completo: https://makelinux.github.io/kernel/map

Tema 2: Sistema de Ficheros

En Linux y Unix todo es un fichero. Los directorios son ficheros, los ficheros son ficheros, y los dispositivos son ficheros.
Los sistemas de ficheros de Linux y Unix se organizan en una estructura jerárquica, de tipo árbol. El nivel más alto del sistema de ficheros es / o directorio raíz (root). Todos los demás ficheros y directorios están bajo el directorio raíz, justo debajo de este hay un importante grupo de directorios comunes en la mayoría de distribuciones Linux. Este diagrama es un ejemplo de ello:



Tema 2: Comandos Básicos

CODE SPACE ACADEMY

Comando de sistema	Función
Sudo	Ejecuta como super usuario
Man	Muestra la ayuda del comando (manual)
(comando) | (comando)	Pipe (tubería), redirige la salida al siguiente comando
(comando) && (comando)	Enlaza comandos
> / >> (archivo)	Mete la salida en un archivo
Echo	Imprime por pantalla texto
Clear	Limpia pantalla de comandos anteriores

Atajos teclado	Función
Enter	Ejecuta la línea
Control + Z	Suspende la tarea en ejecución
Control + C	Mata el proceso en ejecución

Tema 2: Comandos Básicos

CODE SPACE ACADEMY

Comando de manejo de Ficheros	Función
Ls	Lista el contenido del directorio
Cd	Accede al directorio
Mkdir	Crea directorio
Touch	Crea archivo
Rm (-r -f)	Elimina archivos (-r para directorios –f forzar)
Cp	Copia archivos y directorios
Mv	Mueve un archivo y directorio
Find	Busca un archivo
Pwd	Muestra la ruta en la que estamos
Grep	Imprime líneas que incluyan un patrón
Cat	Imprime el interior de un archivo
Vi/nano	Abre editor visual / Abre editor nano

CODE SPACE ACADEMY

DEMO

Linux Básico

Probar cada comando de Linux y enseñar el manejo básico de estos

Tema 2: Editores de Linux

- Vim: Versión mejorada de vi el editor visual por defecto de Linux. Es un editor de texto altamente configurable y avanzado creado para permitir una edición de texto eficiente. Admite varios tipos de archivos, por lo que podemos decir que vim es un editor de programadores. Podemos usar complementos vim según nuestros requisitos.

- Nano: Nano es un editor de texto simple y fácil de usar que mejora la experiencia del usuario. Su GUI (interfaz gráfica de usuario) lo hace fácil de usar y permite a los usuarios interactuar directamente con el texto sin cambiar entre los modos. Por lo general, se instala por defecto en el sistema operativo Linux, a diferencia de CentOs y Fedora.

CODE SPACE ACADEMY

DEMO

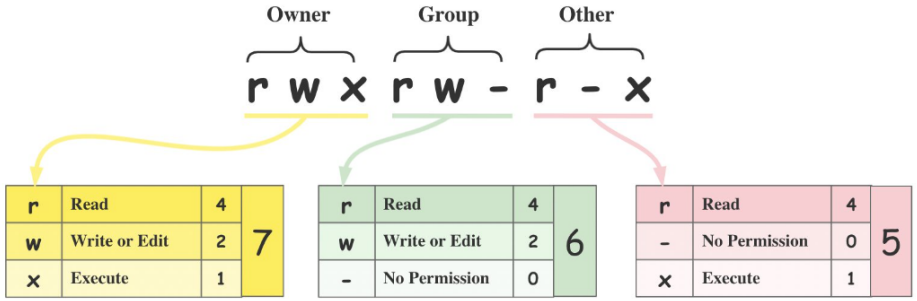
Linux Básico

Usar ambos editores Vim y Nano para editar algún fichero

Tema 2: Permisos

Los permisos son uno de los aspectos más importantes de Linux. Sirven principalmente para proteger el sistema y los archivos de los diferentes usuarios.
Solo el super usuario (root) tiene acciones sin limites en el sistema, justamente por ser el usuario responsable de la configuración, administración y mantenimiento de Linux. Depende de este, por ejemplo, determinar lo que cada usuario puede ejecutar, crear, modificar, etcétera.

Binary	Octal	String Representation	Permissions
000	0 (0+0+0)	---	No Permission
001	1 (0+0+1)	--x	Execute
010	2 (0+2+0)	-w-	Write
011	3 (0+2+1)	-wx	Write + Execute
100	4 (4+0+0)	r--	Read
101	5 (4+0+1)	r-x	Read + Execute
110	6 (4+2+0)	rw-	Read + Write
111	7 (4+2+1)	rwx	Read + Write + Execute



Tema 2: Variaciones de DevOps

Comandos de privilegios	Función
Chmod (+r) (664)	Cambia permisos de archive
Chown	Cambia owner del archive
Chgrp	Cambia permisos de grupo
Adduser	Crea un usuario básico
Deluser	Elimina usuario
Passwd	Asigna contraseña

Comandos comprobar	Función
Cat /etc/passwd | grep (user)	Busca en el fichero de contraseñas el user
Cat /etc/group | grep (user)	Busca el grupo del user

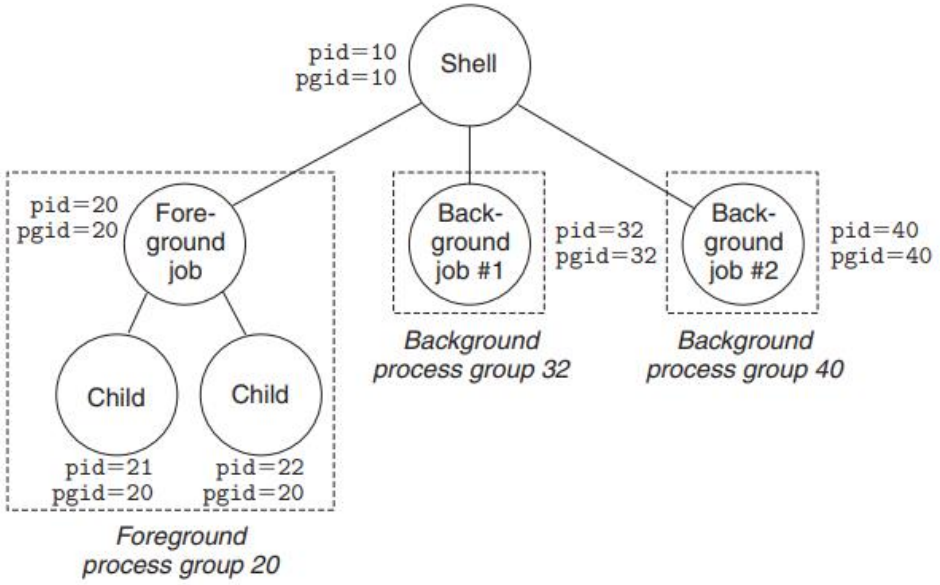
DEMO

Linux Básico

Crear nuevo usuario, cambiar su contraseña y probar sus permisos asignados

Tema 2: Procesos

Un proceso es una instancia de un programa en ejecución. Cada proceso que se inicia es referenciado con su ID de proceso (PID), que es siempre un entero positivo entre 0 y 65535. Prácticamente todo lo que se está ejecutando en el sistema en cualquier momento es un proceso, incluyendo el shell, el ambiente gráfico, el stack de protocolos de la red, etc.
Un proceso puede tener procesos hijos que son creados por él para realizar otras tareas. Los hijos de un proceso comienzan siendo copias exactas del proceso padre. De esta manera el proceso principal (proceso padre), comparte con sus hijos los recursos del sistema que este consumiendo, sus atributos de seguridad (tales como su propietario y permisos de archivos así como roles y semaforización.



Tema 2: Procesos

Comando de manejo de Ficheros	Función
Ps	Lista de procesos
Kill	Manda una señal al proceso
Nice	Cambia prioridad de un proceso

Nombre	ID	Descripción
SIGINT	2	Para el proceso y desaparece. También se envía con Control + C.
SIGKILL	9	Mata un proceso incondicionalmente y en el acto.
SIGTERM	15	Termina o mata un proceso de forma controlada.
SIGCONT	18	Continuar. Cuando un proceso detenido recibe esta señal continua su ejecución.
SIGSTOP	19	Para el proceso pero se queda preparado para continuar. También se envía con Control + Z.

DEMO

Linux Básico

Revisar los procesos del sistema y mandar alguna señal a algún proceso. (usamos sleep)

Tema 2: Scripting

Un script es una secuencia de comandos que se utiliza para manipular, personalizar y automatizar las acciones de un sistema existente. Normalmente, almacenadas en un archivo de texto que deben ser interpretados línea a línea en tiempo real para su ejecución, se distinguen de los programas, pues deben ser convertidos a un archivo binario ejecutable para que funcionen.

Si creamos un archivo con #! /bin/bash al comienzo y lo ejecutamos con ./ Linux lo tratará como un script del sistema.

EJERCICIO

Scripts

Crea un script llamado “script.sh” que cree una carpeta con nombre “carpeta”, cree un archivo de texto dentro de esta con nombre “notas.txt” y escriba en su interior vuestro nombre y apellidos.

EJERCICIO

Scripts

1. Ejecutamos script anterior.
2. Cambiamos permisos a 666. Y volvemos a ejecutar ¿Qué ha ocurrido?
3. Cambiamos con Vim o Nano su interior. (Por ejemplo, ponemos nuestra comida favorita)
4. Cambiamos permisos a 555. Volvemos a editar ¿Qué ha ocurrido?
5. Cambiamos a 111 y hacemos cat. Qué ha ocurrido?

EJERCICIO

Scripts

Creamos un archivo passwords.txt con contraseñas inventadas (lo editamos con vim). Creamos un script llamado script2.sh que cree una carpeta Secret y copie el contenido de passwords.txt a un archivo contra.txt dentro de Secret y por último que cambie a 555 los permisos de contra.txt. (Estamos creando una copia de seguridad de un archivo en una carpeta sin permisos de escritura, para que nadie pueda modificarla)

Tema 2: Networking con Linux

Linux también nos permite comprobar nuestra Ip y configuración de red, a parte, nos permite refinar y encontrar soluciones cuando hay problemas de conexión en nuestra red.

Comando	Función
Ping	Envía paquetes ping
Curl	Obtiene la información de la url
Nslookup	Comprueba la DNS de una IP y viceversa
Traceroute/tracepath	Describe el camino de los paquetes
Ifconfig	Muestra tu IP y más detalles
Hostname	Muestra nombre de máquina (DNS)

DEMO 4

Linux Básico

Revisar los comandos de redes.

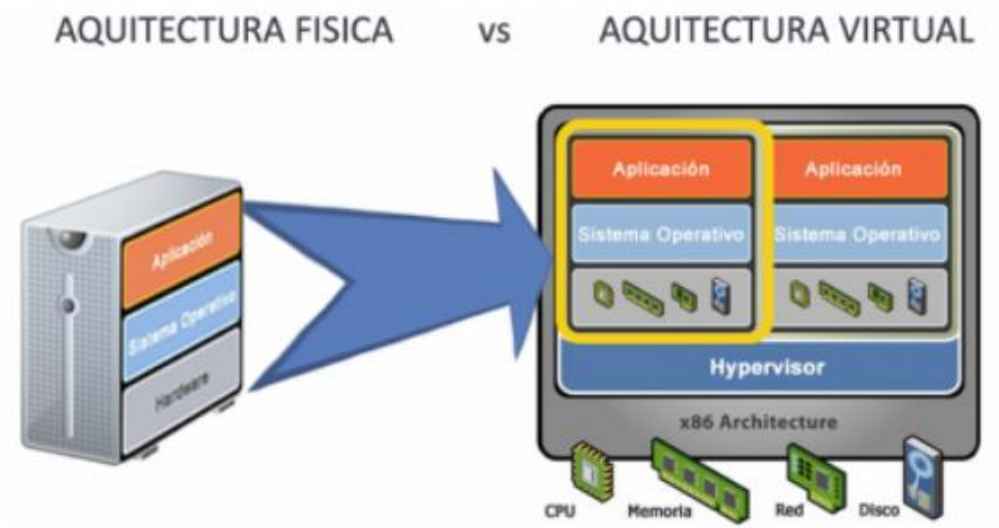
Tema 3 : Docker

CODE SPACE ACADEMY

- Virtualización
- Arquitectura de Microservicios
- Docker Arquitectura
- DockerFiles, Imágenes y Contenedores
- Repositorios y Registros
- Volúmenes

Tema 3: Virtualización

La virtualización es una tecnología que se puede usar para crear representaciones virtuales de servidores, almacenamiento, redes y otras máquinas físicas. El software virtual imita las funciones del hardware físico para ejecutar varias máquinas virtuales a la vez en una única máquina física.
[Fuente: Amazon]

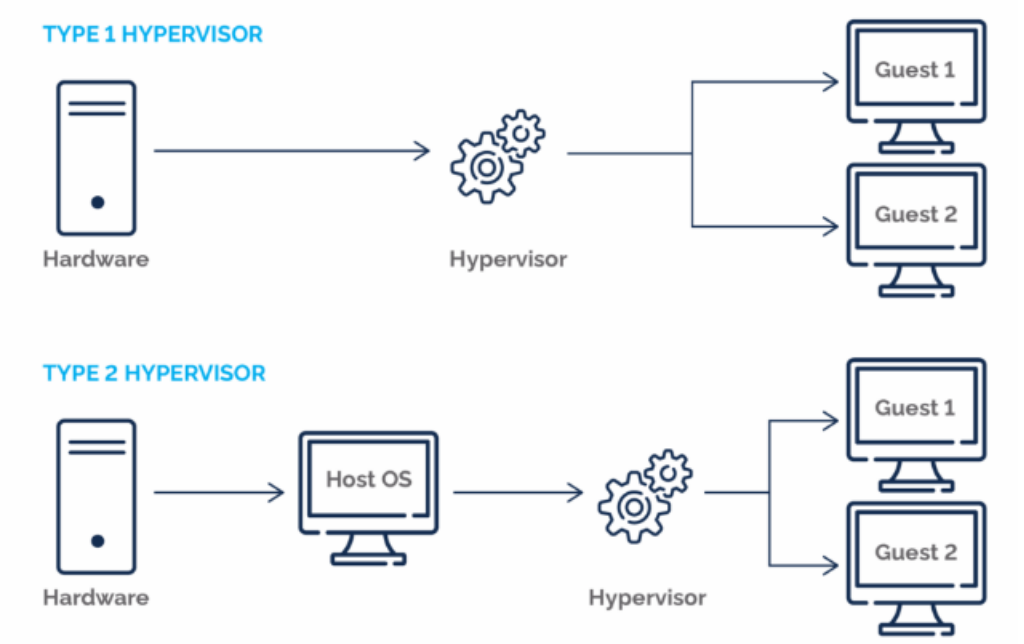


Tema 3: Tipos de Virtualización

- Tipo1: Un hipervisor simple (Tipo 1) es una capa de software que instalamos directamente sobre un servidor físico y su hardware subyacente.
- Tipo 2: Es por esto que llamamos hipervisores alojados de hipervisores de tipo 2. A diferencia de los hipervisores de tipo 1 que se ejecutan directamente en el hardware, los hipervisores alojados tienen una capa de software debajo

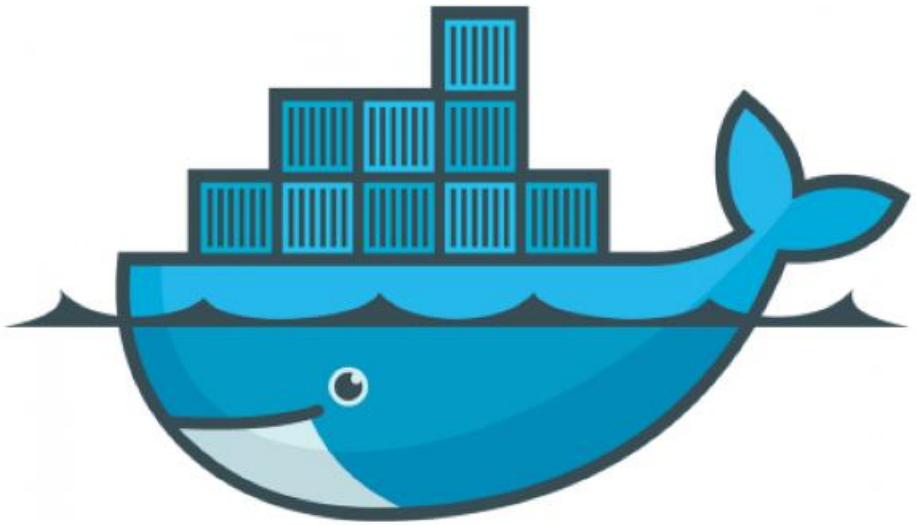
Microsoft Hyper-V

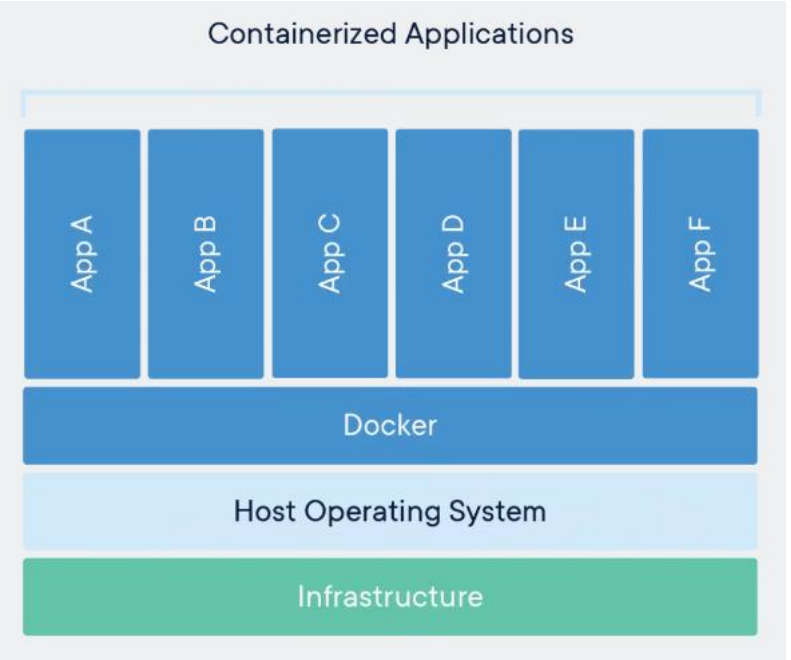
VirtualBox



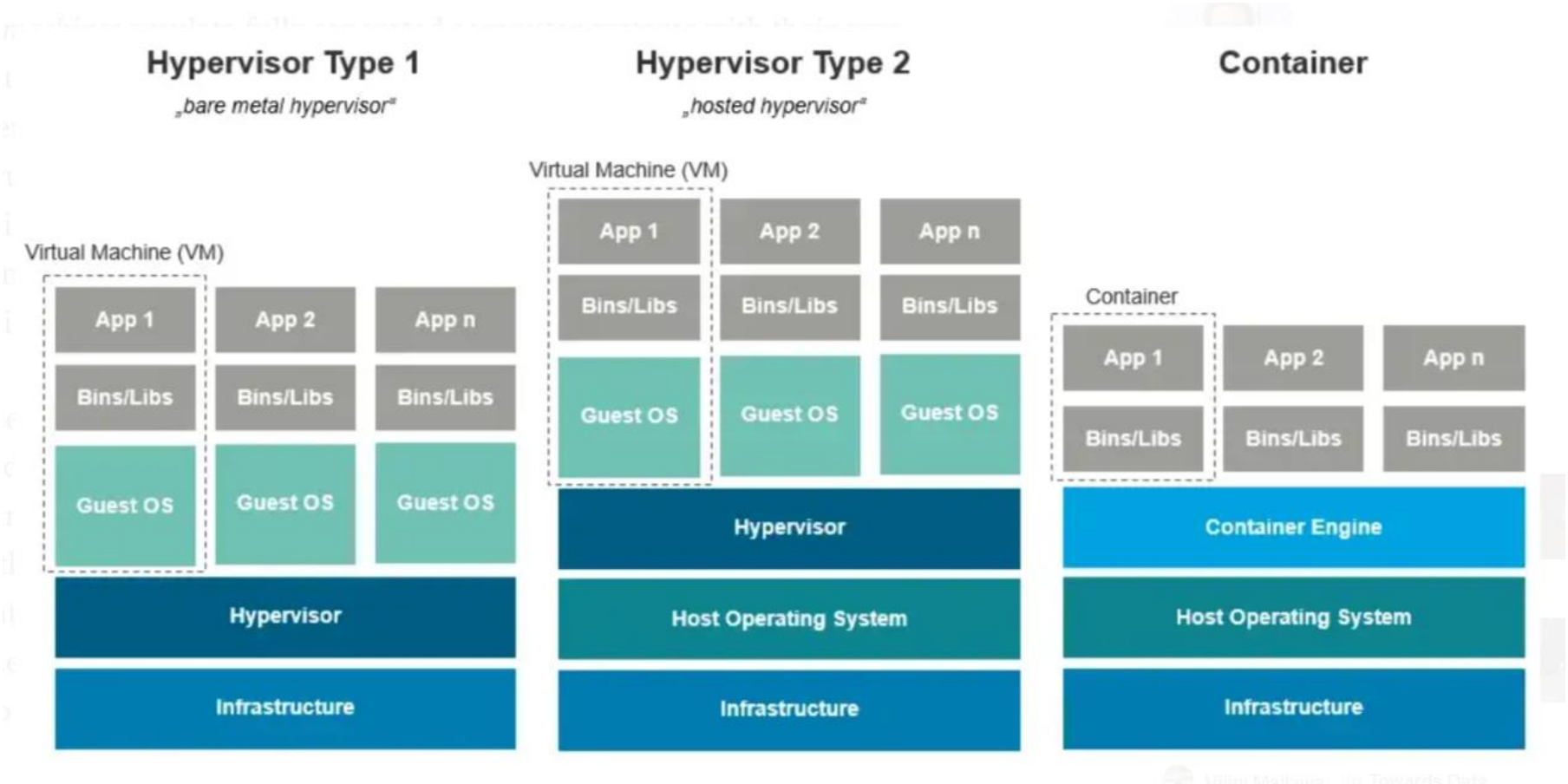
Tema 3: Docker

Docker es una plataforma de software que le permite crear, probar e implementar aplicaciones rápidamente. También proporciona la capacidad de empaquetar y ejecutar una aplicación en un entorno poco aislado llamado contenedor. El aislamiento y la seguridad permiten ejecutar muchos contenedores simultáneamente en un host determinado. Los contenedores son ligeros porque no necesitan la carga adicional de un hipervisor, sino que se ejecutan directamente dentro del kernel de la máquina.



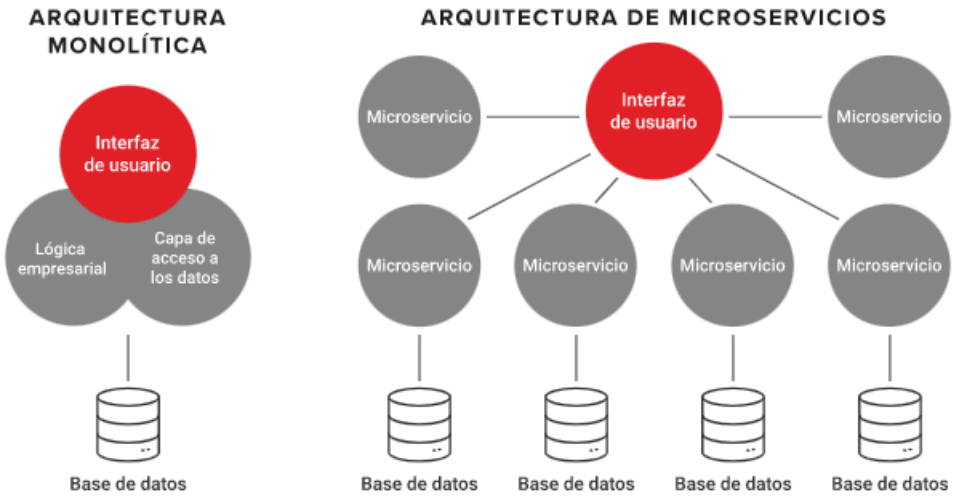


Tema 3: Hipervisores vs Contenedores



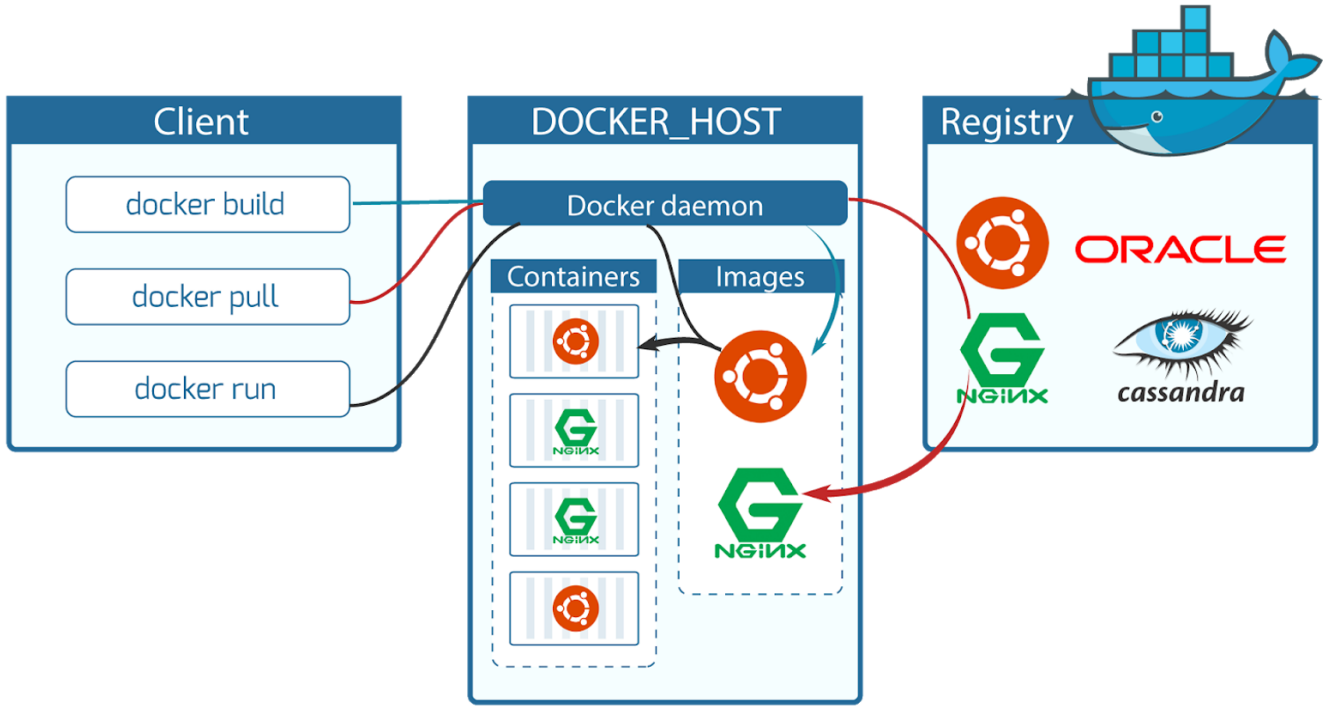
Tema 3: Arquitectura de Microservicios

- Arquitectura monolítica: Las aplicaciones y servicios están centralizados, están en una sola máquina o en unas pocas.
- Arquitectura de microservicios: Cada aplicación y servicio se encuentran descentralizadas, máquina o instancia por microservicio.



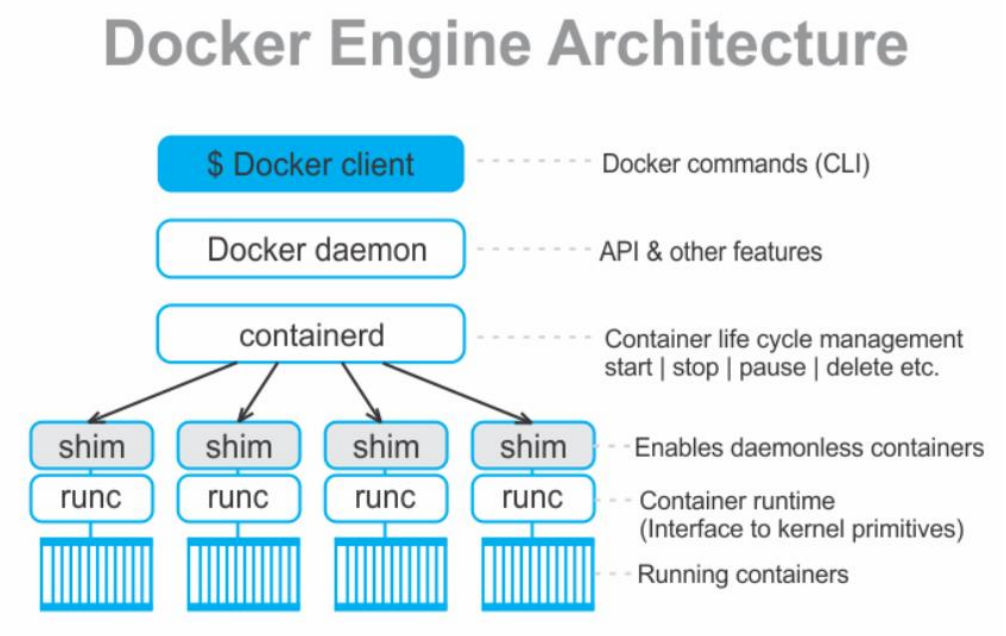
Tema 3: Arquitectura de Docker

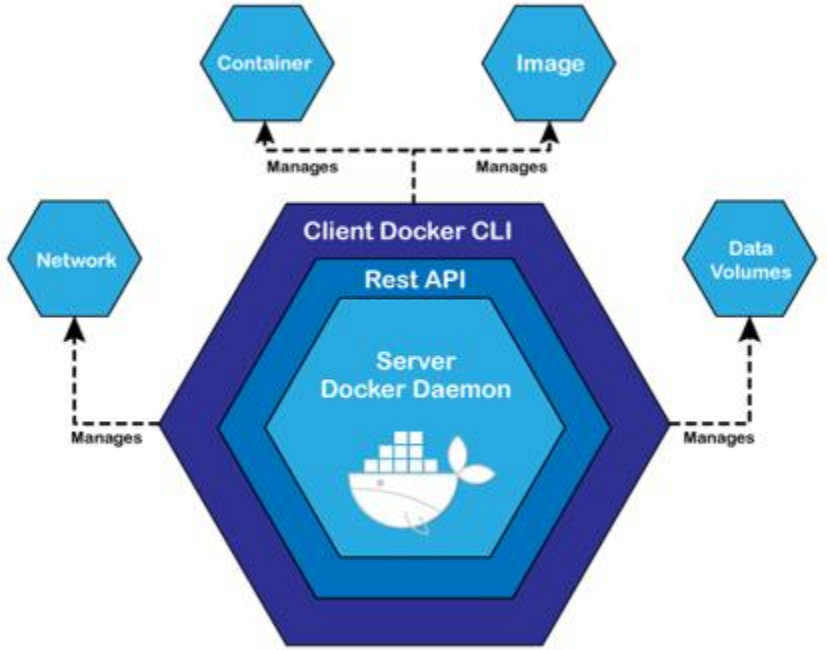
Docker usa una arquitectura cliente-servidor. El cliente Docker habla con el demonio (daemon) Docker, que hace el trabajo pesado de construir, ejecutar y distribuir sus contenedores Docker. El cliente Docker y el demonio pueden ejecutarse en el mismo sistema, o se puede conectar un cliente Docker a un demonio Docker remoto. El cliente Docker y el demonio se comunican utilizando una API REST, sobre sockets UNIX o una interfaz de red.



Tema 3: Docker Engine

El Docker Engine como su propio nombre indica, es el motor de Docker. Crea, encapsula y ejecuta los contenedores Docker.

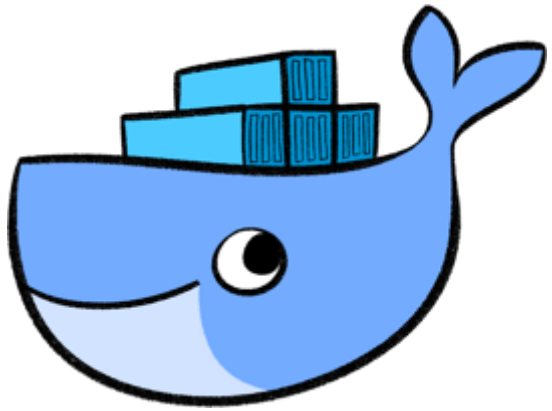




Tema 3: Docker Engine

- El Docker daemon (dockerd) escucha las solicitudes de la API de Docker y gestiona los objetos de Docker, como imágenes, contenedores, redes y volúmenes. Un daemon también puede comunicarse con otros daemons para administrar los servicios de Docker.

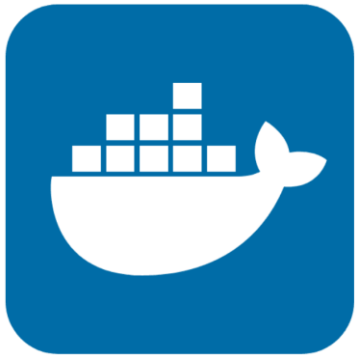
- El cliente Docker (docker) es la forma principal en que muchos usuarios de Docker interactúan con Docker. Cuando se utilizan comandos como docker run, el cliente envía estos comandos a dockerd, que los lleva a cabo. El comando docker usa la API Docker. El cliente de Docker puede comunicarse con más de un daemon.



Tema 3: Docker Desktop

Docker Desktop es una aplicación disponible para Mac, Linux y Windows que permite construir, compartir contenedores e imágenes de aplicaciones y microservicios.
Provee una interfaz de usuario gráfica (GUI) que permite administrar contenedores, aplicaciones e imágenes de forma sencilla e intuitiva directamente desde tu ordenador. Docker Desktop incluye:

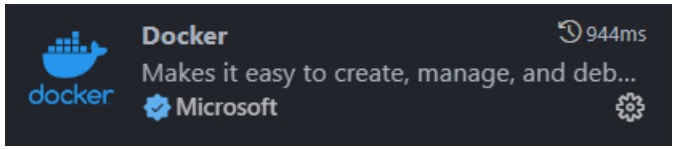
- Docker Engine
- Docker CLI
- Docker Compose
- Kubernetes
- Extensiones útiles

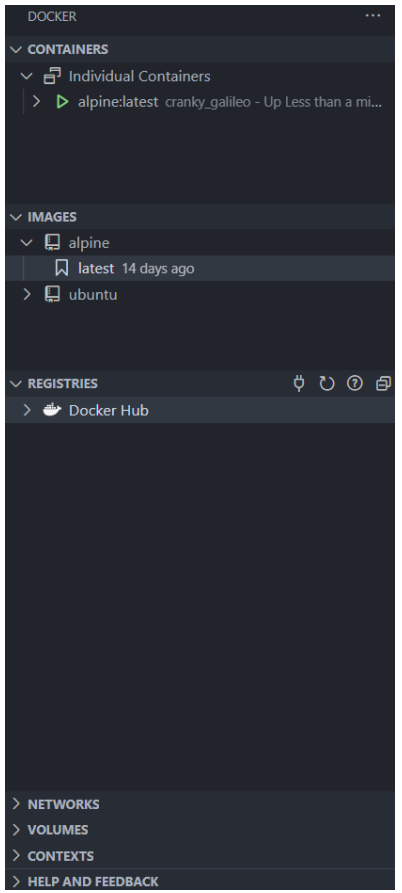


* Probemos si funciona con un Hello World: **docker run -d -p 80:80 docker/getting-started**

Tema 3: Extensión Docker VSC

Nos permite de manera muy sencilla y gráfica crear imágenes y ejecutar contenedores con apenas un par de clicks sin tener que memorizar los comandos Docker.

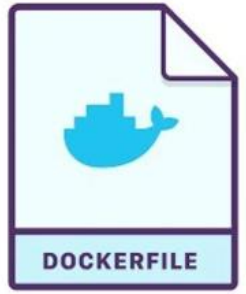




Tema 3: Dockerfile

Un Dockerfile es un archivo o documento de texto simple que incluye una serie de instrucciones que se necesitan ejecutar de manera consecutiva para cumplir con los procesos necesarios para la creación de una nueva imagen.

Instrucción	Explicación
FROM <imagen>	Imagen base usada
COPY <SRC> <DST>	Copia el archivo a la ruta del contenedor
RUN <comando>	Ejecuta un comando en el contenedor
CMD <proceso>	Es el comando por defecto que va a ejecutar nuestro contenedor
ENV <variable> <valor>	Especifica una variable de entorno
USER <usuario>	Usuario usado para ejecutar todo
WORKDIR <carpeta>	Carpeta de trabajo (igual a cd <carpeta>)
EXPOSE <PORT>	Expone el puerto en el contenedor



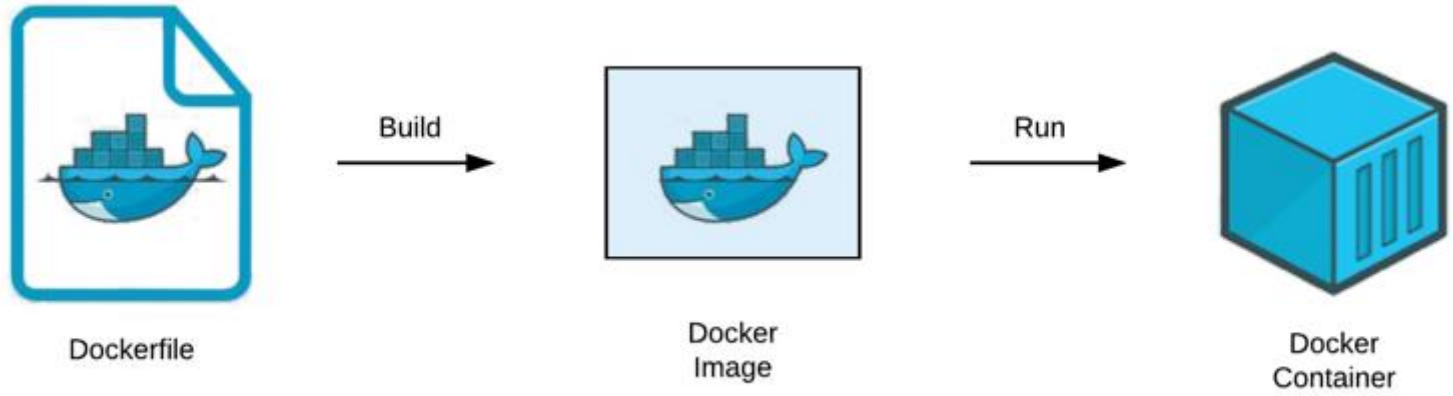
Ejemplo Dockerfile

```
FROM debian
RUN apt-get update && apt-get install -y apache2 && apt-get
clean && rm -rf /var/lib/apt/lists/*
COPY index.html /var/www/html/
EXPOSE 80
CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

Tema 3: Imágenes y Contenedores

- Imagen: Es la base usada para inicializar un contenedor Docker puede ser creada a partir de un archivo Dockerfile.

- Contenedor: Unidad estándar en la que se aloja el servicio de una aplicación.



Tema 3: Comandos Docker

CODE SPACE ACADEMY

Comando	Explicación
Docker pull <imagen>	Trae la imagen de un registro
Docker images	Lista las imégenes locales
Docker inspect <imagen/contenedor>	Da información adicional
Docker login	Logea en un registro privado con docker
Docker run <imagen>	Inicia un contenedor
Docker ps (-a)	Lista contenedores en ejecución
Docker exec <contenedor> <comando>	Ejecuta un commando en un contenedor
Docker tag <imagen> <tag>	Etiqueta una imagen
Docker logs <contenedor>	Muestra logs del contenedor
Docker push <imagen>	Sube imagen a un registro
Docker build .	Crea una imagen a partir de un Dockerfile

Tema 3: Comandos Docker

CODE SPACE ACADEMY

Comando	Explicación
-d	Ejecuta en segundo plano
-e	Añade variables de entorno adicionales
-i / -it	Ejecuta en modo interactivo
-l	Añade una etiqueta de metadata
--name	Asigna un nombre al contenedor
--rm	Elimina el contenedor tras acabar
--restart no	Política de reinicio
-p <local>:<contenedor>	Realiza el mapeado de puertos
-u	Usuario con el que se ejecuta el contenedor
-v / --mount	Monta un volumen en el contenedor

Tema 3: Creación Imágenes

DEMO

Docker

Probamos algunos comandos docker de la tabla

Tema 3: Creación Imágenes

DEMO

Docker

1. Creamos archivo Dockerfile con las instrucciones deseadas:

```
FROM alpine
CMD echo "Hello, World!"
```

2. Construimos la imagen con un nombre asociado (tag):

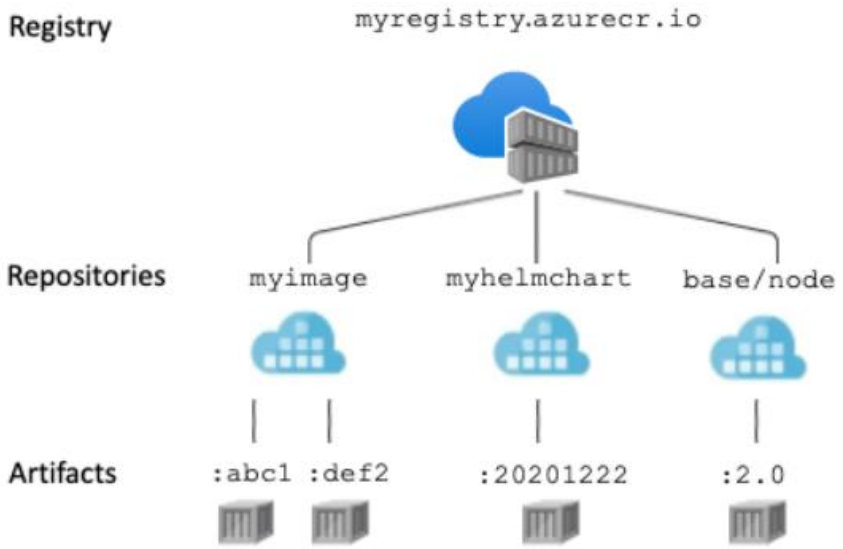
```
docker build -f Demo1.Dockerfile -t mi-imagen:latest .
```

3. Ejecutamos la imagen creada:

```
docker run --rm mi-imagen:latest    (-it)
```

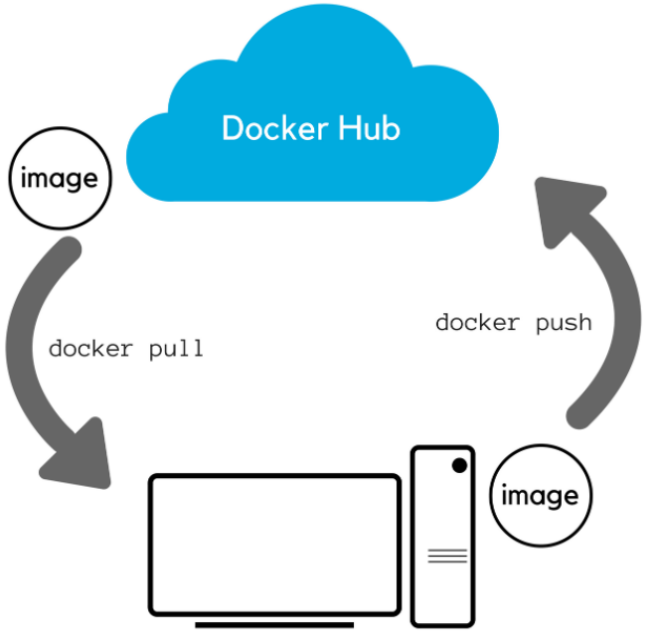
Tema 3: Registros Repositorios y Artefacto

- Registro: Es un servicio que almacena y distribuye no solo imágenes de contenedores, sino también los artefactos relacionados.
- Repositorio: Es una colección de imágenes u otros artefactos de un registro que tienen el mismo nombre pero etiquetas diferentes (tag).
- Artefacto: una imagen de contenedor u otro artefacto dentro de un registro está asociado con una o varias etiquetas, tiene una o más capas y se identifica mediante un manifiesto.



Tema 3: DockerHub

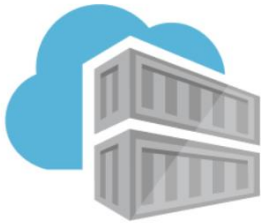
Usaremos DockerHub en este curso como registro de imágenes: https://hub.docker.com



*Nos crearemos una cuenta todos.

MÓD. 3 - Tema 1: Registros Privados

ECR

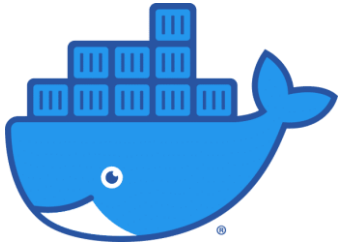


ACR

Gitlab

Nexus

GCR



Docker Hub

GitHub

JFrog

Harbor

Tema 3: Imágenes útiles

Imágenes docker más descargadas:

Tags más usadas (Sistemas operativos base):

- Alpine: Basada en una imagen linux Alpine (5mb) especialmente diseñada para contenedores, solo se instala los requerimientos mínimos.
- Slim: Es una version Debian base en el que solo se instala los requerimientos mínimos (20mb)
- Bullseye: Basada en una imagen linux Debian:Bullseye (50mb)
- Buster: Basada en una imagen linux Debian:Buster (50mb)

DEMO

DockerHub

Nos iremos a DockerHub y descargaremos algunos de las imágenes más descargadas, las probaremos y las eliminaremos tras ello

DEMO

DockerHub

Nos descargamos imágenes con diferentes tags y las comprobamos, miramos su interior a ver si tienen lo mismo

CODE SPACE ACADEMY

DEMO

DockerHub

Usaremos la imagen creada anteriormente para etiquetarla correctamente y la subiremos a nuestro repositorio de DockerHub

Ejercicio

DockerHub

Probaremos a crear un repositorio privado con otro nombre e intentaremos subir nuestra imagen al nuevo repositorio

DEMO

Registry local

Usaremos la imagen de registry para crearnos un repositorio local, así comprobaremos y veremos la diferencia entre traer imágenes desde dockerhub y un registro local:

docker run -d -p 5000:5000 --restart always --name registry registry:latest

EJERCICIO

Docker

Ciclo completo: Nos crearemos a partir de un Dockerfile una imagen Docker Ubuntu que contenga un archivo de texto con vuestro nombre. Probaremos que funciona ejecutando el contenedor de forma interactiva o ejecutando comandos. Tras esto etiquetamos correctamente la imagen y la subimos al repositorio de DockerHub. Para acabar eliminamos la imagen, nos la volveremos a traer a nuestro repositorio local y volveremos a ejecutar el contenedor.

Demo

Docker, contenedores en paralelo

Creamos un primer contenedor con imagen nginx (nginx es un servidor web de webs estáticas)

docker run -p 80:80 -d nginx

También creamos otro contenedor esta vez de alpine que devuelva la fecha

docker run -d alpine sh -c "while true; do date; sleep 1; done“

Podemos ver que ambos contenedores pueden coexistir sin problemas

docker logs -f

Demo

Docker, contenedores comunicantes (herramienta de debug de redes)

Creamos un primer contenedor con imagen nginx en el puerto 80 (usaremos –name para que sea más fácil llamarlo desde el otro contenedor).

docker run --name nginx-container -p 80:80 -d nginx

Creamos un segundo contenedor que se dedique solo a hacer ping al primero. Y miramos sus logs. (usaremos el flag --link para enlazar dos contenedores)

docker run --name ping-container --link nginx-container -d alpine ping nginx-container

CODE SPACE ACADEMY

EJERCICIO

Docker

Hacer una web personalizada con un servidor apache dentro de un contenedor docker.
1- Máquina Linux e instalar apache
2- Usando imagen apache (httpd)

DEMO

Docker base de datos

Base de datos MySQL:
docker pull mysql:8.0
docker run --name mysql80 -e MYSQL_ROOT_PASSWORD=12345678 -d -p 3306:3306 mysql:8.0
docker exec -it mysql80 mysql -u root -p

CODE SPACE ACADEMY

DEMO

Base de datos MongoDB

```
docker pull mongo:latest
docker run -d -p 27017:27017 --name test-mongo mongo:latest
docker exec -it test-mongo mongosh
```

EJERCICIO

Docker

Crear contenedor docker que ejecute un script.sh que imprima la fecha (date)

EJERCICIO

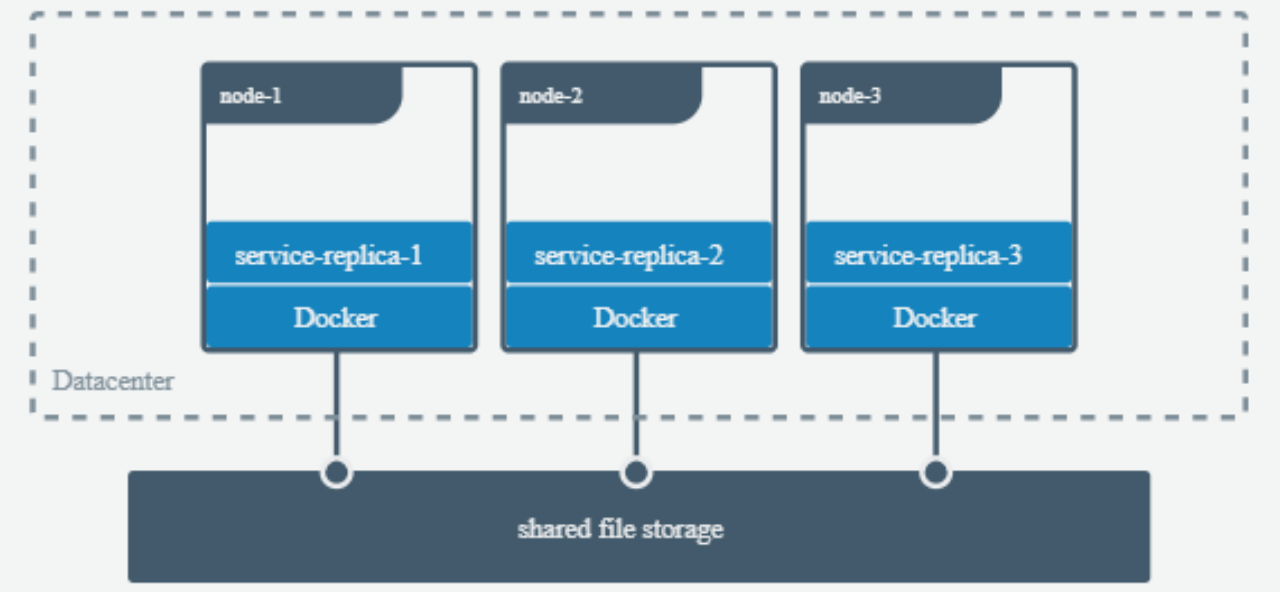
Docker

Replicamos el script que creamos en la primera clase (Creamos un archivo passwords.txt con contraseñas inventadas. Creamos un script llamado script2.sh que cree una carpeta Secret y copie el contenido de passwords.txt a un archivo contra.txt dentro de Secret y por último que cambie a 555 los permisos de contra.txt) y comprobamos con comandos docker o ejecutando el contenedor de manera interactive si realmente funciona.

(Usamos RUN, si usamos CMD va a terminar el contenedor tras la ejecución)

MÓD. 3 - Tema 1: Volúmenes

Volumen: Almacenamiento persistente, incluso si el contendor está parado, los datos siguen almacenados y pueden ser usados por otros contenedores



docker volume create my-vol	**docker run –v <ruta local>:<ruta contenedor>**

DEMO

Volumen

Creamos volumen:
docker volume create mi_volumen

Ejecutamos ambos contenedores 4 y 4b con el volumen como parámetro e interactivo para que no se cierren:
docker run -it -v /mi_volumen:/app ejercicio4:v1

Creamos y borramos archivos en uno y comprobamos en el otro

DEMO

Volumen

Crea una imagen de Apache

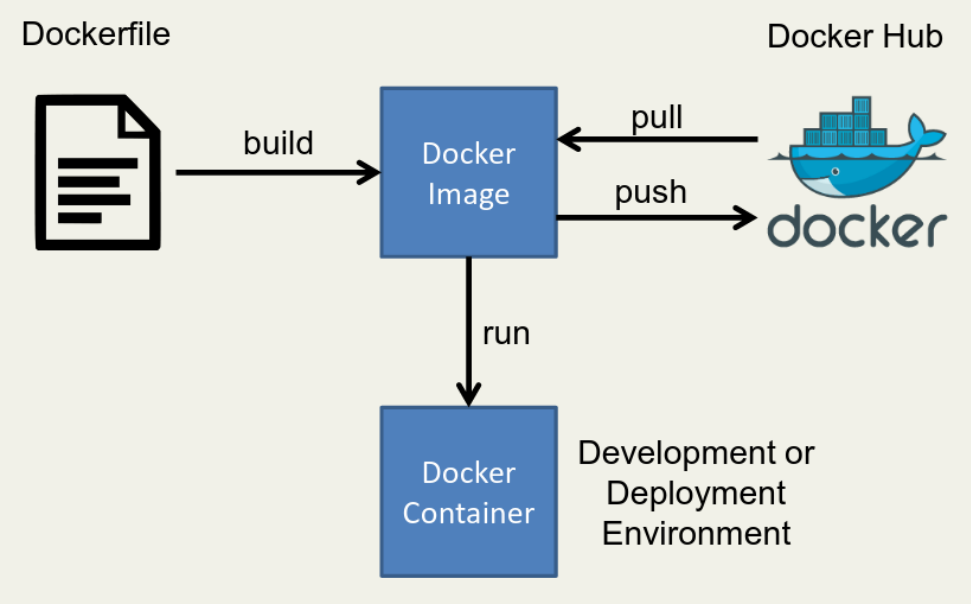
Ejecuta un contenedor a partir de la imagen y monta un volumen en el directorio /var/www/html en el host para que puedas acceder a los archivos HTML desde tu sistema.

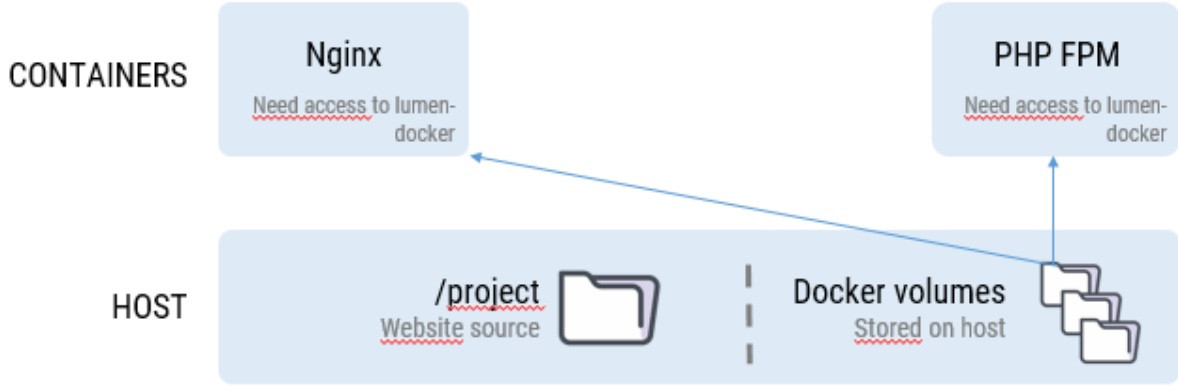
Verifica que la página Apache se está ejecutando correctamente en el contenedor visitando la dirección http://localhost en tu navegador.

Modifica el archivo HTML en el sistema host y verifica que los cambios se reflejan en la página Apache en el contenedor.

```
docker run -it --rm -p 80:80 -v ~/html:/usr/local/apache2/htdocs/
```

Diapositiva de Refuerzo





- V $(pwd)/project:/volumen1

Tema 3: Creación Imágenes

Diapositiva de refuerzo

- Creamos archivo Dockerfile con las instrucciones deseadas
- Construimos la imagen con un nombre asociado (tag): **docker build -f Dockerfile -t imagen:v1 .**
- Ejecutamos la imagen creada (--rm -it): **docker run --name contenedor imagen:v1**
- Ejecutamos comandos (-it): **docker exec contenedor /bin/sh**
- Tageamos la imagen para ser subida a un repo: **docker tag imagen:v1 repo/imagen:v1**
- Subimos al repositorio: **docker push repo/imagen:v1**
- Eliminamos contenedor: **docker rm -f contenedor**
- Eliminamos imagen: **docker rmi repo/imagen:v1**
- Traemos imagen desde el local: **docker pull repo/imagen:v1**

DEMO 4

Volumen

Base de datos mongo con almacenamiento persistente

```
docker pull mongo
mkdir ~/data/mongodb
docker run --name mongodb-container -v ~/data/mongodb:/data/db -d mongo
docker exec -it mongodb-container mongosh
```

CODE SPACE ACADEMY

DEMO

Web con nginx

Veremos como crear una web de ejemplo con nginx

Ejercicio

Web con nginx

Nos gustaría que nuestra web estuviera en el puerto 8080 en vez de en el 80, también nos gustaría que nuestro archivo html se llamara inicio.html. ¿Y si también queremos añadir un CSS o un archivo JS?

CODE SPACE ACADEMY

DEMO

Docker go

Vemos como buildear una imagen y ejecutar un contendor con una aplicación de go

CODE SPACE ACADEMY

DEMO

Docker python

Vemos como buildear una imagen y ejecutar un contendor con una aplicación de Python.

CODE SPACE ACADEMY

DEMO

Docker nodejs

Vemos como buildear una imagen y ejecutar un contendor con una aplicación de nodejs

CODE SPACE ACADEMY

DEMO

Docker react ui

Vemos como buildear una imagen y ejecutar un contendor con una aplicación de react que contiene una interfaz gráfica UI

DEMO-PROYECTO

Sonarqube

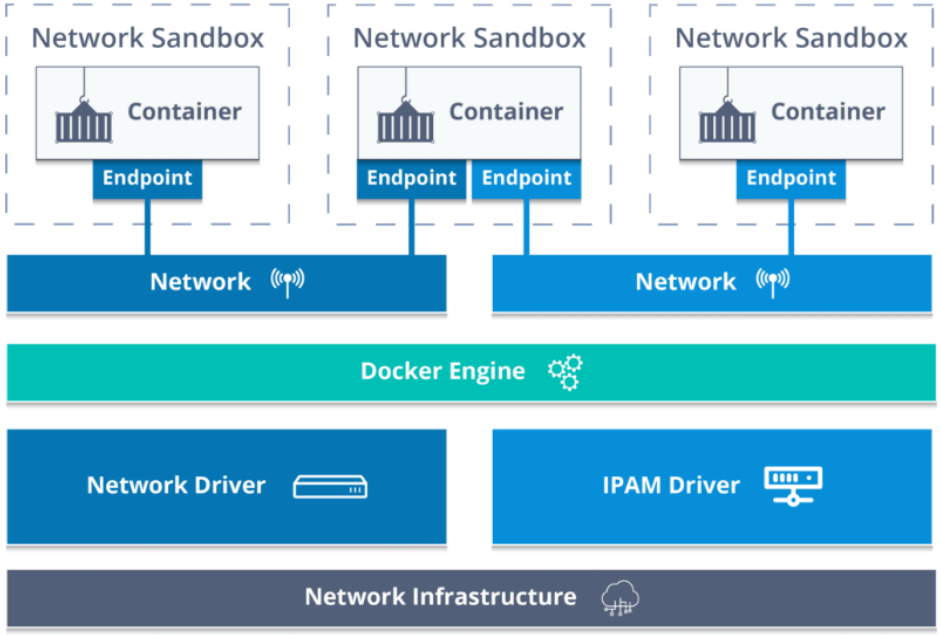
Vamos a construir un servidor de Sonarqube en un contenedor y crear otro con el cliente de sonar.
Una vez creados y enlazados vamos a lanzar un escaneo y ver que en nuestro servidor sonar aparece.

```
docker run -d --name sonarqube -p 9000:9000 sonarqube:latest
```

```
docker run -v ~/node:/usr/src -it --rm --name sonar-scanner --link sonarqube:sonarqube sonarsource/sonar-scanner-cli:latest sonar-scanner -Dsonar.host.url=http://sonarqube:9000 -Dsonar.projectKey=proyecto -Dsonar.sources=. -Dsonar.login=sqp_2d07d26b4f2c4bad189239e3c0d3e5815aa50315
```

Tema 3: Redes Docker

Las redes Docker se usan para conectar contenedores Docker entre sí mediante endpoints, pueden usarse para conectar los contenedores con otros que se encuentren incluso fuera de nuestro entorno mediante el uso de Puentes (Bridges).



Tema 3: Redes Docker

Estos son los comandos más usados relacionados con redes Docker:

Comando	Función
Docker network ls	Lista las redes
Docker network create	Crea una red
Docker network connect	Conecta un contenedor a una red
Docker network inspect	Da detalles sobre la red

CODE SPACE ACADEMY

DEMO

Docker redes
Creamos dos contenedores, uno web y otro con una base de datos:
docker run -d --name web-server -p 80:80 nginx
docker run -d --name database -e MYSQL_ROOT_PASSWORD=mysecretpassword mysql

Creamos una red:
docker network create my-network

Conectamos ambos contenedores a la red:
docker network connect my-network web-server
docker network connect my-network database

Probamos que se conectan entre sí:

docker exec -it web-server bash
$ **curl database:3306**

Ejercicio

Docker redes

Creamos 2 contenedores Ubuntu que se comuniquen mediante una red, probamos su conectividad con comandos de redes vistos en el tema de Linux. (esta vez no usaremos --link sino las network)

CODE SPACE ACADEMY

DEMO

Docker flask

Vemos como buildear una imagen y ejecutar un contendor con una aplicación de Python usando Flask.

CODE SPACE ACADEMY

DEMO

Docker C++ multistage

Vemos como buildear una imagen y ejecutar un contenedor con una aplicación de C++ usando Dockerfiles multistage.

Ventajas de las imágenes multistage: más fáciles de entender, solo usamos las primeras imágenes para descargar dependencias y copiamos lo que necesitamos

Imágenes multistage con más sentido

Docker multistage JAVA

```
FROM maven AS build
WORKDIR /app
COPY . .
RUN mvn package

FROM tomcat
COPY --from=build /app/target/file.war /usr/local/tomcat/webapps
```

Imágenes multistage con más sentido

Docker Python multistage REACT

```
FROM node:18 AS build
WORKDIR /app
COPY package* yarn.lock ./
RUN yarn install
COPY public ./public
COPY src ./src
RUN yarn run build

FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
```

CODE SPACE ACADEMY

Ejercicio

Docker Python multistage REACT

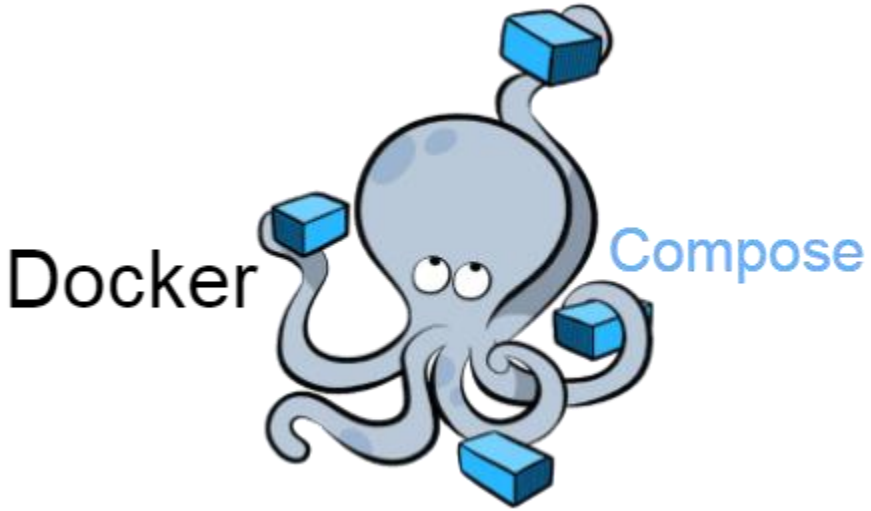
Modificamos la aplicación REACT que hemos dockerizado anteriormente y la convertimos en multistage usando nginx

Tema 4 : Docker Compose

- Tema 1: Docker Compose
- Tema 2: Orquestación de Contenedores

Tema 4: Docker Compose

Docker Compose es una herramienta para definir y ejecutar aplicaciones de Docker de varios contenedores. En Docker Compose, se usa un archivo YAML para configurar los servicios de la aplicación. Después, con un solo comando, se crean y se inician todos los servicios de la configuración y con otro se pueden parar todos a la vez.



Tema 4: Docker Compose

Comando	Función
Docker-compose up	Levantamos los contenedores afectados
Docker-compose down	Para los contenedores afectados
Docker-compose ps	Vemos los contenedores activos
Docker-compose restart	Reinicia los contenedores afectados
Docker-compose exec <servicio> <comando>	Ejecuta comandos en el servicio seleccionado

Tema 4: Docker Compose

Ejemplo Docker Compose YAML

```yaml
version: '3'
services:
  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: mydb
    volumes:
      - db_data:/var/lib/mysql
  web:
    image: php:7.2-apache
    depends_on:
      - db
    volumes:
      - .:/var/www/html
    ports:
      - "8080:80"
volumes:
  db_data:
```

Compose YAML

- Version: la versión de formato de yaml para Docker compose (Por defecto usamos 3, la versión más nueva, en la documentación de Docker aparecen las diferencias)
- Services: define los servicios que queremos levantar
- Image: imagen usada
- Environment: variables de entorno adicionales para el servicio
- Volumes: volúmenes asociados
- Depends on: Selecciona el orden de ejecución, espera a que se ejecute para ejecutarse.
- Ports: mapeo de puertos
- Network: usa una red
- Build: equivale a hacer un Docker build en el directorio seleccionado

CODE SPACE ACADEMY

DEMO

Docker Compose

Creamos un archivo yaml para crear una web con nginx y una base de datos mysql.

EJERCICIO

Docker Compose

Crear una web personalizada usando docker compose y Docker. (Pista: crear propia imagen Docker)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mi sitio web</title>
  </head>
  <body>
    <h1>Bienvenido a mi sitio web</h1>
    <p>Este es un ejemplo básico de web HTML.</p>
  </body>
</html>
```

DEMO

Docker Compose

Ejecutamos el Docker-compose mostrado en el primer ejemplo, es otra forma de usar volúmenes para pasar archivos al interior de las imágenes

EJERCICIO

Docker Compose

Creamos un archivo yaml para crear una web con nginx, un servidor back con una imagen de mongo-express y una base de datos mongo.

CODE SPACE ACADEMY

DEMO

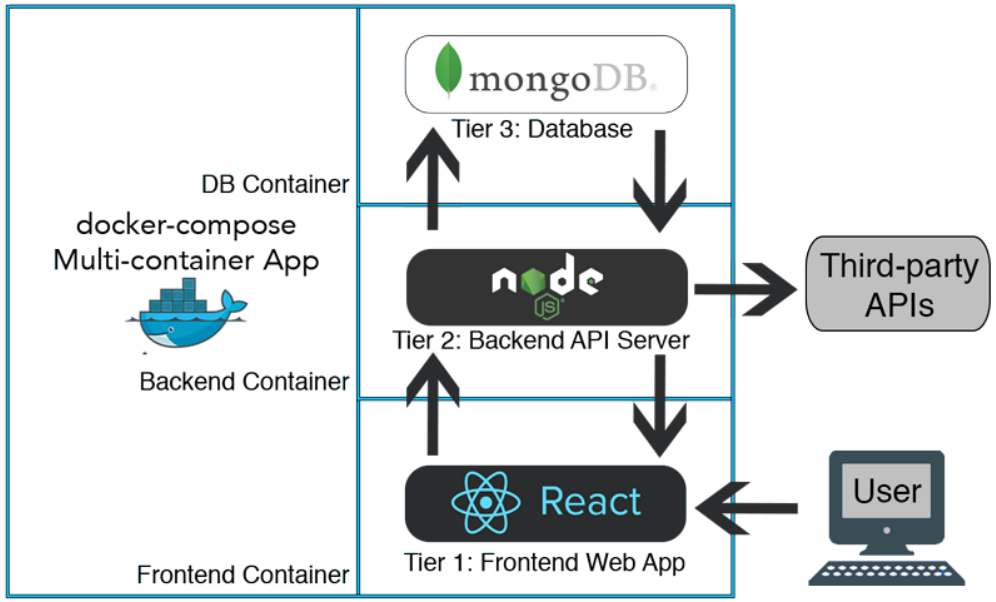
Docker Compose

Mostramos un Docker compose de phpmyadmin

DEMO/EJERCICIO REAL

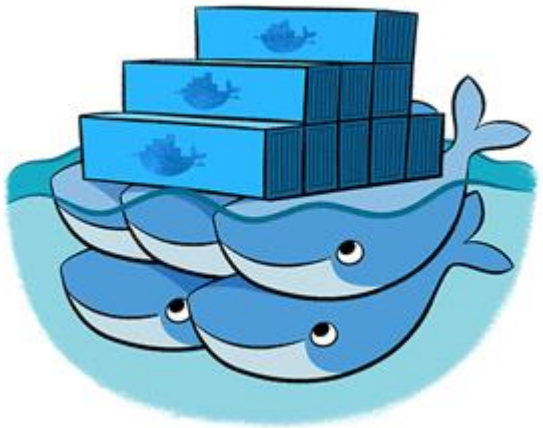
Docker Compose

Desplegar aplicación full stack con Docker compose: react UI, nodejs backend y mongodb



Tema 4: Orquestación de Contenedores

El uso de contenedores permite ejecutar un conjunto de procesos separados del sistema facilitando su portabilidad a otros entornos y siendo muy rápidos de implementar. Los contenedores permiten empaquetar y aislar del sistema, las aplicaciones y el entorno necesario para que se ejecuten, permitiendo mover dichas aplicaciones entre distintos sistemas. Debido a su popularidad el uso de contenedores se ha extendido y han surgido herramientas que permiten administrarlos y organizarlos como son Kubernetes de Google y Swarm de Docker.

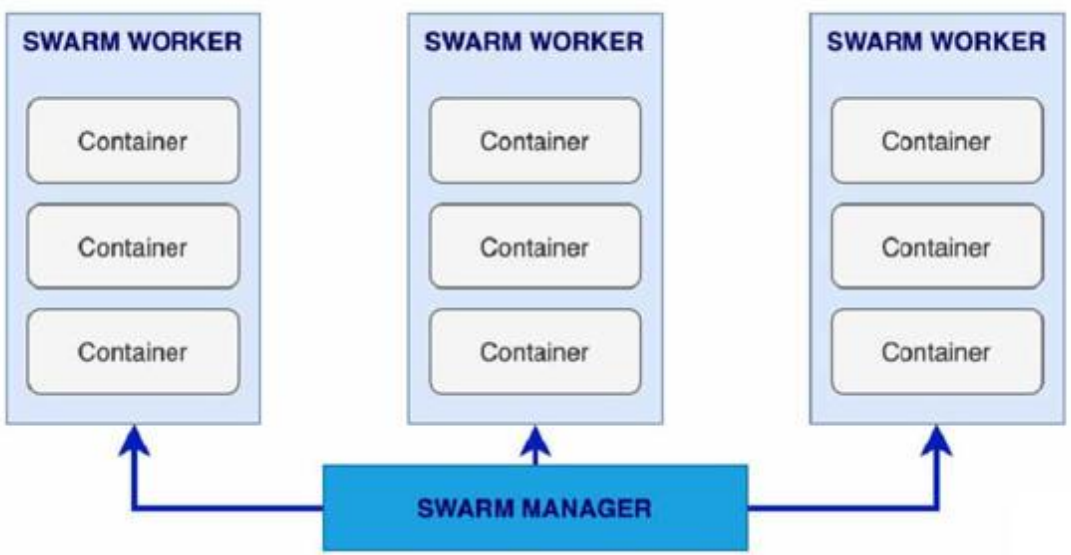


kubernetes

Tema 4: Docker Swarm

Docker Swarm es una herramienta integrada en Docker que permite agrupar una serie de hosts de Docker en un clúster y gestionarlos de forma centralizada, así como orquestar sus contenedores.

En el Docker Swarm se tendrán múltiples servidores que se pueden comunicar entre sí y tienen corriendo diversos contenedores. Alguno de estos servidores serán managers y podrán decidir dónde debe ir la carga, asignando en que nodo correrá cada contenedor.



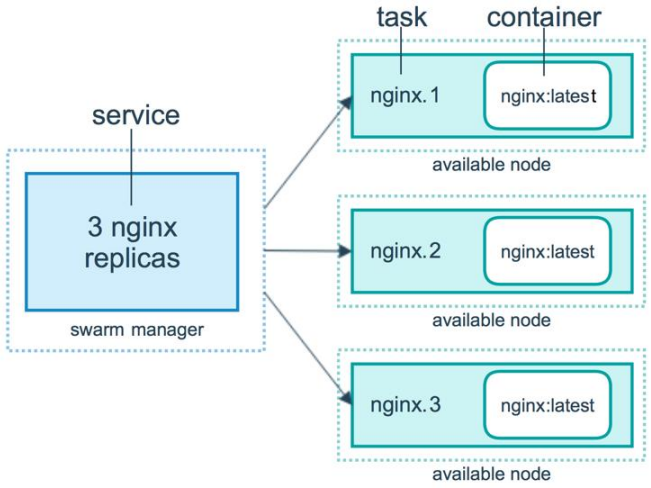
Tema 4: Docker Compose

Comando	Función
Docker swarm init	Iniciamos un swarm
Docker swarm join --token	Añade un nuevo nodo al swarm
Docker node ls	Lista los nodos
Docker service ls	Lista los servicios
Docker service update	Actualiza algún valor del servicio
Docker stack deploy -c compose_file nombre	Despliega Docker compose en el swarm
Docker swarm leave --force	Eliminamos el swarm

DEMO

DOCKER SWARM

Iniciamos Docker swarm: **docker swarm init**
Desplegamos servicio en el swarm: **docker service create --name web --replicas 1 -p 80:80 nginx**
Chequeamos que el servicio está ejecutándose: **docker service ls**
Cambiamos el número de réplicas a 3: **docker service update --replicas 3 web**



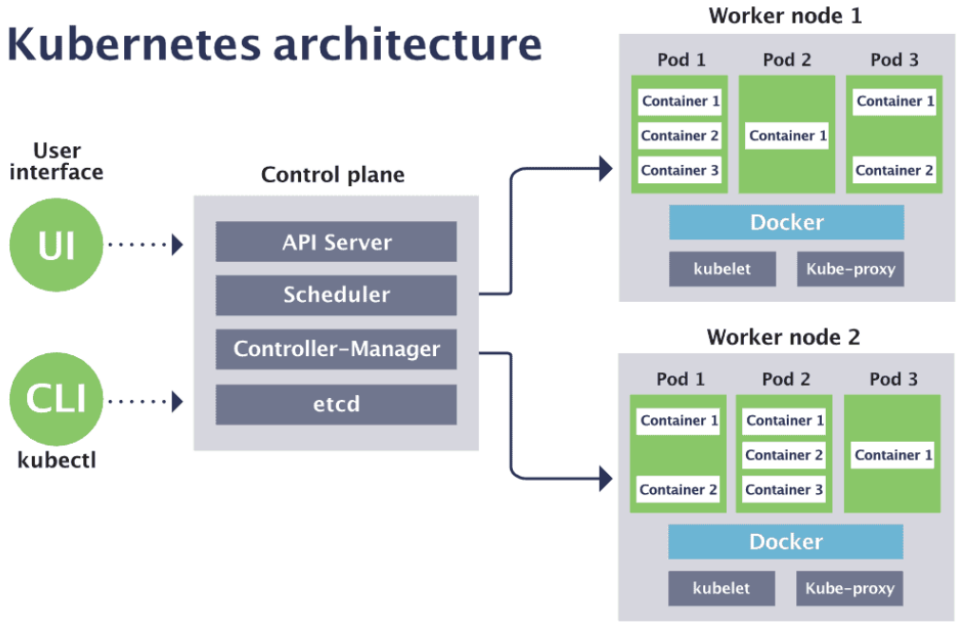
DEMO

DOCKER SWARM

Desplegamos demo1 en el swarm: **docker stack deploy -c docker-compose.yml stack_app**

Tema 4: Kubernetes

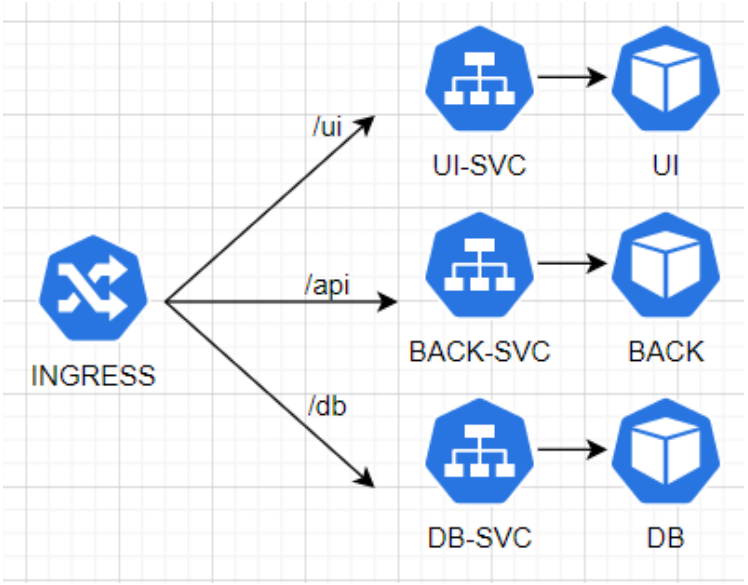
Kubernetes (K8s) es una plataforma de código libre cuyo objetivo es el de administrar y organizar clusters de contenedores, evitando los procesos manuales relacionados para escalar e implementar aplicaciones en contenedores. Cuando se trabaja con grupos de contenedores, Kubernetes facilita su administración de forma rápida y efectiva.

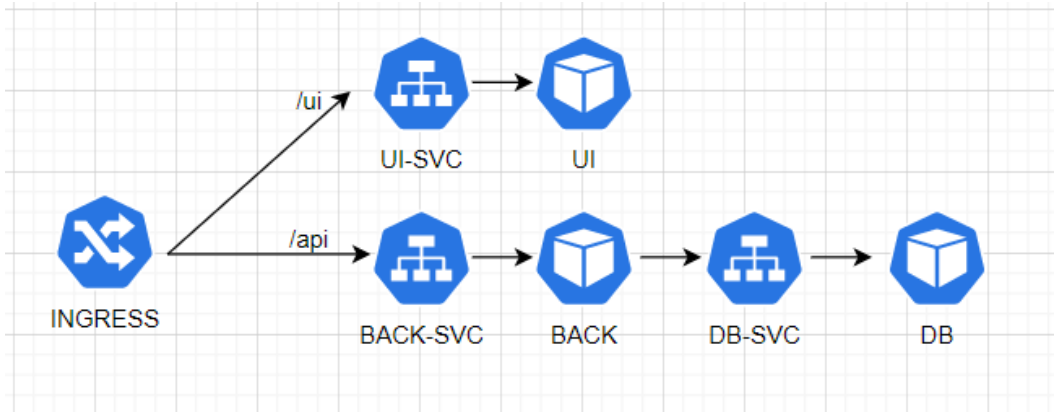


Ejemplo

KUBERNETES

Posibilidad de desplegar con arquitecturas definidas por yamls





Tema 4: Kubernetes vs Docker Swarm

CODE SPACE ACADEMY

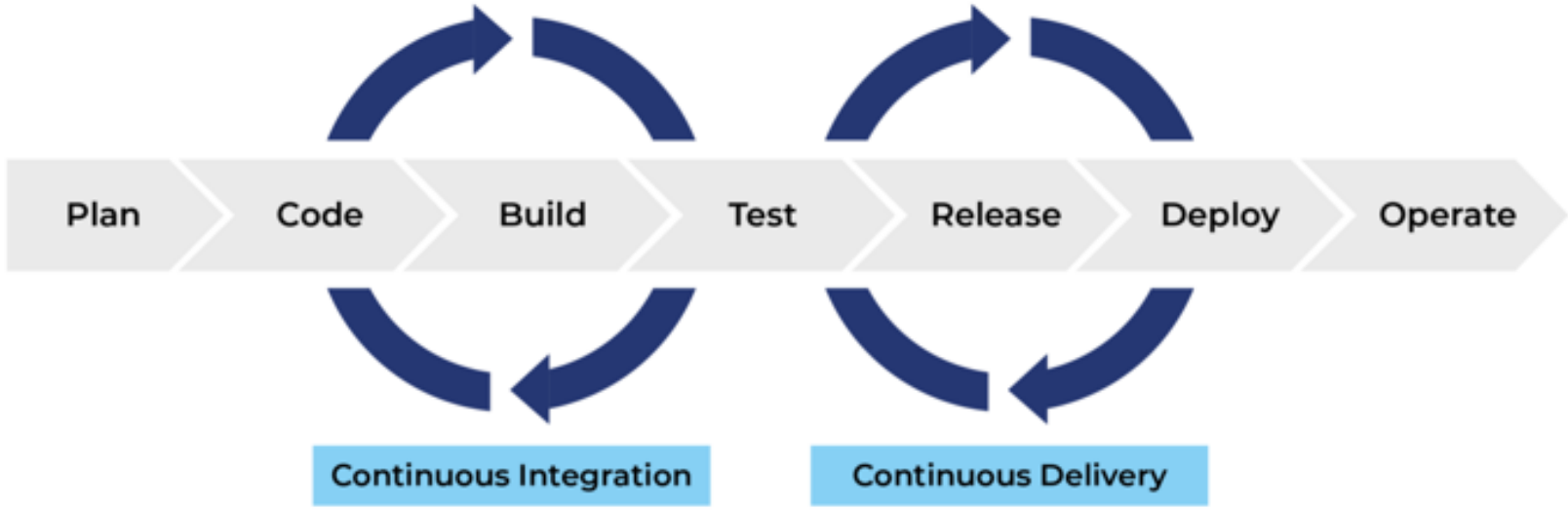
Característica	Kubernetes	Docker Swarm
Instalación y Configuración	Complicada pero es muy configurable	Fácil pero poco configurable
GUI	Kubernetes dashboard y más herramientas externas	No tiene
Escalabilidad	Muy escalable y rápido	Escalable
Auto escalado	Sí	No
Balance de Carga	Definición de servicios para ello	Es automático
Rollbacks	Sí	No
Logs y monitorización	Muy monitorizable y fácil de ver logs gráficos	Se necesitan herramientas externas para monitorización y logs muy básicos

Tema 5 : CICD

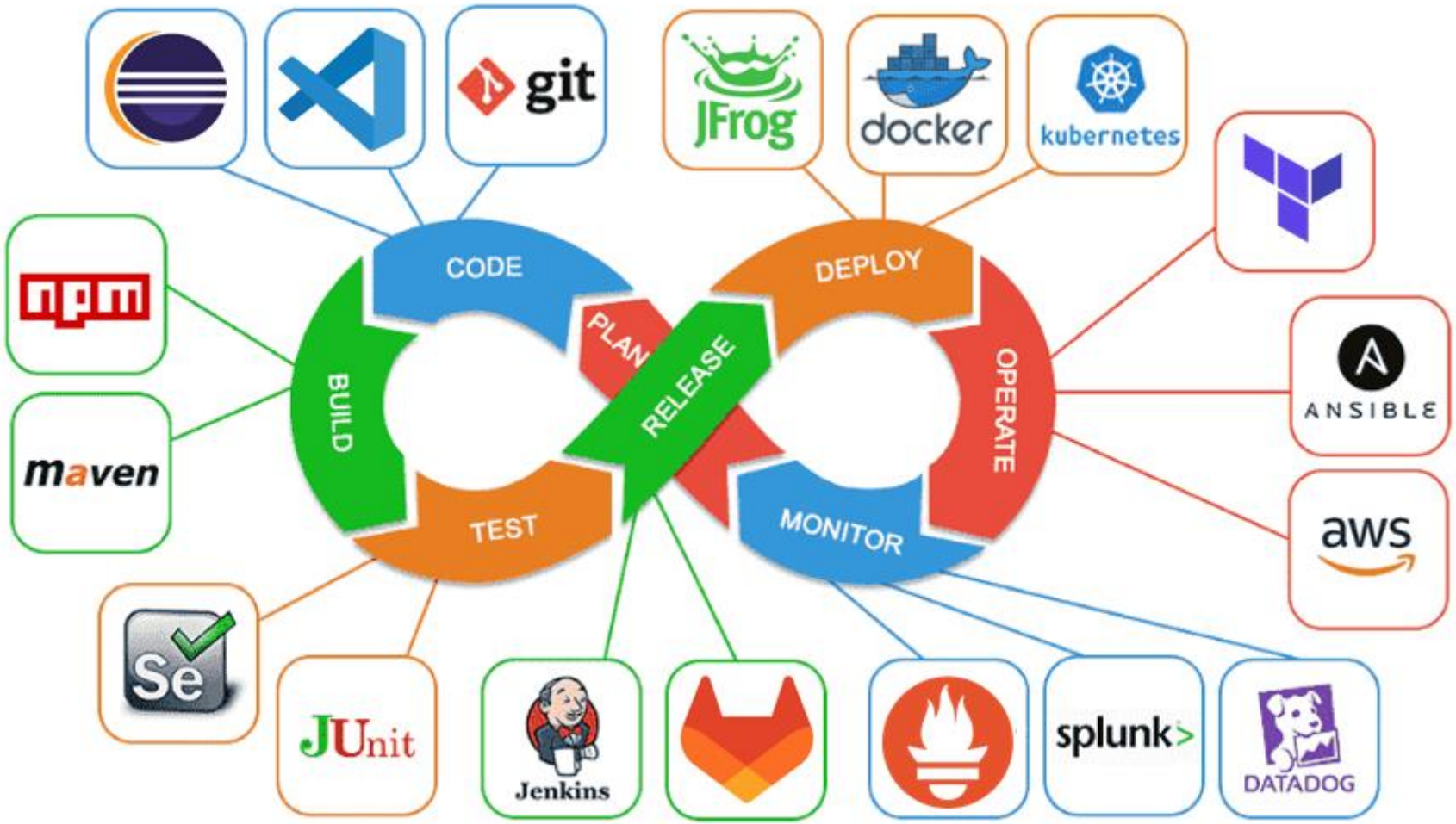
- ¿Qué es?
- Linux (Conceptos Básicos)
- Permisos
- Scripting

Tema 5: ¿Qué es?

La integración continua (CI) y la entrega continua (CD), abreviada como CI/CD, es una parte integral de la cultura DevOps en la que combina los procesos operativos y de desarrollo en un solo flujo de trabajo. En otras palabras, es una metodología secuencial que se utiliza para entregar las aplicaciones desarrolladas a sus usuarios finales mediante la introducción de la automatización en las etapas de desarrollo de aplicaciones.

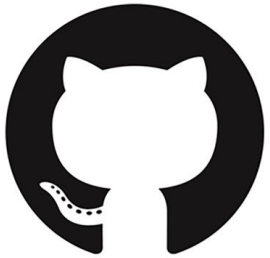


Tema 5: Flujo Completo



Tema 5: Herramientas de CICD

CODE SPACE ACADEMY



DEMO

Jenkins

Crearemos un contenedor de Jenkins y explicaremos su funcionamiento, aunque no crearemos ninguna pipeline (usaremos otra herramienta más actualizada para ello)
docker pull jenkins/Jenkins
docker run -p 8080:8080 -p 50000:50000 -v jenkins_home:/var/jenkins_home jenkins/jenkins

CODE SPACE ACADEMY

DEMO

CICD

Usaremos gitlab para hacer una pequeña demo sobre pipelines: www.gitlab.com

Tema 5: Pipelines

Ejercicio

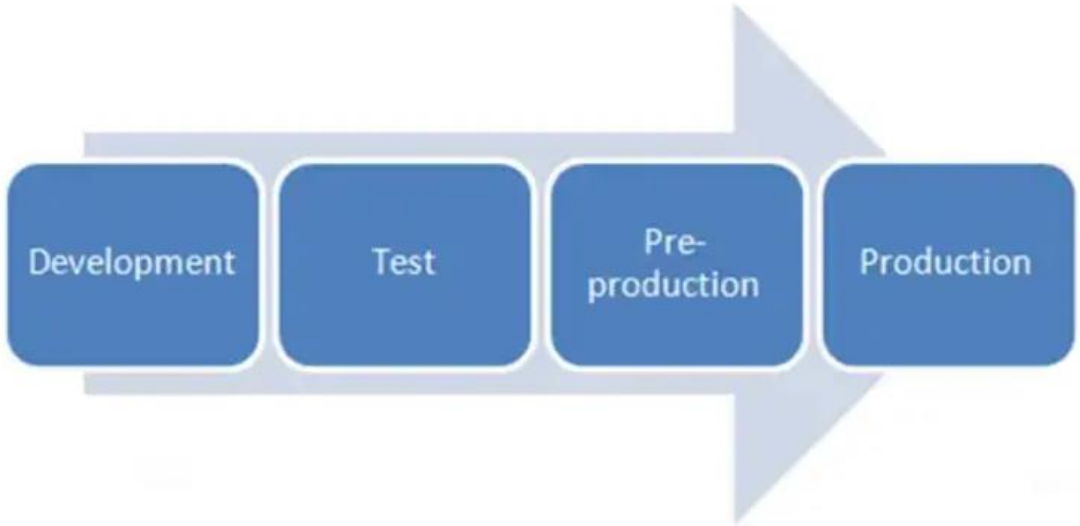
CICD

Crearemos una pipeline en Gitlab que “despliegue” falsamente un UI con react, usaremos imágenes docker.

Documentación Gitlab CI: https://docs.gitlab.com/ee/ci/

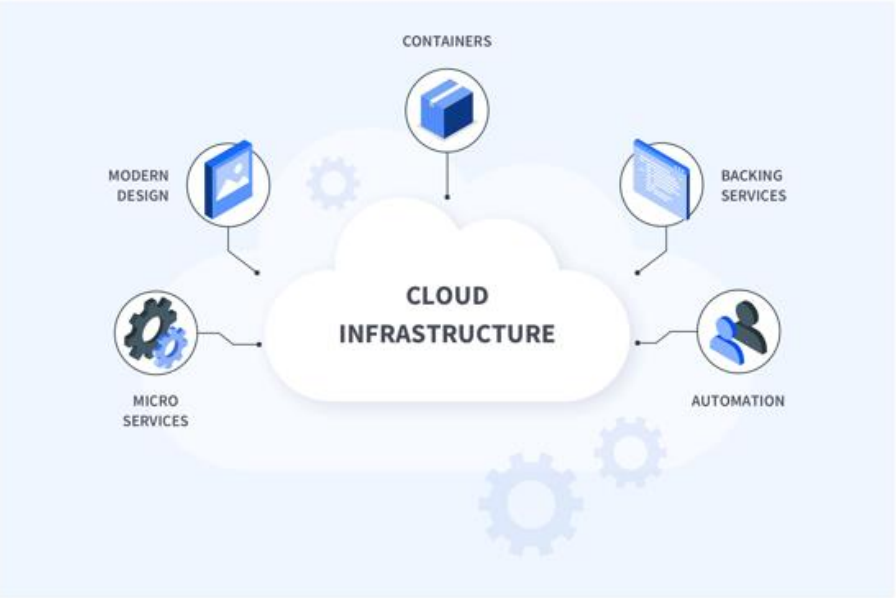
Tema 5: Entornos

- Entorno de desarrollo: Aquí es donde el desarrollador prueba el código y comprueba si la aplicación se ejecuta correctamente. Una vez que la implementación ha sido probada y el desarrollador considera que el código funciona de forma correcta, la aplicación se mueve al siguiente entorno.
- Entorno de pruebas: Aquí es donde trabaja el equipo de testeo y analistas. Comprueban que el código desarrollado funciona como es debido tras pasar las pruebas pertinentes.
- Entorno de integración (pre-producción): Este entorno se hace para que se vea exactamente igual que el entorno de producción. La aplicación se prueba en el servidor de integración para comprobar la fiabilidad y para asegurarse de que no falla en el servidor de producción real.
- Entorno de producción: Es el entorno donde reside la aplicación final y tienen acceso los clientes.



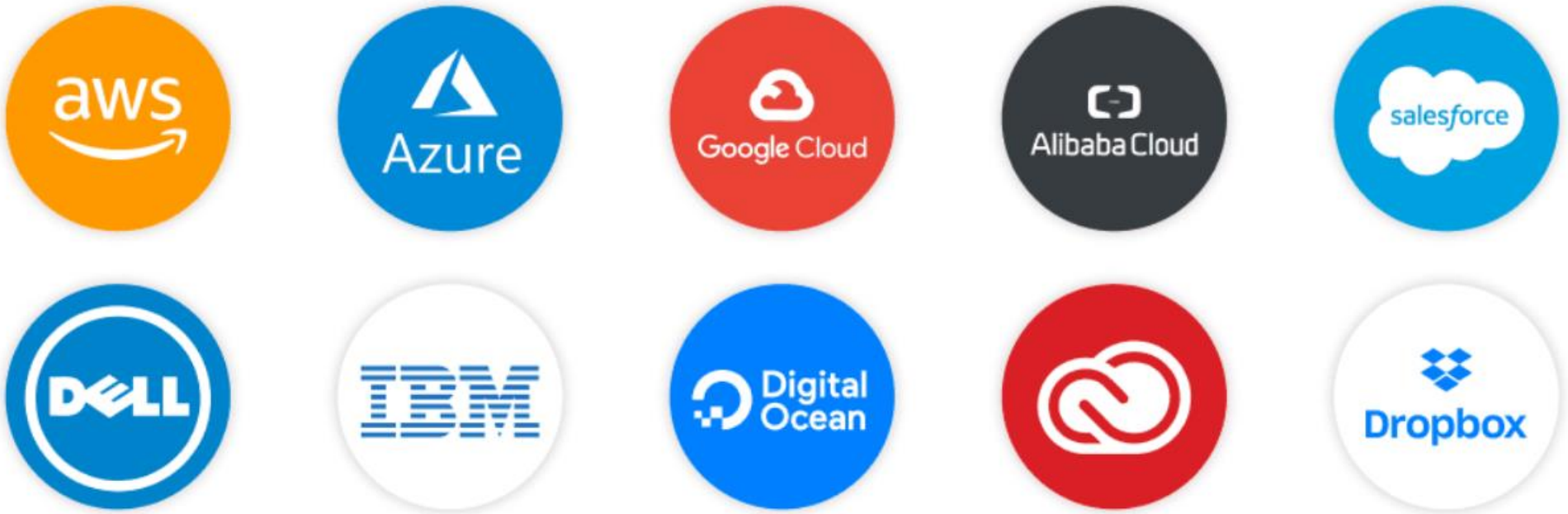
Tema 5: Cloud

En términos prácticos, tanto cloud como cloud computing son términos que se utilizan para describir el concepto de almacenar y acceder a la información en Internet, generalmente a través de servicios de terceros.
Especificando más diríamos que mientras que cloud es Internet en general y nos referimos a ella cuando hablamos de tener datos, aplicaciones o infraestructura fuera de las instalaciones de nuestra empresa, cloud computing nos habla de los productos y servicios que funcionan en la nube (cloud) y los cuales podemos acceder a través de Internet. Se refiere al acceso a ordenadores, otros elementos de TI y aplicaciones de software a través de una conexión de red



Tema 5: Proveedores Cloud

CODE SPACE ACADEMY



PREGUNTAS GENERALES

CODE SPACE ACADEMY

DESCANSO

CODE
SPACE
ACADEMY