Curso Docker

Elmer Orlando Real Ixcayau
Carne 201503936
Abril 2020

Indice

INTRODUCCIÓN

Para ver la lección presione el icono…

Regresar al Indice

Siguiente Leccion

Despliegue

- Una de las etapas más importantes en el desarrollo de software es el despliegue que consiste en migrar cada versión nueva de una aplicación del entorno de desarrollo al entorno de producción
- Existen diferentes tipos de despliegues
  - Tradicional
  - Utilizando tecnologías de virtualizacion
  - Utilizando contenedores.

Inconvenientes

- Las configuraciones no son portables
- Se utiliza mucho tiempo para su configuracion.
- No se pueden desplegar aplicaciones que funcionen con diferente sistema operativo o versión de una librería.
- Recursos computacionales sub-utilizados.

DESPLIEGUE TRADICIONAL

Tradicionalmente el proceso de despliegue de una aplicación se realiza sobre una maquina física administrando manualmente el sistema operativo, versiones de archivos binarios y librerías requeridas para el correcto funcionamiento de la aplicación a desplegar.

DESPLIEGUE EN MÁQUINAS VIRTUALES

Los problemas del despliegue tradicional son solucionados utilizando la **tecnología de virtualizacion**, la cual permite simular múltiples entornos aislados, inclusive con sistemas operativos diferentes, en una misma computadora.

A estos entornos aislados se les conoce como **Máquina Virtuales** y pueden haber tantas como el hardware soporte, permitiendo así desplegar múltiples aplicaciones y aprovechar de mejor manera los recursos físicos del servidor.

DESPLIEGUE EN MÁQUINAS VIRTUALES

Los software de virtualizacion se pueden instalar

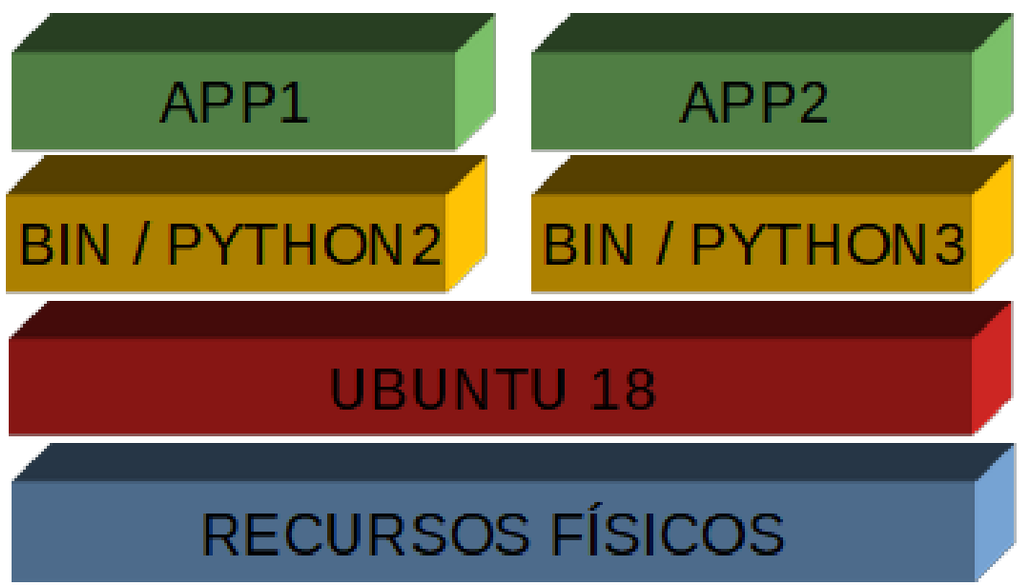
- Directamente sobre computadoras sin sistema operativo (también conocido como Bare-Metal)
- Sobre algún sistema operativo

Para funcionar crea una simulación de todos los componentes físicos de la computadora y utiliza un programa centinela conocido como **hipervisor** para monitorear la interacción que se realiza con las componentes simulados para replicarlos en la maquina física.

Inconvenientes

- Si existen varias maquinas de un mismo sistema operativo se duplica el Sistema Operativo (Kernel).
- No son completamente portables por que cada Hipervisor funciona de diferente manera en cada maquina, por lo que a veces es necesario configurar manualmente.
- El Hipervisor que hace posible la virtualizacion también consume recursos computacionales para funcionar.
- Se debe de reservar recursos como Ram o disco duro específicamente para cada máquina virtual,limitando así la cantidad de Maquinas virtuales que una computadora pueda soportar.
- Una maquina virtual puede reservar recursos que no utiliza

DESPLIEGUE EN CONTENEDORES



Otra manera de desplegar aplicaciones es empaquetando el software en unidades estandarizadas denominadas contenedores. Docker es una tecnología de contenerizacion.

Los contenedores son otra tecnología de virtualizacion que resuelve algunos problemas de las máquinas virtuales al aprovechar ciertas características del Kernel de linux

DESPLIEGUE EN CONTENEDORES

Docker a diferencia de las maquinas virtuales, que proporcionan entornos completamente aislados (con sistema operativo y kernel propio), ofrece entornos aislados parcialmente, es decir, aislá el espacio de trabajo de cada contenedor (equivalente a una maquina virtual) pero todos los contenedores comparten el kernel con el sistema operativo anfitrión, por lo que no hay necesidad de simular los componentes físicos de la computadora, eliminando así la necesidad del uso del hipervisor ya que todo se realiza sobre la maquina anfitrión a través de un servicio (conocido como demonio Docker).

Ventajas

- **Portabilidad**: puede ejecutarse en cualquier maquina y siempre tendrá el mismo comportamiento.
- **Tiempo**: Se puede automatizar utilizando scripts
- **Costo**: Debido a que elimina la duplicación de kernel y el uso de hipervisor se aprovechan mas los recursos fisicos.
- **Ligero**: Un contenedor utiliza solamente los recursos necesarios para funcionar.

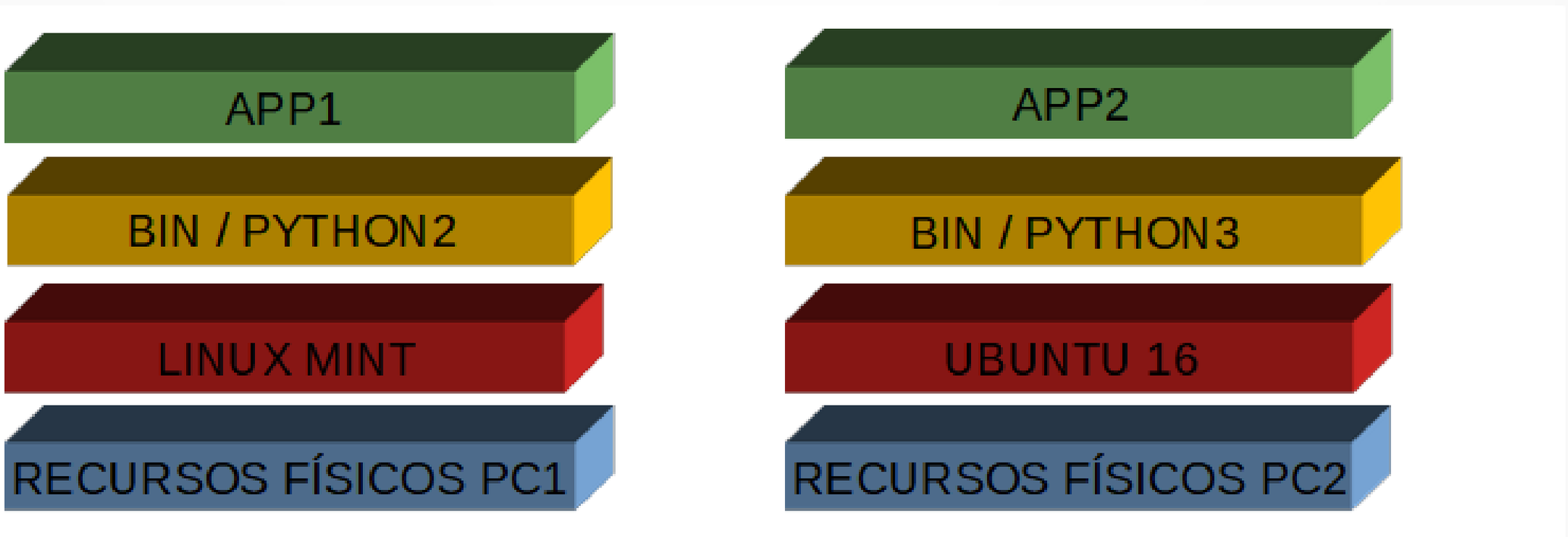
Ejemplo

- Suponga que se necesita desplegar dos aplicaciones con las siguientes características

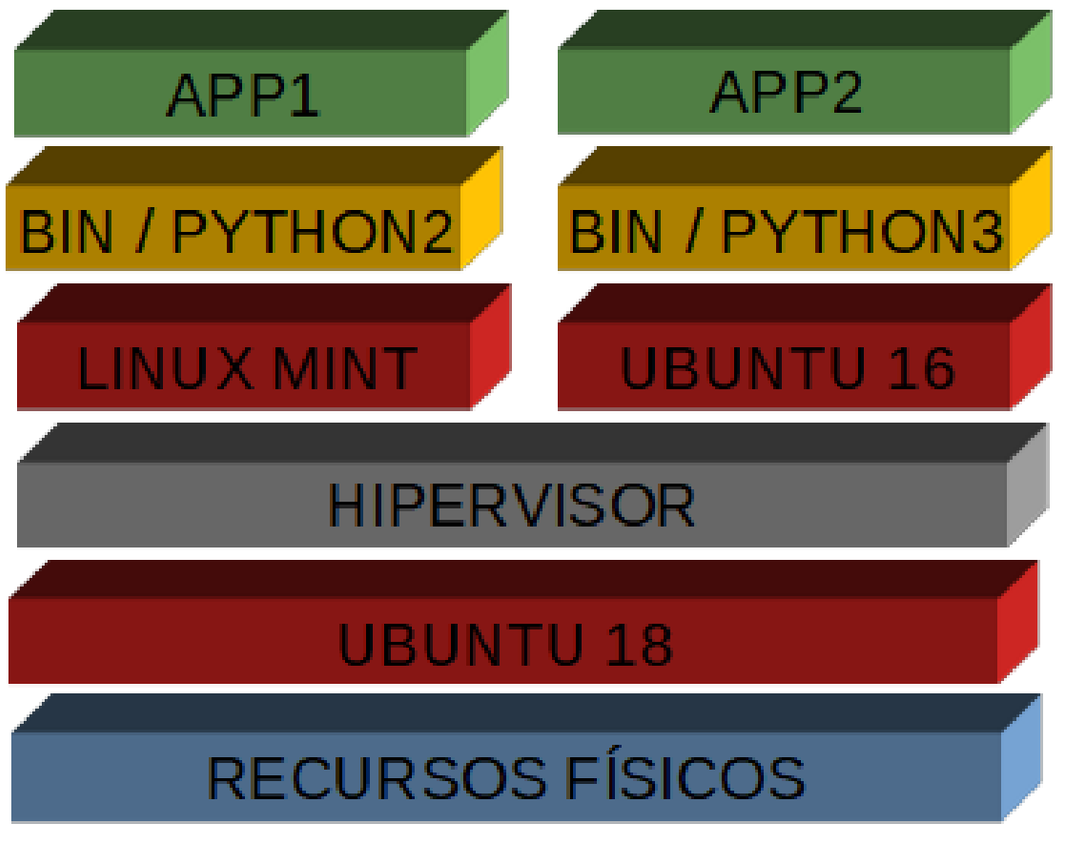
Aplicación	Sistema Operativo	Librerías extras
APP1	Linux Mint	Python 2
APP2	Ubuntu 16	Python3

DESPLIEGUE TRADICIONAL

Es imposible desplegar ambas aplicaciones en una sola máquina de la manera tradicional por que cada aplicación utiliza una librería python y sistema operativo completamente diferente, por lo que la única solución es despegarlo en maquinas independientes.

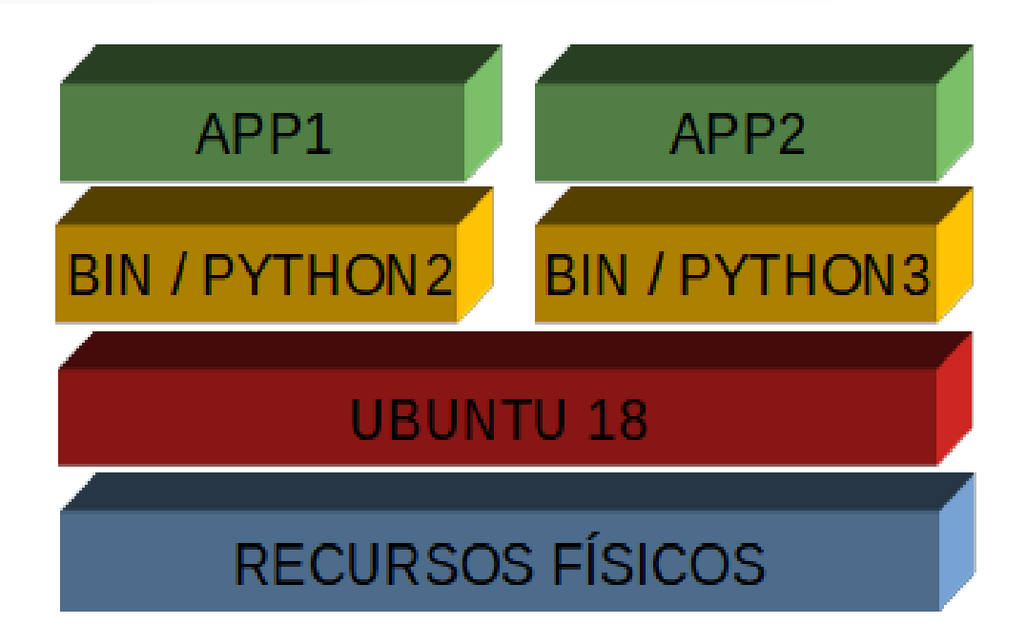


DESPLIEGUE EN MÁQUINAS VIRTUALES



Utilizando máquinas virtuales si se pueden desplegar en una sola máquina, sin embargo, Linux tiene la característica que todas sus distribuciones se basan del mismo Kernel por lo que en este caso se replica tres veces consumiendo así espacio de almacenamiento, además cada máquina reserva memoria primaria y secundaria condicionando la posibilidad de lanzar otra maquina virtual en esta misma máquina.

DESPLIEGUE EN CONTENEDORES



Ambas aplicaciones se despliegan en una misma máquina física compartiendo con ella el Kernel Linux de manera que solo sera necesario descargar las librerías propias de cada sistema operativo, así como las distintas versiones de Python. Con esta arquitectura se aprovecha de mejor manera los recursos y se elimina la necesidad de un Hipervisor y la duplicidad de librerías Además, como no se reservan recursos se pueden crear tantos contenedores como se requieran (es importante estimar la carga de cada contenedor para no afectar a los demás contenedores que están funcionando).

Conceptos Básicos

Para ver la lección presione el icono…

Regresar al Indice

Siguiente Leccion

¿ Qué es Docker?

- Docker es una herramienta de virtualizacion a nivel de sistema operativo para el despliegue de software en unidades estandarizadas denominadas contenedores.
- Cada contenedor es un entorno único y aislado que posee su propio software, bibliotecas y configuración
- Funciona reutilizando los módulos del kernel linux anfitrión.

Arquitectura utilizando Contenedores

Ventajas de Docker

Docker ofrece las siguientes ventajas:

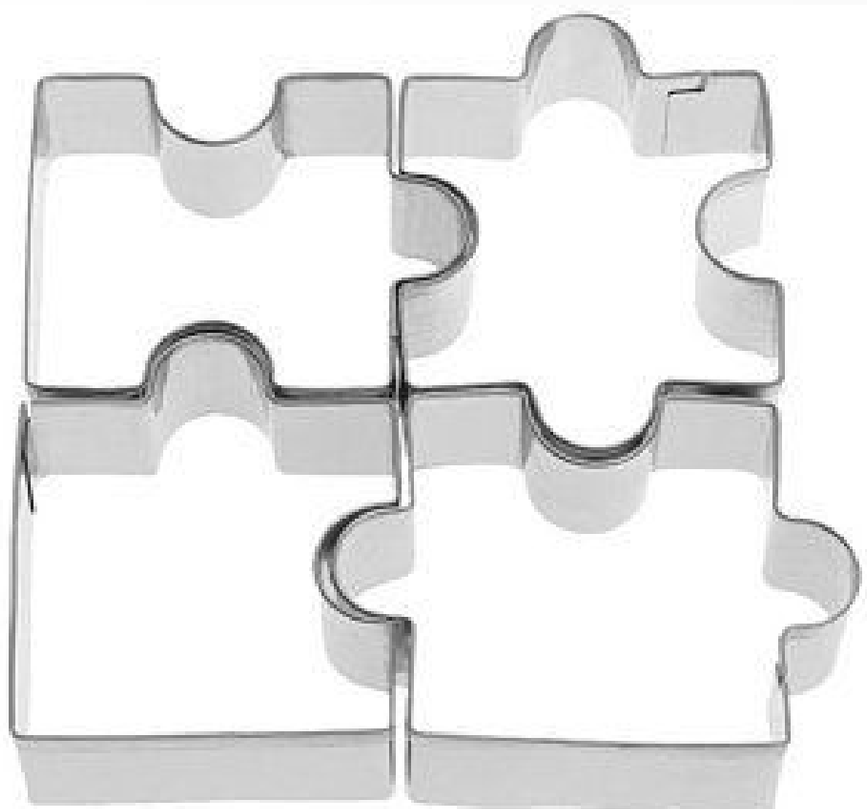
- Portabilidad: puede ejecutarse en cualquier maquina y siempre tendrá el mismo comportamiento.
- Tiempo: Se puede automatizar utilizando scripts
- Costo: Debido a que elimina la duplicación de kernel y el uso de hipervisor se aprovechan mas los recursos fisicos.
- Ligero: Un contenedor utiliza solamente los recursos necesarios para funcionar.

Términos importantes en Docker

- Imagen
- Contenedor
- Registro

Imagen

- En Docker una imagen es una plantilla de solo lectura con un conjunto de archivos binarios, librerías y aplicaciones que se utilizan para construir contenedores.
- Se puede hacer la analogía con el concepto de clase en programación orientada a objetos, de la cual pueden crearse múltiples objetos (contenedores).

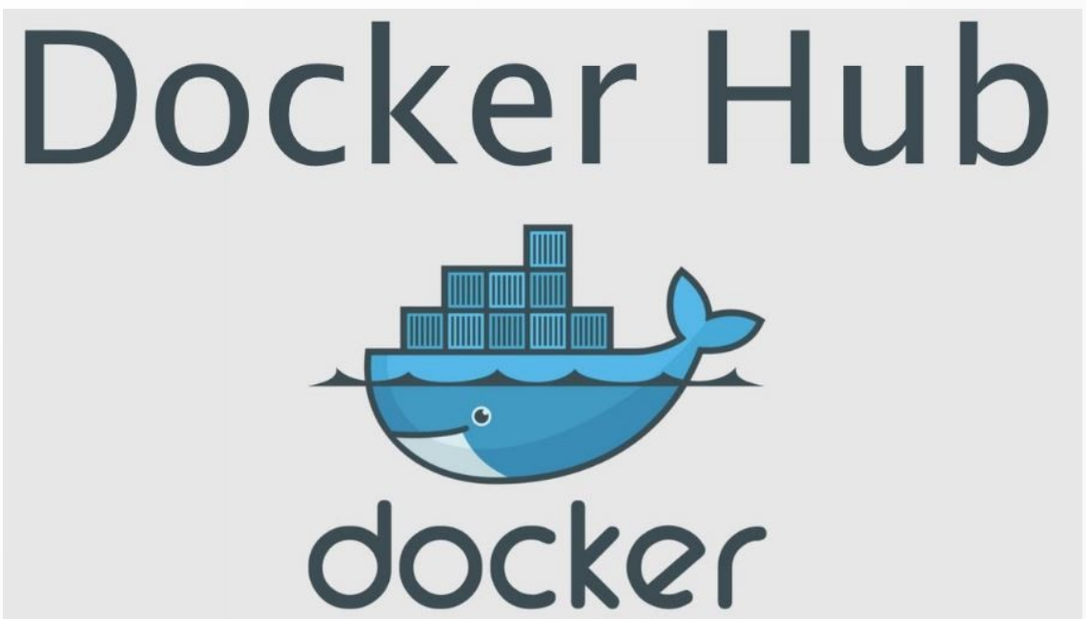


Contenedor

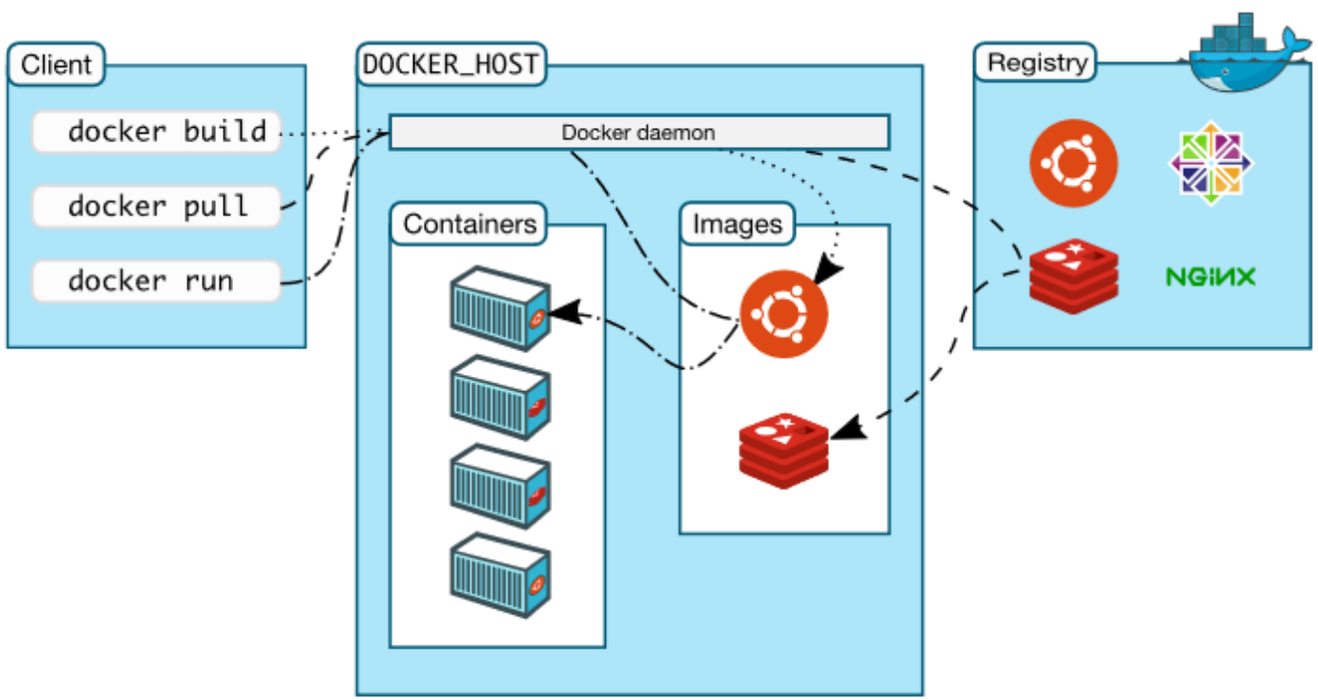
- Un contenedor es la instancia de una imagen, es decir es un entorno único y aislado que se creo a partir de una imagen.
- Cada contenedor que se crea de una imagen es idéntico en cuanto a librerías, binarios y aplicaciones instaladas pero es un entorno completamente diferente e independiente.

Registro

- Un registro es una biblioteca de imágenes Docker.
- Se acceso puede ser publico o privado
- Se puede alojar on-premise o cloud.
- Docker Hub es el registro por defecto de Docker.



Arquitectura Docker



La arquitectura Docker se basa de la arquitectura cliente-servidor. Donde el cliente es el usuario que se conecta mediante una linea de comandos (De manera local o remota) y el servidor es un servicio que corre en la maquina anfitrión denominado demonio docker el cual permite interactuar tanto con las imágenes, contenedores e incluso con docker hub.

Interactuar con imágenes Docker

Para ver la lección presione el icono…

Regresar al Indice

Siguiente Leccion

Comandos

- Buscar una imagen en el registro.
  - **docker search <imagen>:<tag>**
- Descargar una imagen del registro.
  - **docker pull <imagen>:<tag>**
- Listar todas las imágenes almacenadas localmente.
  - **docker images**
- Eliminar una imagen especifica.
  - **docker rmi <Id_Imagen>**
    - No se recomienda usar el nombre de la imagen para eliminarla porque se puede eliminar la versión incorrecta.
- Convertir una imagen en un contenedor (básico).
  - **Docker run <Id_Image>**

Interactuar con contenedores.

Para ver la lección presione el icono…

Regresar al Indice

Siguiente Leccion

Interactuar contenedores.

- Mostrar metadata de un contenedor
  - **Docker inspect <ID_Container>**
- Mostrar la bitácora de la consola del contenedor.
  - **Docker logs <ID_Container>**
- Listar los contenedores que están ejecutados.
  - **Docker ps**
- Listar todos los contenedores (Corriendo, detenidos y muertos)
  - **Docker ps -a**

Interactuar contenedores.

- Enviar una señal de detención al contenedor, para que se detenga correctamente.
  - **Docker stop <ID_Container>**
- Ejecutar de nuevo un contenedor detenido.
  - **Docker start <ID_Container>**
- Matar un contenedor repentinamente, es poco probable que el contenedor vuelva a iniciar correctamente.
  - **Docker kill <ID_Container>**
- Eliminar un contenedor detenido o muerto
  - **Docker rm <ID_Container>**
- Iniciar una linea de comandos dentro de un contenedor.
  - **Docker exec -it <ID_Container> bash**

Configurar atributos de un contenedor utilizando Docker run

Para ver la lección presione el icono…

Regresar al Indice

Siguiente Leccion

Opciones Docker run
Docker run [Opcion1,Opcion2,...OpcionN] <ID_Imagen> <Comandos>

- Ejecutar el contenedor en segundo plano .
  - **--detach , -d**
- Nombrar un contenedor
  - **--name**
- Declarar variables de entorno.
  - **--env , -e**
- Indicar directorio de trabajo dentro del contenedor
  - **--workdir , -w**
- Documentar los puertos que el contenedor utiliza.
  - **--expose**
    - **No publica los puertos en las interfaz de red del anfitrión.**

Opciones Docker run
Docker run [Opcion1,Opcion2,...OpcionN] <ID_Imagen> <Comandos>

- Publicar el puerto en la interfaz de red del host.
  - **--publish , -p**
- Enlazar una carpeta de la maquina física con una carpeta dentro del contenedor.
  - **--volume , -v** :
- Crear un directorio temporal en un contenedor.
  - **--tmpfs**
- Enlazar un contenedor con otro para comunicarse.
  - **--link**

Crear nuevas imágenes

Para ver la lección presione el icono…

Regresar al Indice

Siguiente Leccion

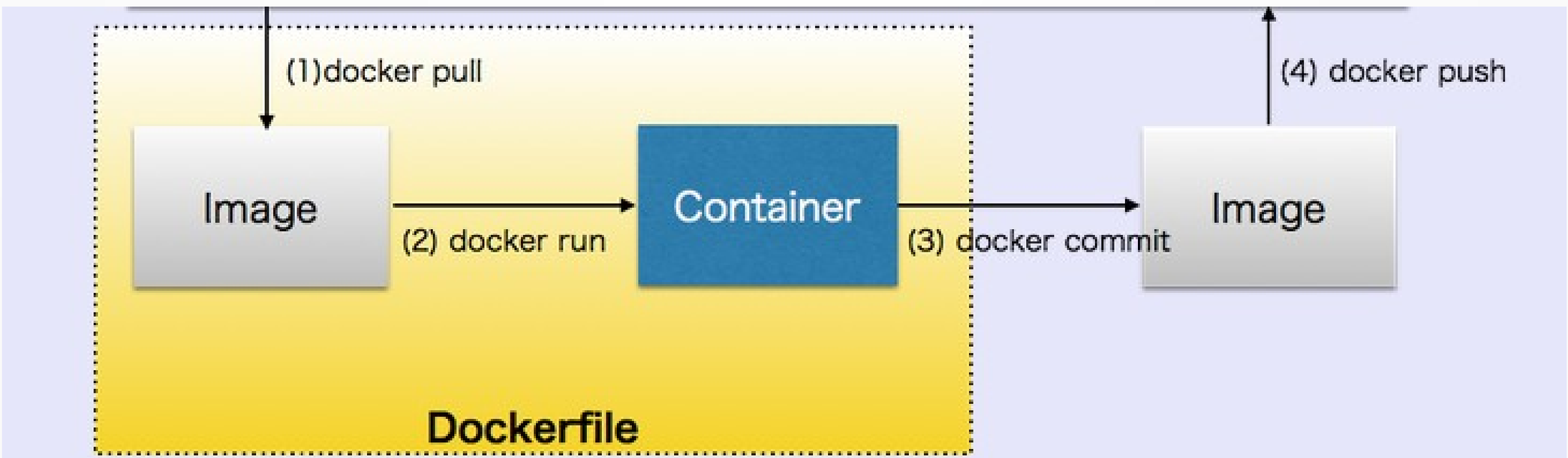
Crear nuevas imagenes.

Una imagen es una plantilla con binarios, librerias y aplicaciones instaladas, comúnmente es necesario que un contenedor tenga algunas aplicaciones extra a las que incluye la imagen, por lo que se deben de instalar via linea de comandos. Cuando es solo un contenedor no hay problema, cuando se tratan de varios contenedores del mismo tipo es aconsejable crear una imagen con las aplicaciones previamente instaladas en orden de automatizar el despliegue de las aplicaciones al máximo.

Se pueden crear imágenes de dos maneras

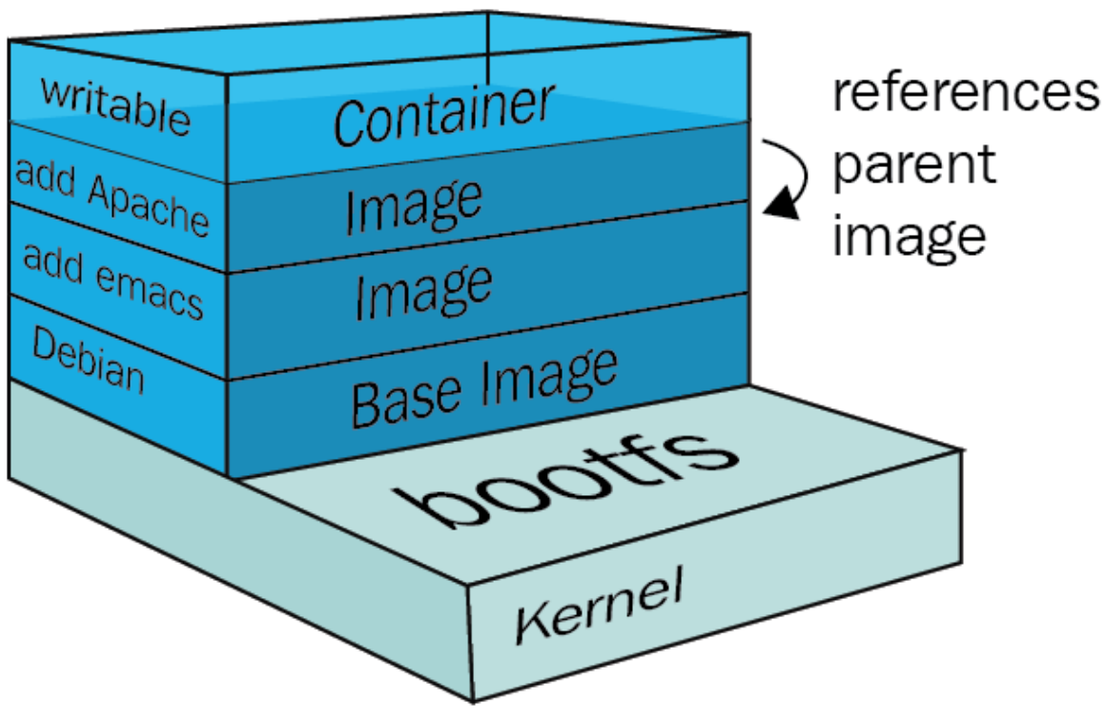
- Tomándole una instantánea a un contenedor.
- Escribiendo una ‘receta’, es decir un archivo Dockerfile.

Flujo para crear una imagen.



Como una instantánea de un contenedor

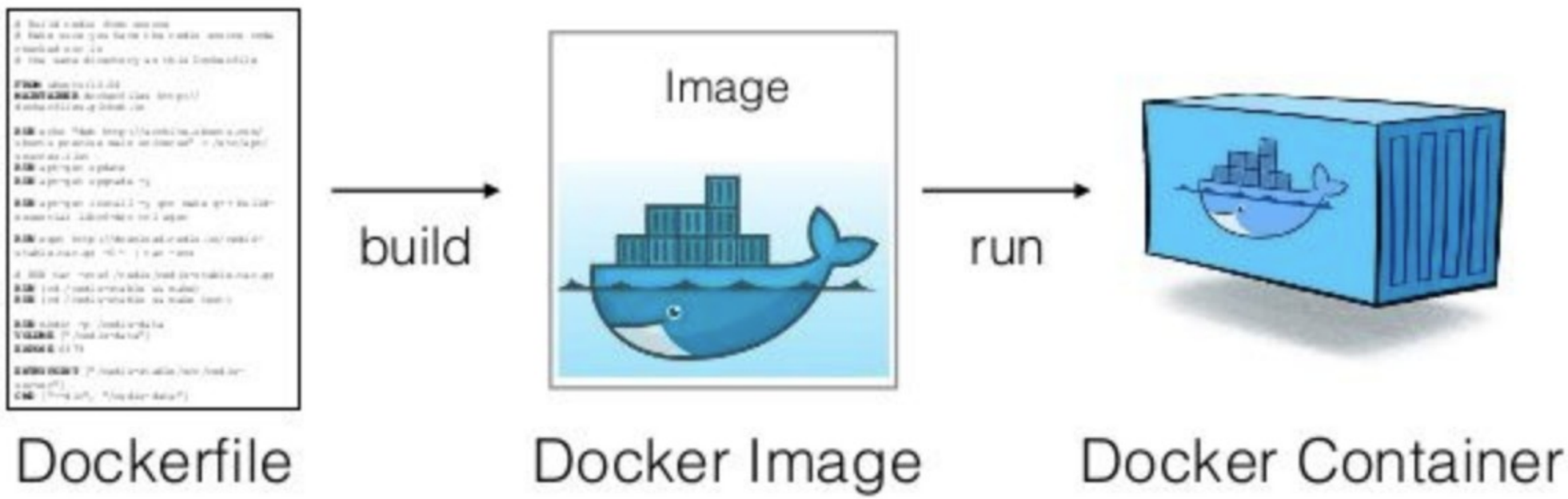
- Un contenedor compuesto de múltiples capas.
- Su capa superior es de lectura y escritura. El resto de capas son solo de lecturas y se les conoce como imágenes intermedias.
- Su capa mas baja se le conoce como capa base.
- Al realizar commit se guarda la ultima capa y referencia a su capa padre mas cercan.



Crear imagen a partir la instantánea de un contenedor

- Para crear una imagen a partir de la instantánea de un contenedor
  - **Docker commit <ID_Container> [-t <NombreRepositorio>[:tag]]**
    - La bandera -t es opcional, si no se indica crea la imagen con nombre y tag Nulo.
- Etiquetar una imagen
  - **Docker tag <ID_Image>[:Tag] <NombreRepositorio>[:Tag]**
- Autenticar la consola en Docker Hub
  - **Docker login**
- Subir a Docker Hub una imagen
  - **Docker push <NombreRepositorio>[:Tag]**

Mediante una ‘receta’ utilizando un archivo Dockerfile.



Mediante una ‘receta’ utilizando un archivo Dockerfile.

- El archivo debe de llamarse exactamente: **Dockerfile** (sin extension).
- El archivo se ejecuta con el comando Docker build <ruta> -t <Nombre>: <tag>
- Es un archivo con sintaxis YAML (Tabulado)
- Comandos:
  - **FROM**: Indica cual es la imagen base
  - **WORKDIR** : Indica el directorio al cual se le aplicaran los comandos RUN, CMD y ENTRYPOINT
  - **RUN**: Ejecuta comandos de consola

Mediante una ‘receta’ utilizando un archivo Dockerfile.

- Comandos:
  - **CMD**: Sirve para proporcionar valores predeterminados para un contenedor en ejecución
  - **ENTRYPOINT**: Sirve para configurar imágenes ejecutables.
  - **COPY**: Copia directorios y/o archivos de la maquina principal a la computadora.
  - **VOLUME**: Crea un volumen en el host y copia el contenido de la ruta indicada a ese volumen.
  - **ENV**: Establece variables de entorno
  - **EXPOSE**: Documenta que puertos utiliza la imagen.

Enlazar múltiples contenedores

Para ver la lección presione el icono…

Regresar al Indice

Siguiente Leccion

Orquestar contenedores

Para ver la lección presione el icono…

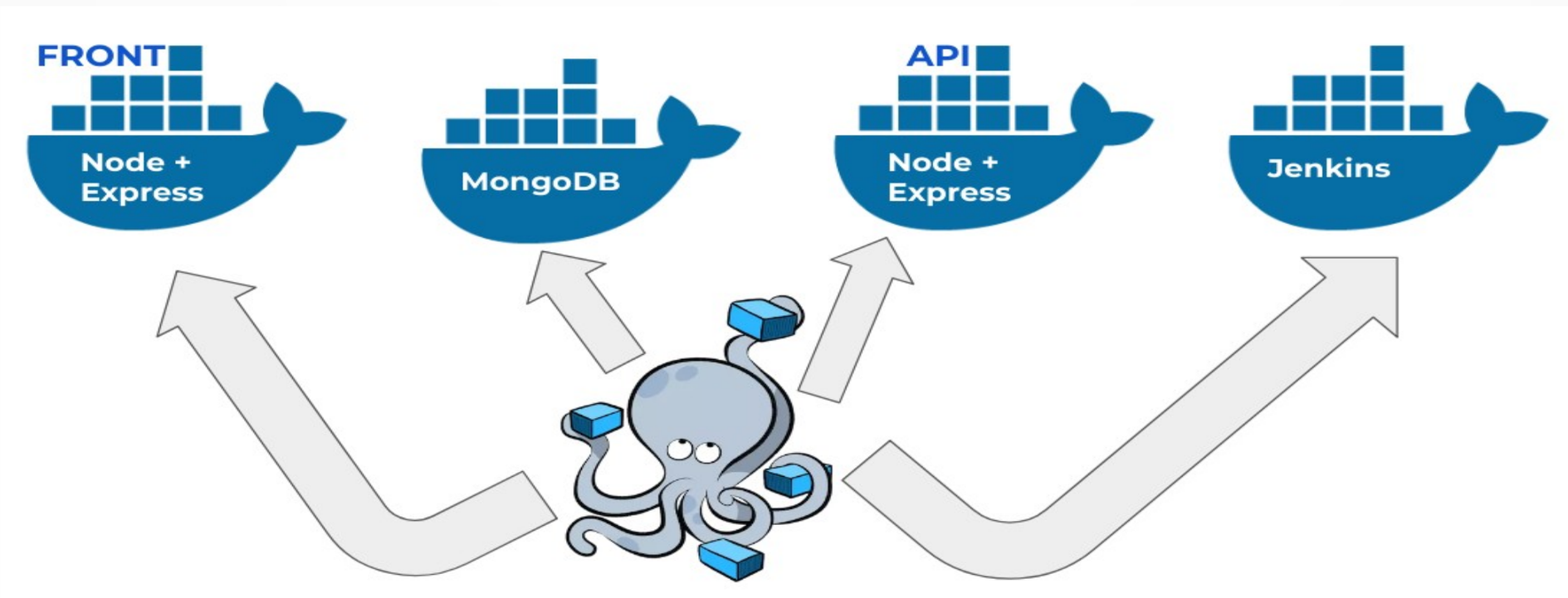
Regresar al Indice

Orquestar contenedores

- Cuando se despliega una aplicación orientada a micro-servicios, existe la necesidad de crear múltiples contenedores simultáneamente, pero crearlos manualmente utilizando docker run es completamente impráctico por lo cual se utiliza docker-compose para facilitar la definición y creación de contenedores.
- Docker compose es una herramienta de orquestación pero a menor escala.

Docker Compose

- Es una herramienta para definir y ejecutar múltiples contenedores utilizando código.



En que escenarios se recomienda utilizarlo?

- Ambientes de desarrollo
- Ambientes de prueba
- Despliegues en una sola maquina.

Porque docker-compose no es capaz de responder a alta disponibilidad y escalamiento al nivel que se requiere en un entorno de producción.

Orquestar contenedores con Docker Compose

- El archivo debe de llamarse exactamente: **docker-compose.yml**
- Es un archivo con sintaxis YAML (Tabulado)
- Sintaxis:

```
version: 2

services:

    NombreServicio1:

        atributo1: valor
        atributo2: valor
    NombreServicio2:

        atributo1: valor
        atributo2: valor
```

Atributos básicos para definir un servicio

- **container_name**: Nombre del contenedor
- **build:** Sirve para indicar que debe de construir la imagen a partir de un Dockerfile
- **image**: Indica que imagen utiliza el servicio.
- **expose**: Documenta puertos
- **ports**: Se indican una lista de puertos asociados al contenedor que se publicarán en la interfaz de red.
- **depends_on**: Se indica la lista de servicios que deben de estar funcionando para iniciar este servicio.
- **links**: Para comunicar varios contendores. (Legacy, es decir puede desaparecer en las nuevas versiones de docker compose)

Atributos básicos para definir un servicio

- **network**: Es una nueva opcion para comunicar varios contenedores. Se recomienda utilizar este atributo.
- **volume**: Se indican una lista de volumenes asociados al servicio.
- **environment**: Se indica una lista de variables de entorno
- **Utiles**
  - Extends: Para utilizar importar un servicio de algún archivo.
  - Command: Sobre escribe el comando por defecto de la imagen
  - Entrypoint: Sobre escribe el ejecutable de la imagen
  - Para mantener una consola activa
  - stdin_open:true
  - tty (-t):true

Comandos para ejecutar el archivo docker-compose

- Levantar todos los servicios
  - **Docker-compose up [-d]**
    - **-d para ejecutarlos en segundo plano.**
- Terminar todos los servicios.
  - **Docker-compose down [--volume, --build]**
    - **--volume:** elimina todos los volumenes creados
    - **--build:** forza a crear de nuevo las imagenes de los servicios que utilicen un Dockerfile (por defecto si no detecta cambios no lo vuelve a crear)

Comandos para ejecutar el archivo docker-compose

- Listar el estado de los servicios
  - **Docker-compose ps**
- Iniciar un servicio específico
  - **Docker-compose start <Nombre_Servicio>**
- Detener un servicio específico
  - **Docker-compose stop <Nombre_Servicio>**