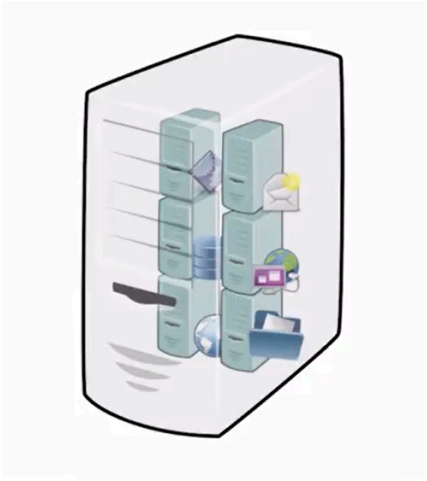
Minería de Datos II

Clase 4 - Virtualización y Docker

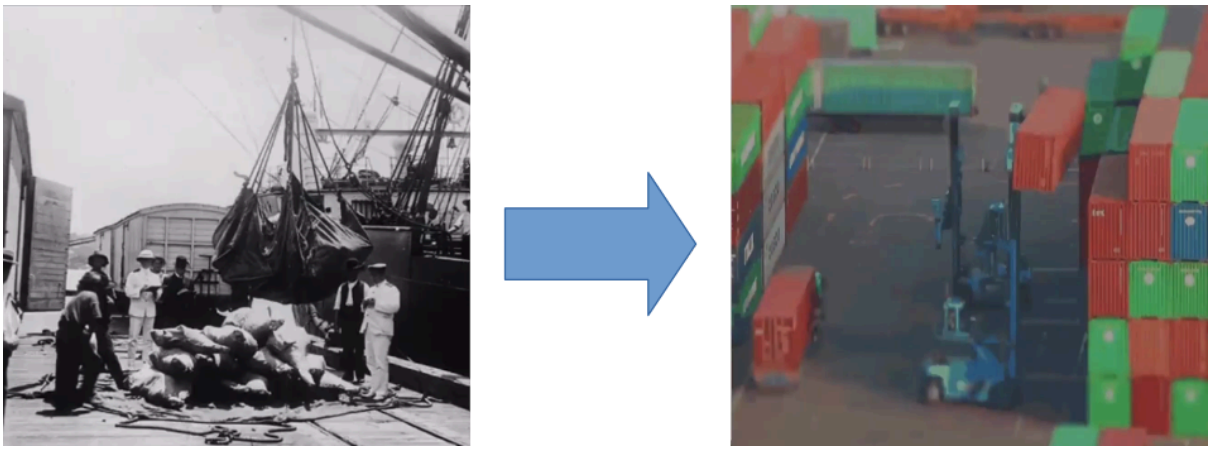
-Máquinas Virtuales

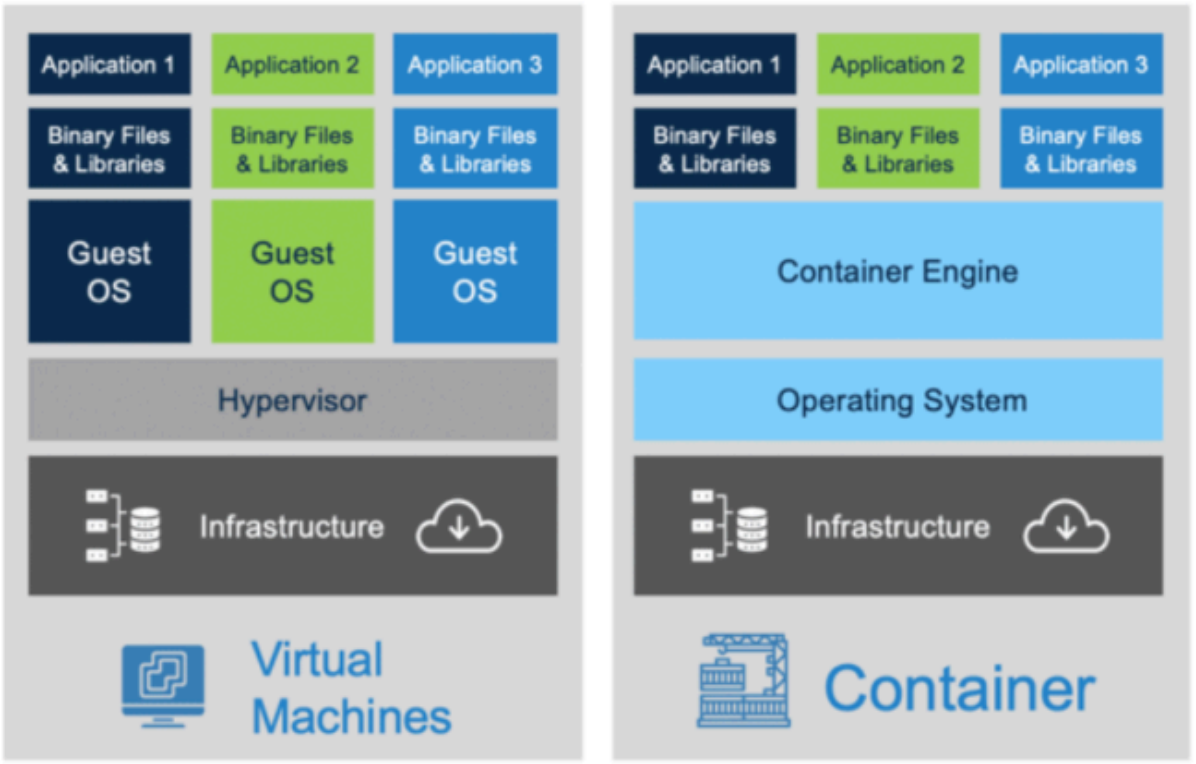
- Es la versión virtual de algún recurso tecnológico, como Hardware, un sistema operativo, un dispositivo de almacenamiento o recurso de red.
- Esa virtualización, es un sistema huésped que ejecuta sobre un sistema anfitrión, sin embargo tiene su propio sistema de archivos, que pueden tener múltiples formatos, como ser VDI, VMDK, VHD ó raw entre otros.



-Docker

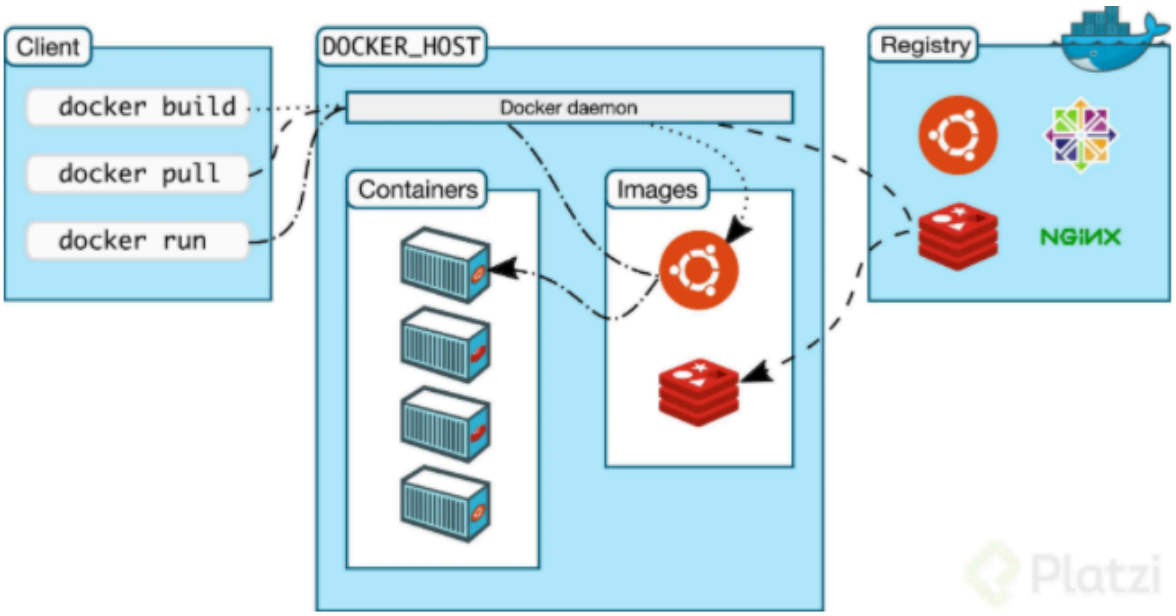
Utiliza contenedores, y lo que hacen es reutilizar el kernel, que es la parte mas profunda del SO de la máquina anfitriona, manejando de forma más óptima recursos que ya están disponibles en el SO anfitrión.
Esa containerización, trae consigo las ventajas de ser más liviana, portable, de bajo acoplamiento debido a que los contenedores son autocontenidos (no afecta a los demás para su funcionamiento), escalable y segura.





Corre nativamente en Linux, por eso para otros SO levanta una máquina virtual.

Componentes del Docker Engine:
• Docker daemon: Es el centro de docker, por medio del cual, es posible la comunicación con los servicios de docker.
• REST API: Como cualquier otra API, es la que nos permite visualizar docker de forma “gráfica”.
• Cliente de docker: Permite la comunicación con el centro de docker (Docker Daemon) que por defecto es la línea de comandos.



Dentro de la arquitectura de Docker encontramos:

1. Contenedores: Se encapsulan las imágenes para llevarlas a otra computadora o servidor, etc.
2. Imágenes: Se puede correr una aplicación específica.
3. Volúmenes de datos: Se puede acceder con seguridad al sistema de archivos de la máquina anfitrión.
4. Redes: Permiten la comunicación entre contenedores.

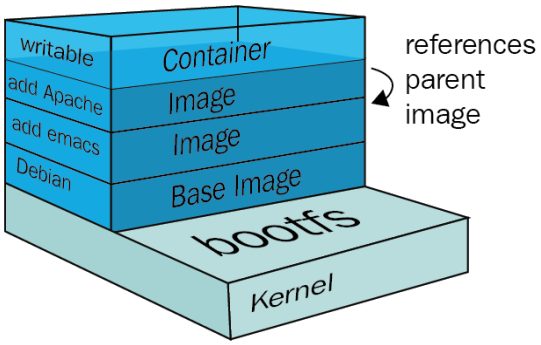
Es una arquitectura cliente-servidor, se comunican mediante una API para poder gestionar el ciclo de vida de los contenedores y así poder construir, ejecutar y distribuirlos.

-¿Qué es una imágen?

Se parte desde la base del SO Linux, y se agrega capas de personalización hasta obtener la imágen deseada:

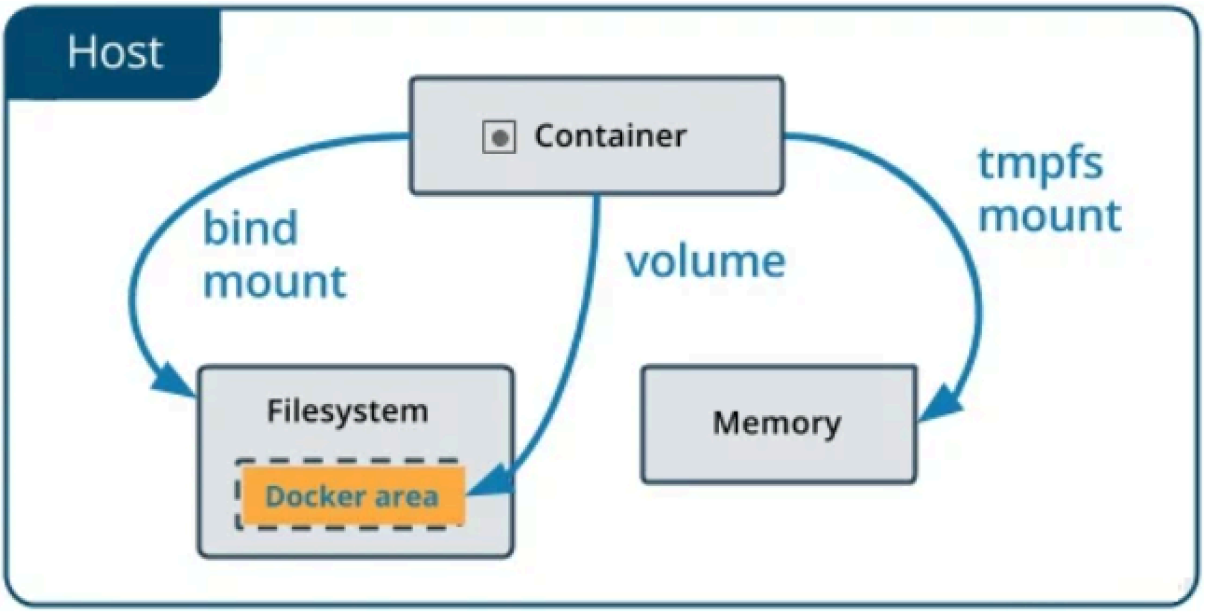
Ejemplo:

1. Distribución Debian
2. Editor emacs
3. Servidor Apache
4. Permisos de escritura para la carpeta /var/www de Apache



-¿Qué es un contenedor?

• Agrupación de procesos.
• Entidad lógica, no tiene el límite estricto de las máquinas virtuales.
• Ejecuta sus procesos de forma nativa.
• Los procesos que se ejecutan dentro de los contenedores ven su universo como el contenedor lo define, no pueden ver más allá del contenedor, a pesar de estar corriendo en una máquina más grande.
• No tienen forma de consumir más recursos que los que se les permite.
• Sector del disco: Cuando un contenedor es ejecutado, el daemon de docker establece a qué parte puede acceder.
• Docker hace que los procesos dentro de un contenedor estén aislados del resto del sistema, no le permite ver más allá.
• Cada contenedor tiene un ID único, también tiene un nombre.



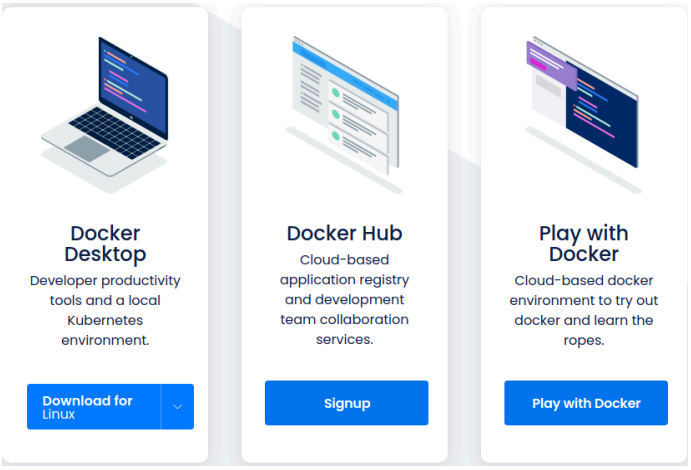
-Docker Compose

• Herramienta que permite simplificar el uso de Docker a partir de archivos YAML, con los que es más sencillo crear contenedores, conectarlos, habilitar puertos, volúmenes, etc.
• Se pueden crear diferentes contenedores y al mismo tiempo, en cada contenedor, diferentes servicios, unirlos a un volúmen común, iniciarlos y apagarlos, etc.
• Componente fundamental para poder construir aplicaciones y microservicios.
• Permite poder instruir al Docker Engine a realizar tareas, programáticamente siendo ésta la clave: La facilidad para dar una serie de instrucciones, y luego repetirlas en diferentes ambientes.
• Describe de forma declarativa la arquitectura de servicios necesaria en un archivo donde se declara lo que debe suceder.

Laboratorio:

Consideraciones generales:

- Es necesario tener instalado Docker.
- Registrarnos en Docker Hub.
- Al ejecutar las instrucciones, anteponer “sudo”.



https://hub.docker.com/

Laboratorio Docker 1:

1) Ejecutar en la consola el contenedor “hello-world” del Docker-Hub y luego verificar si está ejecutando:

1) $ docker run hello-world (corro el contenedor hello-world)
2) $ docker ps (muestra los contenedores activos)

2) Ejecutar una inspección de un contenedor específico

1) $ docker ps -a (muestra todos los contenedores)
2) $ docker inspect \<container ID\> (muestra el detalle completo de un contenedor)
3) $ docker inspect \<name\> (igual que el anterior pero invocado con el nombre)

3) Ejecutar el contenedor “hello-world” asignándole un nombre distinto.

1) $ docker run -d –-name hola-mundo hello-world (le asigno un nombre custom “hola-mundo”)
2) $ docker rename hola-mundo hola-a-todos (cambio el nombre de hola-mundo a hola-a-todos)

4) Ejecutar la eliminación de un contenedor (usar rm y prune)

1) $ docker rm \<ID o nombre\> (borro un contenedor)
2) $ docker container prune (borro todos lo contenedores que estén parados)
3) Explorar Docker Hub y probar ejecutar alguna de las imágenes.
https://hub.docker.com/

5) Ejecutar la imagen “ubuntu”:

1) $ docker run ubuntu (corre un ubuntu pero lo deja apagado)
2) $ docker run -it ubuntu (lo corre y entro al shell de ubuntu)
-i: interactivo

-t: abre la consola
corre el siguiente comando en la consola de linux $ cat /etc/lsb-release (veo la versión)

6) Ejecutar la imagen “nginx” y probar los comandos “stop” y “rm”

1) $ docker run -d --name proxy nginx (corro un nginx)
localhost:8081 (Desde mi navegador compruebo que funcione. Lo hace? por qué?)
2) $ docker stop proxy (apaga el contenedor)
3) $ docker rm proxy (borro el contenedor)
4) $ docker rm -f \<contenedor\> (lo para y lo borra)

7) Ejecutar nginx exponiendo el puerto 8080 de mi máquina

1) Exponer contenedores:
$ docker run -d --name proxy -p 8081:80 nginx (corro un nginx y expongo el puerto 80 del contenedor en el puerto 8080 de mi máquina)
2) localhost:8081 (Desde mi navegador compruebo que funcione nuevamente)

8) Ejecutar el comando logs para ver los logs del contenedor de nginx:

1) $ docker logs proxy (veo los logs)
2) $ docker logs -f proxy (hago un follow del log)

9) Ejecutar comando “logs –tail” para ver las últimas N entradas de log

1) $ docker logs --tail 10 -f proxy (veo y sigo solo las 10 últimas entradas del log)

Laboratorio Docker 2:

1) Ejecutar la imagen “mongodb” y asociarla con un directorio en mi máquina

1) $ mkdir dockerdata (creo un directorio en mi máquina)
2) $ cd dockerdata (me ubico dentro del directorio creado)
3) $ docker run -d --name mongodb -v "$(pwd)":/data/db mongo:4.4
(Corro un contenedor de mongo y creo un bind mount. El comando "pwd" retorna la ubicación donde estamos, se usa luego del parámetro "-v" para dar la ubicación dentro de mi máquina)
De arrojar error Exited (132) debemos usar otra versión (4.4 por ejemplo) usando mogno:X.X en vez de mongo solamente.2

2) Ejecutar el comando “exec” para introducirse en el shell de un contenedor:

1) $ docker ps (veo los contenedores activos)
2) $ docker exec -it mongodb bash (entro al bash del contenedor)

3) Ejecutar los siguientes comandos:

1) $ mongo (me conecto a la base de datos)
2) show dbs (listo las bases de datos)
3) use prueba (creo la base “prueba”)
4) db.prueba.insert({"color":"azul"}) (inserto un nuevo dato)
5) db.prueba.find() (veo el dato que cargué)
6) Salir del contenedor de Mongo (comando "exit") y revisar el contenido del directorio "dockerdata"
7) Detener y volver a ejecutar el contenedor mongodb y verificar que el dato insertado en una ejecución previa ya se pueda ver, debido a que la nueva ejecución levanta lo datos ligados mediante Bind.
Se debe usar el comando "docker container stop mongodb" y luego "docker container start mongodb".

4) Crear un volúmen y volver a ejecutar los pasos del punto anterior, pero esta vez ligando el contenedor con el volúmen creado

1) $ docker volume create dockerdata-vol
2) $ docker run -d --name mongodb --mount type=volume,source=dockerdata-vol,target='/data/db' mongo:4.4
3) $ docker exec -it mongodb bash (ingresar al contenedor)
4) $ mongo (me conecto a la base de datos)
5) show dbs (listo las bases de datos)
6) use prueba (creo la base “prueba”)
7) db.prueba.insert({"color":"azul"}) (inserto un nuevo dato)
8) db.prueba.find() (veo el dato que cargué)
9) Salir del contenedor de Mongo (comando "exit") y detenerlo

5) Volver a ejecutar un contenedor de Mongo ligado al volumen creado en el punto anterior pero con otro nombre, ingresar y chequear que se vean los datos ya cargados

1) $ docker run -d --name mongodb2 --mount type=volume,source=dockerdata-vol,target='/data/db' mongo:4.4
2) $ docker exec -it mongodb2 bash (ingresar al contenedor)
3) $ mongo (me conecto a la base de datos)
4) show dbs (listo las bases de datos)
5) use prueba (creo la base “prueba”)
6) db.prueba.find() (veo el dato que cargué)

6) Utilizar la opción "mount" con el parámetro "bind" para ligar a un contenedor de mongo, la carpeta creada

1) $ docker run -d --name mongodb3 --mount type=bind,source='/home/ubuntu/dockerdata',target='/data/db' mongo:4.4

7) Ejecutar un contenedor de Ubuntu y copiar desde el mismo un archivo hacia tu máquina y viceversa.

1) $ docker run -d --name ubuntu_test ubuntu tail -f /dev/null

2) $ docker exec -it ubuntu_test bash
3) En el contendedor, se crea el directorio “test”: $ mkdir test
4) Se crea un archivo dentro de esa carpeta: $ touch /test/archivo.txt
5) Al salir del contenedor se copia el archivo creado en la maquina anfitrión: $ docker cp ubuntu_test:/test/archivo.txt archivo.txt
6) Copiar desde la máquina anfitrión hacia el contenedor:
  * $ touch archivo2.txt
  * $ docker cp archivo2.txt ubuntu_test:/test/archivo2.txt