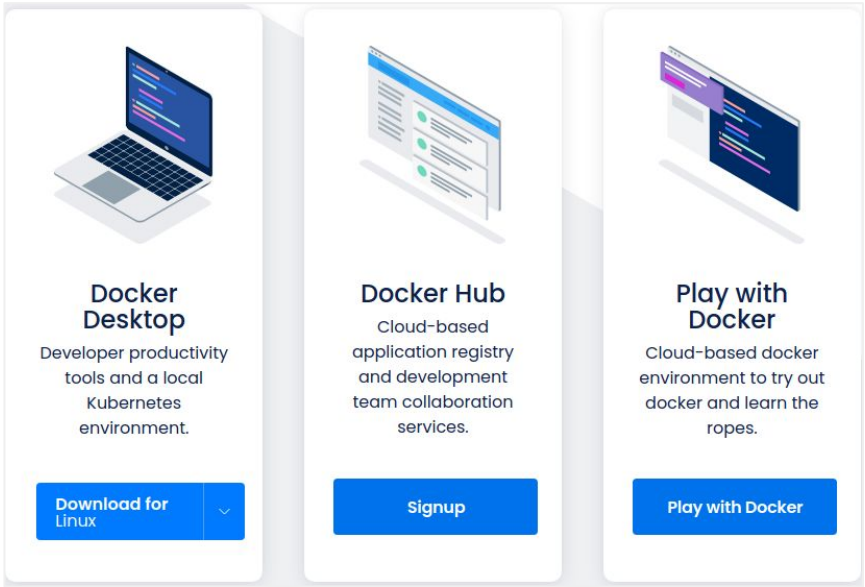
Big Data con Hadoop y Spark

Módulo 02 - Resolución del desafío

Consideraciones

1. Es necesario tener instalado **Docker.**
2. Registrarnos en Docker Hub. https://hub.docker.com/
3. Al ejecutar las instrucciones, anteponer **“sudo”**.



Resolución del ejercicio 1

1. La parte del disco donde Docker crea los volúmenes es accesible sólo por Docker, por lo tanto es más seguro y ordenado.

   a. **`$ docker volume ls`** (ver los volúmenes creados).

   b. **`$ docker volume create dbdata`** (crea el volumen de nombre *"dbdata"*).

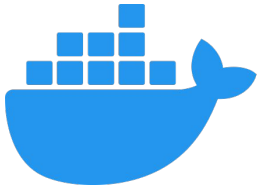
   c. **`$ docker run -d --name db --mount src=dbdata,dst=/data/db mongo`** (monta el volumen *"dbdata"* al contenedor en el destino **/data/db** y ejecuta la base de datos Mongo).

d. **`$ docker exec -it db bash`** (ingresar al contenedor)

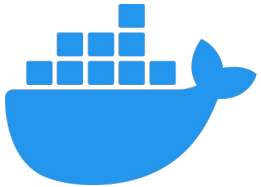
e. **`$ mongo`** (conectarse a la BBDD)

f. **`shows dbs`** #se listan las BBDD
`use prueba` #se crea la BBDD prueba
`db.prueba.insert({“color”:“azul”})` #se carga un dato
`db.prueba.find()` #se visualiza el dato cargado

g. Y al crear un nuevo contenedor, se usa el mismo
`$ docker run -d --name db --mount src=dbdata,dst=/data/db mongo`



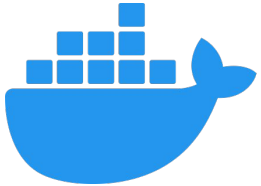
2. **`$ docker run -d --name ubuntu_test ubuntu tail -f /dev/null`**
   (correr ubuntu)
3. **`$ docker exec -it ubuntu_test bash`** (acceder al bash)
4. En el contendedor, se crea el directorio **“test”**, al salir del contenedor para copiar un archivo dentro del contenedor:
   `$ docker cp test.txt ubuntu_test:test`
5. Copiar desde el contenedor a la máquina anfitrión:
   `$ docker cp ubuntu_test:test [carpeta local]`



Ejercicio 2: Docker

Contienen distintas capas de datos (distribución, diferente software, librerías y personalización).

1. **`$ docker image ls`** (ver las imágenes que tengo localmente).
    a. **`$ docker pull ubuntu:20.04`** (bajo la imagen de ubuntu con una versión específica).
2. **`$ mkdir imagenes`** (creo un directorio en mi máquina).
    a. **`$ cd imagenes`** (entro al directorio).
    b. **`$ touch Dockerfile`** (creo un Dockerfile).
    c. **`$ vi Dockerfile`** (abro code en el directorio en el que estoy).



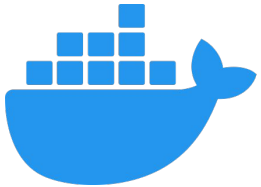
3. **##Contenido del Dockerfile##**
   FROM ubuntu:latest
   RUN touch /ust/src/hola.txt (comando a ejecutar en tiempo de build)
   ##fin##

4. **$ docker build -t ubuntu:ubuntu2** (creo una imagen con el contexto de build <directorio>).

   a. **$ docker run -it ubuntu:ubuntu2** (corro el contenedor con la nueva imagen).

   b. **$ docker login** (me logueo en docker hub).

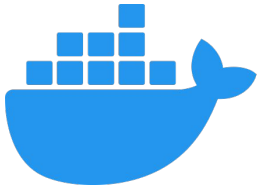
   c. **$ docker tag ubuntu:ubuntu2 miusuario/ubuntu:ubuntu2** (cambio el tag para poder subirla a mi docker hub).



5. **`$ docker push miusuario/ubuntu:ubuntu2`**
   (publico la imagen a mi docker hub).

La importancia de entender el **sistema de capas** consiste en la **optimización de la construcción del contenedor para reducir espacio** ya que cada comando en el dockerfile crea una capa extra de código en la imagen.

Con **docker commit** se crea una nueva imagen con una capa adicional que modifica
la capa base.



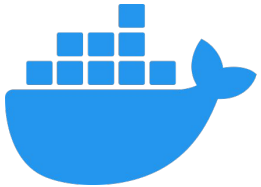
Ejercicio 3: Imágenes

1. Ejemplo: crear una nueva imagen a partir de la imagen de Ubuntu.

   `docker pull ubuntu`
   `docker images`
   `docker run -it cf0f3ca922e0 bin/bash`
   (modificar el contenedor: Ej `apt-get install nmap`)
   `docker commit deddd39fa163 ubuntu-nmap`

2. **`$ docker history ubuntu:ubuntu2`** (ver la info de cómo se construyó cada capa).

3. **`$ dive ubuntu:ubuntu2`** (ver la info de la imagen con el programa dive).



¡Terminaste el módulo!

Estás listo para rendir el examen