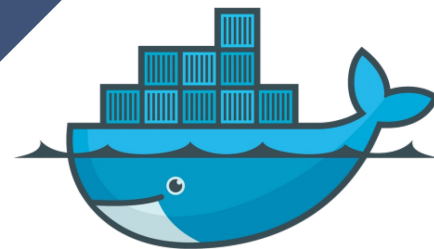


Docker de 0 a 100

Israel Oña Ordoñez | DevOps Engineer



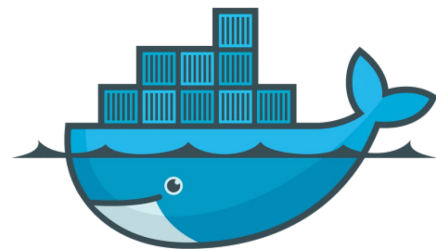
devisra



1

Fundamentos

Aspectos básicos





¿Qué es Docker?

Docker es una **herramienta** que permite implementar fácilmente las aplicaciones en un espacio aislado (contenedores) para ejecutarlas en un SO host.

El principal **beneficio** de Docker es que permite empaquetar una aplicación en un objeto denominado imagen con todas sus dependencias.

Contenedor

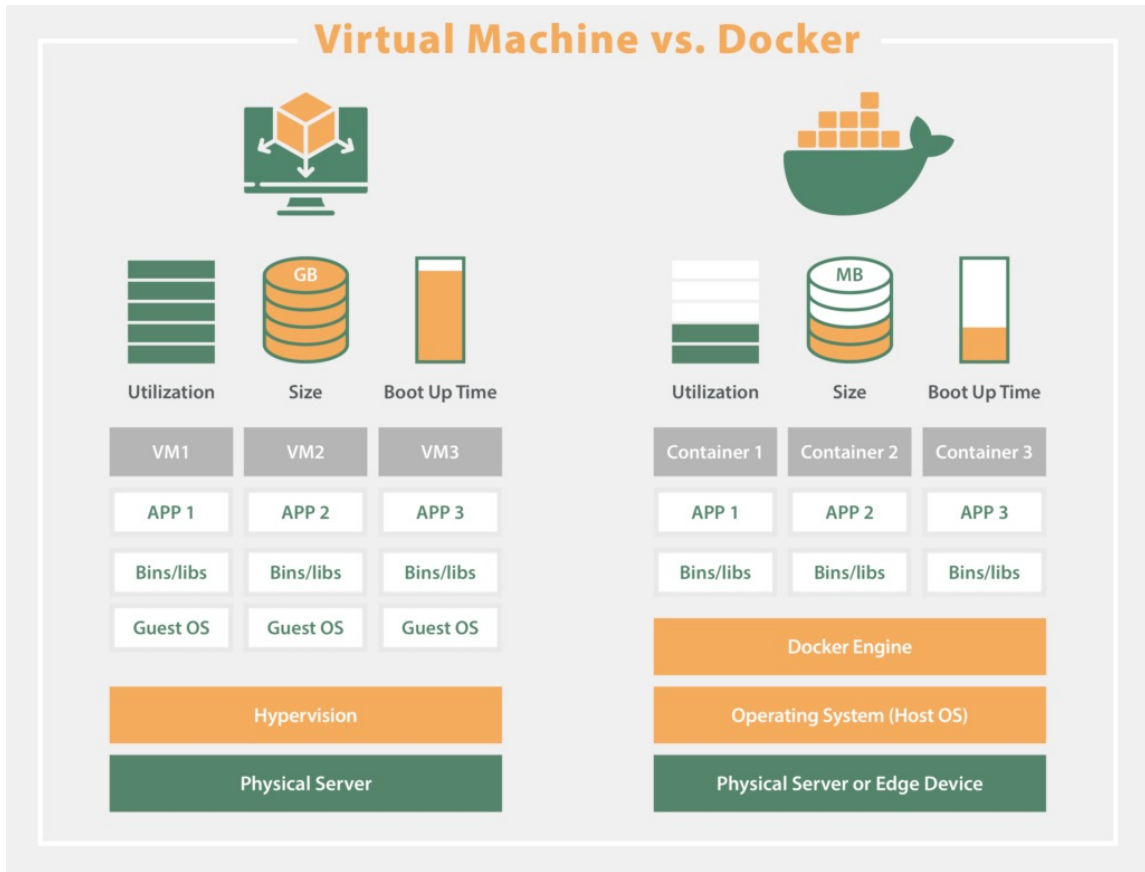
Es la instancia de una imagen ligera e independiente que contiene todo lo necesario para ejecutar una aplicación (código, runtime, herramientas del sistema, bibliotecas)

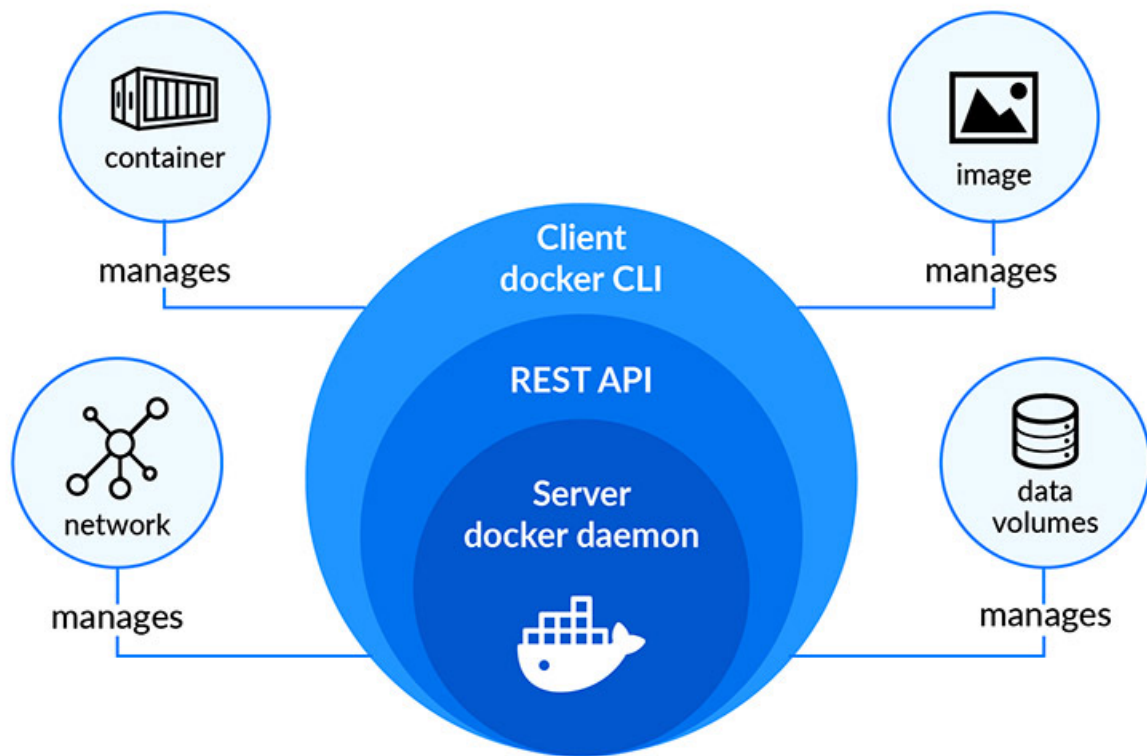
- En mi computadora si funciona
- Si pero no le vamos a dar tu computadora al cliente

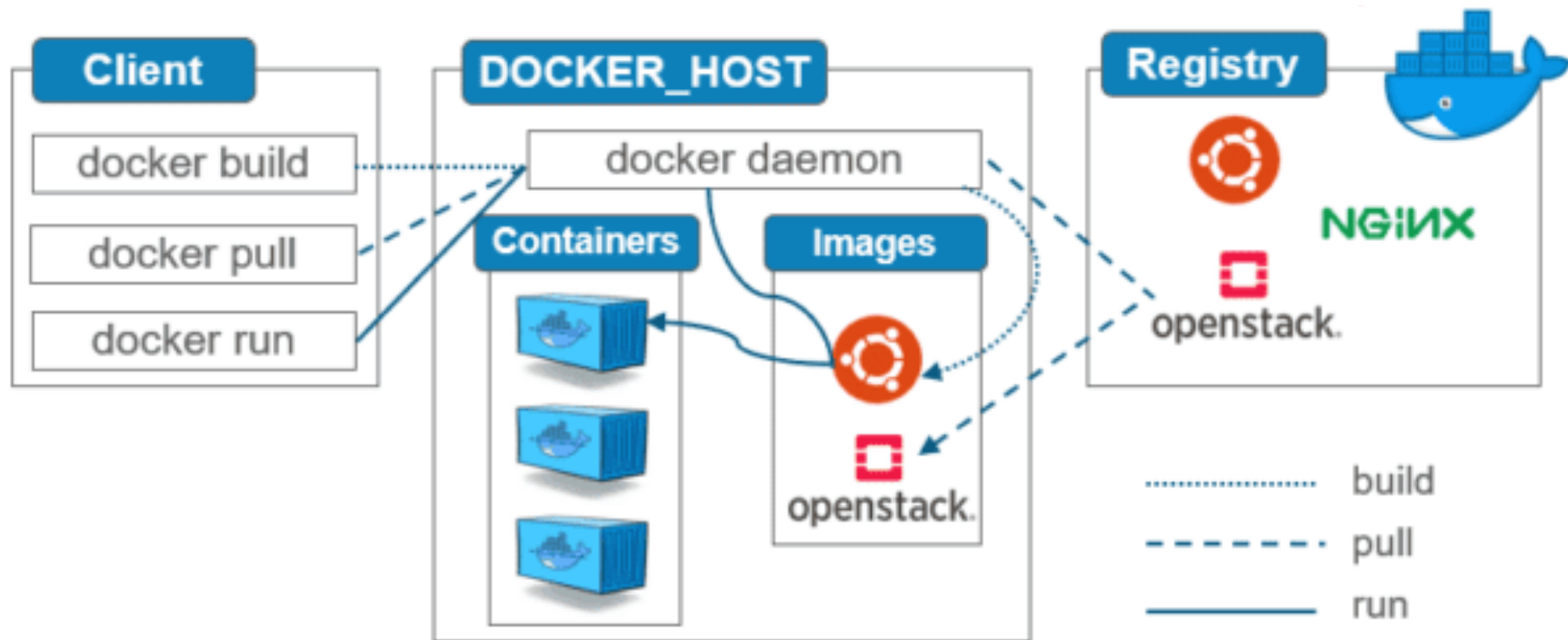


Las **máquinas virtuales** son excelentes para proporcionar aislamiento completo, pero a un gran costo, la **sobrecarga** computacional gastada en la virtualización del hardware para el SO invitado.

Los **contenedores** brindan la mayor parte del aislamiento de las máquinas virtuales pero a una **fracción** de la potencia computacional.







docker run hello-world

```
> docker run hello-world
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://hub.docker.com/>

For more examples and ideas, visit:

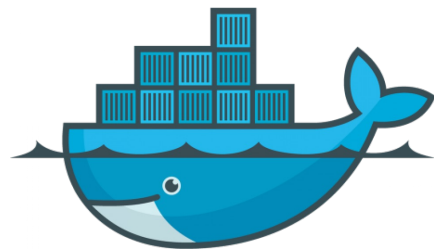
<https://docs.docker.com/get-started/>



2

Comandos principales

Navegación entre contenedores e imágenes





docker run

Crea y ejecuta un contenedor a partir de una imagen

Crea un contenedor

Enciende un contenedor

`docker run nginx`

`docker create nginx`

`docker start container_<id/name>`

Opciones

`--detach, -d`: Corre el contenedor en segundo plano (imprime el ID del contenedor)

`--name`: Asigna un nombre al contenedor

`-it`: Ejecuta el contenedor en modo interactivo

`--env, -e`: Establece una variable de entorno

`docker run --detach nginx` / `docker run -d nginx`

`docker run --name my_nginx nginx`

`docker run -it ubuntu bash`

`docker run -e var=val nginx`



docker ps

Lista los contenedores, por defecto solo los que estén en estado “running”

```
docker ps
```

Opciones

--all, -a: Lista todos los contenedores, independientemente de su estado

```
docker ps -a
```

--no-trunc: No trunca el ID del contenedor en la salida

```
docker ps --no-trunc
```

--quiet, -q: Solo muestra el ID de los contenedores

```
docker ps -q
```

--size, -s: Muestra el tamaño total de los archivos

```
docker ps -s
```



docker stop

Detiene uno o más contenedores en ejecución

```
docker stop container1_<id/name> \  
container2_<id/name>
```



docker rm

Elimina uno o más contenedores, por defecto solo se pueden eliminar contenedores cuyo estado no sea "running"

Opciones

--force, -f: Fuerza la eliminación de un contenedor que se esté ejecutando

```
docker rm container1_<id/name> \  
container2_<id/name>
```

```
docker rm -f container_<id/name>
```



docker images

Lista las imágenes disponibles en el docker host, por

`docker images`

Opciones

--all, -a: Lista todas las imágenes, por defecto se ocultan las intermedias

`docker images -a`

--digests: Muestra el digest (identificador único e inmutable)

`docker images --digests`

--no-trunc: No trunca el ID de la imagen en la salida

`docker images --no-trunc`

--quiet, -q: Solo muestra el ID de las imágenes

`docker images -q`



docker rmi

Elimina una o más imágenes, por defecto solo se pueden eliminar imágenes que no estén siendo ocupadas por algún contenedor, independientemente de su estado

Opciones

--force, -f: Fuerza la eliminación de una imagen aunque algún contenedor la esté usando (también se eliminan el o los contenedores que la estén usando)

```
docker rmi image1_<id/name> \  
image2_<id/name>
```

```
docker rmi -f image_<id/name>
```



docker pull

Descarga una imagen de un repositorio, por defecto Docker Hub

```
docker pull image_name
```

Opciones

--all-tags, -a: Descarga todas las versiones (tags) de la imagen en el repositorio

```
docker pull image_name -a
```

--quiet, -q: No muestra el detalle de la descarga en la consola

```
docker pull image -q
```



docker image prune

Elimina todas las imágenes sin usar, aquellas cuyas capas no guardan relación con las imágenes tagueadas (dangling images)

`docker image prune`

Opciones

--all, -a: Elimina todas las imágenes sin usar, no solo las dangling images sino también las que no están referenciadas en ningún contenedor, sin importar su estado

`docker image prune -a`

--force, -f: No pide confirmación para ejecutar el comando

`docker image prune -f`



docker container prune

Elimina todos los contenedores detenidos, sin importar la razón por la cual se detuvieron

Opciones

--force, -f: No pide confirmación para ejecutar el comando

docker container prune

docker container prune -f



docker inspect

Muestra la información de un objeto de docker
(imagen, contenedor, volume o red)

```
docker inspect object _<id/name>
```

Opciones

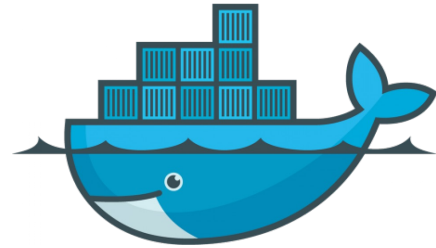
--format, -f: No pide confirmación para ejecutar el
comando

```
docker inspect -f='{{PropertyPath}}' object  
_<id/name>
```

3

Mapeo de puertos y volúmenes

Exposición de puertos del contenedor al host y persistencia de datos

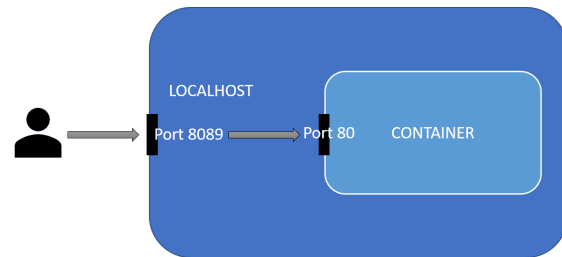




Port Mapping

Para establecer comunicación con un contenedor desde el host es necesario realizar un **mapeo de puertos**.

Se abre un puerto desde el host (8089) para tener acceso a un puerto abierto dentro del contenedor (80), de esta manera todas la solicitudes que se realizan al puerto del host se redirigen al puerto del contenedor.



El mapeo de puertos solo se puede realizar el momento de la instanciación del contenedor.

```
docker run -p 8089:80 nginx
```

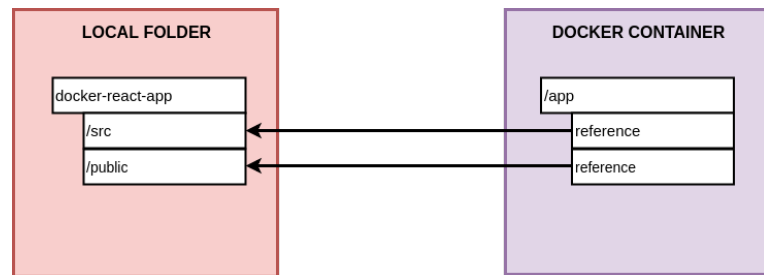
Generalmente se debe evitar abrir puertos de host para mantener los servicios del contenedor privados o solo visibles para el resto de contenedores y solo los servicios de frontend deberían exponerse.



Volume Mapping

Por definición, los contenedores son objetos que se crean de manera aislada y una vez que se eliminan, toda la data que estos contienen también será eliminada.

El uso de volúmenes representa una manera de preservar la data que los contenedores generan, evitando así que esta se pierda, y si el contenedor se elimina pero se crea otro a partir de la misma imagen con la misma configuración, se pueda continuar la operación con la data previamente creada.

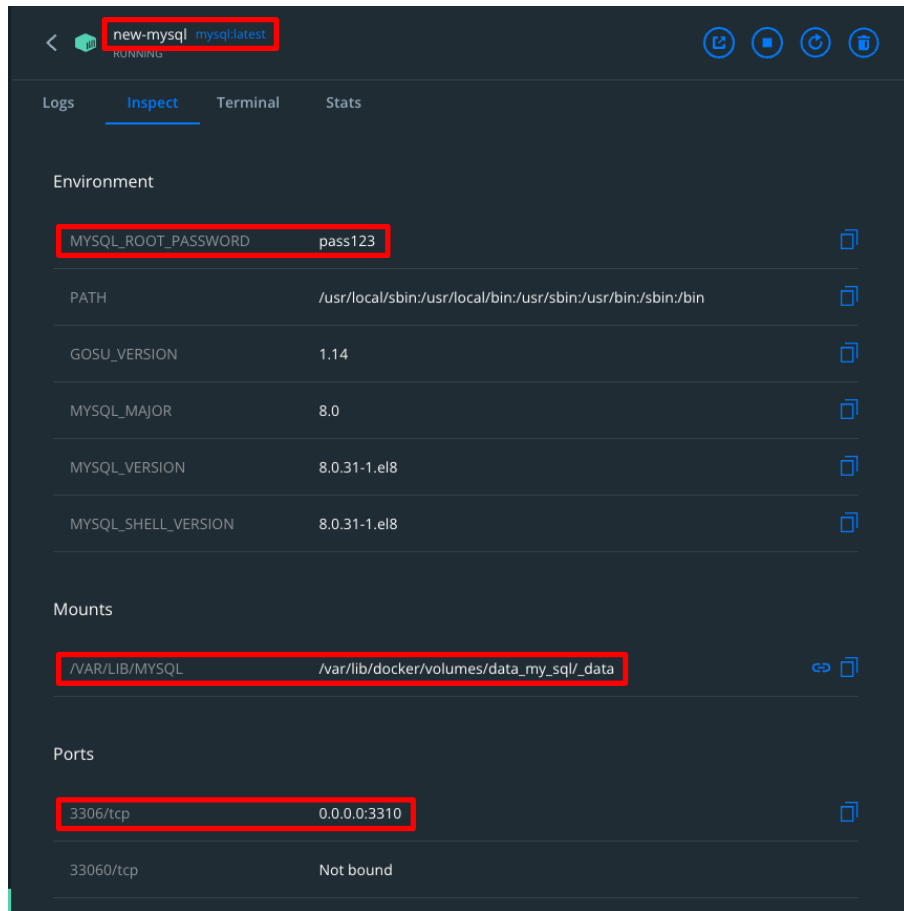


```
docker run -v /datadir:/var/lib/mysql mysql
```

En linux se puede acceder a los volúmenes desde `/var/lib/docker/volumes`.

En MacOS es más complicado debido a que Docker para Mac corre sobre una máquina virtual solo que esta está "oculta" para simplificar su uso.

MySQL instance



The screenshot shows the Docker Desktop interface for a container named 'new-mysql' using the 'mysql:latest' image. The container is in a 'RUNNING' state. The 'Inspect' tab is selected, showing the following configuration:

- Environment:**
 - `MYSQL_ROOT_PASSWORD` is set to `pass123`.
 - `PATH` is `/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin`.
 - `GOSU_VERSION` is `1.14`.
 - `MYSQL_MAJOR` is `8.0`.
 - `MYSQL_VERSION` is `8.0.31-1.el8`.
 - `MYSQL_SHELL_VERSION` is `8.0.31-1.el8`.
- Mounts:**
 - `/VAR/LIB/MYSQL` is mounted to `/var/lib/docker/volumes/data_my_sql/_data`.
- Ports:**
 - `3306/tcp` is mapped to `0.0.0.0:3310`.
 - `33060/tcp` is 'Not bound'.



```
docker run -d \  
  --name mysql_instance \  
  -e MYSQL_ROOT_PASSWORD=pass123 \  
  -v data_my_sql:/var/lib/mysql \  
  -p 3310:3306 \  
  mysql:latest
```