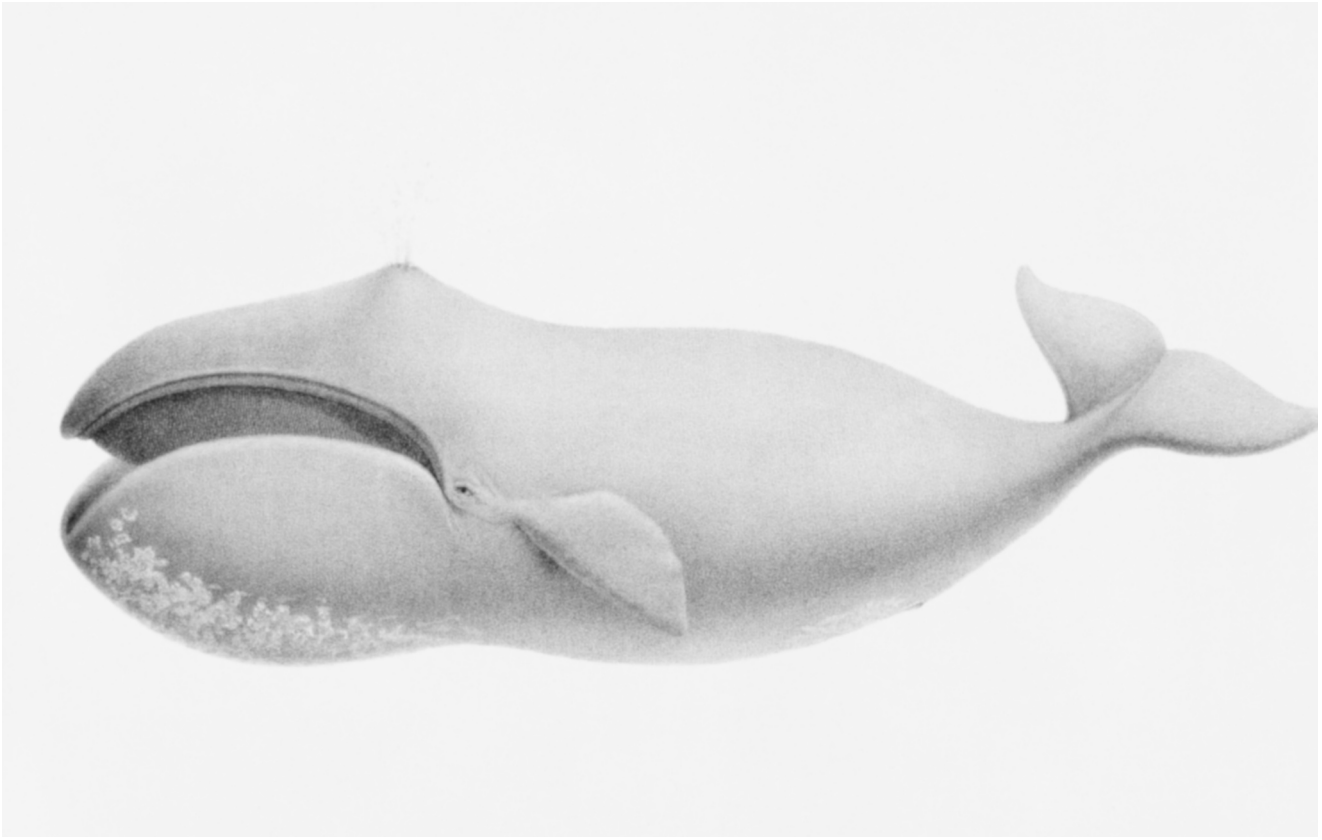
DOCKER



Primeros pasos

Esta publicación fue escrita en marzo de 2020 en Buenos Aires, Argentina. Las imagened usadas son de dominio publico, obtenidas del sitio www.rawpixel.com, las imagenes de los cetáceos son del archivo son originales de **Charles Melville Scammon's "Natural History of the Cetaceans and other Marine Mammals of the Western Coast of North America" (1872)** y la imagen del pinguino **Vintage Victorian style penguins engraving. Original from the British Library**. El texto se encuentra bajo licencia Creative Commons atribución no comercial.

Version 0.1

¿Qué es Docker?

Docker Inc es una compañía que desarrolla tecnología para el trabajo con contenedores Linux, Docker hace uso de Linux Containers(LXC). Anteriormente era desarrollado por DotCloud, la cual cambia su nombre de Docker Inc en Octubre de 2013. DotCloud era una empresa que venía trabajando con contenedores desde 2008.

Docker Inc tiene varias soluciones como Docker Swarm, Docker Compose, Docker Registry. Nosotros veremos algunas de estas en este zine, ya que no alcanza un tomo para todo.

Docker como tecnología de container nos permite trabajar en diferentes arquitecturas(on premise y cloud), esto es debido que el container nos permite aislar el código(isolation) que está dentro del imagen con el resto del sistema operativo anfitrión (Host). Permitiendo crear ambientes con bibliotecas, paquetes específicos(.deb, .rpm,.apk, etc), configuraciones, etc como un sistema operativo completo. Esto nos permite mantener el mismo ambiente para todo los estadíos de vida del software, desde el desarrollo hasta su puesta en producción, ganando agilidad en todo el ciclo.

Docker marcó un estándar de container es por ello que junto a Linux Foundation y otros desarrolladores de productos similares crearon OCI (open container Initiative), para establecer un estándar en conjunto. Al ser un desarrollado en Golang, permite su optimización en cuanto a recursos y tener una gran comunidad de desarrollo.

Container Linux

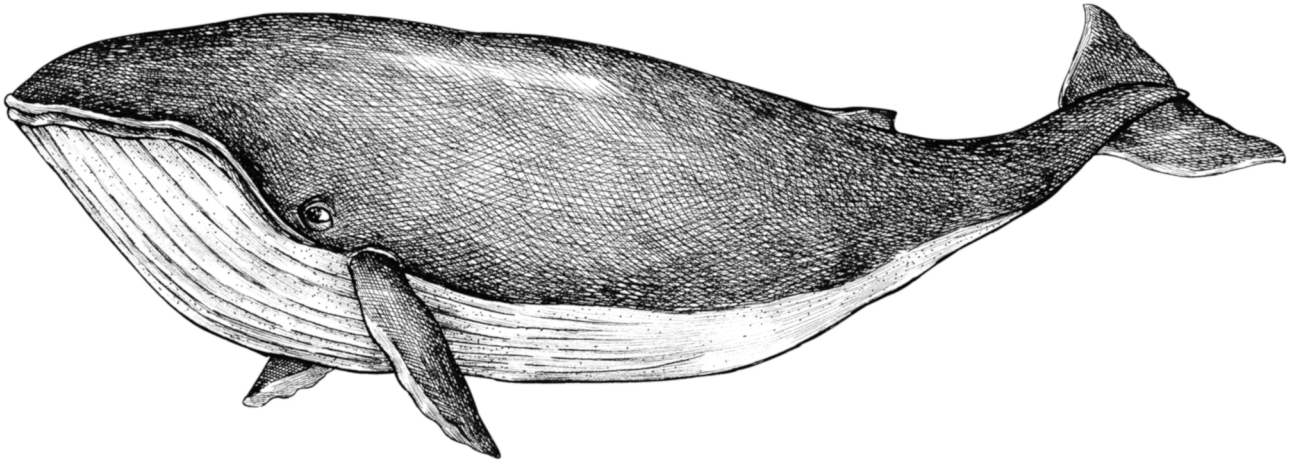
Los containers en Linux tienen origen en 2008, con la versión de kernel 2.6.29, que es un método de virtualización a nivel sistema operativo. Lo que difiere de los sistemas de virtualización tradicionales que hacen uso del hypervisor,lo cual permite emular el hardware posibilitando que varios sistemas operativos corran a la vez sobre un mismo host. La virtualización a nivel container hace uso del kernel de host, es decir que el contenedor que corramos en nuestro host debe ser de la misma arquitectura de microprcesador. Si en nuestro host tenemos una arquitectura x64, debemos correr un contenedor x64,lo mismo para ARM por ejemplo.

Esta tecnología hace uso de los cgroups, lo cual nos permite controlar los recursos de los mismos. Se puede ver más información de cgroups en http://man7.org/linux/man-pages/man7/cgroups.7.html .
Muchas empresas como Google, hacen uso de los contenedores desde su aparición, ya es una tecnología de más de 10 años al escribir esto. Si embargo, su gran auge fue en el 2013 con Docker.

Imagenes en Docker

Venimos nombrando el concepto imagen, ¿pero qué significa?

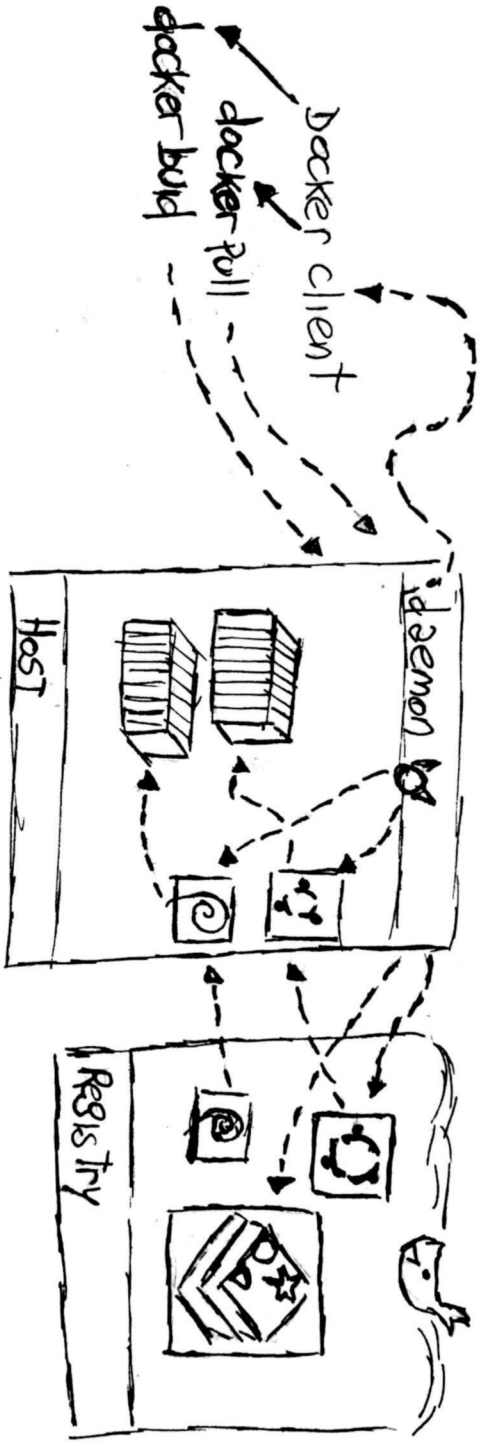
Una image es un archivo inmutable, que contiene las especificaciones establecidas en la creación , bibliotecas y código de nuestra app si es que agregamos . Las Imágenes pueden ser basadas en un S.O GNU/LINUX existente como Ubuntu, Debian, Fedora o un S.O Linux como Alpine. Aclaro esto, porque Alpine es muy usado dentro de la comunidad pero el mismo no está basado en GNU, lo cual no cuenta con bibliotecas básicas del proyecto GNU.
Cuando creamos una imagen , se genera un hash único que nos permite verificar la integridad de la misma. Cada imagen debe tener un tag, esto nos permite versionar la misma, más adelante veremos bien el uso de los tag.
Podemos contar con varias imágenes con software especifico como MySQL, Redis entre otros como también un S.O base basado en Debian. Estas imágenes se pueden encontrar en Docker Hub . Existen otros registrys públicos pero no vamos ahondar sobre ellos.



Docker Registry y repositorios

Un registry es un software que nos permite contener las distintas imágenes de Docker. Cada imagen se encuentra dentro de un repositorio el cual está dentro de un registry . La estructura seria registry.com/repositorio/imagen.
Hay distintas soluciones que nos da el servicio de registry, tanto públicos como privados. Podemos tener nuestro propio registry o utliza hub.docker.com, como habia menciado es el más popular pero tiene sus limitaciones como todo producto, a momento de escribir esto solo permite un repositorio privados en su versión free, pero según el plan contratado varía.

Docker Arquitectura



Instalación

Contamos con dos versiones de Docker, la versión community y entreprise. Nosotros vamos a instalar Docker CE. Desde GNU/ Linux, es relativamente fácil, debes descargar el script de instalación desde https://get.docker.com , guardarlo y ejecutarlo, por ejemplo:

```
$ curl https://get.docker.com –o /home/$USER/docker.sh

$ sudo sh  docker.sh
```

Esto nos instala segun la version de nuestro S.O la última versión estable de Docker. Para comprobar la versión instalada, solo hace falta ejecutar el siguiente comando:

```
$ sudo docker -- version
```

Nos deberá devolver la versión exacta de docker instalada. Si no queremos usar sudo para invocar al cliente de docker, puede usar el siguiente comando:

```
$ sudo usermod –aG docker your–user
```

Esto no es recomendable en un entorno productivo, porque está asignando al grupo docker un usuario con acceso root, el cual puede ser usado desde un contenedor para obtener root del host.

Hola mundo!

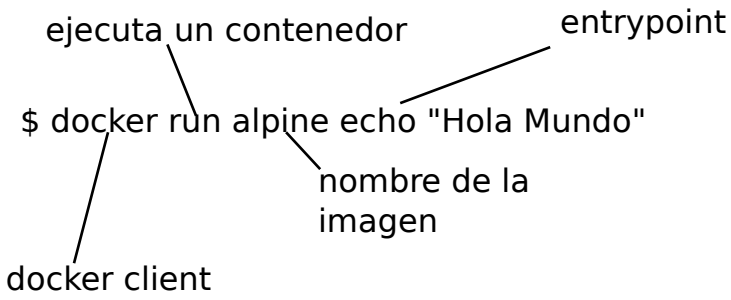
Vamos a correr nuestro primer contenedor, el mismo va está basado en la imagen de alpine y ejecutará un hola mundo y finalizara nuestro container.

```
$ docker run alpine echo "Hola Mundo"
```

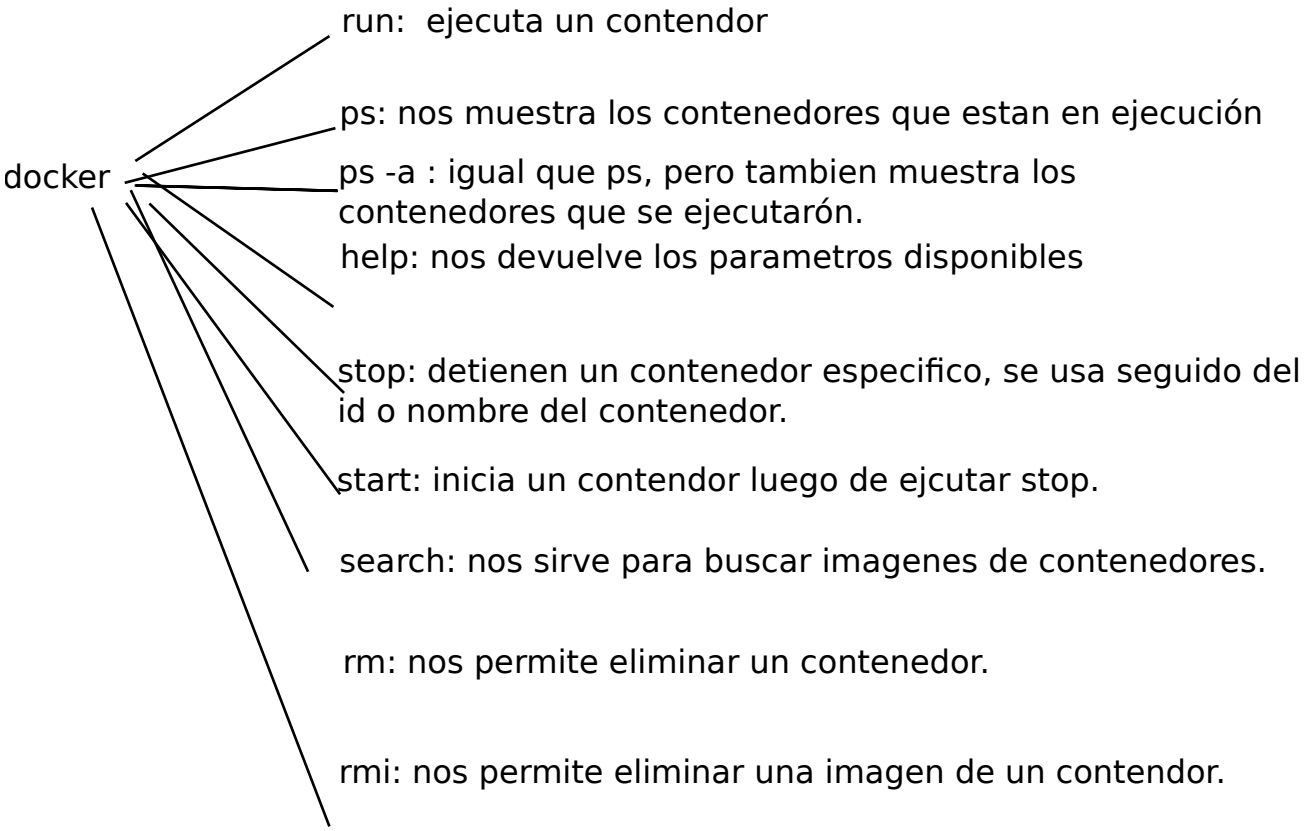
Lo que estamos haciendo acá es hacer uso del entrypoint del container , diciendo que ejecute un echo con el cadena de texto “Hola Mundo”. Nuestro container finaliza porque el propósito de su creación es solo el echo. Ahora vamos a ver como hacer para que nuestro container siga con “vida”.

Comando RUN.

El parámetro run nos permite ejecutar contenedores, si no pasamos ningún argumento solo descargar la imagen sino existe en el host, la ejecutara y se detendrá .



Comandos utiles



Dockerfile

Dockerfile es un archivo de texto plano que nos permite construir nuestras imágenes personalizadas. Está conformado por instrucciones que nos permite agregar nuestro software como así también las dependencias necesarias para que nuestro código se pueda ejecutar. El archivo se puede guarda con el nombre Dockerfile o también con un nombre personalizado, en este ejemplo vamos usar el nombre por defecto. Una práctica es guardarlo en el directorio raíz del proyecto, esto nos va a facilitar muchas cosas.
Una cosa para tener en cuenta, van a ver muchos ejemplos en internet, pero siempre vayan a la documentacion oficial https://docs.docker.com/engine/reference/builder/. Con nuevas versiones pueden quedar deprecado algunas cosas como también sumando nuevas funcionalidades.
El formato que maneja es el siguiente:

```
# Comment

INSTRUCTION arguments
```

Así de simple.Pero armemos un ejemplo de cero para un app de python.
Nuestra primera línea será la imagen base que va utilizar, en nuestro caso python 3.8 y utilizamos el sentencia FROM :

```
FROM python:3.8
```

Tenemos que identificar quién es el manteiner de esta imagen, por eso vamos a especificar. Para eso utilizamos LABEL:

```
LABEL maintainer="SvenDowideit@home.org.au"
```

Ahora queremos copiar nuestros codigo, contamos con el sentencia COPY, necesitamos pasar ubicación y destino, por ejemplo COPY app/archivo.py . , esto copia del directorio app al directorio actual, por eso el punto.

```
COPY app/main.py .
COPY requirement .
```

Ahora vamos a instalar nuestros requirement,vale aclarar que requirement es similar a package.json en javascript por ejemplo.Vamos a utilizar RUN, que nos permite ejecutar gestores de paquetes entre otras cosas.

RUN pip install –f requirement.txt

Ahora, si nuestro código necesita trabajar con variables de entorno, debemos definir las mismas, para eso se usa ENV.

ENV FLASK_ENV=”production”

Ahora, definamos el puerto TCP que va a necesitar esta app, vamos a utilizar EXPOSE.

EXPOSE 8080

Ahora para que nuestro código se ejecute una vez iniciado el contenedor, debemos utilizar CMD.

CMD ["python","–u","main.py"]

Nuestro Dockerfile debió quedar de la siguiente forma.

```
FROM python:3.8
LABEL maintainer="SvenDowideit@home.org.au"
COPY requirements.txt .
RUN python –m pip install –r requirements.txt
COPY main.py .
ENV FLASK_ENV="production"
EXPOSE 8080CMD ["python","–u","main.py"]
```

Para crear nuestra imagen solo necesitamos correr el siguiente comando:

docker build –t nombredelaimagen:version .

Esto crear la imagen y la guarda localmente, para guardarla en un registry, por ejemplo docker hub, tenes que crear una cuenta en el sitio y ejecutar el siguiente comando:

docker login –u nombredeusuario –p password

docker push nombrelaimagen:version

Recordar que tu repositorio debe llamarse como tu imagen.

Otras sentencias utilies:

ADD: Es similar a COPY, necesita un source y dest pero nos permite copiar archivos remotos, por ejemplo archivos alojados en un servidor web.

WORKDIR: nos permite definir un directorio de trabajo, todo lo que copiemos dentro de nuestro imagen va ir al directorio de trabajo definido, si no existe se creará. Ejemplo:

```
WORKDIR app/
```

ENTRYPOINT:

ENTRYPOINT permite crear un container como un ejecutable, es decir iniciar el container ejecutar una instrucción y finalizar. Esto puede ser útil para una prueba, un job. Se define de la siguiente dos maneras:

```
ENTRYPOINT ["executable", "param1", "param2"]
ENTRYPOINT command param1 param2
```

La primera es la más utilizada y es conocida como el formato exec y es utilizado dentro de CMD. El segundo ejemplo es el formato shell y es utilizado para ejecutar script dentro del contenedor, hace uso de /bin/sh –c lo cual hace no necesario invocar el sh como el primer formato. Este último ignora los argumentos de CMD y el stop signal de docker stop, no utilice este formato para iniciar el contenedor, no es el adecuado.

CMD:

CMD se usa como valor predeterminado para iniciar un contenedor, este argumento puede ser usado para ejecutar un archivo ejecutable o pasar un ENTRYPOINT. Si usamos varios CMD dentro de nuestro dockerfile, sera el ultimo que se define el utilizado. Si usamos CMD como ENTRYPOINT debemos especificar el comando dentro de corchetes [ ] y comillas dobles “ “ cada parámetro. Esto se debe que una vez que se ejecuta es convertido en una matriz json.

Si CMD esta definido en la imagen base y en la imagen nueva utiliza ENTRYPOINT, este genera un CMD vacio, lo cual debera usar CMD en la nueva imagen.

Buenas practicas

Una de las buenas prácticas es el uso de .dockerignore, este archivo oculto de tipo plano, nos permite definir qué archivos queremos que ignore al crear una imagen.
Creamos un archivo con el nombre .dockerignore, y agregamos el nombre o extensión de archivos a ignorar y listo. Por ejemplo:

```
# comment
*/temp*
*/*/temp*
temp?
*.md
```

* No usar varios COPY o RUN en el mismo Dockerfile.

Podes usar en la misma linea de RUN varios comandos haciendo uso de && y \

* Evitar dependencia innecesarias, borrar paquetes luego de instalar.

* Instalar los paquetes que necesita y borrar luego los paquetes en el cache.

* Evitar modificaciones sobre el Dockerfile.

Cada sentencia que usamos dentro de nuestro Dockerfile genera una capa, de no ser modificada la sentencia, en el próximo build será usada y evitando la creación de la misma, ganando tiempo de generación de la imagen.

* Usar imagenes oficiales.

Esto nos brinda seguridad que la imagen tiene lo que necesitamos, cuenta con el soporte de los desarrolladores del software, por ejemplo si voy a correr un mysql, voy usar la imagen oficial de mysql.

* Usar tag especifico, no latest.

Esto nos posibilita trackear que version especifica estamos corriendo, atengo que latest puede ser nuevo o una version vieja de latest.

Esta publicacion es un breve resumen para inciarse en el uso Docker. Este es el inicio de un conjuntos de publicaciones sobre el tema. Tratando de reunir un poco de experiencia y documentación oficial.

Felino Ediciones