Trabajo Práctico 2 - Introducción a Docker

1- Objetivos de Aprendizaje

- Familiarizarse con la tecnología de contendores
- Ejercitar comandos básicos de Docker.

2- Unidad temática que incluye este trabajo práctico

Este trabajo práctico corresponde a la unidad Nº: 2 (Libro Ingeniería de Software: Unidad 18)

3- Consignas a desarrollar en el trabajo práctico:

A continuación, se presentarán algunos conceptos generales de la tecnología de contenedores a manera de introducción al tema desde el punto de vista práctico.

¿Que son los contenedores?

Los contenedores son paquetes de software. Ellos contienen la aplicación a ejecutar junto con las librerías, archivos de configuración, etc para que esta aplicación pueda ser ejecutada. Estos contenedores utilizan características del sistema operativo, por ejemplo, cgroups, namespaces y otros aislamientos de recursos (sistema de archivos, red, etc) para proveer un entorno aislado de ejecución de dicha aplicación.

Dado que ellos utilizan el kernel del sistema operativo en el que se ejecutan, no tienen el elevado consumo de recursos que por ejemplo tienen las máquinas virtuales, las cuales corren su propio sistema operativo.

¿Que es docker?

Docker es una herramienta que permite el despliegue de aplicaciones en contenedores. Además, provee una solución integrada tanto para la ejecución como para la creación de contenedores entre otras muchas funcionalidades.

¿Porque usar contenedores?

Los contenedores ofrecen un mecanismo de empaquetado lógico en el cual las aplicaciones pueden estar aisladas del entorno en el cual efectivamente se

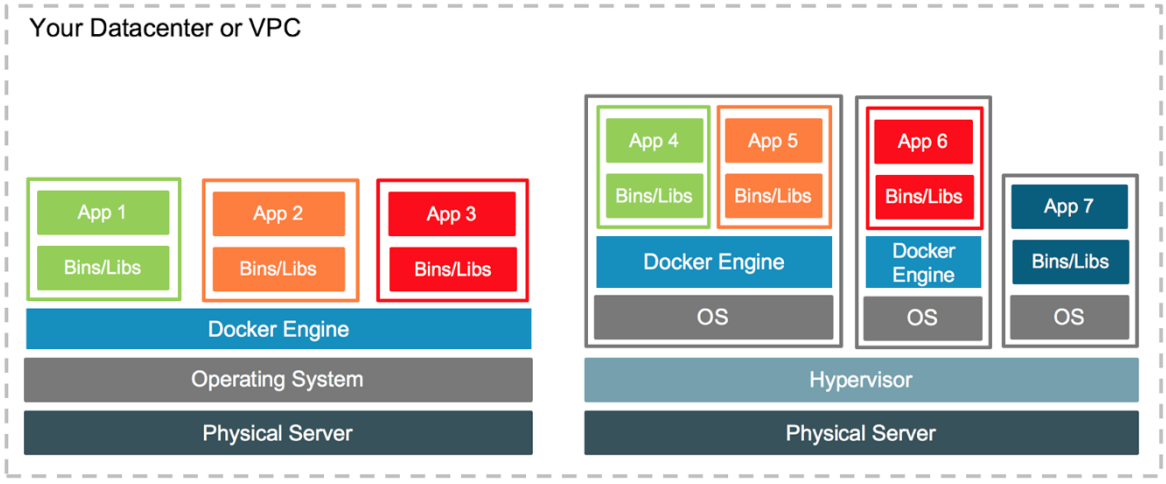
ejecutan. Este desacoplamiento permite a las aplicaciones en contenedores ser desplegadas de manera simple y consistente independientemente de si se trata de un Data Center privado, una Cloud publica, o una computadora de uso personal. Esto permite a los desarrolladores crear entornos predecibles que están aislados del resto de las aplicaciones y pueden ser ejecutados en cualquier lugar.

Por otro lado, ofrecen un control más fino de los recursos y son más eficientes al momento de la ejecución que una máquina virtual.

En los últimos años el uso de contenedores ha crecido exponencialmente y fue adoptado de forma masiva por prácticamente todas las compañías importantes de software.

Máquinas Virtuales vs Contenedores

Los contenderos no fueron pensados como un remplazo de las máquinas virtuales. Cuando ambas tecnologías se utilizan en forma conjunta se obtienen los mejores resultados, por ejemplo, en los proveedores cloud como AWS, Google Cloud o Microsoft Azure.



(Imagen: https://blog.docker.com/2016/04/containers-and-vms-together/)

Analogía

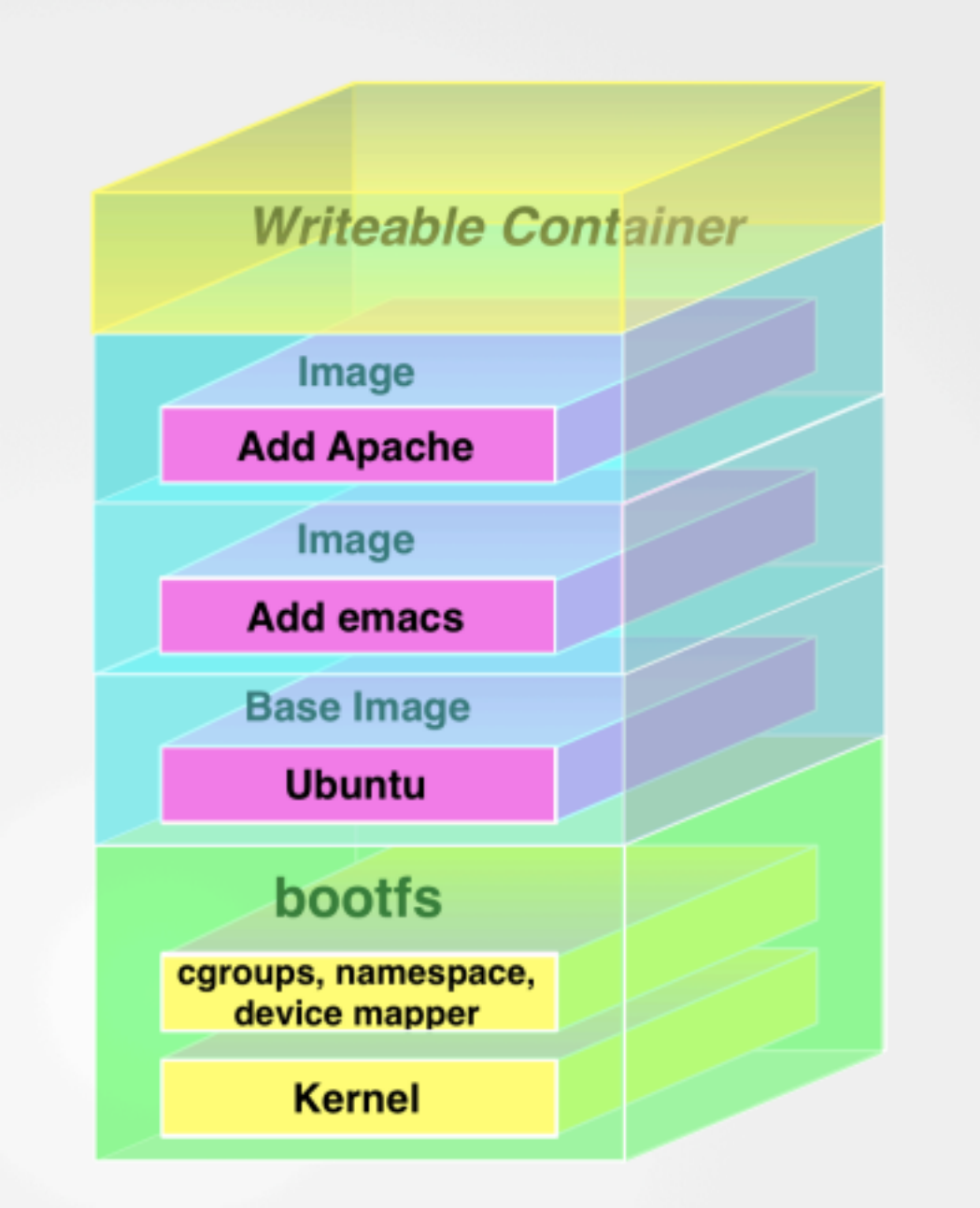
(Imagen: https://github.com/SteveLasker/Presentations/tree/master/DockerCon2017)

Conceptos Generales

- **Container Image**: Una imagen contiene el sistema operativo base, la aplicación y todas sus dependencias necesarias para un despliegue rápido del contenedor.
- **Container**: Es una instancia en ejecución de una imagen.
- **Container Registry**: Las imágenes de Docker son almacenadas en un Registry y pueden ser descargadas cuando se necesitan. Un registry pude ser público, por ejemplo, DockerHub o instalado en un entorno privado.
- **Docker Daemon**: el servicio en segundo plano que se ejecuta en el host que gestiona la construcción, ejecución y distribución de contenedores Docker. El daemon es el proceso que se ejecuta en el sistema operativo con el que los clientes hablan.
- **Docker Client**: la herramienta de línea de comandos que permite al usuario interactuar con el daemon. En términos más generales, también puede haber otras formas de clientes, como Kitematic, que proporciona una GUI a los usuarios.
- **Dockerfile**: Son usados por los desarrolladores para automatizar la creación de imágenes de contenedores. Con un Dockerfile, el demonio de Docker puede automáticamente construir una imagen.

Layers en Docker

Las imágenes de docker están compuestas de varias capas (layers) de sistemas de archivos y agrupadas juntas. Estas son de solo lectura. Cuando se crea el contenedor, Docker monta un sistema de archivos de lectura/escritura sobre estas capas el cual es utilizado por los procesos dentro del contenedor. Cuando el contenedor es borrado, esta capa es borrada con él, por lo tanto, son necesarias otras soluciones para persistir datos en forma permanente.



(Imagen: https://washraf.gitbooks.io/the-docker-ecosystem/content/Chapter%201/Section%203/union_file_system.html)

4- Desarrollo:

1- Instalar Docker Community Edition

- Diferentes opciones para cada sistema operativo
- https://docs.docker.com/
- Ejecutar el siguiente comando para comprobar versiones de cliente y demonio.

```
docker version
```

2- Explorar DockerHub

- Registrase en docker hub: https://hub.docker.com/
- Familiarizarse con el portal

3- Obtener la imagen BusyBox

- Ejecutar el siguiente comando, para bajar una imagen de DockerHub

```
docker pull busybox
```

- Verificar qué versión y tamaño tiene la imagen bajada, obtener una lista de imágenes locales:

```
docker images
```

4- Ejecutando contenedores

- Ejecutar un contenedor utilizando el comando **run** de docker:

```
docker run busybox
```

- Explicar porque no se obtuvo ningún resultado
- Especificamos algún comando a correr dentro del contendor, ejecutar por ejemplo:

```
docker run busybox echo "Hola Mundo"
```

- Ver los contendores ejecutados utilizando el comando **ps**:

```
docker ps
```

- Vemos que no existe nada en ejecución, correr entonces:

```
docker ps –a
```

- Mostrar el resultado y explicar que se obtuvo como salida del comando anterior.

5- Ejecutando en modo interactivo

- Ejecutar el siguiente comando

```
docker run –it busybox sh
```

- Para cada uno de los siguientes comandos dentro de contenedor, mostrar los resultados:

```
ps
uptime
free
ls –l /
```

- Salimos del contendor con:

```
exit
```

6- Borrando contendores terminados

- Obtener la lista de contendores

```
docker ps –a
```

- Para borrar podemos utilizar el id o el nombre (autogenerado si no se especifica) de contendor que se desee, por ejemplo:

```
docker rm elated_lalande
```

- Para borrar todos los contendores que no estén corriendo, ejecutar cualquiera de los siguientes comandos:

```
docker rm $(docker ps –a –q –f status=exited)
docker container prune
```

7- Montando volúmenes

Hasta este punto los contenedores ejecutados no tenían contacto con el exterior, ellos corrían en su propio entorno hasta que terminaran su ejecución. Ahora veremos cómo montar un volumen dentro del contenedor para visualizar por ejemplo archivos del sistema huésped:

- Ejecutar el siguiente comando, cambiar myusuario por el usuario que corresponda. En linux/Mac puede utilizarse /home/miusuario):

```
docker run –it –v C:\Users\misuario\Desktop:/var/escritorio busybox
/bin/sh
```

- Dentro del contenedor correr

```
ls –l /var/escritorio
touch /var/escritorio/hola.txt
```

- Verificar que el Archivo se ha creado en el escritorio o en el directorio home según corresponda.

8- Publicando puertos

En el caso de aplicaciones web o base de datos donde se interactúa con estas aplicaciones a través de un puerto al cual hay que acceder, estos puertos están visibles solo dentro del contenedor. Si queremos acceder desde el exterior deberemos exponerlos.

- Ejecutar la siguiente imagen, en este caso utilizamos la bandera -d (detach) para que nos devuelva el control de la consola:

```
docker run –d daviey/nyan–cat–web
```

- Si ejecutamos un comando ps:

```
PS D:\> docker ps
CONTAINER ID        IMAGE                  COMMAND
CREATED             STATUS              PORTS               NAMES
87d1c5f44809        daviey/nyan–cat–web   "nginx –g 'daemon of…"   2
minutes ago       Up 2 minutes        80/tcp, 443/tcp
compassionate_raman
```

- Vemos que el contendor expone 2 puertos el 80 y el 443, pero si intentamos en un navegador acceder a http://localhost no sucede nada.

- Procedemos entonces a parar y remover este contenedor:

```
docker kill compassionate_raman
docker rm compassionate_raman
```

- Vamos a volver a correrlo otra vez, pero publicando uno de los puertos solamente, el este caso el 80

```
docker run -d -p 80:80 daviey/nyan-cat-web
```

- Accedamos nuevamente a http://localhost y expliquemos que sucede.

9- Utilizando una base de datos

- Levantar una base de datos PostgreSQL

```
mkdir $HOME/.postgres

docker run --name my-postgres -e POSTGRES_PASSWORD=mysecretpassword -v
$HOME/.postgres:/var/lib/postgresql/data -p 5432:5432 -d postgres:9.4
```

- Ejecutar sentencias utilizando esta instancia

```
docker exec -it my-postgres /bin/bash

psql -h localhost -U postgres

#Estos comandos se corren una vez conectados a la base

\l
create database test;
\connect test
create table tabla_a (mensaje varchar(50));
insert into tabla_a (mensaje) values('Hola mundo!');
select * from tabla_a;

\q

exit
```

- Conectarse a la base utilizando alguna IDE (Dbeaver - https://dbeaver.io/, eclipse, IntelliJ, etc...). Interactuar con los objectos objectos creados.

- Explicar que se logro con el comando `docker run` y `docker exec` ejecutados en este ejercicio.

10- Presentación del trabajo práctico.

Subir un archivo md (puede ser en una carpeta) trabajo-practico-02 con las salidas de los comandos utilizados. Si es necesario incluir también capturas de pantalla.