Instituto Tecnológico de Costa Rica

Área Académica de Ingeniería en Computadores

(Computer Engineering Academic Area)

Programa de Licenciatura en Ingeniería en Computadores

(Licentiate Degree Program in Computer Engineering)

Curso: CE-4303 Principios de Sistemas Operativos
(Course: CE-4303 Principles of Operative Systems)

Tarea Corta 2: Container Jobs

(Homework 2: Container Jobs)

Realizado por:
Made by:

Giovanni Villalobos Quirós 2013030976

Profesor:
(Professor)

Ing. Alejandra Bolaños Murillo

Fecha: Cartago, Octubre 24, 2017
(Date: Octubre 24th, 2017)

1. Investigación

1.1 Docker

Es una compañía que maneja el movimiento llamada “contenedor”. Siendo el único proveedor de plataformas de contenedores para cada aplicación en la nube híbrida. Docker es la única compañía de dicho movimiento que construye, asegura y administra una gran cantidad de aplicaciones desde su desarrollo hasta su etapa de producción estando o no en la nube. Existen dos principales ediciones de Docker, el Comunity Edition que le ayuda a los desarrolladores a construir aplicaciones y el Enterprise edition, que provee servicios a nivel de IT con operaciones multi-arquitectura a escala. Además una plataforma abierta que permite desarrollar, entregar y correr aplicaciones permitiendo separar las aplicaciones de la infraestructura para poder entregar software de una manera mas rapida.

Entre las principales que brinda Docker se encuentran.
Simplicidad: puesto que ofrece poderosas herramientas accesibles a todos.
Open Source: Se puede construir utilizando tecnología open source.
Independencia: Permite brindar una separación entre desarrolladores e IT.

1.2 Docker volumes

En Docker se puede hacer datos persistentes mediante tres maneras, una de ellas son los volúmenes, que permite guardar datos directamente en el sistema, no en el contenedor. Estos son guardados en una parte del host file system que es manejado por Docker (/var/lib/docker/volumes en Linux), por lo que son aislados de la funcionalidad de la máquina host. Realizan un mayor desempeño que guardar datos en la capa de escritura del contenedor, por lo tanto son la mejor manera de almacenar usando Docker.
Son creados y manejados por Docker, se pueden crear con el comando: *docker volume create,* cuando es creado se guarda en un directorio en el host y al momento de montar el volumen en un contenedor es justamente este directorio el que es montado.

Un sólo volúmen puede ser montado en múltiples contenedores al mismo tiempo y cuando no está siendo usado de igual manera persiste aunque puede ser eliminado con *docker volumen prune*. Al momento de montar un volumen puede ser nombrado o anónimo. En este último caso docker les da un nombre al azar.
También existe la posibilidad de usar driver para los volúmenes, que permite almacenar volúmenes en host remotos. Generalmente son usados cuando se necesita tener acceso compartido a exactamente los mismos datos (dos contenedores deben tener acceso a los datos de su capa de escritura entonces se guardan en un volumen y cada container puede utilizar los mismos datos).

1.3 Container

Un contenedor es definido en [3] como un sistema operativo ligero que corre dentro del sistema de la máquina host que corre instrucciones nativas del núcleo del CPU. Eliminando la necesidad de compilación dinámica. Proveen una alternativa que salva el consumo de recursos evitando el overhead de las máquinas virtuales, además de un gran nivel de aislación. En la figura 1.1 se puede observar una comparación de las máquinas virtuales con los contenedores donde se evidencia la ventaja de estos.

Parameter	Virtual Machines	Containers
Guest OS	Each VM runs on virtual hardware and Kernel is loaded into in its own memory region	All the guests share same OS and Kernel. Kernel image is loaded into the physical memory
Communication	Will be through Ethernet Devices	Standard IPC mechanisms like Signals, pipes, sockets etc.
Security	Depends on the implementation of Hypervisor	Mandatory access control can be leveraged
Performance	Virtual Machines suffer from a small overhead as the Machine instructions are translated from Guest to Host OS.	Containers provide near native performance as compared to the underlying Host OS.
Isolation	Sharing libraries, files etc between guests and between guests hosts not possible.	Subdirectories can be transparently mounted and can be shared.
Startup time	VMs take a few mins to boot up	Containers can be booted up in a few secs as compared to VMs.
Storage	VMs take much more storage as the whole OS kernel and its associated programs have to be installed and run	Containers take lower amount of storage as the base OS is shared

Figura 1.1. Contenedores comparadas con máquinas virtuales, tomado de [3]

.

Desde el punto de vista de docker [1], un contenedor es una instancia en ejecución de una docker image.
Está compuesto por una docker image, un ambiente de ejecución y un set de instrucciones estándar. Una imagen de contenedor es una ligera pieza de software que opera independientemente capaz de ejecutarse puesto que ya incluye todo lo necesario para correr como código, bibliotecas, tools, etc. Actualmente están disponibles tanto para windows como para linux permitiendo que las aplicaciones construidas bajo este modelo se puedan ejecutar de igual manera sin importar el SO siempre y cuando se tenga Docker instalado.
Sus principales características son.
Ligero: Los contenedor dockers corriendo en una máquina comparten el kernel del sistema operativo de dicha máquina, se inician instantáneamente y usan menos RAM. Las imágenes de los contenedores son creadas de capas del file system y comparten archivos en común. Esto minimiza el uso de disco y la descarga de las imágenes es mucho más rápido.

Estándar: Corren en Linux, Windows e inclusive en VMs, bare-metal y en la nube.
Seguridad: Las aplicaciones están totalmente aisladas, limitando los problemas de las aplicaciones dentro del contenedor en lugar de en toda la máquina.

1.4 Docker Filesystem

- Es llamado como Union file system.
- Implementa *union mount* [1] y opera creando distintas capas.
- Combina esto con la técnica de *copy on write* [2] para construir bloques de contenedores haciéndolos muy ligeros y rápidos.

Para Debian y Ubuntu el AUFS es el file system predeterminado, el cual es un Union file system, en este caso colocando en capas múltiples directorios de un único Linux host. Su infraestructura se puede ver en la imagen de la figura 1.2.[1]

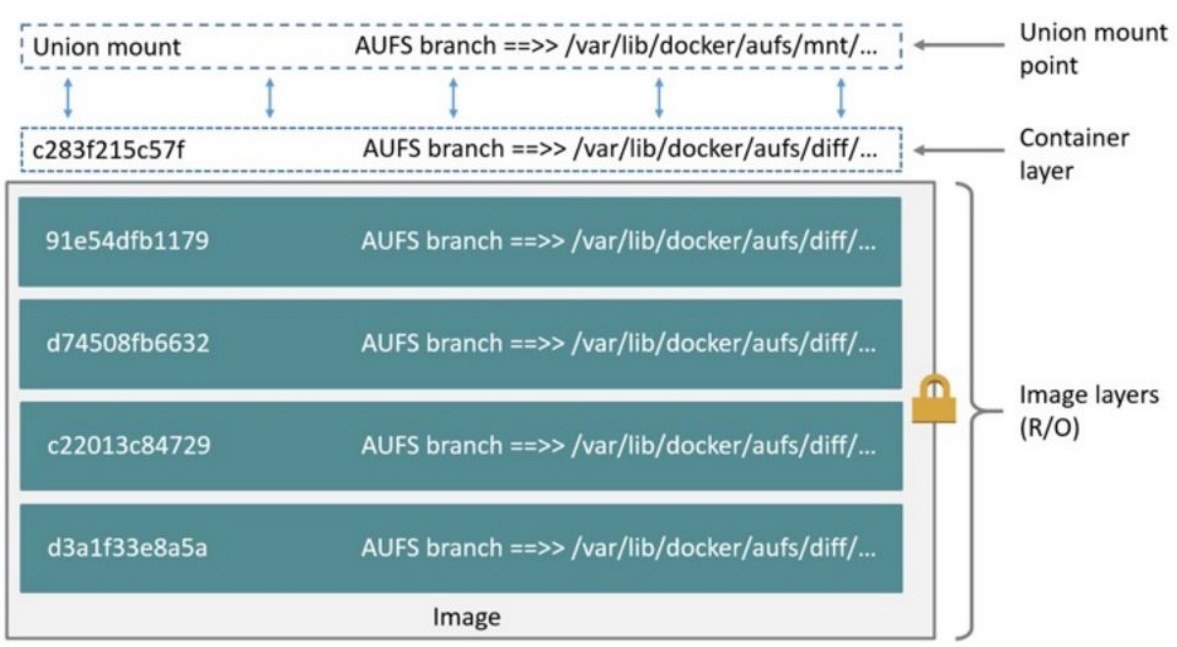

Figura 1.2. AUFS union file system. [1]

Esta es un ejemplo de un contenedor basado en la imagen de ubuntu:latest para ejemplificar el file system. Cada capa en el contenedor son representados en el Docker host como subdirectorios dentro de /var/lib/docker, y el union mount permite ver todas las capas unificadas como una sola. [1]

[1] Combina múltiples directorios en uno que parece contener los contenidos combinados como uno solo, esto es que al montar dos elementos diferentes en un único punto de montaje, generalmente el contenido que existían antes en ese punto de montaje sigue existiendo pero no se puede acceder, con unión mount ambos contenidos (el anterior y nuevo montaje) son vistos como uno.

[2] Múltiples copias de una entidad comparten la misma instancia “copy” y cada una hace cambios a su única y específica capa para “write”, and that changes are only accessible by that entity and deleted when the entity is deleted.

1.5 Docker images

Son un conjunto de capas, cada una de las cuales representa una instrucción en la imagen , son creadas a partir del DockerFile, un documento de texto que contiene todos los comandos necesarios para construir una Docker image.

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Figura 1.3. Ejemplo del dockerfile. [1]

Por ejemplo en el Dockerfile de la figura 1.1 existen 4 comandos y cada uno de ellos crea una capa. Cada capa es solamente un conjunto de diferencias de la anterior y están apiladas una sobre otras.

Generalmente se suele confundir la idea de Imagen con contenedor, por lo que es necesario dejar en claro las principales diferencias. En la imagen de la figura 1.2 se puede observar la relación contenedor-imagen.

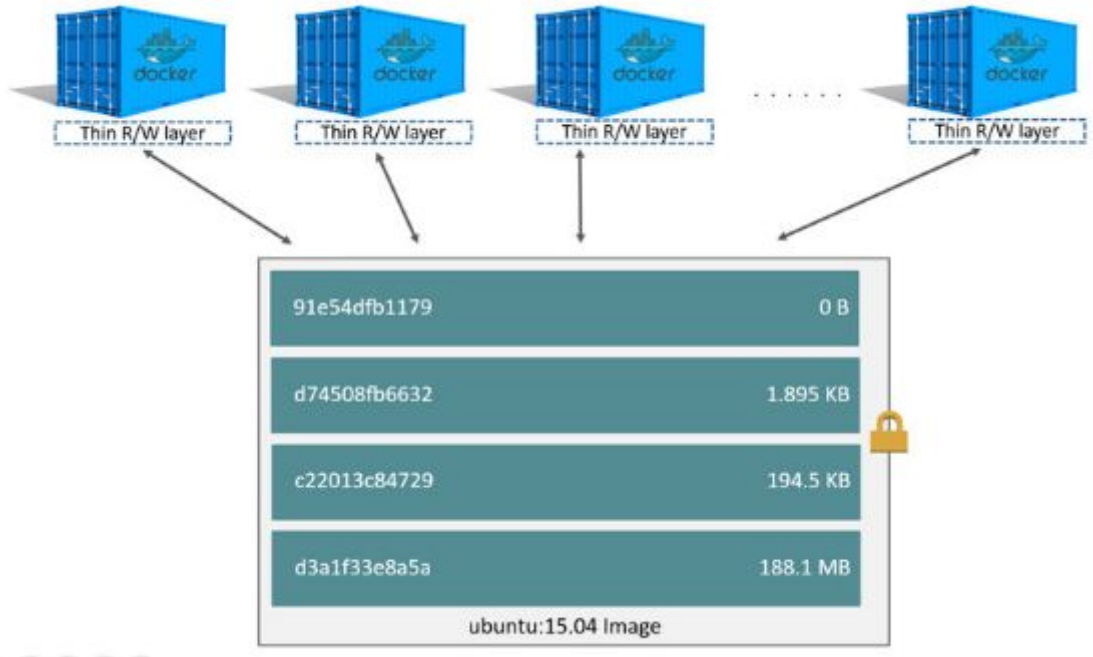

Figura 1.4. Diferencias contenedor-imagen. [1]

Se puede observar en la imagen de la figura 1.2 como los contenedores pueden usar todos una misma imagen y para realizar cambios sobre ella existe la capa de R/W como se observa abajo de cada contenedor donde se guardan todos los cambios realizados a la imagen. Esto para poder compartir una imagen con distintos contenedores. Cuando el contenedor se borra, se borra estos cambios locales de la capa de R/W pero la imagen se mantiene intacta.

1.6 Docker Registry

Un Docker Registry es una aplicación server altamente escalable que almacena las Docker images. Existen diferentes registros, algunos son públicos como Docker Hub (por defecto) y Docker Cloud, que requieren ningún tipo mantenimiento. Los comandos utilizados para obtener las imágenes son *docker pull* y *docker run*, para almacenar imágenes se utiliza *docker push.*
Su uso es generalmente recomendado cuando se quiere tener altamente controlado donde las imágenes se están almacenando.

1.7 RSA

RSA es un algoritmo de llave pública ampliamente usado en firma digital, su nombre significa Ron Rivest, Addi Shamir, Len Adleman. Es un cifrado de bloque en el cual cada mensaje es mapeado a un entero. Consiste de una llave pública (Conocida por todos) y una llave privada(Conocida sólo por el dueño del dato).
Consiste de tres partes, la generación de la llave, encriptación y desencriptación. [2]

Generación de la llave

Antes de cifrar el texto se genera la llave.

1. Se escogen dos números primos a y b. Por seguridad deben ser random y de similar tamaño.

2. Se aplica n = a * b

3. Se aplica la función totient de Euler Ø(n) = (a-1) * (b - 1).

4. Se escoge un entero e tal que 1 < e < Ø(n) y el divisor común de ambos es 1. Este llamado el exponente de la llave pública.
5. Se determina d como d = $e^{-1}(mod\ Ø(n))$

6. Se mantiene d como componente clave privada.

7. La llave pública consiste en el módulo n y el exponente de llave pública.

8. La llave privada consiste en el módulo n y el componente privado d que debe mantenerse en secreto.

Encriptación

Convierte el texto original plano en texto cifrado.

1. El servidor debe transmitir la llave publica al usuario que quiere enviar el dato

2. El dato del usuario es mapeado en un entero usando un protocolo que tenga inversa, el entero es llamado m.
3. Los datos son encriptados con C = $m^e (mod\ n)$.

4. El texto cifrado es almacenado o enviado.

Desencriptación

Devolver el texto cifrado a los datos originales

1.Se verifica la autenticidad y se envia C
2. Al recibir C se puede obtener m con m = $C^d (mod\ n)$

3. Con m obtenido se puede obtener el dato original con el protocolo de reversa.

2. Dockerfile creado

```
# Use an official Python runtime as a parent image
FROM centos:7

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages needed, in this case pip for python installation modules
RUN yum -y update; yum clean all
RUN yum -y install epel-release; yum clean all
RUN yum -y install python-pip; yum clean all
RUN yum -y install net-tools
RUN yum -y install python-netifaces

#Install some pythons needs
RUN pip install -r requirements.txt

# Make port 6666 available to the world outside this container
EXPOSE 6666

# Run python server.py when the container launches
CMD ["python", "server.py"]
```

2.1 El archivo requirements.txt

```
colorthief
```

3.Referencias

[1]"Docker", Docker, 2017. [Online]. Available: https://www.docker.com/. [Accessed: 21- Oct- 2017].

[2] Kalpana, P., Singaraju,S., "Data Security in Cloud Computing using RSA Algorithm," Ijrcct, vol. 1, (4), 2012. Available: http://ijrcct.org/index.php/ojs/article/view/53/40.

[3] R. Dua, A. R. Raja and D. Kakadia, "Virtualization vs containerization to support PaaS," in 2014 IEEE International Conference on Cloud Engineering, 2014, . DOI: 10.1109/IC2E.2014.41.