Instituto Tecnológico de Costa Rica

Área Académica en 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 Operating Systems)

Tarea Corta II
(Short task II)

Estudiante:
(Student)

Wagner Coto Meneses, 201070638

Profesora:
(Professor)

Ing. Alejandra Bolaños Murillo, Licda.

Fecha de entrega: 24 de mayo de 2017
(Due date: Wednesday 24th May, 2017)

Índice

1. Docker

Docker es una plataforma que permite crear, desarrollar y ejecutar cualquier aplicación, en donde sea. Ha recorrido un gran camino en muy poco tiempo, esto lo convierte en un estándar cuya vía es resolver uno de los mayores problemas en cuanto al costo de programación: su despliegue [1].

Previo a la llegada de Docker, el desarrollo de los programas era llevado a cabo mediante la combinación de diferentes tecnologías para la administración de todo el proceso involucrado en el software tales como máquinas virtuales, herramientas para la configuración y mantenimiento, diferentes paquetes para la administración de sistemas y complejas dependencias a ciertas bibliotecas web.

Todas estas herramientas necesitaban ser administradas y mantenidas por diferentes ingenieros especialistas; haciendo mención que la gran mayoría de estas tenían una configuración propia y única [1].

Docker ha permitido revolucionar la manera en que se ha desarrollado anteriormente el software, de forma que ahora los diferentes ingenieros puedan "hablar el mismo lenguaje" tal como se ilustra en la Figura 1 y 2.

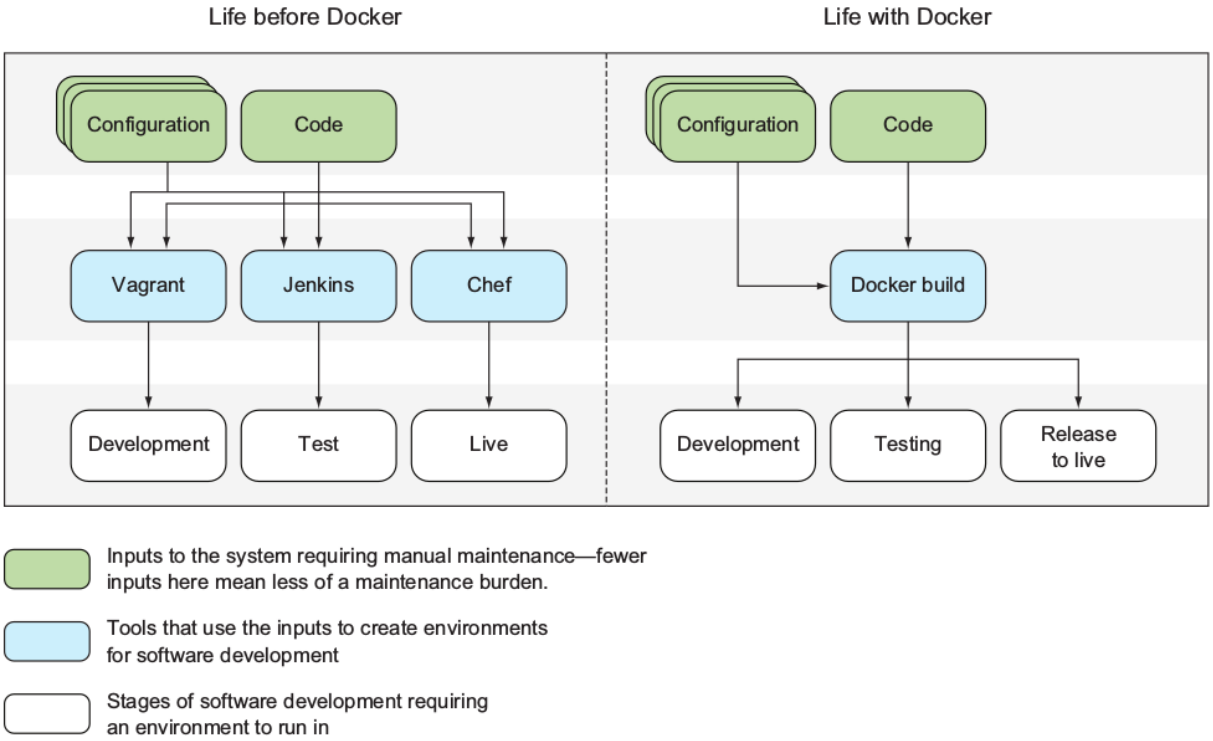

Figura 1: Cómo Docker ha facilitado el agobiante mantenimiento [1]

Con Docker es posible separar la configuración (y todo el esfuerzo que esto conlleva) de la administración de los recursos. Basta con ejecutar `docker run` y el ambiente de la imagen se descargará y estará listo para ejecutarse en la máquina (que puede estar utilizando RedHat, Ubuntu, etc) sin ningún problema tanto como este esté dentro de Docker.

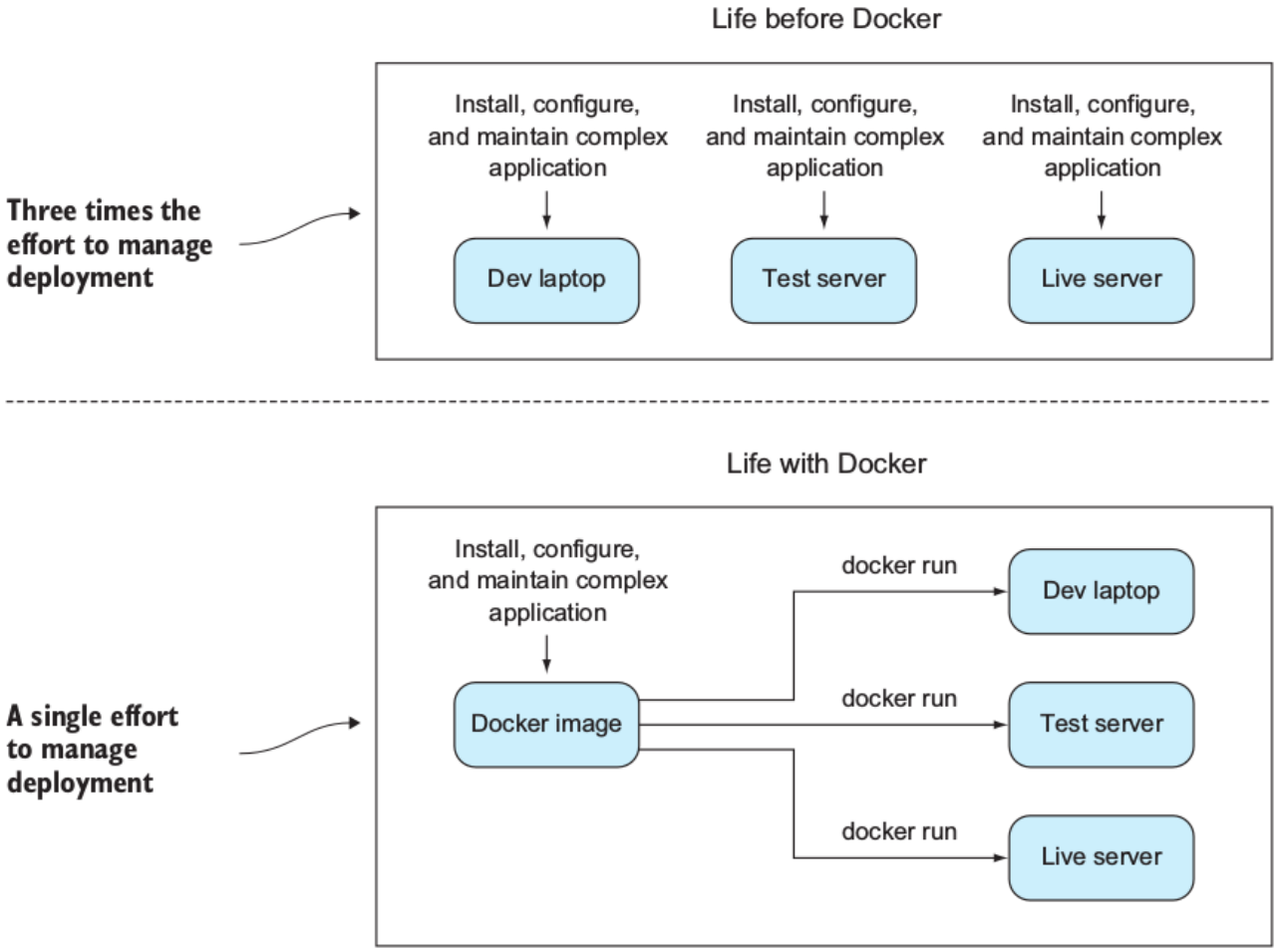

Figura 2: Despliegue de software antes y después de Docker [1]

2. Docker volumes

Un volumen puede ser descrito como un folder compartido; estos se inicializan cuando un contenedor es creado. Se diseñan principalmente para mantener la persistencia de los datos, independientemente del ciclo de vida de los contenedores. Estos son directorios especialmente diseñados dentro de uno o más contenedores con el fin de mantener los datos de forma persistente, independientemente del ciclo de vida del contenedor [2].

3. Container

Los contenedores son instancias en ejecución que encapsulan el software requerido, estos son creados a partir de imágenes. En la Figura 3 se muestra en forma general y gráfica los principales conceptos involucrados en el contexto de Docker, los cuales se explican mas detalladamente en secciones posteriores.

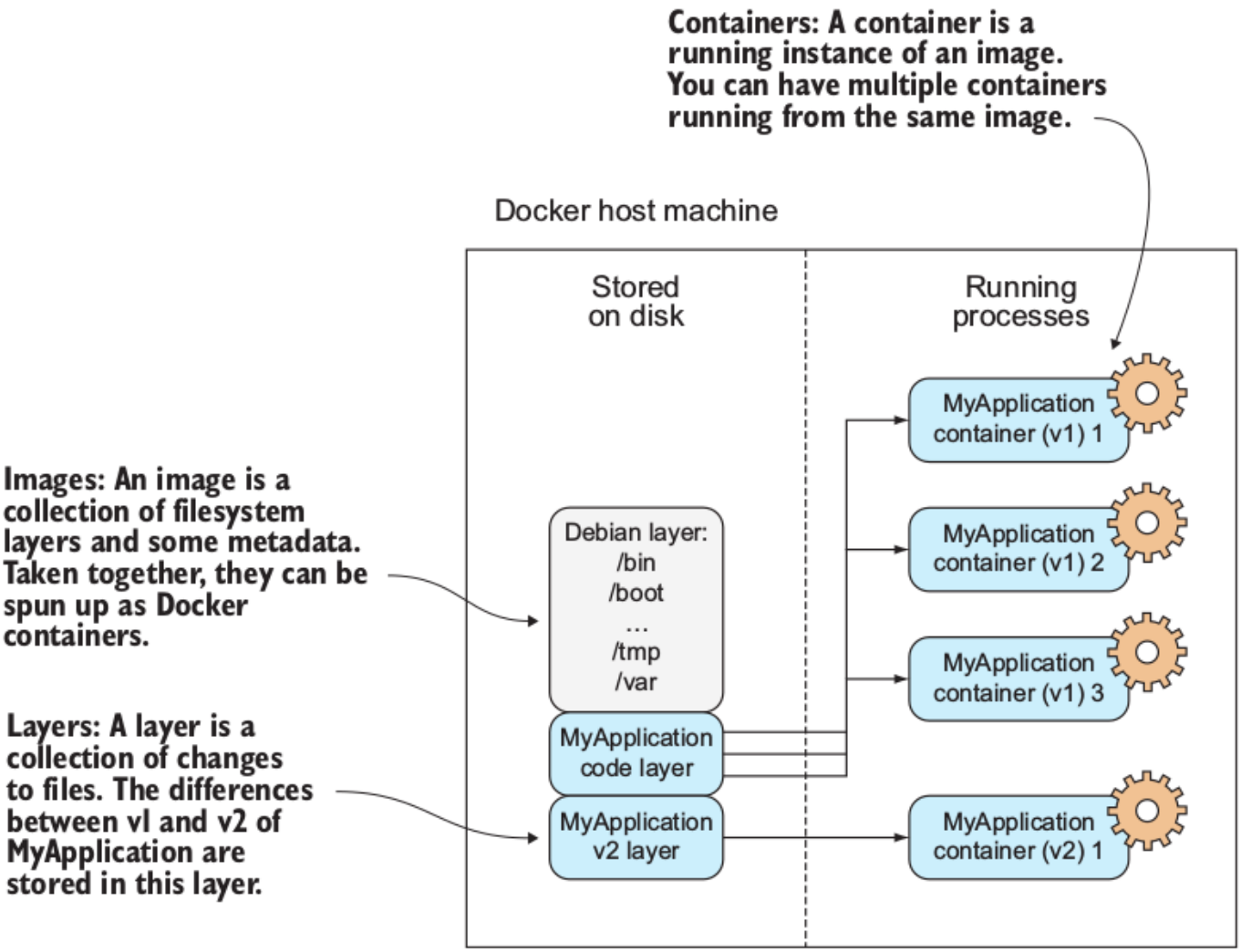

Figura 3: Conceptos del núcleo de Docker [1]

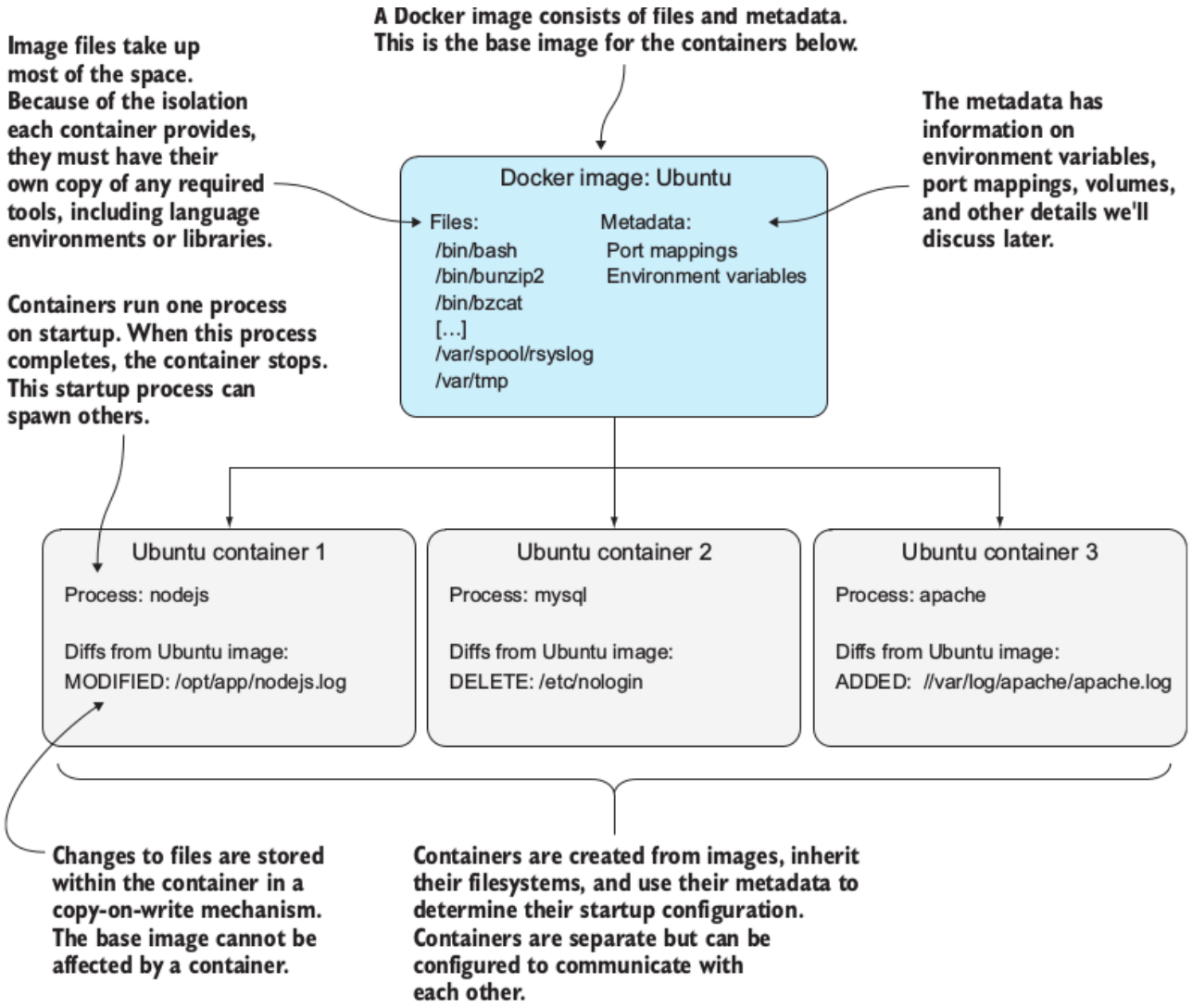

Figura 4: Definición de contenedor y una imagen de Docker [1]

4. Docker filesystem

Un *Dockerfile* es un archivo de texto que contiene una serie de comandos limitado dentro. En dicho archivo, la secuencia de comandos es ejecutada en un orden estricto lo que provoca un impacto directo en los archivos y metadatos de la imagen resultante.

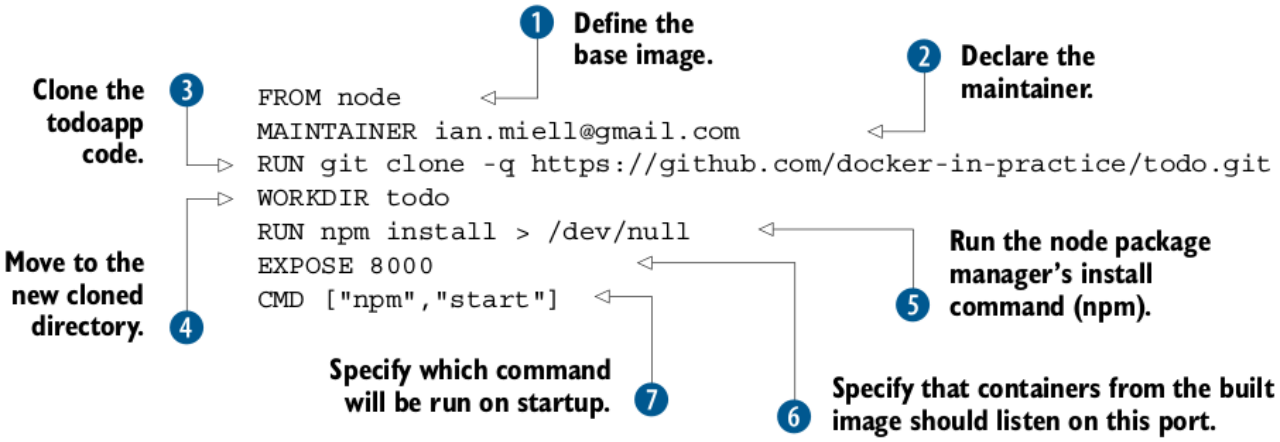

Figura 5: Estructura de un archivo Docker [1]

5. Docker images

Las imágenes son el elemento básico de cualquier contenedor. Cuando estas son creadas, cada instrucción es almacenada en caché para ser reutilizada posteriormente (modelo *Copy on Write*).

Existen cuatro formas mediante las cuales se pueden crear imágenes de Docker tal como se muestran en las Figuras 6 y 7

Method	Description
Docker commands / "By hand"	Fire up a container with `docker run` and input the commands to create your image on the command line. Create a new image with `docker commit`.
Dockerfile	Build from a known base image, and specify build with a limited set of simple commands.

Figura 6: Creación de imágenes Docker I [1]

Method	Description
Dockerfile and configuration management (CM) tool	Same as Dockerfile, but hand over control of the build to a more sophisticated CM tool.
Scratch image and import a set of files	From an empty image, import a TAR file with the required files.

Figura 7: Creación de imágenes Docker II [1]

La primera opción es viable cuando se quiere hacer pruebas y análisis del funcionamiento de la instalación. En el caso que se deseen crear y definir pasos para la creación de una nueva imagen es recomendable utilizar la segunda opción.

En caso de querer realizar desarrollos mas complejos, es factible utilizar la tercera opción; en especial cuando aquellas características del archivo Docker no son tan sofisticadas como la imagen que se desea tener y finalmente cuando se tenga una imagen nula por medio de la superposición del conjunto de archivos requerido para ejecutar la imagen se insta a utilizar el cuarto método. Este método es muy útil en caso que que se desee importar un conjunto de archivos autocontenidos creados en diferentes espacios, pero que son raramente vistos en las secciones principales de uso.

Una forma de analizar las imágenes y contenedor es viéndolo desde una perspectiva análoga de programas y procesos. De la misma forma en que un proceso se puede modelar como un programa en ejecución, un contenedor de Docker puede ser visto como como una imagen de Docker en ejecución.

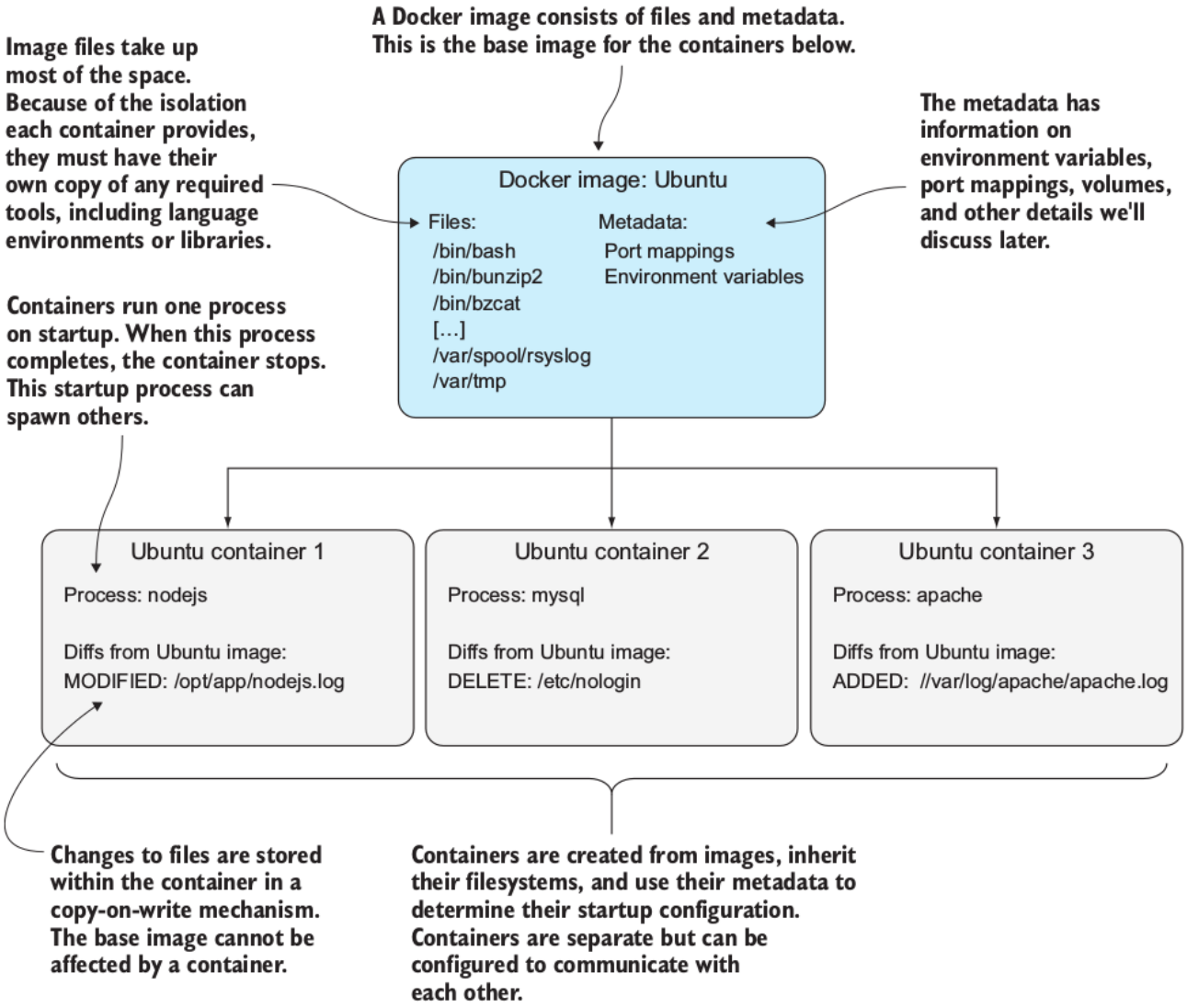

Figura 8: Imágenes y Contenedor del Docker [1]

6. Docker registry

Una vez que se han creado las imágenes, posiblemente estas se quieran compartir con otros usuarios. En este escenario es donde surge el concepto de **registro**, el cual se puede ver como un servidores en donde se almacenan las diferentes imágenes de Docker (puede ser comparado con el sistema Github)

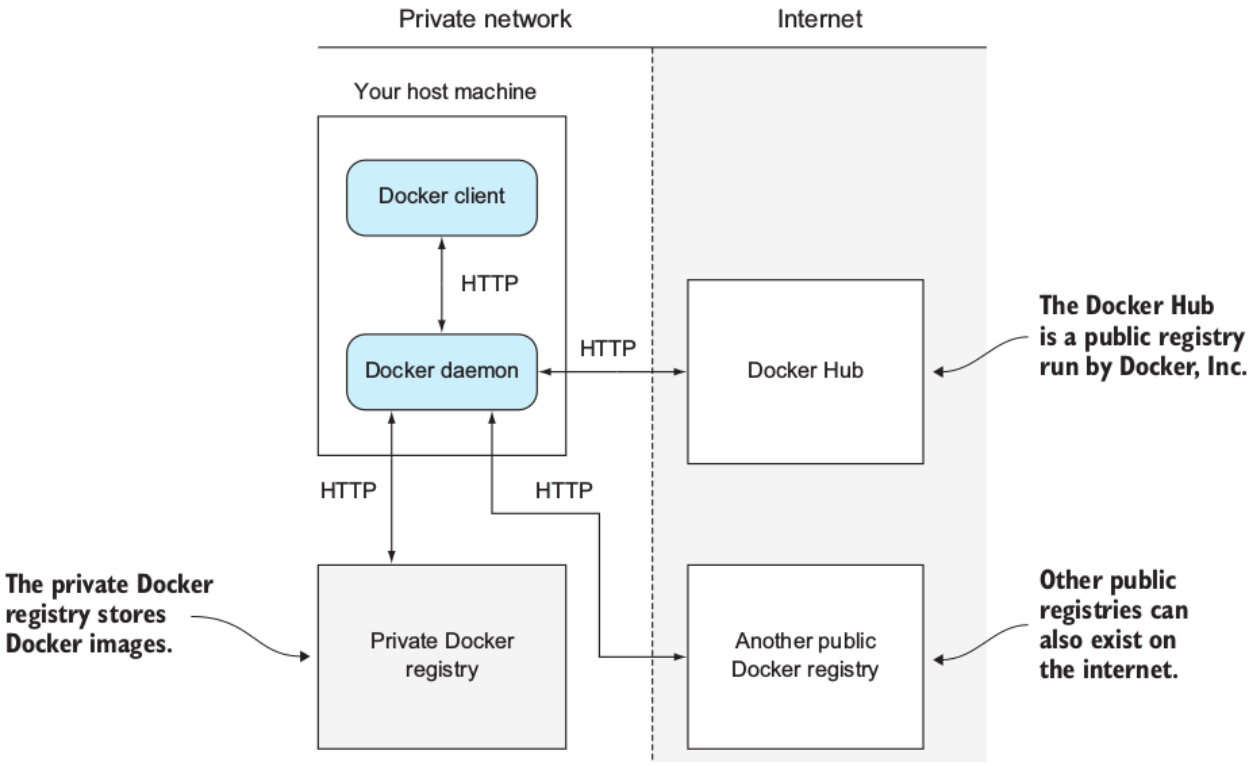

Figura 9: Registros en Docker [1]

Los tres diferentes registros mostrados en la Figura 9 difieren en cuanto a su accesibilidad. Uno de ellos se realiza en una red privada, el otro es mediante una red pública y el tercero es público, pero accesible solo para aquellos registrados en Docker.
Un registro en Docker permite múltiples usuarios que puedan realizar tanto *push* como *pull* desde un sitio central a la API RESTful. El código de registro es, al igual que Docker, de código abierto.

7. RSA

El método de encriptado de datos conocido como algoritmo RSA, por los nombres de sus inventores (Ron **R**ivest, Adi **S**hamir y Leonard **A**dleman) es uno de los más usados hoy día para la transmisión segura de datos a través de canales inseguros, el cual fue diseñado con el fin de reemplazar el entonces usado (e inseguro) NBS (*National Bureau of Standards*). Se implementa un sistema de encriptación con llave pública.

La encriptación mediante llaves públicas es conocida también como *criptografía mediante llaves asimétricas*. Básicamente lo que esto significa es que el cifrado y el desencriptación del mensaje se realiza mediante un par de llaves diferentes conocidas como *llave pública* y *llave privada*.

Supongamos que el usuario A desea comunicarse de manera segura con el usuario B. Si el usuario B desea enviar un mensaje de manera confidencial, lo que hace es encriptarlo con la llave pública del usuario A y enviarlo... a su vez, cuando el usuario A recibe el mensaje, la forma en que lo desencripta es utilizando su propia llave privada, la cual solamente él posee.

La seguridad de este algoritmo radica en el problema de la factorización de números enteros. Los mensajes enviados se representan mediante números, y el funcionamiento se basa en el producto, conocido, de dos números primos grandes elegidos al azar y mantenidos en secreto. Actualmente estos primos son del orden de 10^{200}. El algoritmo consta de tres pasos: generación de claves, cifrado y descifrado [3].

Generación de claves

1. Cada usuario elige dos números primos distintos p y q. Por motivos de seguridad, estos números deben escogerse de forma aleatoria y deben tener una longitud en bits parecida. Se pueden hallar primos fácilmente mediante test de primalidad.

2. Se calcula $n = pq$, n se usa como el módulo para ambas claves: pública y privada.

3. Se calcula $\varphi(n) = (p-1)(q-1)$, donde φ es la función de Euler.

4. Se escoge un entero positivo ε menor que $\varphi(n)$, que sea coprimo con $\varphi(n)$.

   a) ε se da a conocer como el exponente de la clave pública.

   b) Si se escoge un ε con una suma encadenada corta, el cifrado será más efectivo. Un exponente ε muy pequeño (p. ej. ε=3) podría suponer un riesgo para la seguridad.

5. Se determina un δ (mediante aritmética modular) que satisfaga la congruencia $\varepsilon \cdot \delta \equiv 1$.

   a) Expresado de otra manera, $\varepsilon \cdot \delta - 1$ es dividido exactamente por $\varphi(\mathrm{n}) = (\mathrm{p}\text{-}1)(\mathrm{q}\text{-}1)$.

   b) Esto suele calcularse mediante el algoritmo de Euclides extendido.

   c) δ se guarda como el exponente de la clave privada.

La clave pública es (n, ε), esto es el módulo y el exponente de cifrado.

La clave privada es (n, δ), es es el módulo y exponente de descifrado que debe mantenerse en secreto.

Cifrado

El usuario A comunica su clave pública (n, ε) al usuario B y guarda su clave privada en secreto. Ahora B desea enviar un mensaje M a A.

Primero B convierte M en un número entero m menor que n mediante un protocolo reversible acordado de antenamo. Luego calcula el texto de cifrado c mediante la operación: $c \equiv m^{\epsilon}$.

Esto puede hacerse rápido mediante el método de exponenciación binaria. Ahora B transmite c a A.

Descifrado

A puede recuperar m a partir de c usando su exponente ϵ de la clave privada mediante el siguiente cálculo: $m \equiv c^{\delta}$.

Ahora que tiene m en su poder, puede conocer M realizando la inversión según el protocolo.

8. Guía de ejecución

Construcción y ejecución de la imagen Docker

```
$ docker build -t <tag> .
$ docker run -p <port1>:<port2> -v <pathHost>:<pathDocker> <tag>
```

Para el caso particular de este proyecto, se clona el repositorio en Github

```
git@github.com:wcoto/Tarea2_SO.git
```

Y una vez dentro del mismo, se compila y ejecuta con los siguientes comandos

```
$ docker build -t tarea2 .
$ docker run -p 15951:15951 -v /home/wagcm/Escritorio/Tarea2_SO/carpetaDocker
:/carpetaDocker tarea2
```

Importante: Es necesario modificar los siguientes archivos de texto localizados dentro /CarpetaDocker

- **PublicKey.txt:** se almacena la llave con la que se va a realizar el procedimiento de encriptación o desencriptación.
- **mensaje.txt:** en este archivo se almacena la palabra local que se va concatenar. En caso de ser el penúltimo contendor de la lista, se debe escribir la palabra *fin* para indicar el cierre del proceso.
- **ip.txt:** en este archivo debe guardarse la ip local de la máquina, se puede obtener con el comando IFCONFIG.
- **Configuracion.config:** este archivo debe ir ordenado de acuerdo con las ip's de los diferentes contenedores. Es necesario incluir tanto el ip local y posteriormente el ip al que se enviarán los datos encriptados (en caso de no encontrarse la palabra *fin*).

Referencias

[1] A. H. Miell, Ian; Sayers, *Docker in Practice*. ISBN: 978-607-442-046-3, Shelter Island, NY 11964: Manning Publications Co., first ed., 2016.

[2] D. docs, "Definition of: volume," may 2017.

[3] A. Kak, "Lecture 12: Public-key cryptography and the rsa algorithm," *Avinash Kak, Purdue University*, p. 103, feb 2017.