CONTENIDO

DOCKER

Consiste en una plataforma líder mundialmente en contenedores de software.
Se utiliza para eliminar problemas de "trabajo en mi maquina" colaborando así con el código de otros desarrolladores.

Por tanto lo utilizan:

- Operadores, para ejecutar y administrar apps lado a lado en contenedores aislados para obtener una mejor densidad de cálculo.
- Empresas, para construir 'tuberías' agiles para entrega de software para enviar características más rápido, confiables y seguros en aplicaciones Linux y Windows server.

Automatiza tareas repetitivas de configurar y configurar entornos de desarrollo para que los desarrolladores puedan enfocarse en lo que importa: crear un gran software.

Al estar “dockerizada” una aplicación, la complejidad que tiene se introducen en contenedores que se pueden construir, compartir y ejecutar fácilmente. Con Dockerfiles es fácil compartir con algún compañero de trabajo algún código sin tener que instalar software. [1]

DOCKER IMÁGENES

Una imagen de Docker, es una estructura de directorios y paquetes mínima, creada para una función básica. Se trata de que sea una plantilla, que se puede modificar, pero que ha sido pensada para una uso básico y específico. Por ejemplo, una imagen Debian con un Apache. De esta forma, para tener múltiples servidores web apache, cada uno encapsulado y aislado de los demás, sólo tendremos que crear contenedores desde ella, y cada uno será un servidor web distinto, con su propia IP, al que se puede configurar de manera independiente y que a su vez de puede clonar o mover donde se quiera. [2]

CONTAINER

Una imagen de contenedor es un paquete liviano, independiente, ejecutable de un software que incluye todo lo necesario para ejecutarlo:

- Código
- tiempo de ejecución
- herramientas del sistema
- bibliotecas del sistema
- configuración.

Disponible para aplicaciones basadas en Linux y Windows, el software en contenedor siempre funcionará igual, independientemente del entorno. Los contenedores aíslan el software de su entorno, por ejemplo las diferencias entre los entornos de desarrollo y de estadificación, y ayudan a reducir los conflictos entre equipos que ejecutan diferentes programas en la misma infraestructura. [3]

Por tanto a partir de una única imagen, podemos ejecutar varios contenedores.

Como podrás deducir a partir de la imagen anterior, esto es una buena manera de tener copias de tu aplicación ejecutándose en varios contenedores, para luego, a través de balanceadores de carga, distribuir los accesos a tu aplicación, y ofrecer servicios con más garantías y con menos carga de peticiones en cada contenedor.

Como las imágenes no cambian, si creas un contenedor a partir de una imagen, y mientras que se está ejecutando el contenedor cambias algo o instalas alguna herramienta, al parar dicho contenedor y después volver a ejecutar otra vez la misma imagen, esos cambios no se verán reflejados.

Docker va trackeando los cambios en los contenedores como si fuera una herramienta de control de versiones, por lo que si realmente deseas esos cambios, haciendo un commit del contenedor puedes crear otra imagen que contenga dichos cambios.

Por lo tanto otro beneficio de Docker, es el versionado de los contenedores. Si algo va mal en nuestra aplicación, podremos volver de forma sencilla a una versión anterior del contenedor, a una versión anterior del entorno. [4]

DOCKER FILESYSTEM

La base del uso del filesystem en Docker es la abstracción del backend de almacenamiento. Un backend de almacenamiento le permite almacenar un conjunto de capas cada una dirigida por un nombre único. Cada capa es un árbol de fileSystem que se puede montar cuando se necesita y se modifica. Las nuevas capas se pueden iniciar desde cero, pero también se pueden crear con un padre especificado. Esto significa que comienzan con el mismo contenido que la capa primaria, pero con el tiempo pueden divergir. Esto se implementa típicamente en el backend por alguna forma de copia sobre escritura para permitir que la operación de creación sea rápida.

Algunos backends también tienen operaciones adicionales que permiten un cómputo eficiente de las diferencias entre capas. Todas estas operaciones tienen implementaciones de repliegue, pero pueden ser más lentas, ya que tienen que comparar todos los archivos de las capas.

Basado en la abstracción de capa Docker implementa los conceptos de alto nivel de imágenes y contenedores.

Cada imagen de Docker en el sistema se almacena como una capa, siendo el padre la capa de la imagen principal. Para crear una imagen de este tipo se crea una nueva capa (basada en el padre derecho) y luego los cambios en esa imagen se aplican al sistema de archivos recién montado.

Los contenedores son un poco más complicados. Cada contenedor tiene dos capas, una (llamada capa init), que se basa en una capa de imagen y un hijo de aquella que contiene el contenido real del contenedor. La capa init contiene algunos archivos que siempre deben existir en los contenedores de Docker (por ejemplo, /.dockerinit). Cometer un contenedor (y, por lo tanto, crear una imagen) implica encontrar todos los cambios de la capa de init a la capa de contenedor y aplicarlos a una nueva capa basada en la misma imagen que el contenedor utilizado. [5]

DOCKER VOLUMENES

Unas de las funcionalidades mas importantes de los docker son los volumenes, a pesar de que son simples carpetas en nuestro sistema de ficheros son capaces de sobrevivir al ciclo de vida normal del contenedor.
Nos permite compartir varios ficheros con otros contenedores o con el host, por lo que nos permite consultar todos los logs, hacer backups desde un contenedor a otro.

Los volúmenes pueden ser de tres tipos distintos:

- Data volúmenes
  - Anonymous data volúmenes
  - Named data volumenes
- Mounted volumes

Data volumes

Se trata de carpetas que se crean en */var/lib/docker/* y que pueden compartirse entre diferentes contenedores.

Anonymous data volumes

Se crean cuando se levanta un contenedor, mediante el comando *docker run*, por ejemplo:

```
gerard@sirius:~$ docker run -ti --rm -v /data alpine:3.4 sh
```

Esto nos crea un volumen asociado al contenedor creado.

```
root@sirius:~# docker volume ls
DRIVER              VOLUME NAME
local               1b39e6601cd3711c27f3a1a4eb50d82e182151fd14b82048f47b0d50ad22b97a
root@sirius:~# tree /var/lib/docker/volumes/
/var/lib/docker/volumes/
├── 1b39e6601cd3711c27f3a1a4eb50d82e182151fd14b82048f47b0d50ad22b97a
│   └── _data
└── metadata.db
2 directories, 1 file
root@sirius:~#
```

A su vez, otro contenedor puede montar los volúmenes de otro contenedor, ya sea porque los creó o porque los ha montado de un tercero.

```
root@sirius:~# docker run -ti --rm --volumes-from adoring_lovelace alpine:3.4 sh
```

Ahora mismo, la carpeta */data/* pertenece al primer contendor, pero es la misma para ambos contenedores.
Docker mantiene una cuenta de los contenedores que están usando un volumen, y estos solo se eliminan cuando el último contenedor que lo usa sale con el parámetro *--rm* o si se hace un *docker rm -v*. En cualquier otro caso, el volumen se queda parasitando, hasta que lo eliminamos manualmente usado *docker volume rm*.

Named data volumes

Estos volúmenes no dependen de ningún contenedor concreto, y se pueden montar en cualquier contenedor. Se crean específicamente usando el comando *docker volume create*, o al ejecutar un contenedor si le damos un nombre en la línea de comandos.

```
gerard@sirius:~$ docker volume create --name vol1
vol1
gerard@sirius:~$ docker run -ti --rm -v vol2:/data alpine:3.4 true
gerard@sirius:~$ docker volume ls
DRIVER              VOLUME NAME
local               vol1
local               vol2
gerard@sirius:~$
```

Estos volúmenes no se eliminan por si solos nunca y persisten cuando su contenedor desaparece. Para eliminarlos se necesita una intervención manual mediante el comando *docker volume rm*.

```
gerard@sirius:~$ docker volume ls
DRIVER              VOLUME NAME
local               vol1
local               vol2
gerard@sirius:~$ docker volume rm vol1 vol2
vol1
vol2
gerard@sirius:~$ docker volume ls
DRIVER              VOLUME NAME
gerard@sirius:~$
```

Mounted volumenes

Otras veces nos interesa montar ficheros o carpetas desde la máquina *host*. En este caso, podemos montar la carpeta o el fichero especificando la ruta completa desde la máquina *host*, y la ruta completa en el contenedor. Es posible también especificar si el volumen es de lectura y escritura (por defecto) o de solo lectura.

```
gerard@sirius:~/docker$ docker run -ti --rm -v /etc/hostname:/root/parent_name:ro -v /opt/:/data alpine:3.4 sh
/ # cat /root/parent_name
sirius
/ # ls /data/
```

```
/ #
```

Este último caso es ideal para recuperar *backups* o ficheros generados en un contenedor, en vistas a su utilización futura por parte de otros contenedores o del mismo *host*. [6]

DOCKER REGISTRY

El Registro es una aplicación de servidor apilada y altamente escalable que almacena y le permite distribuir imágenes de Docker. El Registro es de código abierto, bajo licencia permisiva de Apache.

Debe utilizar el Registro si desea:

- Controlar con precisión dónde se almacenan las imágenes
- Totalmente dueño de su pipeline de distribución de imágenes
- Integrar el almacenamiento y la distribución de imágenes en su flujo de trabajo interno de desarrollo

El registro es compatible con la versión 1.6.0 o superior del motor Docker. [7]

Crear un registro en un contenedor Docker es muy simple. Simplemente hay que ejecutar el contenedor provisto por Docker, así:

```
$ docker run -p 5000:5000 registry:2
```

Esto ejecutará un contenedor que utiliza la versión 2.0 de la aplicación de registro y comunica por el puerto 5000 de la máquina local. [8]

En resumen los registros docker contienen imágenes creadas por los usuarios y puestas a disposición del público. Podemos encontrar repositorios públicos y totalmente gratuitos o repositorios privados donde tendremos que comprar las imágenes que necesitemos. Estos registros permiten desarrollar o desplegar aplicaciones de forma simple y rápida en base a plantillas, reduciendo el tiempo de creación o implementación de aplicaciones o sistemas. [9]

RSA

En un algoritmo asimétrico de encriptación usando clave publica RSA.
Una clave puede ser un archivo binario o una cadena de bits o bytes.

Al enviar un msj entonces el emisor utiliza la clave publica del receptor y cuando llega este mensaje al receptor entonces este se ocupa de descifrarlo usando su clave oculta.
La seguridad de este algoritmo se basa en que no hay maneras rapidas de factorizar un numero grande en su factores primos usando computación tradicional.

Este algoritmo fue desarrollado en 1977 en el MIT y en el 2000, expirando su patente, paso a ser un algoritmo de dominio público.

Básicamente funciona de la siguiente manera:

1. Inicialmente es necesario generar aleatoriamente dos números primos grandes, a los que llamaremos **p** y **q**.
2. A continuación calcularemos n como producto de **p** y **q: n = p * q**
3. Se calcula **fi: fi(n)=(p-1)(q-1)**
4. Se calcula un número natural **e** de manera que MCD(**e, fi(n))=1** , es decir e debe ser primo relativo de **fi(n).** Es lo mismo que buscar un número impar por el que dividir **fi(n)** que de cero como resto.
5. Mediante el algoritmo extendido de Euclides se calcula **d: e.d mod fi(n)=1** Puede calcularse **d=((Y*fi(n))+1)/e** para **Y=1,2,3**,… hasta encontrar un **d** entero.
6. El par de números **(e,n)** son la clave pública.
7. El par de números **(d,n)** son la clave privada.
8. Cifrado: La función de cifrado es **C = M^e mod n**
9. Descifrado: La función de descifrado es **M = C^d mod n**

Ejemplo:

1. Escogemos dos números primos, por ejemplo p=3 y q=11.
2. n = 3 * 11
   = 33
3. fi(n) = (3-1) * (11-1)
   = 20
4. Buscamos
   e: 20/1=0, 20/3=6.67.
   e=3
5. Calculamos d como el inverso multiplicativo módulo z de e, por ejemplo, sustituyendo Y por 1,2,3,… hasta que se obtenga un valor entero en la expresión:
   d = ((Y * fi(n)) + 1) / e = (Y * 20 + 1) / 3 = 21 / 3 = 7
6. e=3 y n=33 son la clave pública
7. d=7 y n=33 son la clave privada
8. Cifrado:
   Mensaje = 5,
   C = M^e mod
   n = 5^3 mod 33
   = 26
9. Descifrado:
   M = C^d mod
   n = 26^7 mod 33
   = 8031810176 mod 33
   = 5

Fuente:

BIBLIOGRAFIA

[1] "What is Docker," Docker, 14-May-2015. [Online]. Available: https://www.docker.com/what-docker. [Accessed: 17-May-2017].

[2] A. rootsudo, "Docker, manual en español," *root sudo*, 29-Jun-2016.

[3] "What is a Container," *Docker*, 29-Jan-2017. [Online]. Available: https://www.docker.com/what-container. [Accessed: 17-May-2017].

[4] A. M. del C. G. O. S. Q. at Bq. linkedin com/in/amgarciaoApasionada por la calidad del softwarey buenas prácticas en general E. en testing and A. D. P. E. I. Continua, "Entendiendo Docker. Conceptos básicos: Imágenes, Contenedores, Links…," *Javier Garzás*, 31-Jul-2015.

[5] "Supported Filesystems — Project Atomic." [Online]. Available: http://www.projectatomic.io/docs/filesystems/. [Accessed: 20-May-2017].

[6] Gerard, "Tipos de volúmenes en Docker," *Linux Sysadmin*, 18-Jul-2016. [Online]. Available: http://www.linuxsysadmin.tk/2016/07/tipos-de-volumenes-en-docker.html. [Accessed: 20-May-2017].

[7] "Docker Registry," *Docker Documentation*, 20-May-2017. [Online]. Available: https://docs.docker.com/registry/. [Accessed: 20-May-2017].

[8] P. por jalapuente, "FUNCIONAMIENTO DE NUESTRO PROPIO REGISTRO DOCKER." .

[9] "Docker, Qué es y sus principales características.," *OpenWebinars.net*, 05-May-2014. [Online]. Available: https://openwebinars.net/blog/docker-que-es-sus-principales-caracteristicas/. [Accessed: 20-May-2017].

[10] W. E. C. M. dice, “¿Qué es RSA?,” *Seguridad Informática*, 14-Sep-2007. .