

MAQUINAS VIRTUALES VS CONTENEDORES

Huichi Contreras, Franklin Carlos (2016056193), Huillca Umpiri, Willian (2015053793), Robles Flores, Anthony Richard (2016056192)

Escuela Profesional de Ingeniería de Sistemas

Universidad Privada de Tacna

Tacna, Perú

Abstract

In this article we will learn concepts, differences and uses of virtualization and containers, this will allow us to choose the most recommendable option for the performance of a system that is being developed or to make changes in it.

Keywords:

virtualization, containers, tools, processes, simulation, resources.

1. Resumen

En este artículo aprenderemos conceptos, diferencias y usos de la virtualización y contenedores, esto nos permitirá elegir la opción más recomendable para el desempeño de un sistema que se esté desarrollando o para realizar cambios en él.

Palabras clave:

virtualización, contenedores, herramientas, simulación, procesos, recursos.

2. Introducción

Antes de ahondar en el concepto de contenedores, volvamos unos años atrás para recordar el nacimiento de la virtualización. A medida que el hard-

ware se hacía más poderoso nos encontramos con que el software no ocupaba todas las capacidades de la maquina física donde se encontraba siendo ejecutada (en muchos casos ni siquiera una fracción de estos recursos). Dado lo anterior se crearon recursos “virtuales” para simular el hardware base sobre el cual se ejecuta el software, permitiendo que múltiples aplicaciones puedan ser ejecutadas al mismo tiempo, cada una usando una fracción de los recursos del hardware físico disponible. A a esta “simulación” que permite de compartir recursos la denominamos comúnmente “virtualización”.

La mayoría de nosotros cuando escuchamos el concepto de virtualización pensamos inmediatamente en máquinas virtuales, pero es importante entender que este es solo un tipo de virtualización en el cual se habilita un sistema operativo el cual tiene la ilusión de que posee recursos dedicados para operar. Entendiendo lo anterior podemos ahora definir a los Contenedores como creadores de la percepción de un ambiente aislado exclusivo para la aplicación mientras que en la virtualización “tradicional” de máquinas virtuales la aplicación se ejecuta en un sistema operativo virtualizado donde convive con otros aplicativos.

3. Marco teorico

3.1. ¿Que es la virtualización?

La virtualizacion es la creación de una versión virtual (no física) de algo. Esta basada en software, se puede aplicar a sistemas operativos, almacenamiento, servidores, aplicaciones, redes, etc. y es una manera de reducir gastos y aumentar eficiencia y agilidad en las empresas.

3.1.1. Virtualizacion de servidores

La virtualizacion de servidores ayuda a evitar ineficiencias ya que permite ejecutar varios sistemas operativos en una maquina fisica con maquinas virtuales con acceso a los recursos de todos. Tambien permite generar un cl uster de servidores en un unico recurso para así mejorar mucho mas la eficiencia y la reduccion de costes. Tambien permite el aumento de rendimiento de las aplicaciones y la disponibilidad al aumentar la velocidad en la carga de trabajo.

3.1.2. Virtualizacion de escritorios

La implementacion de escritorios virtualizados permite ofrecer a las sucursales o empleados externos de forma rapida y sencilla un entorno de trabajo y una reduccion de la inversion a la hora de gestionar cambios en estos.

3.1.3. Virtualizacion de red

Se trata de reproducir una red fisica completa mediante software para poder ejecutar los mismos servicios que una red convencional y sus dispositivos. Cuentan con las mismas características y garantías que las redes físicas con las ventajas que nos ofrece la virtualización además de la liberación del hardware.

3.1.4. Almacenamiento definido por software

La virtualización del almacenamiento permite prescindir de los discos de los servidores. Los combina en depósitos de almacenamiento de alto rendimiento y los distribuye como software. Este nuevo modelo permite aumentar la eficiencia en el guardado de datos.

3.2. Ventajas de la virtualización

Como se ha podido apreciar en los tipos de virtualización presentados anteriormente, esta conlleva una mejora considerable tanto en el rendimiento, agilidad, flexibilidad, escalabilidad, etc. como en una reducción considerable de los costes económicos y de tiempo y una simplificación en la gestión de la infraestructura.

- Reduce los costes de capital y los gastos operativos.
- Minimiza o elimina los tiempos de inactividad
- Aumenta la productividad, la eficiencia, la agilidad y la capacidad de respuesta
- Implementa aplicaciones y recursos con más rapidez.
- Garantiza la continuidad del negocio y la recuperación ante desastres.
- Simplifica la gestión del centro de datos.

3.3. Contenedores

Un contenedor de software se puede considerar como una aplicación para el servidor. Estos se encargan de proporcionar a las aplicaciones archivos, variables y bibliotecas que sean necesarios para ejecutarse y maximiza su portabilidad. Para poder instalar una aplicación, el contenedor se carga en el ordenador en un formato portable o imagen (Image) que incluye todos los datos necesarios para su funcionamiento y, en el ordenador, se inicia en

un entorno virtual. Los contenedores permiten que estos equipos de desarrollo alcancen una eficiencia muy alta en la entrega y el despliegue de software, al solucionar muchos de los retos que presenta la virtualización tradicional.

Beneficios:

- Ahorro de espacio y consumo. No se tendrá la necesidad de crear máquinas virtuales en las que instalarlas.
- Reutilización. Se puede crear tantas instancias como necesitemos, destruirlas y reproducir el entorno inicial.
- No "ensucia". No se tendrá que instalar dentro de nuestro equipo con la problemática que ello conlleva en algunos casos.
- Compartir. Estas imágenes las podremos distribuir comodamente entre los componentes de nuestro equipo.
- Se obtiene mayor modularidad. El desarrollo con contenedores es ideal para un enfoque basado en microservicios para el diseño de aplicaciones.

Ventajas de los contenedores:

Los contenedores de aplicaciones "empaquetan" los recursos necesarios para el funcionamiento de una aplicación sin embargo, las mayores ventajas de tales contenedores radican, sobre todo, en la gestión y en la automatización de software basado en contenedores.

- Instalación más sencilla: los contenedores de software se inician a partir de imágenes o representaciones portables de un contenedor, incluyendo un programa y todos los componentes requeridos. De esta manera se compensan las diferencias entre sistemas operativos. Su instalación se reduce a la introducción de una línea de comando.
- Independiente de la plataforma: las imágenes se pueden transportar cómodamente de un sistema a otro y se caracterizan por una considerable independencia de la plataforma. Lo único que se necesita para iniciar un contenedor desde una imagen es un sistema operativo que soporte contenedores.

- Pérdidas por virtualización mínimas: con un Linux y Docker container, la instalación de contenedores requiere alrededor de 100 MB y unos pocos minutos, aunque no es solo esto a lo que se oponen los administradores de sistemas. Mientras que la virtualización de hardware trae consigo una pérdida de rendimiento para el hipervisor y otros sistemas operativos, los contenedores, al prescindir de todo esto, reducen esta pérdida al mínimo.
- Aplicaciones aisladas: cada programa funciona independientemente de otros contenedores, de forma que aplicaciones con requerimientos opuestos pueden funcionar en paralelo en el mismo sistema.
- Administración y automatización unitarias: debido a que en una plataforma como Docker todos los contenedores son gestionados con las mismas herramientas, es posible automatizar todas las aplicaciones de manera centralizada. Por esto, estas soluciones están indicadas sobre todo para arquitecturas de servidor en las cuales los componentes están distribuidos en varios servidores, de forma que se carga con los pesos de instancias diferentes. En estos ámbitos de aplicación, el Docker container dispone de herramientas con las cuales configurar automatismos. Esto posibilita, por ejemplo, iniciar instancias nuevas de forma automática en momentos puntuales de sobrecarga.

Al final, el uso de los contenedores es muy conveniente, también puede traer un nivel bajo de seguridad estaríamos dejando de usar sistemas operativos separados. Pues, los contenedores no son tan herméticos como las máquinas virtuales con sistema operativo propio. En consecuencia, aunque los contenedores constituyen una alternativa para la virtualización de hardware, de momento no la pueden sustituir por completo.

4. Funcionamiento de Docker

4.1. *Arquitectura*

Docker usa una arquitectura cliente servidor. El cliente de Docker se comunica con el Daemon de Docker para crear, ejecutar y distribuir los contenedores. Tanto el cliente como el Daemon pueden estar en el mismo sistema o pueden conectarse remotamente. Como Docker usa el kernel de Linux para

su ejecución, si el sistema operativo del sistema no es este, se deberá usar una pequeña capa extra en la arquitectura de tipo VM (boot Docker) para poder correr Docker en la máquina.

4.2. *Cliente de Docker*

Es la principal interfaz de usuario para Docker. Acepta los comandos del usuario y se comunica con el Daemon de Docker.

4.3. *Imágenes de Docker (Docker Images)*

Las imágenes de Docker son plantillas de solo lectura, que nos permitan crear contenedores basados en su configuración.

4.4. *Registros de Docker (Docker Registries)*

Los registros de Docker guardan las imágenes. Estos son repositorios públicos o privados donde se pueden subir o descargar imágenes. Sería similar a GitHub para imágenes de Docker (Docker Hub).

5. Creación de Imágenes

Como se comenta en el punto anterior, las imágenes de Docker son las plantillas para poder levantar los contenedores. Por eso la importancia de saber crear imágenes y personalizarlas ya que solo permiten lectura y los cambios que hagamos en los contenedores no se verán reflejados en estas. La manera más sencilla de crear una imagen es descargarla del Docker Hub con el comando explicado anteriormente:

```
docker pull [OPTIONS] NAME[:TAG|@DIGEST]
```

Este comando nos permite descargar una imagen en una versión concreta o tag dependiendo de nuestras necesidades. Por defecto, si no se pone nada, descargará la última.

En la figura podemos apreciar cómo se descarga la última versión de la imagen de MongoDB y se genera la imagen. Una vez obtenida la imagen se

```
alfredo@alfredo:~$ docker pull mongo
Using default tag: latest
latest: Pulling from library/mongo

8ad8b3f87b37: Pull complete
5947be99d359: Pull complete
d5a4577c6007: Pull complete
acb97586a200: Pull complete
d11260d069a3: Pull complete
bf102d35e390: Pull complete
f4964f6a9bfa: Pull complete
8b392ba3e8bf: Pull complete
c023b73abe56: Pull complete
Digest: sha256:8ff7bd4acdb123e3922a7fae7f73efa35fba35af33fad0de946ea31370a23cc4
Status: Downloaded newer image for mongo:latest
alfredo@alfredo:~$
```

(a) Docker pull

```
alfredo@alfredo:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mongo	latest	48b8b08dca4d	2 weeks ago	366.4 MB

```
alfredo@alfredo:~$
```

(b) Docker images

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

pasara a levantar el contenedor para poder ejecutar el servicio con otro de los comandos explicados.

En la figuras 2.4 se puede apreciar como utilizando el comando `docker ps` que permite ver que contenedores estan levantados, no hay ninguno (a) y como al inicializar con el comando `docker run mongo` levanta el servicio (b) y esta vez sí aparece el contenedor (c).

La segunda manera de crear y personalizar imagenes es mediante un DockerFile, que es un documento de texto donde se encuentran los comandos que se deben ejecutar para generar nuestra imagen.

El comando `docker build` comunica al Daemon de Docker que debe de leer el DockerFile del directorio actual y seguir las instrucciones linea por linea para la creacion de nuestra imagen. Este proceso va pintando los resultados por pantalla y generando imagenes intermedias para obtener asi una cache que nos permitira en caso de errores, una vez corregido el DockerFile,

```
alfredo@alfredo:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
alfredo@alfredo:~$
```

(a) Docker ps

```
alfredo@alfredo:~$ docker run mongo
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] MongoDB starting : pid=1 port=27017 dbpath=/data/db 64-bit host=alfredo
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] db version v3.2.9
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] git version: 22ecae3b40c85fc7cae7d56e7d6a82fd01188c
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.0.1t  3 May 2016
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] allocator: tcmalloc
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] modules: none
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] build environment:
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten]   distmod: debian11
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten]   distarch: x86_64
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten]   target arch: x86_64
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] options: {}
2016-09-19T14:54:09.631+0000 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=3G,session_max=20000,eviction=(threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000),checkpointer=(wait=60,log_size=2GB),statistics_log=(wait=0),
2016-09-19T14:54:10.421+0000 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2016-09-19T14:54:10.421+0000 I CONTROL [initandlisten] ** We suggest setting it to 'never'
2016-09-19T14:54:10.421+0000 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2016-09-19T14:54:10.421+0000 I CONTROL [initandlisten] ** We suggest setting it to 'never'
2016-09-19T14:54:10.421+0000 I CONTROL [initandlisten]
2016-09-19T14:54:10.422+0000 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory '/data/db/diagnostic.data'
2016-09-19T14:54:10.422+0000 I NETWORK [HostNameCanonicalizationWorker] Starting hostname canonicalization worker
2016-09-19T14:54:10.492+0000 I NETWORK [initandlisten] waiting for connections on port 27017
```

(b) Docker run

```
alfredo@alfredo:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
eee379c1602   mongo    "/entrypoint.sh mongo"   4 seconds ago  Up 3 seconds  27017/tcp    kickass_pike
alfredo@alfredo:~$
```

(c) Docker ps

continuar desde el punto conflictivo.

```
FROM docker/whalesay:latest
CMD echo "Proyecto CoIoTe" | cowsay
```

Aquí tenemos un ejemplo sencillo de DockerFile que nos servirá para explicar de una manera rápida como crearlos.

FROM indica la imagen base que va a utilizar para seguir futuras instrucciones. Buscamos si la imagen se encuentra localmente, en caso de que no, la descargamos. En nuestro ejemplo utilizamos la última versión de la imagen docker/whalesay.

6. Docker Compose

Docker Compose es un orquestador que nos permite ejecutar aplicaciones que utilicen varios contenedores a la vez. Se crea un archivo docker-

compose.yml donde se configuraran todos los servicios necesarios para nuestra aplicacion. Una vez ejecutado este archivo nos generara todas las imagenes y con estas los contenedores especificados a la vez que arrancara la aplicacion. Los comandos que se utilizaran seran similares a los utilizados en la creacion de imagenes.

Para ejecutar el servicio y levantar todos los contenedores.

```
docker-compose up
```

Para detener el servicio y detener los contenedores.

```
docker-compose stop
```

7. Conclusiones

- Los Contenedores son un método de virtualización de un sistema operativo que permite ejecutar una aplicación junto con sus elementos dependientes, en un ambiente aislado e independiente.
- La virtualizacion lo que hace es crear a traves del software una version virtual de un recurso, como virtualiza de manera completa un software, consume los recursos del hardware tal como lo hicieramos en una maquina física reduciendo el rendimiento y el performance de la maquina que aloja a la virtual