Título

Pruebas con Título

Daniel Moreno

April 27, 2019

IES Francesc de Borja Moll

Índice

Contenedores

Los contenedores no tienen un sistema operativo dentro, el contenedor aísla el espacio de usuario.

Son muy ligeros porque corren como un proceso sobre el SO del host.

Los contenedores escalan en función de la demanda, mientras que las MV tienen que ser aprovisionadas previamente.

Contenedores Windows y contenedores Linux

Los contenedores Windows corren sobre Windows y los contenedores Linux sobre Linux
Se diferencian en los tipos de aislamiento que tienen ambos

Los contenedores Linux:

- Se ejecutan sobre el host de modo que son visibles a ellos
- Se ejecutan sobre el kernel como un proceso visible desde el host

En Hyper-V Windows lanza una MV super fina que tiene su propio kernel y por tanto los contenedores no son visibles desde el propio SO.

Imágenes

Una imagen es como una aplicación compilada. Un contenedor es una instancia en ejecución de una imagen concreta.

Las imágenes se construyen sobre una tecnología de **sistema de ficheros por capas**.

Para crear una imagen, generalmente se crea un archivo de texto llamado docker file formado por diferentes **instrucciones**. Cada línea representa una instrucción, y cada vez que se ejecuta el dockerfile se ejecutan dichas instrucciones de arriba a abajo.

Estos dockerfiles se almacenan como texto y se pueden compartir con facilidad, así como almacenarse en sistemas de control de versiones.

Cada **instrucción** que se ejecuta cambia ligeramente el estado del sistema de archivos respecto a la instrucción anterior.

La diferencia entre el estado del sistema de ficheros antes y después de cada instrucción se guarda en disco como un archivo, que conforma una **capa**.

Aprovechamiento de las capas

Cada vez que se ejecuta una instrucción, se crea un contenedor y se etiqueta con un hash creado para obtener un nombre único. De este modo, podemos reutilizar estas capas intermedias y solo tener que construirlas una vez.

Si dos docker files comienzan por el mismo set de instrucciones desde el comienzo, todas las capas que coincidan solo existirán en el sistema de ficheros una vez, para ambos contenedores.

Para cada RUN se crea una capa, por lo que se pueden agrupar varios comandos en un solo RUN y crear una única capa.

Docker

Docker permite automatizar el despliegue de aplicaciones dentro de contenedores.

Veremos diferentes configuraciones de Docker que podemos hacer.

Dockerfiles

Permiten automatizar la construcción de una imagen, a través de un fichero que contiene instrucciones para fabricar una imagen, a través de una serie de pasos.

Formato de un dockerfile
Formato del Dockerfile, que se creará dentro de la carpeta donde tengamos el proyecto

```
FROM ubuntu:14.04
ENTRYPOINT ["/bin/echo"]
```

Ejecutar un comando
Si en lugar de utilizar entrypoints queremos pasar parámetros, podemos utilizar CMD

```
CMD ["/bin/echo" , "Hi Docker !"]
```

Construir una imagen

Construimos la imagen con el siguiente comando. Al haber un Dockerfile en la carpeta la detecta y monta la imagen a partir de las instrucciones del Dockerfile.

```
docker build .
```

Definir etiqueta para una imagen

Al no definir repositorios ni tags, se asigna a la imagen una ID hexadecimal. Podemos especificar un nombre y una etiqueta al vonstruir la imagen:

```
docker build -t cookbook:hello .
```

Consultar imágenes
Consultar las imágenes disponibles en nuestra máquina. A partir de una imagen, podemos levantar tantos contenedores como queramos.

```
docker images
```

Nos mostrará:

```
REPOSITORY              TAG                 IMAGE ID            CREATED
node.js-mongodb         latest              2ce837b1b0ca        11 minutes ag
danimrtic/apache2       first               4223faeadc8f        2 months ago
danimrprofe/apache2     first               4223faeadc8f        2 months ago
danimrprofe/apache2     latest              4223faeadc8f        2 months ago
```

Docker compose

Si quisiera levantar varios contenedores de diferentes imágenes, me vería obligado a definir diferentes Dockerfiles para cada uno y levantar los contenedores uno a uno.

Docker-compose es una herramienta que nos permite:

- Definir varios **servicios** en un solo archivo de configuración
- Con un solo comando podemos poner en marcha varios contenedores al mismo tiempo.
- Podemos decir que un servicio solo se levante si otro se ha iniciado previamente

Definición de servicios

Para ello definimos la configuración de los servicios mediante un archivo en formato YAML, que llamaremos docker-compose.

Ejemplo de docker-compose, en el que podemos ver dos servicios:

- Un servicio **basededatos** que cargará una imagen de **mysql:5.7**
- Un servicio **wordpress** que cargará una imagen **wordpress:latest**

Estas imágenes saldrán de docker hub. Los propios desarrolladores generan las imágenes docker y las publican allí.

Montaje de volúmenes

Como ya sabemos, los contenedores tienen un ciclo de vida y dentro de ellos los datos no persisten, por lo que tendremos que enlazar una carpeta externa a una interna del contenedor.

Por ejemplo, la carpeta `db_data` que tenemos en la carpeta contenedora de los dockerfiles y el código fuente se montará dentro del contenedor en la carpeta `/var/lib/mysql`.

De este modo, al morir el contenedor los datos no serán borrados.

Variables de entorno

Los diferentes servicios pueden tener que compartir variables entre ellos. Por ejemplo, wordpress necesitará saber los datos de acceso a la BD donde guardará la información que necesite.

Esto, que generalmente haríamos a mano modificando archivos de configuración, se puede hacer definiendo variables externamente dentro del dockerfile.

Ejemplo de docker-compose

```yaml
version: '3.3'

services:
  basededatos:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
```

Ejecutar docker-compose

Para ejecutar docker-compose:

```
docker-compose up
```

El inconveniente de hacerlo así es que la consola se verá bloqueada por la ejecución del proceso hasta que el contenedor finalice.

De todos modos, también podemos ejecutar los contenedores en segundo plano (detached), utilizando:

```
docker-compose up -d
```

Visualizar contenedores

Si queremos visualizar los contenedores que se están ejecutando:

```
docker ps
```

Monitorizar contenedores

- Comprobar recursos que utiliza un contenedor: docker stats xxx
- Comprobar los logs de un contenedor: docker logs
- Comprobar todos los eventos que han ocurrido a un contenedor: docker events
- Listar procesos de un contenedor: docker top xxx

Eliminar contenedores

Eliminar todos los contenedores existentes (subshell):

```
docker stop $(docker ps -q)
docker rm -v $(docker ps -aq)
```

Persistencia

Los contenedores por defecto están completamente aislados del mundo exterior.

Podemos montar un volumen que funcionará como si fuer una carpeta compartida a la que puedes acceder desde otro lugar.

Dive

Existe una herramienta llamada dive que permite monitorizar una imagen para ver detalles, espacio ocupado, sistema de archivos, etc. e intentar optimizarla.

No he tenido tiempo de probarlo, pero el proyecto está en:

https://github.com/yosifkit/dive

Multi stage builds

Definimos un **dockerfile** y ponemos en primer lugar una línea FROM. A continuación ponemos otra línea FROM. Por ejemplo:

- FROM alpine golang
- FROM alpine

Puedes cargar el código fuente en una imagen y compilarlo Luego borrar el código fuente Etiquetamos el estado y utilizarlo como un nuevo statement. Todo lo que hay antes del segundo FROM se elimina

Buenas prácticas para construir contenedores

- Una sola aplicación por contenedor. Por ejemplo, PHP y Mysql en dos contenedores.
- Agregar el código fuente de la aplición lo más tarde posible. Las capas y dependencias se pueden cachear y acelerar las builds posteriores
- Eliminar todo lo que no sea necesario, o utilizar una imagen scratch o distroless.
- Hacer las imágenes lo más pequeñas posibles. Reduce downtimes, tiempo de arranque y espacio en disco.
- Etiquetar las imágenes
- Utiliza volúmenes para manejar guardar la configuración y los datos fuera de los contenedores

Ventajas de dockerizar

Algunas ventajas de cambiar a un entorno docker son:

- Ciclos de desarrollo más rápidos
- Aplicaciones más pequeñas
- Ayuda a tener más documentados los pasos de aprovisionamiento del servidor, al tenerlos que definir como instrucciones en un Dockerfile.
- Al hacer los dockerfiles tratas tu infraestructura como código que puede ser commited y compartido, así como ver un histórico de cambios.