Instituto Tecnológico Superior” TECNOECUATORIANO”

Nombre: Isaac Meza.

Estudio de los contenedores Docker

¿Qué es Docker?

Docker es un sistema de código abierto para correr aplicaciones, sistemas operativos y cualquier otra herramienta que usualmente necesitaría una instalación normal.

Al usar imágenes y contenedores los mismos proveen la gran ventaja de ocupar mucho menos espacio en disco, así como reducir el uso de memoria. Docker al correr en el kernel de Linux es compatible con casí todos los SO actuales.

Al descargar imágenes de aplicaciones como mysql, se pueden crear uno o varios contenedores que necesiten utilizar estas imágenes, de esta manera se ahorra todo el tedioso proceso de instalación al igual que los posibles errores que en muchas ocasiones ocurren al momento de instalar una aplicación de manera habitual. Al descargar la imagen de MySql:5 por ejemplo, y la usamos en un contenedor cualquiera, si creáramos otro contenedor que use esa misma imagen, Docker no la vuelve a descargar sino que utiliza la ya existente, de esta manera se ahora espacio en disco.

Debido a las facilidades que implica Docker, las compañías han optado por**dockerizar** **sus aplicaciones**, pues de esta manera es más sencillo tener toda la información necesaria para el despliegue de aplicaciones sin las necesidades de máquinas virtuales.

**¿Qué es Dockerizar?**

Dockerizar se refiere a la **implementación de**[**Docker**](https://bambu-mobile.com/devops/)**para empaquetar una aplicación** (software), para luego distribuirla y ejecutarla a través de los contenedores. También se le conoce como **contenerizar aplicaciones**. Dockerizar permite:

* La creación y distribución de códigos de forma más sencilla.
* Trabajar bajo la metodología DevOps

A grandes rasgos, la dockerización es el **proceso de generación de imágenes que contienen las aplicaciones** para poder ejecutarla o publicarla en el momento deseado.

* 1. Se elige una imagen base. Estas se pueden descargar desde Docker Hub. Depende de la solución tecnológica que se requiera, como lo puede ser Java, Ngnix o Angular. En caso de no poder ocupar la imagen deseada, se tendría que instalar todo lo respectivo a esta para hacer empleo de ella. Para este ejemplo se eligió node.
* 2. En el código, se crea un Dockerfile, es decir, todos los comandos iniciales para ensamblar una imagen.
* 3. En el Dockerfile se especifican varios comandos:
* *FROM node : versión* para determinar de dónde viene el servidor elegido.
* *WORKDIR / nombre de la carpeta* este se usa para decir qué información tendrá dentro el contenedor cuando se inicia en el equipo.
* *COPY . / nombre de la carpeta* para copiar los elementos dentro de esa carpeta.
* *EXPOSE + puerto* especificando el puerto que se usa.
* Debe especificar el punto de entrada y el comando: *ENTRYPOINT “[“npm”]”* y *CMD “[“start”]”* respectivamente.
* 4. Una vez terminado, se origina como archivo de Docker Compose: docker-compose.yml.
* 5. Ya que se ha creado el archivo, se agrega la versión elegida para ejecutar el archivo: version :"3".
* 6. Después se especifican los servicios, a su vez, los servicios pueden tener otras especificaciones como el puerto a usar, el archivo de Docker que va a usar o el volumen que va a ocupar. En esta instrucción es necesario especificar que los cambios producidos deben guardarse siempre.
* 7. Por último, se delimita el volumen: volumes + el nombre y su lugar.
* 8- Se revisa que el Dockerfile funcione de manera correcta, en la terminal se agrega el comando docker build .
* 9. Si no se marca algún error, significa que la imagen corre de manera correcta. En el caso contrario, se marcarán los errores y es posible realizar los cambios pertinentes.
* 10. Se crea un contenedor con la imagen para verificar de nuevo que todo esté funcionando: docker-compose up. En este paso se descargará todo lo necesario para que la imagen corra de manera eficiente.
* 11. En caso de existir errores, se pueden corregir sobre el Dockerfile docker-compose.yml
* 12. Una vez que todos los errores han sido corregidos, la app ya está dockerizada.

Aunque aquí se habla de manera general para ejecutar una imagen en específico, este proceso puede cambiar de una imagen a otra, así como la serie de dependencias y puertos que se deseen utilizar.

**Beneficios de Dockerizar**

Las aplicaciones dockerizadas han mostrado una serie de beneficios frente a aquellas que aún no se han convertido en una imagen para Docker:

* Reduce los recursos de infraestructura de las aplicaciones, así como el tiempo de desarrollo.
* Permite trabajar siguiendo la cultura de DevOps, con lo cual entregarán mejores resultados en menor tiempo.
* Cualquier aplicación se puede ejecutar en cualquier equipo una vez que haya sido Dockerizada.
* Cada aplicación cuenta con sus recursos en un contenedor en particular.
* Docker Hub permite elegir la imagen-base a emplear, así como cualquier plugin adicional.

Dockerizar ha permitido que grandes compañías reduzcan su inversión y obtengan mayores beneficios, pues cumplen con los proyectos.

**Comandos.**

* **Imágenes.**
  + docker pull “image”: permite descargar la imagen indicada alojada en el servidor de repositorios docker.hub.
  + docker images: permite visualizar todas las imágenes descargadas.
  + docker images rm ”image”: Elimina la imagen indicada.

* **Contenedores.**
  + docker create ”image”: Crea un contenedor sobre la imagen indicada (es decir podemos crear un contenedor en base a la imagen de mysql con el comando  docker create mysql) .
  + docker create —name “container\_name” ”image” : Crea un contenedor sobre la imagen indicada pero con el nombre especificado en el parámetro “container\_name".
  + docker create -p “local-port:container-port”: Crea un contenedor indicando que el mismo puede recibir solicitudes desde el puerto de nuestra máquina. En caso de no especificar el puerto al cual queremos dar acceso por ejemplo “-p 27017”, esto le indicará a Docker que debe crear un contenedor el cual corre en el puerto 27017, eligiendo de manera aleatoria un puerto de nuestro PC desde el cual se podrá acceder al contenedor.
  + docker create -p 27017:27017 —name monguito -e MONGO\_INITDB\_ROOT\_USERNAME= MONGO\_INITDB\_ROOT\_PASSWORD=

Este comando indica crear un contenedor que permita el mapeo de puertos permitiendo acceder al puerto de mongodb (27017) desde la direccion IP (0.0.0.0:27017 o localhost:27017), el contenedor se llamará “monguito”, le indicará a la imagen de mongo mediante las variables de entorno de USERNAME y ROOT\_PASSWORD el nombre usuario y las contraseña para acceder a la base de datos de mongo. Todo esto en la imagen de Mongo

* + docker start ”container": Inicia el contenedor indicado (se puede especificar el id del contenedor o el nombre).
  + docker stop ”container”: Detiene la ejecución del contenedor especificado (se puede especificar el id del contenedor o el nombre).
  + docker ps: Muestra una lista de todos los container que están en ejecución.
  + docker ps -a: Muestra una lista de todos los container sin importar que estén o no en ejecución.
  + docker rm “container”: Eliminar el contenedor especificado (se puede especificar el id del contenedor o el nombre).
  + docker logs “container”: Muestra el log o historial de información resultante durante la ejecución del contenedor especificado (se puede especificar el id del contenedor o el nombre).
  + docker logs —f | —follow “container”: Muestra el log o historial de información resultante durante la ejecución del contenedor especificado, escuchando los cambios que ocurran dentro del mismo (se puede especificar el id del contenedor o el nombre).

* **Imágenes/Contenedores.**
  + docker run “image” : Este comando descarga una imagen de no existir, luego de ser descargada crea su contenedor y por último lo pone en ejecución
  + docker run -d“image” : Hace lo mismo que el comando anterior pero sin mostrar constantemente los logs resultantes del comando docker run.

Ejemplificación



