Sistemas Operativos 2020 UEM

LAB: Docker en Google Cloud

Introducción:

Docker es una plataforma abierta para desarrollar, enviar y ejecutar aplicaciones. Con Docker, se puede separar las aplicaciones de la infraestructura y tratarla como una aplicación administrada. Docker ayuda a reducir el tiempo de ciclo entre la escritura del código y su ejecución.

Para esto, Docker combina las características de creación de contenedores de kernel con flujos de trabajo y herramientas que ayudan a administrar y a implementar las aplicaciones.

Los contenedores de Docker pueden usarse directamente en Kubernetes, lo que permite ejecutarlos fácilmente en Kubernetes Engine.

Desarrollo del Lab

En este lab, aprenderemos a realizar las siguientes tareas:

- Cómo compilar, ejecutar y depurar contenedores de Docker
- Cómo extraer imágenes de Docker desde Docker Hub y Google Container Registry
- Cómo enviar imágenes de Docker a Google Container Registry

1.- Activar una máquina virtual en GCP con Ubuntu 18 y disco de 50 GB

2.- Actualizar Ubuntu: $ sudo apt-get update

3.- Instalamos Docker

$ sudo apt install docker.io

4.- Hacemos que arranque desde el inicio

$ sudo systemctl start docker

$ sudo systemctl enable docker

5.- Confirmamos que hemos instalado docker

$ sudo docker –version

6.- Añadimos nuestro usuario a la lista de usuarios de docker

$ sudo usermod -a -G docker $USER

Aquí podemos reiniciar la máquina para que se hagan efectivos los cambios

Por otra parte debemos activar una regla firewall para poder tener tráfico por el puerto 4000. (Atender en clase como se hace)

7.- Hello World

Ejecutamos el siguiente comando para ejecutar un contenedor elemental llamado "hello-world" y comenzar:

```
$ docker run hello-world
```

(Escribir aquí el Resultado del comando)

Este contenedor simple muestra el mensaje `Hello from Docker!` en el terminal. Si bien el comando es simple, observe en el resultado la cantidad de pasos que se ejecutaron. El daemon de docker buscó la imagen hello-world en el equipo local, al no encontrarla, extrajo la imagen desde un registro público llamado Docker Hub, creó un contenedor de esa imagen y ejecutó el contenedor.

Ejecute el siguiente comando para consultar la imagen de contenedor que se extrajo de Docker Hub:

```
$ docker images
```

(Escribir aquí el Resultado del comando)

Esta es la imagen extraída del registro público Docker Hub. El ID de imagen está en formato hash SHA256. Este campo especifica la imagen de Docker que se aprovisionó. Cuando el daemon de Docker no puede encontrar una imagen de forma local, la buscará en el registro público de forma predeterminada. Ejecutemos el contenedor otra vez:

```
$ docker run hello-world
```

(Resultado del comando)

Tener en cuenta que la segunda vez que ejecuta ésto, el daemon de Docker encuentra la imagen en su registro local y ejecuta el contenedor a partir de esa imagen. No necesita extraer la imagen desde Docker Hub.

Finalmente, ejecute el siguiente comando para observar los contenedores en ejecución:

```
$ docker ps
```

(Resultado del comando)

No hay contenedores en ejecución. El contenedor hello-world que ejecutó antes ya se cerró. Para ver todos los contenedores, incluidos los que terminaron de ejecutarse, ejecute :

```
$ docker ps -a
```

(Resultado del comando)

8.- Crear una imagen Docker

Compilemos nuestra propia imagen de Docker a partir de una aplicación de "node" simple. Ejecute el siguiente comando para crear una carpeta llamada `test` y cambiarse a ella.

```
$ mkdir test && cd test
```

Creamos un `Dockerfile`: bien con el comando cat o si prefieren von el editor nano

```
$ cat > Dockerfile <<EOF
# Use an official Node runtime as the parent image
FROM node:6

# Set the working directory in the container to /app
WORKDIR app

# Copy the current directory contents into the container at /app
ADD . /app

# Make the container's port 80 available to the outside world
EXPOSE 80

# Run app.js using node when the container launches
```

```
CMD ["node", "app.js"]
EOF
```

Este archivo le indica al daemon de Docker cómo crear una imagen.

- La línea inicial especifica la imagen base, que, en este caso, es la imagen de Docker oficial para la versión 6 del entorno node js.
- En la segunda, configuramos el directorio de trabajo (actual) del contenedor.
- En la tercera, agregamos los contenidos actuales del directorio (indicados por el "`.`") al contenedor.
- Luego, exponemos el puerto del contenedor, de manera que acepte conexiones en ese puerto. Por último, ejecutamos el comando node para iniciar la aplicación.

Revise las referencias de los comandos de Dockerfile para comprender cada línea de `Dockerfile`.

Ahora escribiremos una aplicación de node y, después, crearemos la imagen.

Ejecute los siguientes comandos para crear la aplicación de node: (o bien a través de nano)

```
$ cat > app.js <<EOF
const http = require('http');

const hostname = '0.0.0.0';
const port = 80;

const server = http.createServer((req, res) => {
    res.statusCode = 200;
      res.setHeader('Content-Type', 'text/plain');
        res.end('Hello World\n');
});

server.listen(port, hostname, () => {
    console.log('Server running at http://%s:%s/', hostname, port);
});

process.on('SIGINT', function() {
    console.log('Caught interrupt signal and will exit');
    process.exit();
});
EOF
```

Este es un servidor HTTP simple que escucha en el puerto 80 y muestra la cadena "Hello World".

Ahora, creamos la imagen.

Observe otra vez el "`.`", que se refiere al directorio actual, por lo que debe ejecutar este comando desde dentro del directorio donde se encuentra el Dockerfile:

```
$ docker build -t node-app:0.1 .
```

Es posible que este comando tarde un par de minutos en terminar de ejecutarse. Una vez que finalice, el resultado debería ser similar al siguiente:

```
Sending build context to Docker daemon 3.072 kB
Step 1 : FROM node:6
6: Pulling from library/node
...
...
...
Step 5 : CMD node app.js
 ---> Running in b677acd1edd9
 ---> f166cd2a9f10
Removing intermediate container b677acd1edd9
Successfully built f166cd2a9f10
```

La opción `-t` es para asignar un nombre y una etiqueta a la imagen con la sintaxis `name:tag`. El nombre de la imagen es `node-app` y la `etiqueta` es `0.1`. Se recomienda usar etiquetas cuando se crean imágenes de Docker. Si no especifica una etiqueta, la etiqueta predeterminada será `latest` y se vuelve más difícil distinguir las imágenes recientes de las anteriores. Además, observe cómo cada línea del `Dockerfile` anterior genera capas intermedias del contenedor a medida que se compila la imagen.

Ahora, ejecute el siguiente comando para ver las imágenes que tenemos:

```
$ docker images
```

(Resultado del comando)

Tengamos en cuenta que node es la imagen base y node-app es la imagen que creamos. Para quitar node, primero debemos quitar node-app. El tamaño de la imagen es relativamente pequeño en comparación con las VM. Otras versiones de la imagen de node, como node:slim y node:alpine pueden proporcionar imágenes aún más pequeñas para aumentar la portabilidad. El tema de reducir los tamaños de los contenedores es un tema de investigación. Podemos ver todas las versiones en el repositorio oficial, aquí.

Ejecución de un contenedor

En este módulo, use este código para ejecutar contenedores a partir de la imagen que construimos:

```
$ docker run -p 4000:80 --name my-app node-app:0.1
```

(Resultado del comando)

La marca --name le permite asignar un nombre al contenedor, si así lo desea. La marca -p indica a Docker que mapee el puerto 4000 del host al puerto 80 del contenedor. Ahora podemos mediante un navegador alcanzar el servidor en http://IP_MAQUINA:4000. Sin el mapeo de puertos, no se podrá alcanzar el contenedor.

9.- Modificar la imagen Docker

Edite app.js con el editor de texto que prefiera (por ejemplo, nano o vim) y reemplace "Hello World" por cualquier otra cadena :

Compile esta imagen nueva y agregue la etiqueta `0.2`:

```
$ docker build -t node-app:0.2 .
```

(Resultado del comando)

Ejecutar el contenedor y comprobar que efectivamente se imprime otra cosa en el navegador.

10.- Depurar una imagen Docker

En algunos casos, necesitaremos iniciar una sesión de Bash interactiva dentro del contenedor en ejecución. Puedemos usar docker exec para hacerlo. Abramos otra terminal (Shell) ingrese el siguiente comando:

```
$ docker exec -it [container_id] bash
```

Las opciones -it permiten interactuar con un contenedor, ya que asignan un pseudo-tty y mantienen la entrada stdin abierta. Observe que Bash se ejecuta en el directorio WORKDIR (/app) especificado en el Dockerfile. A partir de aquí, tenemos la sesión de shell interactiva dentro del contenedor. Podemos ejecytar comandos como “ls”, “cat” etc. Para salir de la sesión de bash ejecutamos “exit”.

Para examinar los metadatos de un contenedor en Docker, utilizamos el comando “docker inspect”:

```
$ docker inspect [container_id]
```

(Resultado del comando)

Podemos utilizar `--format` para inspeccionar campos específicos del JSON que se muestra. Por ejemplo:

```
$ docker inspect --format='{{range
.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' [container_id]
```

(Resultado del comando)

11. Publicar una imagen Docker en Google Cloud.

Ahora vamos a guardar la imagen a Google Container Registry (gcr). Después, borraremos todos los contenedores y las imágenes para simular un entorno nuevo, y luego extraeremos los contenedores y los ejecutaremos. Esto demostrará la portabilidad de los contenedores de Docker.

Para enviar imágenes a un registro privado alojado en gcr, debemos etiquetar las imágenes con el nombre de un registro. El formato es `[hostname]/[project-id]/[image]:[tag]`.

Para gcr:

- `[hostname]`= gcr.io
- `[project-id]`= el ID de su proyecto
- `[image]`= el nombre de su imagen
- `[tag]`= cualquier etiqueta en string que elija. Si no se especifica, el valor predeterminado es "latest".

Instalemos gcloud sdk

$ snap install google-cloud-sdk

Podemos saber el nombre de nuestro proyecto

```
$ gcloud config list project
```

(Resultado del comando)

Creamos la imagen a subir

```
$ docker tag node-app:0.2 gcr.io/[project-id]/node-app:0.2
```

Vemos si realmente se creo

```
$ docker images
```

Activamos la autorización para subir la imagen al registro de google cloud

```
$ sudo su
```

```
# curl -fsSL "https://github.com/GoogleCloudPlatform/docker-credential-gcr/releases/download/v1.5.0/docker-credential-gcr_linux_amd64-1.5.0.tar.gz" | tar xz --to-stdout ./docker-credential-gcr > /usr/bin/docker-credential-gcr && chmod +x /usr/bin/docker-credential-gcr
```

```
# docker-credential-gcr configure-docker
```

Muy probablemente necesitemos habilitar la API de google container registry, lo hacemos desde el menú principal de la consola APIs y servicios->Panel de contral->Habilitar Apis y Sevicios

Buscamos Docker y si no está habilitada la habilitamos.

Subimos la imagen

```
# docker push gcr.io/[project-id]/node-app:0.2
```

(Resultado del comando)

Dejamos de ser superusuarios con “exit”

Podemos navegar a través de la consola a **Herramientas** > **Container Registry** o visitar `http://gcr.io/[project-id]/node-app`. Debería llegar a una página similar y podremos ver nuestra imagen en el repositorio

Probemos esta imagen. Podría iniciar una nueva VM, usar ssh en esa VM y, luego, instalar gcloud.. Para hacerlo más simple, solo quitaremos todos los contenedores y las imágenes para simular un entorno nuevo.

Detenga y quite todos los contenedores:

```
$ docker stop CONTENEDOR
$ docker rm CONTENEDOR
```

Quitemos las imágenes secundarias de node antes de quitar la principal

```
$ docker rmi node-app:0.2 gcr.io/[project-id]/node-app:0.2 node-app:0.1
$ docker rmi node:6
$ docker rmi $(docker images -aq) # remove remaining images
```

```
Si hacemos
```

```
$ docker images
```

Veremos que ya no tenemos ninguna imagen en el sistema

Recuperemos nuestra imagen del repositorio de google cloud

$ sudo su

```
# docker pull gcr.io/[project-id]/node-app:0.2
```

Y veremos como tenemos nuevamente nuestra imagen en la máquina

Salimos de superusuario con “exit”

```
$ docker images
```

(Resultado del comando)