Docker & Docker Tools Basics

Autor: Alberto Herrera Poza
Github: https://github.com/AlbertoIHP

Introducción

El presente documento tiene la finalidad de utilizar las Docker Tools que facilita Docker para el despliegue de ambientes de producción algo complejos. Para esto en primera instancia se revisarán algunos conceptos que son considerados básicos para poder realizar el correcto despliegue de una aplicación full-stack. Posteriormente, se bajará a la práctica los contenidos vistos.

1. Conceptos básicos

Antes de entrar de lleno en Kubernetes, es necesario entender antes que ya existen herramientas nativas de Docker para manejar el despliegue completo de una aplicación, definiremos entonces una serie de conceptos que se vuelven básicos cuando empezamos a hablar de Kubernetes.

- **Docker:**
  - Cree, envíe, publique, descargue y ejecute imágenes docker.
  - Cuando decimos "Docker", normalmente nos referimos al motor Docker, el núcleo de la plataforma docker. Consiste en un docker daemon, una API Rest para interactuar con el demonio y una interfaz de línea de comando (CLI) que se comunica con el demonio a través de la API Rest.

- **Docker Compose:**
  - Defina y ejecute varios contenedores vinculados entre sí en un solo host.
  - Útil para configurar flujos de trabajo de desarrollo y prueba.

- **Docker Machine:**
  - Herramienta para el aprovisionamiento y la administración de hosts docker (hosts virtuales que ejecutan el motor docker).
  - Crea automáticamente hosts, instala Docker Engine en ellos y luego configura los clientes de docker.
  - Puede usar la máquina para crear hosts Docker en su máquina local utilizando un software de virtualización como VirtualBox o VMWare Fusion.
  - La máquina Docker también es compatible con varios proveedores de nube como AWS, Azure, Digital Ocean, Google Compute Engine, OpenStack, RackSpace, etc.

- **Docker Swarm:**
    - Un swarm es un grupo de hosts dockers unidos en un clúster.
    - Un cluster Swarm consiste en un Swarm manager (generalmente el primero que se instancia) y un conjunto de workers (otros host).
    - Interactúa con el clúster ejecutando comandos en el swarm manager.
    - Con Swarm, puedes desplegar y escalar aplicaciones a diferentes host.
    - Swarm ayuda con la gestión, el escalado, la creación de redes, el descubrimiento de servicios y el equilibrio de carga entre los nodos en el clúster.

- **Docker Stack:**
    - Define y ejecuta múltiples contenedores en un cluster Swarm.
    - Al igual que docker-compose le ayuda a definir y ejecutar aplicaciones de contenedores múltiples en un solo host, docker-stack le ayuda a definir y ejecutar aplicaciones de contenedores múltiples en un clúster de enjambre.

Cabe destacar que la unidad base sigue siendo el Dockerfile, y en este caso debido a la limitante de host que supone usar compose, utilizaremos Stack. Se hará uso de Machine para poder crear los host en cuestión. Luego mediante Swarm, definiremos uno de los host creados con Machine, como Swarm Manager, y luego el resto será añadido como worker al Swarm Cluster.

2. *Dockerfile*

El Dockerfile para esta aplicación es.

```
# Dockerfile References:
https://docs.docker.com/engine/reference/builder/

# Start from golang:1.12-alpine base image
FROM golang:1.12-alpine

# The latest alpine images don't have some tools like (`git` and
`bash`).
# Adding git, bash and openssh to the image
RUN apk update && apk upgrade && \
    apk add --no-cache bash git openssh

# Add Maintainer Info
LABEL maintainer="Rajeev Singh <rajeevhub@gmail.com>"

# Set the Current Working Directory inside the container
WORKDIR /app

# Copy go mod and sum files
COPY go.mod go.sum ./

# Download all dependencies. Dependencies will be cached if the go.mod
and go.sum files are not changed
RUN go mod download

# Copy the source from the current directory to the Working Directory
inside the container
COPY . .

# Build the Go app
RUN go build -o main .

# Expose port 8080 to the outside world
EXPOSE 8080

# Run the executable
CMD ["./main"]
```

3. *Uploading the image*

```
# Build the image
$ docker build -t go-docker-swarm .

# Tag the image
$ docker tag go-docker-swarm callicoder/swarm-demo-app:1.0.0

# Login to docker with your docker id
$ docker login

# Push the image to docker hub
$ docker push callicoder/swarm-demo-app:1.0.0
```

4. *Docker Stack*

Para utilizar Stack, es necesario crear un fichero de configuración llamado docker-stack.yml, y su estructura es similar a la de compose.

```
version: '3.7'
services:
  # App Service
  app:
    # Configuration for building the docker image for the service
    image: callicoder/swarm-demo-app:1.0.0
    ports:
      - "8080:8080" # Forward the exposed port 8080 on the container to
port 8080 on the host machine
    deploy:
      replicas: 3
      resources:
        limits:
          cpus: "0.2"
          memory: 50M
      restart_policy:
        condition: on-failure
    environment: # Pass environment variables to the service
      REDIS_URL: redis:6379
    depends_on:
     - redis
    networks: # Networks to join (Services on the same network can
communicate with each other using their name)
      - webnet
  # Redis Service
  redis:
    image: redis:alpine
    ports:
      - "6379:6379"
    networks:
      - webnet
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure
      placement:
        constraints: [node.role == manager]
networks:
  webnet:
    driver: overlay
    attachable: true
```

5. *Docker Machine*

Para crear máquinas virtuales, es necesario utilizar el siguiente comando.

```
$ docker-machine create --driver virtualbox swarm-vm1
$ docker-machine create --driver virtualbox swarm-vm2
$ docker-machine create --driver virtualbox swarm-vm3
```

Docker-Machine le permite crear máquinas virtuales docker en varios proveedores de la nube, así como AWS, google cloud, etc. Por ejemplo, así es cómo puede crear una máquina virtual docker en AWS utilizando docker-machine:

```
$ docker-machine create --driver amazonec2 \
--amazonec2-access-key <ACCESS_KEY> \
--amazonec2-secret-key <ACCESS_SECRET> \
--amazonec2-region "us-east-1" \
--amazonec2-instance-type "t2.micro" \
aws-sandbox
```

6. *Docker Swarm Cluster*

Docker Swarm, necesita un primer worker, el cual es declarado como Administrador, a partir de ese, se pueden unir tantos workers como sean necesarios.

```
docker-machine ssh swarm-vm1 "docker swarm init --advertise-addr
192.168.99.100"

docker-machine ssh swarm-vm2 "docker swarm join --token
SWMTKN-1-073v8uw449vvkwngz6g0mtgivv50dbbqzt2o9f9jmwvwrkihoj-6glz90svgl4t
21z1nnio0oce2 192.168.99.100:2377"

docker-machine ssh swarm-vm3 "docker swarm join --token
SWMTKN-1-073v8uw449vvkwngz6g0mtgivv50dbbqzt2o9f9jmwvwrkihoj-6glz90svgl4t
21z1nnio0oce2 192.168.99.100:2377"
```

7. *Deploy App*

Una vez la estructura del cluster está definida, es posible desplegar la aplicación mediante Docker-Stack.

```
docker stack deploy -c docker-stack.yml swarm-stack
```

Ahora es posible listar los servicios que el daemon de Docker está ejecutando.

```
docker service ls
```

También es posible visualizar los logs.

```
docker service logs swarm-stack_app
```

Listar todas tareas.

```
docker stack ps swarm-stack
```

Por último, detener el proceso.

```
docker stack rm swarm-stack
```