Programa DevOps

UMSA
UNIVERSIDAD
DEL MUSEO SOCIAL ARGENTINO

Softtek®

Working Time #3.2 📝

Docker Compose

Es hora de que pongas en práctica todo lo aprendido. 🤓

Este apartado tiene el objetivo de ayudarte a seguir potenciando tus habilidades, por lo que a continuación encontrarás diferentes **desafíos** que podrás resolver de forma independiente y a tu ritmo.

¡Manos a la obra!

1. **Introducción:**

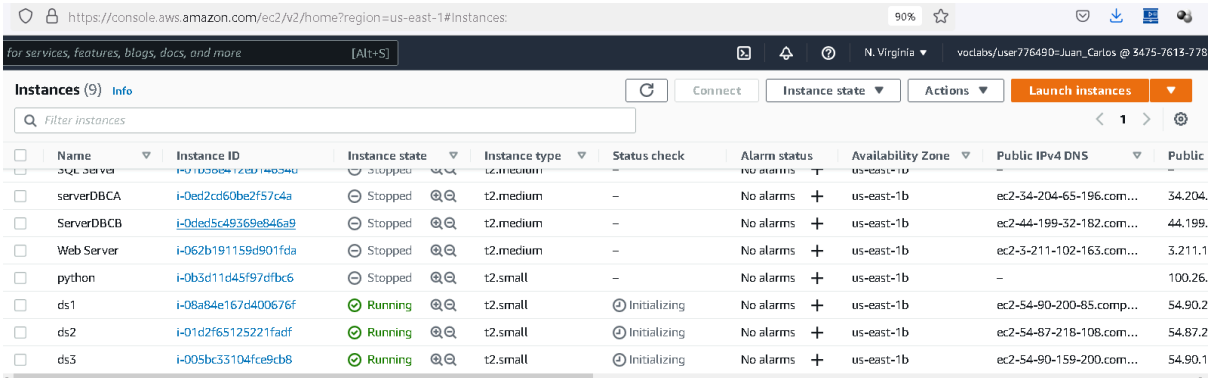
   Docker Swarm permite escalar las aplicaciones basadas en contenedores, gestionándolas en tantas instancias y en tantos nodos de red como se requiera.
   Si, por el contrario, se requiere la ejecución de un clúster una aplicación multi-contenedor, conocida también como “stack” (lote), es mejor recurrir a la herramienta Docker Compose.

   En este tutorial, se aprenderá a:

   - Crear 3 máquinas virtuales e instalar Docker en cada una de ellas
   - Configurar cada instancia
   - Instalar Docker Compose
   - Configurar Swarm
   - Ejecutar las pruebas correspondientes

2. **Requisitos Previos**

- Crear y ejecutar 3 máquinas virtuales AMI Linux en AWS EC2 -t2.small.



- Nombrar los nodos del 1 al 3
- Conectarse por ssh a cada nodo

3. **¿Dónde se lleva a cabo?**
   AWS y línea de comandos con SSH

4. **Tiempo de dedicación**
   30 minutos a 1 hora

5. **Recursos**

   Se utilizarán los amazon-linux-extra que traen una aplicación Python de ejemplo. Pero estos se instalarán de manera automática siguiendo los pasos del tutorial.

6. **Desafío**

En cada máquina creada conectados por ssh, en cada nodo ejecutar:

```
sudo -s
amazon-linux-extras install docker
service docker start
service docker status
```

Se creará una aplicación de ejemplo, que es una app en Python que mantiene una instancia en Redis e incrementa un contador ante cada visita al sitio.

- En el nodo 1, se crea un directorio para el Proyecto y posicionarse en el mismo:

```
mkdir stackdemo
cd stackdemo
```

- Se crea un archivo app.py con el siguiente código.

```
from flask import Flask
from redis import Redis
app = Flask(__name__)
redis = Redis(host='redis', port=6379)
@app.route('/')
def hello():
    count = redis.incr('hits')
    return 'Hello World! I have been seen {} times.\n'.format(count)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000, debug=True)
```

- Se crea otro archivo llamado requirements.txt que contiene estas dos líneas:

```
flask
redis
```

- Se crea otro archivo llamado dockerfile (sin extensión) y con el siguiente código:

```
# syntax=docker/dockerfile:1
FROM python:3.4-alpine
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

- Crear un archivo llamado docker-compose.yml con este contenido:

```
version: "3.9"
services:
  web:
    image: 127.0.0.1:5000/stackdemo
    build: .
    ports:
      - "8000:8000"
  redis:
    image: redis:alpine
```

📝**Nota**:

- La imagen para la web app está creada a partir del archivo Dockerfile definido arriba.

- Está etiquetada (tagged) como 127.0.0.1:5000 que es la dirección de nuestra registry local - normalmente se utiliza una externa-.
- Este paso es importante para distribuir la app al cluster swarm que se configurará luego.

- Instalar Docker compose, para lo cual se ejecuta:

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.26.0/docker-compose-$(uname -s)-$(uname -m)"  -o /usr/local/bin/docker-compose
```

```
sudo mv /usr/local/bin/docker-compose /usr/bin/docker-compose
```

```
sudo chmod +x /usr/bin/docker-compose
```

- Probar la app con Docker Compose

- Este comando genera la imagen de la web app, y crea dos contenedores.

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

- Probar que la app está ejecutándose correctamente:

```
docker-compose ps
```

- Se puede probar con curl desde consola y también a través de un browser:

```
curl http://localhost:8000
```

```
[root@ip-172-31-21-78 ec2-user]# curl http://localhost:8000
Hello World! I have been seen 1 times.
[root@ip-172-31-21-78 ec2-user]# curl http://localhost:8000
Hello World! I have been seen 2 times.
[root@ip-172-31-21-78 ec2-user]#
[root@ip-172-31-21-78 ec2-user]# curl http://localhost:8000
Hello World! I have been seen 3 times.
```

Hello World! I have been seen 33 times.

📝**Nota**: En caso de que no consiga conexión, se debe verificar que el security group de la VM acepte conexiones al puerto 8000.

Configuración de Swarm

- En el nodo1, que va a ocupar el rol de manager, ejecutar:

docker swarm init

- Recibirá un mensaje similar a este:

Swarm initialized: current node (umkuibvdyj89nyb4o2o80p6pd) is now a manager.
To add a worker to this swarm, run the following command:

```
docker swarm join --token
SWMTKN-1-0rromelv7ma636kvz7063wssvksh685n990f70rgcgokq1imab-6qr2aild
bzq0cq07nydwlrwnr 172.31.23.17:2377
```

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

- Dado que se está trabajando en una infraestructura Cloud de AWS EC2, se deben configurar los security groups de las VMs de modo de permitir conexión a los puertos que requiere Docker Swarm.
    - TCP puerto 2377 usado para gestión de comunicaciones del clúster.
    - TCP y UDP puerto 7946 usado para la comunicación intranodos (workers).
    - UDP puerto 4789 para tráfico de red del conjunto.

- Configurar un Docker registry local, dado que Swarm constituye múltiples Docker engines sincronizados, se necesita de una registry a efectos de distribución de las imágenes en los nodos. Se puede usar Docker hub o usar uno propio como en este ejemplo.

- Se inicia un registry como servicio:

docker service create --name registry --publish published=5000,target=5000 registry:2

- Se despliega el stack y se lo llama **stackdemo**

docker stack deploy --compose-file docker-compose.yml stackdemo

- Este último argumento es el nombre. Luego cada objeto network, volume o servicio tendrá este nombre como prefijo asociado.

- Verificar que esté ejecutándose como servicio

```
docker stack services stackdemo
```

- Verificar los nodos en el manager

```
docker node ls
```

- Probar múltiples accesos desde los diferentes nodos, usando curl o vía un browser a las IP públicas de los nodos. Cada acceso incrementa el contador.

- Luego probar la desvinculación de un nodo y se vuelve a probar el acceso desde los nodos restantes,

```
docker swarm leave
```

Verificar que la aplicación continúa funcionando sin inconvenientes.


UMSA
UNIVERSIDAD
DEL MUSEO SOCIAL ARGENTINO
Softtek®