Purpose

The purpose of this lab is to illustrate how Service Discovery works under Swarm Mode.

The application

WordPress is an open-source content management system (CMS) based on PHP and MySQL. It is a very simple containers application often used for demo purposes during meetup and conferences.

Init your swarm

Let’s create a Docker Swarm first. Open up the first instance and initiate Swarm mode cluster.

docker swarm init --advertise-addr $(hostname -i)

This node becomes a master node. The output displays a command to add a worker node to this swarm as shown below:

Swarm initialized: current node (xf323rkhg80qy2pywkjkxqusp) is now a manager.

To add a worker to this swarm, run the following command:

docker swarm join \

--token SWMTKN-1-089phhmfamjor1o1qj8s0l4wdhyvegphg6vtt9p3s8c35upltk-eecvhhtz1f2vpjhvc70v6v

vzb \

10.0.50.3:2377

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

ons.

The above token ID is unique for every swarm mode cluster and hence might differ for your setup. From the output above, copy the join command (*watch out for newlines*).

Next, Open up the new instance and paste the below command. This should join the new node to the swarm mode cluster and this new node becomes a worker node. In my case, the command would look something like this:

docker swarm join \

--token SWMTKN-1-089phhmfamjor1o1qj8s0l4wdhyvegphg6vtt9p3s8c35upltk-eecvhhtz1f2vpjhvc70v6v

vzb \

10.0.50.3:2377

Output:

$ docker swarm join --token SWMTKN-1-089phhmfamjor1o1qj8s0l4wdhyvegphg6vtt9p3s8c35upltk-eecvhh

tz1f2vpjhvc70v6vvzb 10.0.50.3:2377

This node joined a swarm as a worker.

Show members of swarm

Type the below command in the first terminal:

docker node ls

The output shows you both the manager and worker node indicating 2-node cluster:

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS

xf323rkhg80qy2pywkjkxqusp \* node1 Ready Active Leader

za75md1p0hpc2qswefj8uyktk node2 Ready Active

If you try to execute an administrative command in a non-leader node worker, you’ll get an error. Try it here:

docker node ls

Create an overlay network

docker network create -d overlay net1

The above command generates an ID:

4md6wyy0pdpdzku6dj2z7yxjf

List out the newly created overlay network using the below command:

docker network ls

The output should show the newly added network called “net1” holding swarm scope .

NETWORK ID NAME DRIVER SCOPE

c30f13d9c242 bridge bridge local

990fa0ad6ab6 docker\_gwbridge bridge local

c60123ff7abf host host local

v7sp7ev6xfoo ingress overlay swarm

4md6wyy0pdpd net1 overlay swarm

333c7d045239 none null

Creating MYSQL service

docker service create \

--replicas 1 \

--name wordpressdb \

--network net1 \

--env MYSQL\_ROOT\_PASSWORD=mysql123 \

--env MYSQL\_DATABASE=wordpress \

mysql:latest

The above command creates a service named “wordpressdb” which belongs to “net1” network which runs a single replica of the container. It displays service ID as an output as shown:

ip9a8zl9rke256q92itgrm8ov

Run the below command to list out the service:

docker service ls

The output should be like the following one (your ID will display different though).

ID NAME MODE REPLICAS IMAGE

ip9a8zl9rke2 wordpressdb replicated 1/1 mysql:latest

Let’s list the tasks of the wordpressdb service.

docker service ps wordpressdb

You should get an output like the following one where the 1 task of the service are listed.

ID NAME IMAGE NODE DESIRED STATE

CURRENT STATE ERROR PORTS

puoe9lvfkcia wordpressdb.1 mysql:latest node1 Running

Running about a minute ago

Creating WordPress service

docker service create \

--replicas 4 \

--name wordpressapp \

--network net1 \

--env WORDPRESS\_DB\_HOST=wordpressdb \

--env WORDPRESS\_DB\_PASSWORD=mysql123 \

wordpress:latest

The above command creates a service named “wordpressapp” which belongs to “net1” network which runs 4 copies of wordpressapp container. As output, this command displays a service ID as:

m4hca6rliz8wer2aojayv01r5

Listing out the services:

docker service ls

Output:

ID NAME MODE REPLICAS IMAGE

ID NAME MODE REPLICAS IMAGE

ip9a8zl9rke2 wordpressdb replicated 1/1 mysql:latest

m4hca6rliz8w wordpressapp replicated 4/4 wordpress:late

st

You can list the tasks of the wordpressapp service using the command:

docker service ps wordpressapp

Output:

ID NAME IMAGE NODE DESIRED STATE

CURRENT STATE ERROR PORTS

zg7wpvs1rbki wordpressapp.1 wordpress:latest node2 Running

Running 58 seconds ago

8rybe5m4urik wordpressapp.2 wordpress:latest node1 Running

Running about a minute ago

scia4v5i1znj wordpressapp.3 wordpress:latest node2 Running

Running 58 seconds ago

4avyixggcb8n wordpressapp.4 wordpress:latest node1 Running

Running about a minute ago

Service Discovery

Let us try to discover wordpressdb service from within one of wordpressapp container. Open up the manager node instance and run the below command:

Open up instance of worker node and verify what containers are running:

docker ps

This should display number of tasks(containers) running on the worker node locally:

CONTAINER ID IMAGE COMMAND CREATED STATUS

PORTS NAMES

52f16028e12c wordpress:latest "docker-entrypoint..." 2 minutes ago Up 2 minu

tes 80/tcp wordpressapp.1.zg7wpvs1rbkiy4zwo71yk031i

f3271e89d54e wordpress:latest "docker-entrypoint..." 2 minutes ago Up 2 minu

tes 80/tcp wordpressapp.3.scia4v5i1znj378gujluad2ku

As shown above, there are 2 instances of wordpressapp task(container) running on the worker node.

Now, Open up manager node and confirm what task are running:

docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS

PORTS NAMES

b68d99cad3da wordpress:latest "docker-entrypoint..." 5 minutes ago Up 4 minu

tes 80/tcp wordpressapp.2.8rybe5m4urikqsqje6hcpou9t

657cff3e37d5 wordpress:latest "docker-entrypoint..." 5 minutes ago Up 4 minu

tes 80/tcp wordpressapp.4.4avyixggcb8neej1h395ognt2

e71c164c36b3 mysql:latest "docker-entrypoint..." 10 minutes ago Up 10 min

utes 3306/tcp wordpressdb.1.puoe9lvfkciavkrzrkbrhrl6e

As we notice, there are 2 instances of wordpressapp task(container) running on the manager node(shown above) and 1 instance of wordpressdb.

Let’s pick up the wordpressdb task running on the manager node and try to reach out to wordpressapp running on the remote worker node. Because the container is missing the ping command we need to install it first:

docker exec -it e71 bash -c "apt update && apt -y install iputils-ping"

Once installed, we can ping wordpressapp as shown below:

docker exec -it e71 ping wordpressapp

This should work successfully and be able to ping the wordpressapp as service name.

PING wordpressapp (10.0.0.4): 56 data bytes

64 bytes from 10.0.0.4: icmp\_seq=0 ttl=64 time=0.052 ms

^C--- wordpressapp ping statistics ---

1 packets transmitted, 1 packets received, 0% packet loss

round-trip min/avg/max/stddev = 0.052/0.052/0.052/0.000 ms

Let us try to reach out to remote wordpressapp container from one of the wordpressdb instance running on the worker node by its hostname:

docker exec -it e71 ping wordpressapp.3.scia4v5i1znj378gujluad2ku

Output:

PING wordpressapp.3.scia4v5i1znj378gujluad2ku (10.0.0.5): 56 data bytes

64 bytes from 10.0.0.5: icmp\_seq=0 ttl=64 time=6.175 ms

64 bytes from 10.0.0.5: icmp\_seq=1 ttl=64 time=0.131 ms

^C--- wordpressapp.3.scia4v5i1znj378gujluad2ku ping statistics ---

2 packets transmitted, 2 packets received, 0% packet loss

round-trip min/avg/max/stddev = 0.131/3.153/6.175/3.022 ms

Voila ! We are able to ping wordpressapp service from container(running wordpresdb task) using the service name.Also, we were successful in reaching out to remote wordpressapp container using its hostname from one of wordpressdb container running in maanager node.