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.

To join a master node , you need to create a token with a manager privileges. Run the below command to get the master token.

[node2] (local) root@192.168.0.8 ~

$ docker swarm join-token manager

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

docker swarm join --token SWMTKN-1-5j7xka8z4ln2q1435c3ajr5kk69hj1acd3uitjgsh3vq5sfnrb-8mdwc22qbburs35y0ihniyrnj 192.168.0.8:2377

## 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

8mdwc22qbburs35y0ihniyrnj node3 Ready Active Reachable

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 one of wordpressdb task running on the manager node and try to reach out to wordpressapp running on the remote worker node as shown below:

docker exec -it e71 ping wordpressapp

This should work successfully and 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

Detailed:

## Create a Manager node

In this step you’ll initialize a new Swarm, join a single worker node, and verify the operations worked.

Run docker swarm init on **node1**.

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

Swarm initialized: current node (6dlewb50pj2y66q4zi3egnwbi) is now a manager.

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

docker swarm join \

--token SWMTKN-1-1wxyoueqgpcrc4xk2t3ec7n1poy75g4kowmwz64p7ulqx611ih-68pazn0mj8p4p4lnuf4ctp8xy \

10.0.0.5:2377

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

You can run the docker info command to verify that **node1** was successfully configured as a swarm manager node.

docker info

Containers: 2

Running: 0

Paused: 0

Stopped: 2

Images: 2

Server Version: 17.03.1-ee-3

Storage Driver: aufs

Root Dir: /var/lib/docker/aufs

Backing Filesystem: extfs

Dirs: 13

Dirperm1 Supported: true

Logging Driver: json-file

Cgroup Driver: cgroupfs

Plugins:

Volume: local

Network: bridge host macvlan null overlay

Swarm: active

NodeID: rwezvezez3bg1kqg0y0f4ju22

Is Manager: true

ClusterID: qccn5eanox0uctyj6xtfvesy2

Managers: 1

Nodes: 1

Orchestration:

Task History Retention Limit: 5

Raft:

Snapshot Interval: 10000

Number of Old Snapshots to Retain: 0

Heartbeat Tick: 1

Election Tick: 3

Dispatcher:

Heartbeat Period: 5 seconds

CA Configuration:

Expiry Duration: 3 months

Node Address: 10.0.0.5

Manager Addresses:

10.0.0.5:2377

<Snip>

The swarm is now initialized with **node1** as the only Manager node. In the next section you will add **node2** and **node3** as Worker nodes.

## Step 2.2 - Join Worker nodes to the Swarm

You will perform the following procedure on **node2** and **node3**. Towards the end of the procedure you will switch back to **node1**.

Now, take the entire docker swarm join ... command we copied earlier from node1 where it was displayed as terminal output. We need to paste the copied command into the terminal of **node2** and **node3**.

It should look something like this for **node2**. By the way, if the docker swarm join ... command scrolled off your screen already, you can run the docker swarm join-token worker command on the Manager node to get it again.

Remember, the tokens displayed here are not the actual tokens you will use. Copy the command from the output on **node1**. On **node2** and **node3** it should look like this:

docker swarm join \

--token SWMTKN-1-1wxyoueqgpcrc4xk2t3ec7n1poy75g4kowmwz64p7ulqx611ih-68pazn0mj8p4p4lnuf4ctp8xy \

10.0.0.5:2377

docker swarm join \

--token SWMTKN-1-1wxyoueqgpcrc4xk2t3ec7n1poy75g4kowmwz64p7ulqx611ih-68pazn0mj8p4p4lnuf4ctp8xy \

10.0.0.5:2377

Once you have run this on **node2** and **node3**, switch back to **node1**, and run a docker node ls to verify that both nodes are part of the Swarm. You should see three nodes, **node1** as the Manager node and **node2** and **node3** both as Worker nodes.

docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS

6dlewb50pj2y66q4zi3egnwbi \* node1 Ready Active Leader

ym6sdzrcm08s6ohqmjx9mk3dv node3 Ready Active

yu3hbegvwsdpy9esh9t2lr431 node2 Ready Active

The docker node ls command shows you all of the nodes that are in the swarm as well as their roles in the swarm. The \* identifies the node that you are issuing the command from.

Congratulations! You have configured a swarm with one manager node and two worker nodes.

# **Section 3: Deploy applications across multiple hosts**

Now that you have a swarm up and running, it is time to deploy our really simple sleep application.

You will perform the following procedure from **node1**.

## Step 3.1 - Deploy the application components as Docker services

Our sleep application is becoming very popular on the internet (due to hitting Reddit and HN). People just love it. So, you are going to have to scale your application to meet peak demand. You will have to do this across multiple hosts for high availability too. We will use the concept of Services to scale our application easily and manage many containers as a single entity.

You will perform this procedure from **node1**.

Lets deploy sleep as a Service across our Docker Swarm.

docker service create --name sleep-app ubuntu sleep infinity

of5rxsxsmm3asx53dqcq0o29c

Verify that the service create has been received by the Swarm manager.

docker service ls

ID NAME MODE REPLICAS IMAGE

of5rxsxsmm3a sleep-app replicated 1/1 ubuntu:latest

The state of the service may change a couple times until it is running. The image is being downloaded from Docker Store to the other engines in the Swarm. Once the image is downloaded the container goes into a running state on one of the three nodes.

At this point it may not seem that we have done anything very differently than just running a docker run .... We have again deployed a single container on a single host. The difference here is that the container has been scheduled on a swarm cluster.

Well done. You have deployed the sleep-app to your new Swarm using Docker services.

# **Section 4: Scale the application**

Demand is crazy! Everybody loves your sleep app! It’s time to scale out.

One of the great things about services is that you can scale them up and down to meet demand. In this step you’ll scale the service up and then back down.

You will perform the following procedure from **node1**.

Scale the number of containers in the **sleep-app** service to 7 with the docker service update --replicas 7 sleep-app command. replicas is the term we use to describe identical containers providing the same service.

docker service update --replicas 7 sleep-app

docker service scale sleep-app=7

The Swarm manager schedules so that there are 7 sleep-app containers in the cluster. These will be scheduled evenly across the Swarm members.

We are going to use the docker service ps sleep-app command. If you do this quick fast enough after using the --replicas option you can see the containers come up in real time.

docker service ps sleep-app

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

7k0flfh2wpt1 sleep-app.1 ubuntu:latest node1 Running Running 9 minutes ago

wol6bzq7xf0v sleep-app.2 ubuntu:latest node3 Running Running 2 minutes ago

id50tzzk1qbm sleep-app.3 ubuntu:latest node2 Running Running 2 minutes ago

ozj2itmio16q sleep-app.4 ubuntu:latest node3 Running Running 2 minutes ago

o4rk5aiely2o sleep-app.5 ubuntu:latest node2 Running Running 2 minutes ago

35t0eamu0rue sleep-app.6 ubuntu:latest node2 Running Running 2 minutes ago

44s8d59vr4a8 sleep-app.7 ubuntu:latest node1 Running Running 2 minutes ago

Notice that there are now 7 containers listed. It may take a few seconds for the new containers in the service to all show as **RUNNING**. The NODE column tells us on which node a container is running.

Scale the service back down just five containers again with the docker service update --replicas 4 sleep-app command.

docker service update --replicas 4 sleep-app

Verify that the number of containers has been reduced to 4 using the docker service ps sleep-app command.

docker service ps sleep-app

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

7k0flfh2wpt1 sleep-app.1 ubuntu:latest node1 Running Running 13 minutes ago

wol6bzq7xf0v sleep-app.2 ubuntu:latest node3 Running Running 5 minutes ago

35t0eamu0rue sleep-app.6 ubuntu:latest node2 Running Running 5 minutes ago

44s8d59vr4a8 sleep-app.7 ubuntu:latest node1 Running Running 5 minutes ago

You have successfully scaled a swarm service up and down.

# **Section 5: Drain a node and reschedule the containers**

Now you are doing maintenance on one of your servers so you will need to gracefully take a server out of the swarm without interrupting service to your customers.

Take a look at the status of your nodes again by running docker node ls on **node1**.

docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS

6dlewb50pj2y66q4zi3egnwbi \* node1 Ready Active Leader

ym6sdzrcm08s6ohqmjx9mk3dv node3 Ready Active

yu3hbegvwsdpy9esh9t2lr431 node2 Ready Active

You will be taking **node2** out of service for maintenance.

Lets see the containers that you have running on **node2**.

docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

4e7ea1154ea4 ubuntu@sha256:dd7808d8792c9841d0b460122f1acf0a2dd1f56404f8d1e56298048885e45535 "sleep infinity" 9 minutes ago Up 9 minutes sleep-app.6.35t0eamu0rueeozz0pj2xaesi

You can see that we have one of the slepp-app containers running here (your output might look different though).

Now lets jump back to **node1** (the Swarm manager) and take **node2** out of service. To do that, lets run docker node ls again.

docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS

6dlewb50pj2y66q4zi3egnwbi \* node1 Ready Active Leader

ym6sdzrcm08s6ohqmjx9mk3dv node3 Ready Active

yu3hbegvwsdpy9esh9t2lr431 node2 Ready Active

We are going to take the **ID** for **node2** and run docker node update --availability drain yournodeid. We are using the **node2** host **ID** as input into our drain command. Replace yournodeid with the id of **node2**.

docker node update --availability drain yu3hbegvwsdpy9esh9t2lr431

Check the status of the nodes

docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS

6dlewb50pj2y66q4zi3egnwbi \* node1 Ready Active Leader

ym6sdzrcm08s6ohqmjx9mk3dv node3 Ready Active

yu3hbegvwsdpy9esh9t2lr431 node2 Ready Drain

Node **node2** is now in the Drain state.

Switch back to **node2** and see what is running there by running docker ps.

docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

**node2** does not have any containers running on it.

Lastly, check the service again on **node1** to make sure that the container were rescheduled. You should see all four containers running on the remaining two nodes.

docker service ps sleep-app

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

7k0flfh2wpt1 sleep-app.1 ubuntu:latest node1 Running Running 25 minutes ago

wol6bzq7xf0v sleep-app.2 ubuntu:latest node3 Running Running 18 minutes ago

s3548wki7rlk sleep-app.6 ubuntu:latest node3 Running Running 3 minutes ago

35t0eamu0rue \\_ sleep-app.6 ubuntu:latest node2 Shutdown Shutdown 3 minutes ago

44s8d59vr4a8 sleep-app.7 ubuntu:latest node1 Running Running 18 minutes ago

# **Cleaning Up**

Execute the docker service rm sleep-app command on **node1** to remove the service called myservice.

docker service rm sleep-app

Execute the docker ps command on **node1** to get a list of running containers.

docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

044bea1c2277 ubuntu "sleep infinity" 17 minutes ago 17 minutes ag distracted\_mayer

You can use the docker kill <CONTAINER ID> command on **node1** to kill the sleep container we started at the beginning.

docker kill yourcontainerid

Finally, lets remove node1, node2, and node3 from the Swarm. We can use the docker swarm leave --force command to do that.

Lets run docker swarm leave --force on **node1**.

docker swarm leave --force

Then, run docker swarm leave --force on **node2**.

docker swarm leave --force

Finally, run docker swarm leave --force on **node3**.