Getting Started with Docker Services

Getting started with Docker service

We'll cover the following

Let's start with the smallest component of Docker swarm which is Docker services.

Initializing the swarm node

If you are using the Docker Desktop for Windows or for Mac, Docker swarm comes prebuilt. You can check that using `docker swarm -v`. To initialize the swarm mode, type `docker swarm init`.

```
$ docker swarm init
Swarm initialized: current node (ayxwwpio1sksk5hi2bdqxanf4) is now a manager.

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

    docker swarm join --token SWMTKN-1-19vn56bmfcc2l20jmts9gg9b152vkn47ruu6vtf
wgsl3646ha2-8lcseq2vaegpjqytnnw1xgksn 192.168.65.3:2377

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

The node you started will be a manager node. We can add other machines or hosts to this swarm network using the command above with the provided token. But for now, we will see all the operations on a single worker-manager node only.

Creating services

Like we have created containers using `docker run` command, a service does the same thing for a swarm cluster.

`docker service create <service name>` allows us to create a service on swarm with a defined number of containers in it.

Our aim is to create a load-balanced Flask app service and attach a database to it.

> Swarm cluster only works with prebuilt images. This means we cannot build an image using a Dockerfile while creating a container. So, we have to build the image of our app first and then create a service from it.

So, create an image using `docker build -t flask_app:3.0 .` or `docker pull venky8283/flask_app:3.0` to pull the image from Docker Hub.

Now, let's create the first service, which is our Flask login app.

Flask app

Type `docker service create -p 5000:5000 flask_app:3.0`. This will create a service with one container in it, which is also called a task.

```
$ docker service create -p 5000:5000  flask_app:3.0
image flask_app:3.0 could not be accessed on a registry to record
its digest. Each node will access flask_app:3.0 independently,
possibly leading to different nodes running different
versions of the image.

rnlrh4e262aew2c15ldz71c5k
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
```

If the image is not pulled from the Docker Hub, it will warn us saying other swarm nodes may not have the same image, which is fine for now.

At this point, if you open `localhost:5000` to see the app, you will get `MySQLdb._exceptions.OperationalError MySQLdb._exceptions.OperationalError: (2005, "Unknown MySQL server host 'database' (-2)")` error because we don't have

a database running yet.

So, the next task is to create a database service.

Database services

We already have a mysql-server image locally so, let's create a service by using it. Type `docker service create --name database --env-file <.env file location> --mount type=bind,src=<location of init.sql>,dst=/docker-entrypoint-initdb.d/init.sql mysql/mysql-server:5.7`

```
$ docker service create --name database --env-file .env --mount type=bind,src
=/Users/venkateshachint
alwar/Documents/Online_Projects/Docker/db/init.sql,dst=/docker-entrypoint-init
db.d/init.sql mysql/mysql-server:5.7
is4ct57yaqu0mimuywhy05asp
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
```

The command is a little bit longer. Let's understand each argument of the command.

- `--name`: name of the service through which it can be reachable
- `--mount`: bind local file system to the container file system
- `--env_file`: to provide environment variables using the file

Type `docker service ls` to list all the services available on the swarm node.

Networking

Both services are running now, but, you will get the same error as they are not connected. If you remember, we used the `link` argument to connect the database container. In swarm mode, the `link` option is not supported and we have to connect services using a network.

Services connected to the same network can talk to each other. So, to connect our database service to app service, we will create an "overlay network". In swarm mode, we deal with distributed systems, hence, we use an overlay network to establish communication between services.

To create an overlay network, type `docker network create --driver=overlay app`

To create an overlay network, type `docker network create --driver=overlay app`. This will create a new network named 'app' on Docker host.

```
$ docker network create --driver=overlay app
riv3bf55f1xkemfjmm9w2tb5i

$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
riv3bf55f1xk        app                 overlay             swarm
```

Updating services

Now, we have our overlay network. Let's attach it to both the services and let's see if it solves the problem or not.

To update a service, type `docker service update --network-add <network name> <service name/ID>`.

```
$ docker service update --network-add app rnlrh4e262ae
rnlrh4e262ae
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged

$ docker service update --network-add app database
database
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
```

As soon as the network is updated, you will see that your app start working normally.

We have deployed our app using services on the swarm node. Check out all the containers using `docker ps`. Docker service offers a few more commands to help you work with services. You can explore using `docker service --help` and `docker service <COMMAND> --help`.

We keep talking about distributed systems and scaling apps but we aren't done yet. So, in the next lesson, we will scale our services by increasing tasks/containers in it.