LAB 6: Swarm Mode

The application

The voting app is a multi-container application often used for demo purposes during Docker meetups and conferences. It basically allow users to vote between two choices, the default being “cat” and “dog”, but could be “space” or “tab”, too, if you feel like it. This application is available on Github and updated very frequently when new features are developed.

Initialize Your Swarm

First thing we need to do is tell our Docker hosts we want to use Docker Swarm Mode. Swarms *can* be just a single node, but that is unusual as you would have no high availability capabilities and you would severely limit your scalability. Most production swarms have at least three *manager* nodes in them and many *worker* nodes. Three managers is the minimum to have a true high-availability cluster with quorum. Note that manager nodes can run your container tasks the same as a worker node, but this functionality can also be separated so that managers only perform the management tasks.

Initializing Docker Swarm Mode is easy. In the first terminal window labeled [node1] enter the following:

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

That’s it - you now have your first Swarm manager and it is listening on the IP address returned by the (hostname -i) command. You should see some output that looks like this:

```
Swarm initialized: current node (tjocs7ul557phkmp6mkpjmu3f) is now a manager.


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


    docker swarm join --token <token> <host>


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

In the output of your swarm init, you are given a command in the middle that looks like `docker swarm join -token SWMTKN-X-abcdef.....` which you use to join workers nodes to the swarm. You are also given a second command `docker swarm join-token manager` for adding additional managers.

We are going to add a worker. Copy the “docker swarm join…” command from your manager’s output and paste it in the 2nd terminal window on your screen. *Make sure to copy the entire command - it’s likely to break across multiple lines - and do not copy the sample command above because your token will be different*.

You now officially have a Docker Swarm! Currently, you have one manager and one worker. As hinted at above, you would almost always have 3 or more manager nodes and several worker nodes in order to maintain high availability and scalability, but one of each is enough to get started.

Show Swarm Members

From the first terminal window, check the number of nodes in the swarm (running this command from the second terminal worker node will fail as swarm related commands need to be issued against a swarm manager).

```
docker node ls
```

The above command should output 2 nodes, the first one being the manager, and the second one a worker. You should see that your manager node is also the “Leader”. This is because you only have one manager node. The Leader is just what it sounds like: the main control node for all the managers. If your Leader node goes down for some reason, the other manager nodes will elect a new leader; just one reason why you would always have multiple manager nodes in true production.

Here is a view of managers and workers in Docker Swarm Mode. In our exercise, we have just one manager and one worker, but you can see

how multiple managers and workers interact in the diagram:

Clone the Voting App

Now to do something interesting we will retrieve the sample voting app code from Github.

Ensure you are in the first terminal - the manager - and enter the following two commands:

```
git clone https://github.com/docker/example-voting-app
cd example-voting-app
```

Deploy a Stack

A ***stack*** is a group of ***services*** that are deployed together: multiple containerized components of an application that run in separate instances. Each individual *service* can actually be made up of one or more containers, called ***tasks*** and then all the tasks & services together make up a *stack*.

As with Dockerfiles and the Compose files, the file that defines a stack is a plain text file that is easy to edit and track. In our exercise, there is a file called `docker-stack.yml` in the current folder which will be used to deploy the voting app as a stack. Enter the following to investigate the `docker-stack.yml`file:

```
cat docker-stack.yml
```

This YAML file defines our entire stack: the architecture of the services, number of instances, how everything is wired together, how to handle updates to each service. It is the source code for our application design. A few items of particular note:

- Near the top of the file you will see the line "services:". These are the individual application components. In the voting app we have redis, db, vote, result, worker, and visualizer as our services.
- Beneath each service are lines that specify how that service should run:
  - Notice the familiar term *image* from earlier labs? Same idea here: this is the container image to use for a particular service.
  - *Ports* and *networks* are mostly self-explanatory although it is worth pointing out that these networks and ports can be privately used within the stack or they can allow external communication to and from a stack.[2]
  - Note that some services have a line labeled *replicas*: this indicates the number of instances, or tasks, of this service that the Swarm managers should start when the stack is brought up. The Docker engine is intelligent enough to automatically load balance between multiple replicas using built-in load balancers. (The built-in load balancer can, of course, be swapped out for something else.)

Ensure you are in the [node1] manager terminal and do the following:

```
docker stack deploy --compose-file=docker-stack.yml voting_stack
```

You can see if the stack deployed from the [node1] manager terminal

```
docker stack ls
```

The output should be the following. It indicates the 6 services of the voting app's stack (named voting_stack) have been deployed.

```
NAME          SERVICES
voting_stack  6
```

We can get details on each service within the stack with the following:

```
docker stack services voting_stack
```

The output should be similar to the following, although naturally your IDs will be unique:

```
ID            NAME                     MODE        REPLICAS  IMAGE
10rt1wczotze  voting_stack_visualizer  replicated  1/1       dockersample
s/visualizer:stable
8lqj31k3q5ek  voting_stack_redis       replicated  1/1       redis:alpine
nhb4igkkyg4y  voting_stack_result      replicated  1/1       dockersample
s/examplevotingapp_result:before
nv8d2z2qhlx4  voting_stack_db          replicated  1/1       postgres:9.4
ou47zdyf6cd0  voting_stack_vote        replicated  2/2       dockersample
s/examplevotingapp_vote:before
rpnxwmoipagq  voting_stack_worker      replicated  1/1       dockersample
s/examplevotingapp_worker:latest
```

If you see that there are 0 replicas just wait a few seconds and enter the command again. The Swarm will eventually get all the replicas running for you. Just like our `docker-stack` file specified, there are two replicas of the *voting_stack_vote* service and one of each of the others.

Let's list the tasks of the vote service.

```
docker service ps voting_stack_vote
```

You should get an output like the following one where the 2 tasks (replicas) of the service are listed.

```
ID            NAME                 IMAGE
      NODE   DESIRED STATE  CURRENT STATE          ERROR  PORTS
my7jqgze7pgg  voting_stack_vote.1  dockersamples/examplevotingapp_vote:be
fore  node1  Running        Running 56 seconds ago
3jzgk39dyr6d  voting_stack_vote.2  dockersamples/examplevotingapp_vote:be
fore  node2  Running        Running 58 seconds ago
```

From the NODE column, we can see one task is running on each node. This app happens to have a built-in SWARM VISUALIZER to show you how the app is setup and running. You can also access the front-end web UI of the app to cast your vote for dogs or cats, and track how the votes are going on the result page. Try opening the front-end several times so you can cast multiple votes. You should see that the “container ID” listed at the bottom of the voting page changes since we have two replicas running.

The SWARM VISUALIZER gives you the physical layout of the stack, but here is a logical interpretation of how stacks, services and tasks are

inter-related:

Scaling An Application

Let us pretend that our cats vs. dogs vote has gone viral and our two front-end web servers are no longer able to handle the load. How can we tell our app to add more replicas of our *vote* service? In production you might automate it through Docker’s APIs but for now we will do it manually. You could also edit the `docker-stack.yml` file and change the specs if you wanted to make the scale size more permanent. Type the following at the [node1] terminal:

```
docker service scale voting_stack_vote=5
```

Now enter your `docker stack services voting_stack` command again. You should see the number of replicas for the vote service increase to 5 and in a few seconds Swarm will have all of them running. Go back to your front-end voting UI and refresh the page a few times. You should see the *container ID*listed at the bottom cycle through all 5 of your containers. If you go back and refresh your SWARM VISUALIZER you should see your updated architecture there as well.

Here’s our new architecture after scaling:

That’s all there is to it! Docker Swarm can easily and quickly scale your application’s services up and down as needs require. Again, in many situations you would probably want to automate this rather than manually scaling, which is pretty easy through the Docker APIs. You also have the option to swap out the built-in load balancer for something with additional controls, like an F5 or Citrix NetScaler or some other software you prefer.

Conclusion

Using only a couple of commands enables you to deploy a stack of services using Docker Swarm Mode to orchestrate the entire stack, all maintained in the simple, human readable Docker Compose file format.