Docker Orchestration

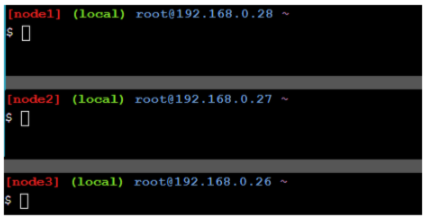
Docker Swarm Hands-On

You will deploy a simple application to a single host and learn how that works. Then, you will configure Docker Swarm Mode, and learn to deploy the same simple application across multiple hosts. You will then see how to scale the application and move the workload across different hosts easily.

Time: Approximately 30 minutes

- Configure Swarm Mode
- Deploy applications across multiple hosts
- Scale the application
- Drain a node and reschedule the containers
- Cleaning Up

Configure Swarm Mode



```
[node1] (local) root@192.168.0.28 ~
$ 
[node2] (local) root@192.168.0.27 ~
$ 
[node3] (local) root@192.168.0.26 ~
$ 
```

An example of running things manually and on a single host would be to create a new container on `node1` by running `docker run -dt ubuntu sleep infinity`.

```
docker run -dt ubuntu sleep infinity
```

Output:

```
# completely the user's responsibilites.                        #
#                                                                #
# The PWD team.                                                  #
##################################################################
[node1] (local) root@192.168.0.28 ~
$ docker run -dt ubuntu sleep infinity
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
d51af753c3d3: Pull complete
fc878cd0a91c: Pull complete
6154df8ff988: Pull complete
fee5db0ff82f: Pull complete
Digest: sha256:747d2dbbaaee995098c9792d99bd333c6783ce56150d1b11e333bbceed5c54d7
Status: Downloaded newer image for ubuntu:latest
51cefaa088d47dc1c7658c0bcd4e6cb3c81bcb963fbc62dded038775e2254d4d
```

This command will create a new container based on the ubuntu:latest image and will run the sleep command to keep the container running in the background. You can verify our example container is up by running `docker ps` on `node1`.

```
docker ps
```

Output:

```
CONTAINER ID        IMAGE               COMMAND             CREATED
51cefaa088d4        ubuntu              "sleep infinity"    53 seconds ago
```

In the following 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)
```

Output:

```
Swarm initialized: current node (omipy80grqsi0dc9l23fk9ne5) is now a manager.

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

    docker swarm join --token SWMTKN-1-46eiygo68w5fgm60dl4xmdbke41dvancdpokwwqik
```

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

```
docker info
```

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.

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.

Deploy the Application

> Let's deploy sleep as a Service across our Docker Swarm.

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

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

```
docker service ls
```

```
ID            NAME       MODE        REPLICAS  IMAGE
of5rxsxsmm3a  sleep-app  replicated  1/1       ubuntu:latest
```

Scale the Application

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
```

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

Note: Another way up scaling up and down of the services is to use `docker service scale <SERVICE-ID>=<NUMBER-OF-TASKS>` command.

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

```
docker service ps sleep-app
```

Scale the service back down to just four containers 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
```

Drain a Node

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

```
docker node ls
```

Output:

```
ID                            HOSTNAME            STATUS              AVAILABILI
tod30iaaxtddacqcj1resbsdh *   node1               Ready               Active
5w20r2jpoksidq38ncdbong1y     node2               Ready               Active
6yt2p3rxgl3xbwma6zrg7i1at     node3               Ready               Active
```

- Let's see the containers that you have running on `node2`.

```
docker ps
```

Output:

```
CONTAINER ID        IMAGE
4e7ea1154ea4        ubuntu@sha256:dd7808d8792c9841d0b460122f1acf0a2dd1f56404f8d1
```

- Now let's jump back to `node1` (the Swarm manager) and take `node2` out of service. To do that, run the command `docker node update --availability drain node2` at manager `node1`. After that let's check the status of the nodes with `docker node ls` command at manager.

```
docker node ls
```

Output:

```
ID                            HOSTNAME            STATUS              AVAILABILI
tod30iaaxtddacqcj1resbsdh *   node1               Ready               Active
5w20r2jpoksidq38ncdbong1y     node2               Ready               Drain
6yt2p3rxgl3xbwma6zrg7i1at     node3               Ready               Active
```

Note the availability of `node2` as `Drain`.

Now, let's write `docker swarm leave` on the `node3` and observe the status.

```
ID                            HOSTNAME            STATUS              AVAILABILI
tod30iaaxtddacqcj1resbsdh *   node1               Ready               Active
5w20r2jpoksidq38ncdbong1y     node2               Ready               Drain
6yt2p3rxgl3xbwma6zrg7i1at     node3               Down                Active
```

The `status` of `node3` now become to `Down`

Cleaning Up

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

```
docker service rm sleep-app
```

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

```
docker ps
```

- You can use the `docker kill <CONTAINER ID>` command on `node1` to kill the sleep container we started at the beginning.
- Finally, let's 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, if not already executed `run docker swarm leave --force` on `node3` then do so.

```
docker swarm leave --force
```