Lab 8: Docker Swarm

In this lab, you will work with Docker Swarm from the command line to manage running nodes, deploy services, and perform rolling updates on your services when needed. You will learn how to troubleshoot your Swarm nodes and deploy entire stacks using your existing Docker Compose files, as well as learning how you can use Swarm to manage your service configuration and secrets. The final part of this lab will provide you with the knowledge you need to get started using Swarmpit, which is a web-based interface for running and managing your Docker Swarm services and clusters.

Exercise 8.01: Running Services with Docker Swarm

This exercise is designed to help you become familiar with using the Docker Swarm commands to manage your services and containers. In the exercise, you will activate a cluster, set up a new service, test scaling up the service, and then remove the service from the cluster using Docker Swarm:

1. Although Swarm is included by default with your Docker installation, you still need to activate it on your system. Use the `docker swarm init` command to put your local system into Docker Swarm mode:

   ```
   docker swarm init
   ```

   Your output might be a little different from what you see here, but as you can see, once the swarm is created, the output provides details on how you can add extra nodes to your cluster with the `docker swarm join` command:

   ```
   Swarm initialized: current node (j2qxrpf0a1yhvcax6n2ajux69) is
   now a manager.
   To add a worker to this swarm, run the following command:
       docker swarm join --token SWMTKN-1-
   2w0fk5g2e18118zygvmvdxartd43n0ky6cmywy0ucxj8j7net1-5v1xvrt7
   1ag6ss7trl480e1k7 192.168.65.3:2377
   To add a manager to this swarm, run 'docker swarm join-token
   manager' and follow the instructions.
   ```

2. Now list the nodes you have in your cluster, using the `node ls` command:

   ```
   docker node ls
   ```

   You should have one node you are currently working on and its status should be `Ready`:

   ```
   ID         HOSTNAME          STATUS    AVAILABILITY
     MANAGER STATUS
   j2qx.. *   docker-desktop    Ready     Active
     Leader
   ```

   For clarity here, we have removed the `Engine Version` column from our output.

3. From your node, check the status of your swarm using the `docker info` command, providing further details of your Swarm cluster and how the node is interacting with it. It will also give you extra information if you need to troubleshoot issues later:

   ```
   docker info
   ```

As you can see from the output, you get all the specific details of your Docker Swarm cluster, including `NodeID` and `ClusterID`. If you don't have Swarm set up correctly on your system, all you will see is an output of `Swarm: inactive`:

```
…
Swarm: active
  NodeID: j2qxrpf0a1yhvcax6n2ajux69
  Is Manager: true
  ClusterID: pyejfsj9avjn595voauu9pqjv
  Managers: 1
  Nodes: 1
  Default Address Pool: 10.0.0.0/8
  SubnetSize: 24
  Data Path Port: 4789
  Orchestration:
   Task History Retention Limit: 5
  Raft:
   Snapshot Interval: 10000
   Number of Old Snapshots to Retain: 0
   Heartbeat Tick: 1
   Election Tick: 10
  Dispatcher:
   Heartbeat Period: 5 seconds
  CA Configuration:
   Expiry Duration: 3 months
   Force Rotate: 0
```

4. Start your first service on your newly created swarm. Create a service named `web` using the `docker service create` command and the `--replicas` option to set two instances of the container running:

```
docker service create --replicas 2 -p 80:80 --name web nginx
```

You will see that the two instances are successfully created:

```
uws28u6yny7ltvutq38166a1f
overall progress: 2 out of 2 tasks
1/2: running   [==========================================>]
2/2: running   [==========================================>]
verify: Service converged
```

5. Similar to the `docker ps` command, you can see a listing of the services running on your cluster with the `docker service ls` command. Execute the `docker service ls` command to view the details of the `web` service created in the *step 4*:

```
docker service ls
```

The command will return the details of the `web` service:

```
ID              NAME  MODE          REPLICAS   IMAGE
  PORTS
uws28u6yny7l    web   replicated    2/2        nginx:latest
  *:80->80/tcp
```

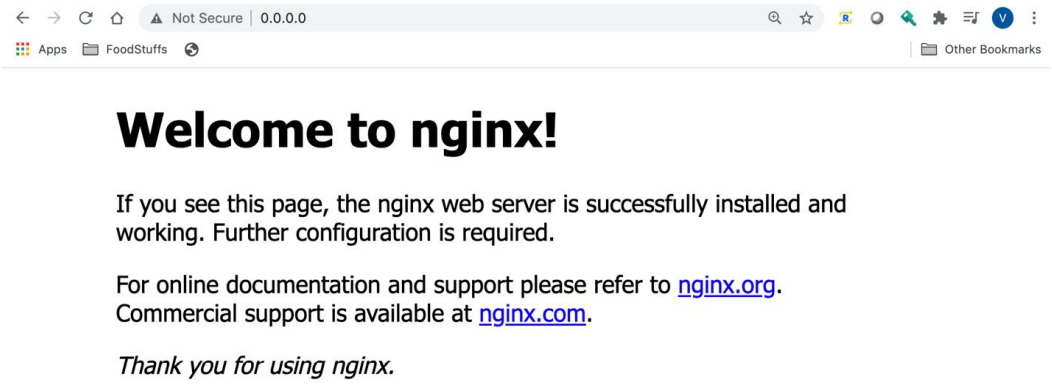
6. To view the containers currently running on your swarm, use the `docker service ps` command with the name of your service, `web`:

```
docker service ps web
```

As you can see, you now have a list of the containers running our service:

```
ID      NAME    IMAGE    NODE               DESIRED
  CURRENT STATE
viyz    web.1   nginx    docker-desktop     Running
  Running about a minute ago
mr4u    web.2   nginx    docker-desktop     Running
  Running about a minute ago
```

7. The service will only run the default `Welcome to nginx!` page. Use the node IP address to view the page. In this instance, it will be your localhost IP, `0.0.0.0`:



8. Scaling the number of containers running your service is easy with Docker Swarm. Simply provide the `scale` option with the number of total containers you want to have running, and the swarm will do the work for you. Perform the command shown here to scale your running web containers to `3`:

```
docker service scale web=3
```

The following output shows that the `web` service is now scaled to `3` containers:

```
web scaled to 3
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
```

9. As in *step 5* of this exercise, run the `service ls` command:

```
docker service ls
```

You should now see three `web` services running on your cluster:

```
ID              NAME    MODE          REPLICAS    IMAGE
    PORTS
uws28u6yny7l    web     replicated    3/3         nginx:latest
    *:80->80/tcp
```

10. The following change is more suited to a cluster with more than one node, but you can run it anyway to see what happens. Run the following `node update` command to set the availability to `drain` and use your node ID number or name. This will remove all the containers running on this node as it is no longer available on your cluster. You will be provided with the node ID as an output:

```
docker node update --availability drain j2qxrpf0a1yhvcax6n2ajux69
```

11. If you were to run the `docker service ps web` command, you would see each of your `web` services shut down while trying to start up new `web` services. As you only have one node running, the services would be sitting in a pending state with `no suitable node` error. Run the `docker service ps web` command:

```
docker service ps web
```

    The output has been reduced to only show the second, third, fifth, and sixth columns, but you can see that the service is unable to start. The `CURRENT STATE` column has both `Pending` and `Shutdown` states:

```
NAME          IMAGE             CURRENT STATE
  ERROR
web.1         nginx:latest      Pending 2 minutes ago
  "no suitable node (1 node…"
\_ web.1      nginx:latest      Shutdown 2 minutes ago
web.2         nginx:latest      Pending 2 minutes ago
  "no suitable node (1 node…"
\_ web.2      nginx:latest      Shutdown 2 minutes ago
web.3         nginx:latest      Pending 2 minutes ago
  "no suitable node (1 node…"
\_ web.3      nginx:latest      Shutdown 2 minutes ago
```

12. Run the `docker node ls` command:

```
docker node ls
```

    This shows that your node is ready but in an `AVAILABILITY` state of `Drain`:

```
ID          HOSTNAME           STATUS    AVAILABILITY
  MANAGER STATUS
j2qx.. *    docker-desktop     Ready     Drain
  Leader
```

13. Stop the service from running. Use the `service rm` command, followed by the service name (in this instance, `web`) to stop the service from running:

```
docker service rm web
```

    The only output shown will be the name of the service you are removing:

```
web
```

14. You don't want to leave your node in a `Drain` state as you want to keep using it through the rest of the exercises. To get the node out of a `Drain` state and prepare to start managing swarm, set the availability to `active` with the following command using your node ID:

    ```
    docker node update --availability active j2qxrpf0a1yhvcax6n2ajux69
    ```

    The command will return the hash value of the node, which will be different for every user.

15. Run the `node ls` command:

    ```
    docker node ls
    ```

    It will now show the availability of our node as `Active` and ready your services to run again:

    ```
    ID          HOSTNAME           STATUS    AVAILABILITY
      MANAGER STATUS
    j2qx.. *    docker-desktop     Ready     Active
      Leader
    ```

16. Use the `docker node inspect` command with the `--format` option and search for the `ManagerStatus.Reachability` status to ensure that your node is reachable:

    ```
    docker node inspect j2qxrpf0a1yhvcax6n2ajux69 --format "{{
    .ManagerStatus.Reachability }}"
    ```

    If the node is available and can be contacted, you should see a result of `reachable`:

    ```
    reachable
    ```

17. Search for `Status.State` to ensure that the node is ready:

    ```
    docker node inspect j2qxrpf0a1yhvcax6n2ajux69 --format "{{ .Status.State }}"
    ```

    This should produce `ready`:

    ```
    ready
    ```

This exercise should have given you a good indication of how Docker Swarm is able to simplify your work, especially when you start to think about deploying your work into a production environment. We used the Docker Hub NGINX image, but we could easily use any service we have created as a Docker image that is available to our Swarm node.

Summary

This lab has done a lot of work in moving our Docker environments from manually starting single-image services to a more production-ready and complete environment with Docker Swarm. We started this lab with an in-depth discussion of Docker Swarm and how you can manage your services and nodes from the command line, providing a list of commands and their uses.

In the next lab, we will introduce Kubernetes, which is another orchestration tool used to manage Docker environments and applications. Here, you will see how you can use Kubernetes as part of your projects to help

reduce the time you are managing services and improve the updating of your applications.