QA Consulting

# Docker Swarm CLI

**TRAINING MATERIALS - MODULE HANDOUT**

---

Contacts

*team.qac.all.trainers@qa.com*

*www.consulting.qa.com*

# Contents

# Overview

The CLI for Swarm is built in to the existing Docker client, so wherever you have Docker installed, you will have this functionality included with it. We can use the CLI for managing all aspects of Swarm including, nodes and services.

# Services

Services are how we recognise are applications and workloads that are running in the Swarm cluster. Services can be replicated and spread across multiple nodes for high availability.

## Create

Services can be created by providing at least a Docker Image to run. The Docker Image provided must be available on the node the container is going to be started on or from a remote registry, the latter being the preferred option.

```
docker service create [OPTIONS] [IMAGE]
```

```
docker service create nginx:latest
```

### Service Naming on Creation

When we don't provide a name for the service, one will be generated. It's much preferred to name our services so that we can easily refer to them later one, whether this be in scripts or manually.

```
docker service create --name [SERVICE_NAME] [OPTIONS] [IMAGE]
```

```
docker service create --name nginx nginx:latest
```

### Replicas on Creation

One of the nicest traits about container orchestration tools is the scalability and Docker Swarm isn't an exception to this. The amount of replicated services can be defined on the service creation. Replicas will be distributed across the available nodes evenly.

```
docker service create --replicas [REPLICA_COUNT] [OPTIONS] [IMAGE]
```

```
docker service create --replicas 2 nginx:latest
```

### Publishing Ports

Ports can be published like usual in Docker Swarm however there are some slightly different behavioural traits in Swarm due to how the routing mesh works. When you publish service in Swarm it is then accessible from the private IP and port of any node in the cluster. There is a load balancer on every node that distributes traffic across all replica containers. Published services cannot be accessed on the loopback interface (localhost).

```
docker service create --publish [PUBLISHED_PORT]:[CONTAINER_PORT] [OPTIONS] [IMAGE]
```

```
docker service create --publish 80:80 nginx:latest
```

## Update

Updating is useful feature for being able to update the published ports for example. Update commands require the name of the service that you are going to be applying the update to.

#### Publishing Ports

```
docker service update --publish-add [PUBLISHED_PORT]:[CONTAINER_PORT] [OPTIONS] [SERVICE]
```

```
docker service update --publish-add 80:80 nginx
```

#### Rolling Updates

When we set a new image for a service, a rolling update is performed across the cluster.

```
docker service update --image [IMAGE] [OPTIONS] [SERVICE]
```

```
docker service update --image nginx:latest nginx
```

### Listing

All the current services that are up with their replica count can be seen by listing them.

```
docker service ls [OPTIONS]
```

```
docker service ls
```

### Deleting

Deleting a service will remove all of the associated containers.

```
docker service rm [SERVICE]
```

```
docker service rm nginx
```

## Nodes

Nodes can join a swarm easily so as they have access to a manager node and the correct token.

### Listing

The current nodes that are up can be listed.

```
docker node ls [OPTIONS]
```

```
docker node ls
```

### Joining Swarms

There are two types of join tokens that can be used, the worker and manager tokens. The token you use when adding the node will effect what type of node it will be.

#### Getting Join Token

```
docker swarm join-token [NODE_TYPE]
```

```
docker swarm join-token worker
```

## Removing Node From Swarms

### Node Draining

Before leaving a swarm the node should first be "drained". This rejigs the containers across the cluster so they are balanced properly again but without the node that has been drained.

```
docker node update --availability drain [OPTIONS][NODE]
```

```
docker node update --availability drain node1
```

### Leave the Swarm

On the node that is leaving you can use the leave command to shutdown the node.

```
docker swarm leave [OPTIONS]
```

```
docker swarm leave
```

### Removing the Node

Once the node has been shutdown, you can remove it using the manager node.

```
docker node rm [OPTIONS] [NODE_NAME]
```

```
docker node rm node1
```

### Removing a Node Forcefully

If a node has been compromised or is unresponsive then you may want to remove it forcefully.

```
docker node rm [OPTIONS] [NODE_NAME]
```

```
docker node rm -f node1
```

# Tasks

## Create Cluster

Put together a cluster with 3 nodes, one manager and two workers.

## Create a Service

Create a service that uses the **bobcrutchley/python-http-server:latest** image and is named **python-http-server**.

## Update the Service

Update the service so that there are 10 replicas and so that the port 9000 (inside the container) has been published to 80 (outside the container).

## Access the Service

Use the curl CLI tool to view the info.json file served by the application, this file shows the name of the host that it is running on.
The info.json file can be accessed at **http://[YOUR_PRIVATE_IP]/info.json**.
You can substitute the private IP address here with the private IP address of your manager node.
Curl the file multiple times, what do you notice about the output?

## Remove the Worker Nodes

Update the amount of replicas to 2.
Drain both of the worker nodes and remove them.

## Jenkins in Swarm

Using the **–mount** option when creating a service, try to get Jenkins running in Swarm with Docker working and the home folder persisted.