Docker & Docker swarm

# What is Docker?

Docker is a tool designed to make it easier to create, deploy and run applications by using containers.

**Container =** Allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies and ship it all out as one package.

Developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

# What is Swarm Mode?

Enables the ability to deploy containers across multiple hosts, using overlay networks for service discovery with a built-in load balancer for scaling the services.

* **Node =** Is an instance of the Docker Engine connected to the Swarm. Nodes are either managers or workers. **Managers** schedules which container to run where. **Workers** execute the tasks. By default, Managers are also workers.
* **Services =** Is a high-level concept relating to a collection of tasks to be executed by workers. An example of a service is HTTP service running as a Docker Container on three nodes.
* **Load Balancing =** Docker includes a load balancer to process request across all containers in the service. (defined as the methodical and efficient distribution of network or application traffic across multiple servers in a server farm.)

# Getting Started with Swarm Mode

## Step 1 - Create Swarm Mode Cluster

1. Overview possible command : ***docker swarm –help***
2. Initialisation of Swarm mode : ***docker swarm init***

After running this command, the Docker Engine knows how to work with a cluster and becomes the manager. A token is made for adding additional nodes in a secure fashion.

## Step 2 - Join Cluster

If nodes happen to disappear, because of a crash, the containers which were running on those hosts will be automatically rescheduled onto other available nodes. This ensures you don’t lose capacity and provides high-availability.

1. Ask manager what the token is : ***docker swarm join-token (worker|manager)***
2. Run the given token on second host : (example) ***docker swarm join --token SWMTKN-1-36mdjplyqmeq7c5fp5c41ajaco5j8ed6kxhhx7pzeq5van1yfr-3ozhaa7doys4tdd3zbkw5t35n 172.17.0.64:2377***
3. View all nodes in the clusters : ***docker node ls***

## Step 3 - Create overlay Network

The overlay network is used to enable containers on different hosts to communicate.

1. Create a new overlay network (called Skynet) : ***docker network create -d overlay Skynet***

All containers registered to this network can communicate with each other, regardless of which node they are deployed onto.

## Step 4 - Deploy Service

The spread approach ensures that containers are deployed across the cluster evenly. This means if one of the nodes is removed from the cluster, the instances would be rescheduled across the remaining available nodes.

1. Make a service which deploys katacoda/docker-http-server and name it http and it should be attached to the newly created Skynet network. To ensure replication and availability, we are running 2 instances, of replicas, of the containers across out cluster. We load balance these 2 containers together on port 80. : ***docker service create –name http –network Skynet –replicas 2 -p 80:80 katacode/docker-http-server***
2. View services running on the cluster : ***docker service ls***
3. View the containers which have started : ***docker ps***
4. Issue an HTTP request to public port, processed by the 2 containers : **curl host01**

## Step 5 - Inspect State

This allows you to inspect the health and state of your cluster and the running applications

1. View list of all the task associated with a service across the cluster: ***docker service ps http***
2. View the details and configuration of a service:  ***docker service inspect –pretty http***
3. Ask what tasks it is currently running (Self refers to the manager node) : ***docekr node ps self***
4. Using ID of a node you can query individual hosts : ***docker node ps $(docker node ls -q | head -n1)***

## Step 6 - Scale Service

A Service allows us to scale how many instances of a task is running across the cluster. As it understands how to launch containers and which

1. Scale our http service to be running across five containers : ***docker service scale http=5***

# Create Overlay Network

Overlay networks allow containers to communicate as if they’re on the same host.

## Step 1 – Initialise Swarm Mode

By default, Docker works as an isolated single-node. All containers are only deployed onto the engine. Swarm Mode turns it into a multi-host cluster-aware engine.

1. **Do 1) Step 1 & 1) Step 2**

## Step 2 – Create Network

When new services are deployed via Swarm Mode, they can utilise this network allowing containers to communicate.

1. Create Overlay Network : ***docker network create -d overlay app1-network***
2. All networks can be viewed using : ***docker network ls***

## Step 3 – Deploy Backend

Once network has been created, services can be deployed and able to communicate with other containers on the network.

1. Deploy a Redis service using the network, name of service will be redis that can be used for discobery via DNS : ***docker service create –name redis –network app1-network redis:alpine***

## Step 4 – Deploy Frontend

With the overlay network and redis deployed, it’s now possible to deploy a Web App to use redis to persist data.

1. Create new service, with a two-node deployment, each container will be deployed onto different hosts :

***docker service create \***

***--network app1-network -p 80:3000 \***

***--replicas 1 --name app1-web \***

***katacoda/redis-node-docker-example***

1. Test by sending a HTTP request will persist the IP of the client in redis : ***curl host01***
2. Also do this in a worker to see of it works : ***curl host01***

# Load Balance and Service Discover in Swarm Mode

## Step 1 – Initialise Cluster

1. **Do 1) Step 1 & 1) Step 2**

## Step 2 – Port Load Balance

By default, request to Services are load balanced based on the public port.

1. Create a new service called Ibapp1 with two containers running. The service is exposed via port 81 : ***docker service create --name lbapp1 --replicas 2 -p 81:80 katacoda/docker-http-server***
2. When request are made to a node in our cluster on port 81, it will distribute the load across the two containers : ***curl host01:81***
3. Do previous command in second host

## Step 3 – Virtual IP and Service Discovery

In this step, you will create a load balanced http that is attached to an overlay network and look up its Virtual IP

1. This network will be a “swamr-scoped” network, this means that only containers launched as a service can attach itself to the network.