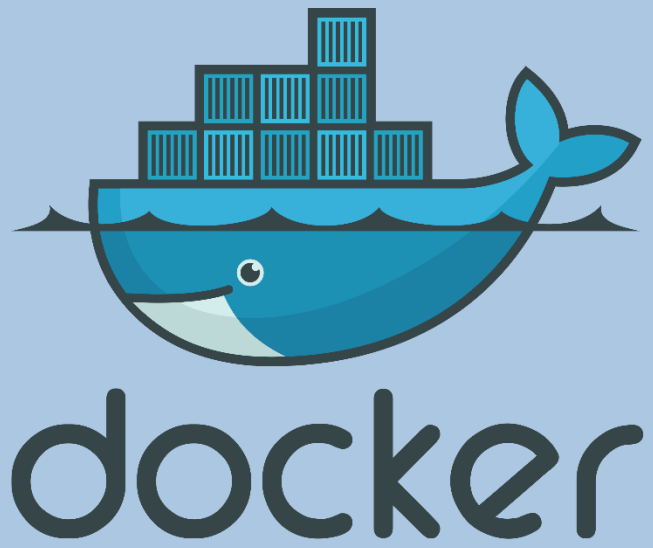


Docker Tutorials

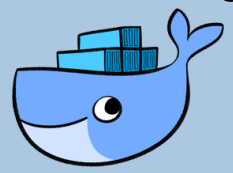
Getting Started With Swarm Mode

Introduction

In this scenario, you will learn how to initialize a Docker Swarm Mode cluster and deploy networked containers using the built-in Docker Orchestration. The environment has been configured with two Docker hosts.

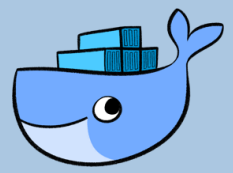
Key Concepts

- **Node**: A Node is an instance of the Docker Engine connected to the Swarm. Nodes are either managers or workers. Managers schedules which containers to run where. Workers execute the tasks. By default, Managers are also workers.
- **Services**: A service is a high-level concept relating to a collection of tasks to be executed by workers. An example of a service is an HTTP Server running as a Docker Container on three nodes.
- **Load Balancing**: Docker includes a load balancer to process requests across all containers in the service.



Step 1 - Initialize Swarm Mode❶

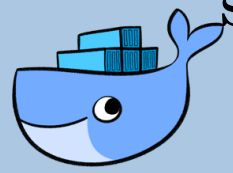
- Turn single host Docker host into a Multi-host Docker Swarm Mode.
  - Becomes Manager By default, Docker works as an isolated single-node. All containers are only deployed onto the engine. Swarm Mode turns it into a multi-host cluster-aware engine.
  - The first node to initialize the Swarm Mode becomes the manager. As new nodes join the cluster, they can adjust their roles between managers or workers. You should run 3-5 managers in a production environment to ensure high availability



Step 1 - Initialize Swarm Mode❷

Task: Create Swarm Mode Cluster

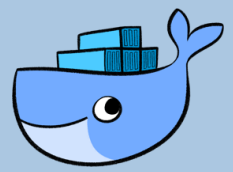
- Swarm Mode is built into the Docker CLI. You can find an overview the possibility commands via
  - docker swarm --help
- The most important one is how to initialise Swarm Mode. Initialisation is done via *init*.
  - docker swarm init
- After running the command, the Docker Engine knows how to work with a cluster and becomes the manager. The results of an initialisation is a token used to add additional nodes in a secure fashion. Keep this token safe and secure for future use when scaling your cluster.



Step 2 - Join Cluster❶

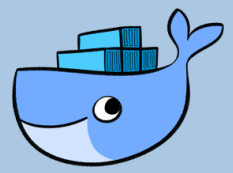
With Swarm Mode enabled, it is possible to add additional nodes and issues commands across all of them. If nodes happen to disappear, for example, because of a crash, the containers which were running on those hosts will be automatically rescheduled onto other available nodes. The rescheduling ensures you do not lose capacity and provides high-availability.

On each additional node, you wish to add to the cluster, use the Docker CLI to join the existing group. Joining is done by pointing the other host to a current manager of the cluster. In this case, the first host.



Step 2 - Join Cluster❷

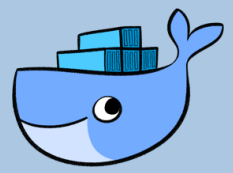
Docker now uses an additional port, *2377*, for managing the Swarm. The port should be blocked from public access and only accessed by trusted users and nodes. We recommend using VPNs or private networks to secure access.



Step 2 - Join Cluster❸

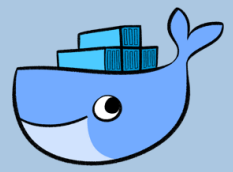
Task

- The first ask is to obtain the token required to add a worker to the cluster. For demonstration purposes, we'll ask the manager what the token is via *swarm join-token*. In production, this token should be stored securely and only accessible by trusted individuals.
  - *token=$(docker -H 172.17.0.26:2345 swarm join-token -q worker) && echo $token*
- On the second host, join the cluster by requesting access via the manager. The token is provided as an additional parameter.
  - *docker swarm join 172.17.0.26:2377 --token $token*
  - *docker node ls*



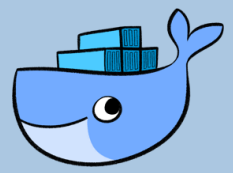
Step 3 - Create Overlay Network❶

- Swarm Mode also introduces an improved networking model. In previous versions, Docker required the use of an external key-value store, such as Consul, to ensure consistency across the network. The need for consensus and KV has now been incorporated internally into Docker and no longer depends on external services.
- The improved networking approach follows the same syntax as previously. The *overlay* network is used to enable containers on different hosts to communicate. Under the covers, this is a Virtual Extensible LAN (VXLAN), designed for large scale cloud based deployments.



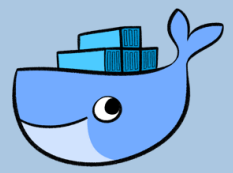
Step 3 - Create Overlay Network❷

- The following command will create a new overlay network called skynet. All containers registered to this network can communicate with each other, regardless of which node they are deployed onto.
  - *docker network create -d overlay skynet*



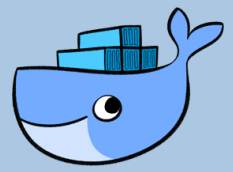
Step 4 - Deploy Service❶

- By default, Docker uses a spread replication model for deciding which containers should run on which hosts. The spread approach ensures that containers are deployed across the cluster evenly. This one of the nodes are removed from the cluster, the containers running are spread across the other nodes available.
- A new concept of Services is used to run containers across the cluster. This is a higher-level concept than containers. A service allows you to define how applications should be deployed at scale. By updating the service, Docker updates the container required in a managed way.



Step 4 - Deploy Service❷

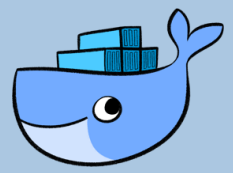
- In this case, we are deploying the Docker Image *katacoda/docker-http-server*. We are defining a friendly name of a service called http and that it should be attached to the newly created Skynet network.
- For ensuring replication and availability, we are running two instances, of replicas, of the container across our cluster.
- Finally, we load balance these two containers together on port 80. Sending an HTTP request to any of the nodes in the cluster will process the request by one of the containers within the cluster. The node which accepted the request might not be the node where the container responses. Instead, Docker load-balances requests across all available containers.



Step 4 - Deploy Service❸

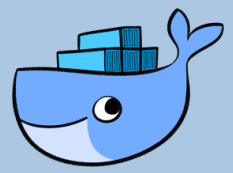
  - *docker service create --name http --network skynet --replicas 2 -p 80:80 katacoda/docker-http-server*

- You can view the services running on the cluster using the CLI command
  - *docker service ls*
- As containers are started you will see them using the ps command. You should see one instance of the container on each host.
  - List containers on the first host -
    - *docker ps*
  - List containers on the second host
    - *docker ps*
- If we issue an HTTP request to the public port, it will be processed by the two containers
  - curl docker



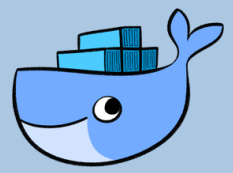
Step 5 - Inspect State

- The Service concept allows you to inspect the health and state of your cluster and the running applications.
  - docker service ps http
  - docker service inspect --pretty http
  - docker node ps self
  - 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 containers are running, it can easily start, or remove, containers as required. At the moment the scaling is manual. However, the API could be hooked up to an external system such as a metrics dashboard.
  - docker service scale http=5
  - docker ps
  - curl docker



Reference

- https://www.katacoda.com/courses/docker/getting-started-with-swarm-mode