Container Orchestration

Container orchestration automates the deployment, management, scaling, and networking of containers.

It is a solution that consists of a set of tools and scripts that can help host containers in a production environment.

Typically a container orchestration solution consists of multiple docker hosts that can host containers so that even if one fails, the application is still accessible to others.

There are multiple container orchestration solutions available.
Docker Swarm
Kubernetes
Apache Mesos

Docker Swarm is easy to set up and get started. It lacks some of the advanced auto scaling features.

Mesos is quite difficult to setup and get started but it supports many advanced features.

Kubernetes is the most popular of all. It is a bit difficult to setup and get started but provides a lot of options to customize departments and has support for many vendors.
Supported on all public cloud service providers(GCP, Azure, AWS).
The Kubernetes project is one of the top ranked projects on github.

Docker Swarm
Combines multiple Docker host machines together into a single cluster.
Docker swarm will take care of distributing services into separate hosts for

high availability and for load balancing across different systems and hardware.

E.g If we want to run 10 instances of nginx. And we would go to the master and tell we want to run 10 instances of nginx. Let's say for some reason we only have 3 worker nodes, then it does it's best to be able allocate each one of the worker nodes to run an X amount of that application.

Swarm was initially a separate product layered on Docker. But since Docker 1.12 it became part of the engine.

Components of Swarm cluster

- Consists of one or multiple Docker nodes
- Nodes are either a manager or a worker. Swarm cluster consists of at least one manager node

Manager node

- Is responsible for receiving instructions from a user or via an API and getting the worker to execute the given tasks.
- It can also run workloads on them by default, but we can optionally configure manager node to perform management tasks alone and have the workloads run exclusively on the worker nodes.
- It is responsible for maintaining the desired state of the Swarm cluster and & takes necessary action if a node fails or a new node is added to the cluster.
- A user provides instruction to the manager node to deploy an application by submitting a service definition.
- The manager then creates tasks and assigns them to the worker nodes, the worker nodes on receiving the tasks, deploys the necessary containers.
- Dispatches tasks to the worker.

Worker node

- Receives instructions or tasks from the manager nodes and runs containers.
- Accepts and executes the tasks.

Features

Cluster management integrated with Docker Engine
Docker CLI allows you to create a swarm of Docker Engines where you can deploy your application. No additional software is needed.

Decentralized design
Docker Engine handles specialization at runtime, allowing you to deploy an entire swarm from a single disk image.

Declarative service
Allows you to define the desired state of various services in your application stack.

Scaling
You can scale up or down and the swarm manager automatically adapts by adding or removing tasks to maintain the desired state.

Desired state reconciliation
The swarm manager constantly monitors the cluster state. It will recognize any differences between the actual state and the desired state and act to reconcile the two.

Multi-host networking
Swarm manager will automatically assign addresses to the containers on an overlay network when the container is initialized or updated.

Service Discovery
Each service in the swarm is assigned a unique DNS name

Load Balancing
Ports for services can be exposed to an external load balancer and internally the swarm lets you configure how to distribute service containers between your worker nodes.

Secure by default
Each worker node in the swarm enforces TLS mutual authentication and encryption between itself and the nodes.

Rolling updates
Swarm lets you control the delay between service deployments to different sets of nodes. If something goes wrong, you can roll back to a previous state.

Details can be learned from Docker official documentation.

Getting started with swarm mode

Setting up 3-node Swarm cluster
Steps

1. Setup 3 virtual machines.
2. Install Docker engine.
3. Configure ip address on the manager node.

Open protocols and ports between the hosts
The following ports must be available. On some systems, these ports are open by default.

- TCP port 2377 for cluster management communications
- TCP and UDP port 7946 for communication among nodes
- UDP port 4789 for overlay network traffic

Create a swarm

To check if the swarm mode is enabled
docker system info | grep Swarm

Initializing the Swarm cluster.
Step1: On the manager node run
docker swarm init
Or
docker swarm init --advertise-addr [Manager_IP]

docker system info | grep Swarm
Now the Swarm status should be active.

The output of the docker swarm init command also gives the join-token for worker node. The command it produces has the address of the master node, so for this reason it is important to have static ip address for the nodes.

Add the worker node to the cluster.
docker swarm join-token [Token] [Manager_IP]:2377

List the nodes in the swarm cluster
docker node ls

Managing the Swarm nodes

Listing nodes
docker node ls
Inspecting the node
docker node inspect [Node_Name]
Or
docker node inspect manager1 -- pretty

Promoting the worker node to a manager
docker node promote [Node_Name]

Demoting manager to a worker
docker node demote [Node_Name]

Removing a node from the Swarm
docker node rm -f [Node_Name]

Have a node leave the swarm, run following on the node
docker swarm leave

Getting the join-token
docker swarm join-token [worker|master]

Have a node rejoin the Swarm
docker swarm join-token [Token] [Manager_IP]:2377

docker node ls

Note:

* On the ID is to represent where the command is being run.
Availability: This column indicates the status of a node for scheduling the processes.
Status can be -Active, -Pause, -Drain

Active - Scheduler can assign tasks to the node.
Pause - Scheduler doesn't assign new tasks to the node but existing tasks remain running
Drain - Scheduler doesn't assign new tasks to the node. The scheduler shutdown any existing tasks

docker node update -- availability drain worker01
docker node ls

To activate the worker node again
docker node update -- availability active worker01

To delete a node, we first drain the node and remove.
docker node update -- availability drain worker01
docker node ls
docker node rm worker01

On the worker01 node
docker swarm leave

Manager status column
Leader - manager node
No value/ blank - worker node
Leader indicates the node is the primary manager node that makes all Swarm management & orchestration decisions for the Swarm cluster.

Leader status -Leader, -Reachable, -Unavailable
Reachable: The node is the manager node but is not a leader node
Unavailable: The node is the manager node but can't communicate with other managers

More details on how nodes work can be learned in the official documentation.

Working with Services

How do services work?

Creating a service
docker service create -d -- name [NAME] -p [host port: container port] -- replicas [replicas] [image] [cmd]

docker service create -d -- name nginx_serivce -p 8080:80 -- replicas 2 nginx:latest

List services
docker service ls

Inspecting a service
docker service inspect [name]

docker network ls -- no-trunc [id of the network in inspect]

Getting logs for a service
docker service logs [name]

Listing all tasks of a service
docker service ps [name]

Scaling a service up and down
docker service scale [name]=[replicas]

docker service scale nginx_service=3

Updating a service
docker service update [option] [name]
-h for help