Docker Swarm

Submitted by:

1. **Dileep Kumar (ERP: 18255)**
2. **Muhammad Arsalan Mubeen (ERP: 23394)**

Docker Swarm

Contents:

Docker Node:

A Docker Node is either a physical or a virtual machine that run the Docker application called Docker Engine.

Docker Swarm:

A Docker Swarm is a group of Docker Nodes that have been configured to join together in a cluster.

We do not need to manage each of the machine individually, as if they are separate machines. We could just teat these as one large machine.

If we need more resources we can scale our the swarm by adding more nodes to it.

Docker SwarmKit:

Docker SwarmKit is a embedded component of Docker Engine.

It provides docker swarm mode. That abstracts the nodes just like a single giant machine.

Docker swarm takes care of everything for us so we do not need worry figure out where to run the software.

We just need to know what to run. While Docker will take cares of how and whare to run the application.

Types of Docker Swarm:

There are two types of Docker Swarm:

1. **Single-node Docker Swarm:**

   When we have both Manager(s) and Worker(s) on same single Docker Node.

2. **Multi-Node Docker Swarm:**

   When we have Manager(s) and Worker(s) on distributed over multiple Docker Nodes.

Networking in Docker Swarm:

There are two types of networking used to create Docker Swarm:

1. **Bridge Networking:**
   - Bridge is virtual switch inside linux operating system.
   - Bridge networking is used to connect two or more containers in a single node docker swarm.

2. **Overlay Networking:**
   - Overlay network are virtual extensible LAN (VxLAN) in linux operating system.
   - Overlay networking is used to connect two or more containers in a multi-node docker swarm.

Features of Docker Swarm

Some of the most essential features of Docker Swarm are:

- **High Service Availability:**

  One of the main benefits of docker swarms is increasing application availability through redundancy. High availability can be ensured by deploying Docker Swarm in a multi-node environment with multiple Manager nodes.

- **Decentralized access:**
  Swarm makes it very easy for teams to access and manage the environment

- **High security:**
  Any communication between the manager and client nodes within the Swarm is highly secure

- **Auto load-balancing:**
  There is autoload balancing within our environment, and we can script that into how we want and structure the Swarm environment.

- **High scalability:**
  Load balancing converts the Swarm environment into a highly scalable infrastructure

- **Roll-back a task:**
  Swarm allow us to roll back environments to previous safe environments

- **Fault tolerance:**
  If a machine goes down, docker swarm migrates all its tasks to others machines.

Docker Swarm Architecture:

Docker Swarm is using is Master / Slave architecture with one or more Managers (masters) and multiple workers (slaves).

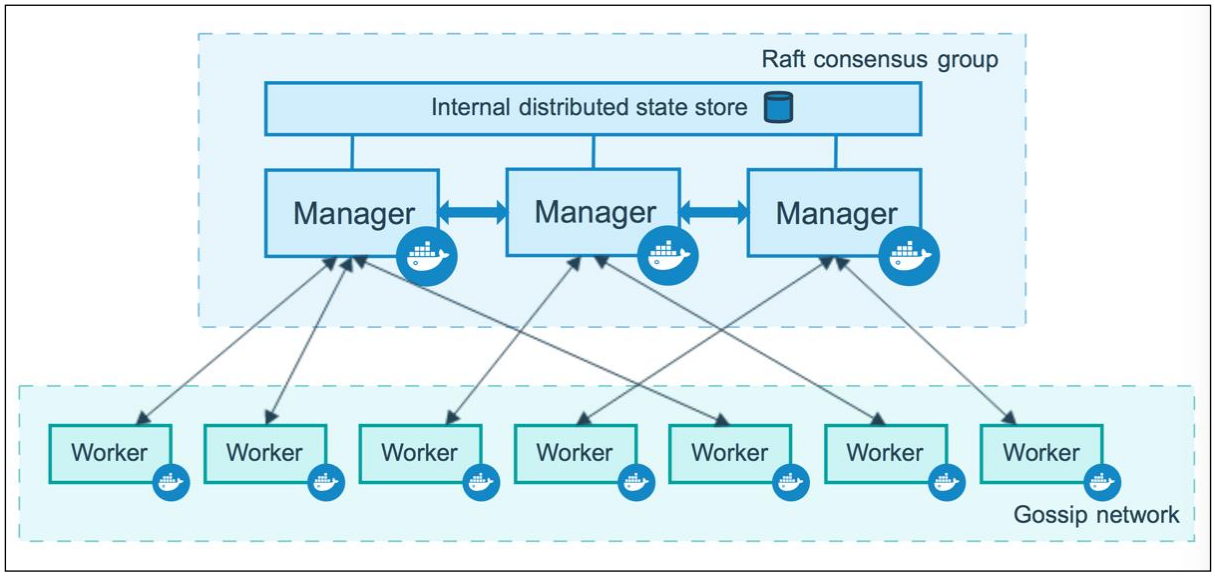

Fig. 1. Docker Swam Typical Architecture

Types of Docker Nodes in Docker Swarm:

There are two types of Docker Nodes in a Docker Swarm:

1. Manager node
2. Worker node

1. Manager Node:

Manager node handles all cluster management tasks, such as:

- maintain a consistent internal state of the entire swarm and all the services running on it using RAFT consensus algorithm.
- allocates tasks to worker nodes
- scheduling services
- serving swarm mode HTTP API endpoints

High-Availability of Manager Nodes:

- A single Docker Swarm can have multiple Manager Nodes for maintaining high-availability.
- Docker recommends a minimum one, and a maximum of seven manager nodes for a swarm.
- **Important Note**: Adding more managers will decrease the performance.
- **Leader Node:** Only one Manager will be a Leader Node.

- A Leader Node is elected among Manager Nodes using Raft consensus algorithm.
- If the leader node becomes unavailable due to an outage or failure, a new leader node can be elected using the Raft consensus algorithm.

2. Worker Node:

Worker nodes are instances of Docker Engine, whose sole purpose is:

- to execute containers
- By default, all managers are also workers in single-node swarm
- multi-node swarm
- Worker nodes accept tasks sent from manager nodes
- Every worker node has an agent which reports the state of node's tasks to the manager
-

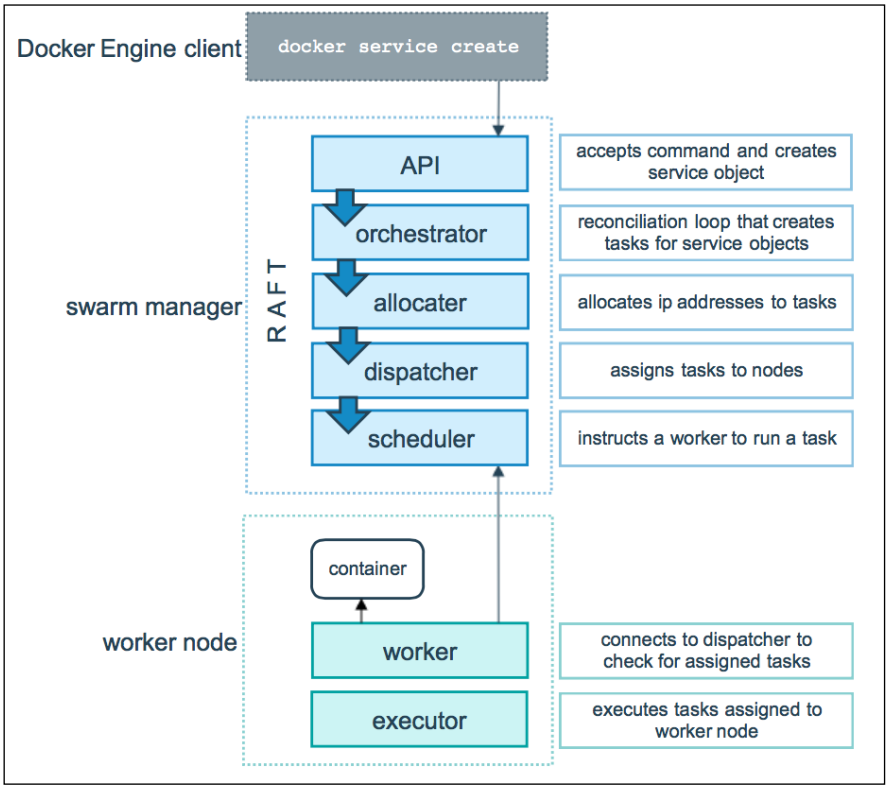

Fig. No. 2. Manager and Worker Nodes in a Docker Swarm

Services in Docker Swarm

- Service is a definition of an application with its multiple instances (containers), deployed over a swarm to scale the application.
- Before we deploy a service in docker swarm, we must have at least one node deployed
- A service is the definition of the tasks to execute on the manager or worker nodes.
- When we create a service, we specify which container image to use and which commands to execute inside running containers.
- In the replicated services model, the swarm manager distributes a specific number of replica tasks among the nodes based upon the scale we set in the desired state.
- For global services, the swarm runs one task for the service on every available node in the cluster.

Types of services in Docker Swarm:

Docker Swarm mode has two types of service deployments:

Replicated services:

For Replicated Services, we specify a number of replica (identical) tasks for the swarm manager to assign to available nodes to run.

Global services:

Global services runs single task on every node.

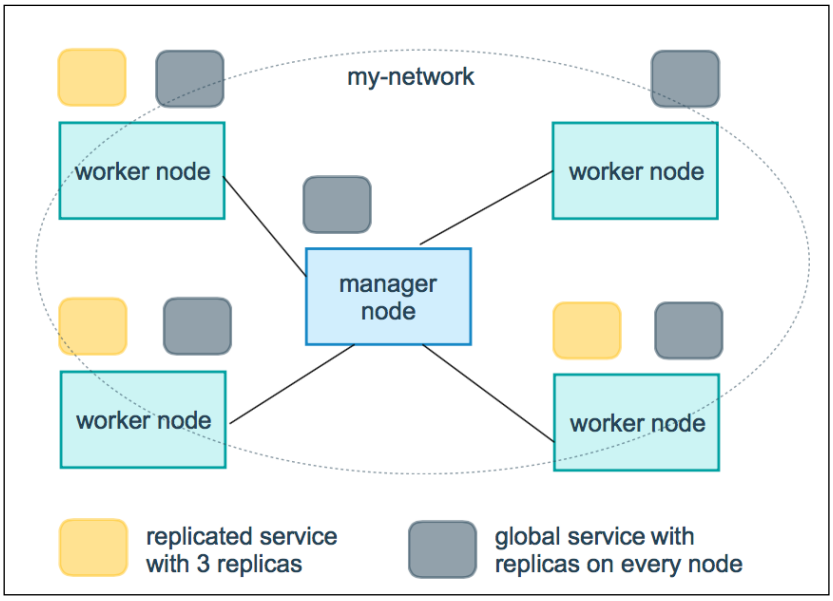

Fig. No. 3. Services in a Docker Swarm

Task in Docker Swarm

- Task is a declaration that describes the containers that we want to run under a service.
- Task is specification of all resources required to run a container in a service.
  - A tasks is created by defining **Replicas:** (creates one task)
  - specification is defined in **TaskTemplate**
  - A single task corresponds to a single container, in a service.
- To deploy our application to a swarm, we submit a service definition to a manager node. The manager node dispatches units of work called tasks to worker nodes.
- Worker nodes receive and execute tasks dispatched from manager nodes.
- By default manager nodes also run tasks as worker nodes.
- But we can configure them to run manager tasks exclusively.
- The worker node notifies the manager node of the current state of its assigned tasks so that the manager can maintain the desired state of each worker.