Docker Swarm

What is Docker Swarm?

Docker Swarm is a container orchestrator — a tool used to manage and coordinate multiple Docker containers running across multiple servers (nodes).

It provides:

- High Availability (HA)
- Fault Tolerance (FT)
- Load Balancing
- Scalability

Swarm allows you to run containers in a cluster of multiple Docker nodes.

What is a Container Orchestrator?

A container orchestrator manages the deployment, scaling, networking, and health of containers automatically.

How does it solve problems?

- Ensures containers are always in the desired state
- Automatically restarts failed containers
- Distributes containers across multiple nodes
- Enables rolling updates and service scaling

Examples of Container Orchestrators:

- Docker Swarm
- Kubernetes
- Apache Mesosphere (DC/OS)

How Orchestrators Work

- A private overlay network for communication between containers

Private IP addresses for each container

Controllers that maintain the desired vs. actual state

When containers crash or nodes go down, orchestrator automatically reschedules containers on healthy nodes.

⚙ Docker Swarm Architecture:

1,. Manager node

2. Worker node

1. Manager node:

Controls and manages the swarm cluster. Executes all management commands.

2. Worker Node

Follows manager instructions. Runs containers (services).

▧ In production, you usually have 3 manager nodes [redacted] and multiple worker nodes.

Containers can run on both manager and worker nodes, but best practice is to avoid running them on managers.

Hands-On Setup

1. Create AWS Instances

4 EC2 instances (Ubuntu 22)

2 Managers, 2 Workers

Open ports:

2377 (Swarm Management)

22 (SSH)

80, 443 (Web access)

All traffic for internal communication

2. Install Docker on All 4 Nodes

```
sudo apt update
sudo apt install docker.io -y
sudo systemctl enable docker
sudo systemctl start docker
systemctl status docker
```

Check swarm status:

```
docker info | grep -i swarm
```

3. Rename Nodes (optional)

```
sudo hostnamectl set-hostname master
exec bash
```

Create and Join Swarm Cluster

Step 1: Initialize Swarm (on Manager1)

```
docker swarm init
```

Check:

```
docker info | grep -i swarm
docker node ls
docker network ls
```

Step 2: Add Second Manager:

```
docker swarm join-token manager
```

Copy the token and run on Manager2.

Step 3: Add Workers

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

Copy the command and run on Worker1 and Worker2.

Check nodes:

```
docker node ls
```

Leader/Reachable → Manager

Blank status → Worker

Network Validation:

Run on all nodes:

```
docker network ls
```

nodes share the same Swarm overlay network.

Swarm provides virtual private networking between all containers in the cluster.

Deploying Services (HA + FT Demonstration)

Create a Service with 3 Replicas

```
docker service create --name first --replicas=3 -p 31000:80 nginx

docker service ls

docker service ps first
```

Check container placement:

```
docker ps -a
```

Now open browser with:

```
http://<Public-IP>:31000
```

App is available from any node (load-balanced through Swarm).

High Availability & Fault Tolerance

Stop any node where a container is running:

Wait a few minutes and observe

```
docker node ls
docker service ps first
```

Swarm automatically reschedules containers on healthy nodes.

Restart the stopped node and check again.

📈 Scaling Services

Scale Up

```
docker service scale first=5
docker service ps first
```

Scale Down

```
docker service scale first=1
```

Remove Service

```
docker service rm first
```

Rolling Updates (Upgrade)

```
docker service create --name myweb -p 32000:80 nginx:1.17
docker service update myweb --image nginx:1.18
```

New container is created, old one removed.

▯ Rollback (Downgrade)

Rollback to the previous version only:

```
docker service rollback myweb
```

Swarm does not support rollback to older-than-last versions.

⚡ Limitations of Docker Swarm:

Auto Scaling	------------------	Manual only
Rollbacks	------------------	Only last version
Community	------------------	Limited

👉 That's why Kubernetes is preferred in production.

Kubernetes

Auto Scaling	------------------	▯ Auto-scaling supported
Rollbacks	------------------	Any version
Community	------------------	Very active

▯ Node Availability Control

Set Manager nodes to Drain state (to avoid running containers):

```
docker node update --availability=drain manager1
docker node update --availability=drain manager2
docker node ls
```

Deploy:

```
docker service create --name testcontainer --replicas=10 nginx

docker service ps testcontainer
```

Observe that containers run only on worker nodes.

Remove Node from Cluster

On Worker2:

```
docker swarm leave

docker info | grep swarm   # Swarm: inactive
```

On Manager:

```
docker node rm worker2
```

To rejoin:

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

Promote or Demote Nodes

Promote Worker to Manager

```
docker node promote worker1
```

Demote Manager to Worker

```
docker node demote manager2
```

Deploying Microservices (WordPress Example)

Swarm uses Stacks to deploy multi-container apps:

```
docker stack deploy -c docker-compose.yml mystack

docker stack services mystack

docker stack ps mystack
```

You can deploy applications like:

- WordPress + MySQL
- Flask + Redis
- Node.js + MongoDB

Conclusion:

Swarm Init	=	Create cluster
Swarm Join	=	Add nodes
Service Create	=	Run containers in cluster
Service Scale	=	Increase/decrease replicas
Service Update	=	Rolling update
Service Rollback	=	Downgrade to previous version
Drain Mode	=	Prevent manager from hosting containers
Stack Deploy	=	Deploy multi-container apps
