**Docker Swarm & Raft Algorithm - Comprehensive Guide**

**Table of Contents**

1. **Introduction to Docker Swarm**
   * What is Docker Swarm?
   * Why Use Docker Swarm?
   * Real-World Use Case
2. **Key Concepts**
   * Master Nodes (Managers)
   * Worker Nodes
   * Raft Consensus Algorithm
3. **Practical Implementation**
   * Setting Up a Swarm Cluster
   * Deploying Services
   * Scaling & Load Balancing
   * Node Management
4. **Docker Swarm vs. Kubernetes**
   * Advantages & Disadvantages
5. **Conclusion & Best Practices**

**1. Introduction to Docker Swarm**

**What is Docker Swarm?**

* Docker Swarm is a **container orchestration tool** that manages multiple Docker hosts as a **single virtual system**.
* It provides **high availability, load balancing, and scaling** for containers.

**Why Use Docker Swarm?**

✅ **Simple to set up** (compared to Kubernetes).  
✅ **Built into Docker** (no additional installation).  
✅ **Good for small-scale deployments** (testing, sandbox environments).

**Real-World Use Case**

* **Scenario:** A company runs **multiple containers** across **multiple Docker hosts** for testing.
* **Problem:** Managing containers manually is inefficient.
* **Solution:** Use **Docker Swarm** to automate deployment, scaling, and failover.

**2. Key Concepts**

**A. Master Nodes (Managers)**

* **Responsible for:**
  + Scheduling tasks (containers).
  + Managing the cluster state.
  + Handling worker nodes.
* **Must be an ODD number** (1, 3, 5…) due to **Raft Consensus Algorithm**.

**B. Worker Nodes**

* **Execute tasks** assigned by master nodes.
* **Cannot manage the cluster**.

**C. Raft Consensus Algorithm**

* Ensures **all manager nodes agree** on the cluster state.
* **Tolerates**(N-1)/2**failures** (where N = number of managers).
  + **Example:**
    - **3 Managers** → Can tolerate **1 failure**.
    - **5 Managers** → Can tolerate **2 failures**.
* **Why odd numbers?**
  + Avoids **split-brain scenarios** (where two managers disagree).

**3. Practical Implementation**

**Step 1: Setting Up a Swarm Cluster**

1. **Launch 6 EC2 Instances** (3 Masters, 3 Workers).
2. **Initialize Swarm on Master 1:**

bash

docker swarm init

* + Output gives **join tokens** for workers & other managers.

1. **Join Other Managers:**

bash

docker swarm join --token <MANAGER\_TOKEN> <MASTER\_IP>:2377

1. **Join Workers:**

bash

docker swarm join --token <WORKER\_TOKEN> <MASTER\_IP>:2377

**Step 2: Deploying Services**

* **Run a Service with Replicas:**

bash

docker service create --name app1 --replicas 6 -p 8000:80 k236193/rolling-update:v1

* **Check Service Status:**

bash

docker service ls

docker service ps app1

**Step 3: Scaling & Load Balancing**

* **Scale Up/Down:**

bash

docker service scale app1=3 *# Reduce to 3 replicas*

docker service scale app1=12 *# Increase to 12 replicas*

* **Routing Mesh:**
  + Access service via **any node’s IP** (even if container isn’t running there).

**Step 4: Node Management**

* **Drain a Node (Maintenance Mode):**

bash

docker node update --availability drain <NODE\_IP>

* **Reactivate a Node:**

bash

docker node update --availability active <NODE\_IP>

* **Deploy Only on Worker Nodes:**

bash

docker service create --name app2 --constraint node.role==worker --replicas 6 -p 8000:80 k236193/rolling-update:v1

* **Global Service (Deploys on Every Node):**

bash

docker service create --name monitor --mode global -p 9100:9100 prom/node-exporter:latest

**4. Docker Swarm vs. Kubernetes**

| **Feature** | **Docker Swarm** | **Kubernetes** |
| --- | --- | --- |
| **Complexity** | Simple | Complex |
| **Setup Time** | Minutes | Hours |
| **RBAC** | ❌ No built-in role-based access control | ✅ Supports RBAC |
| **Namespaces** | ❌ No namespaces | ✅ Supports namespaces |
| **Auto-Healing** | ✅ Basic | ✅ Advanced |
| **Use Case** | Small-scale, testing | Large-scale, production |

**Disadvantages of Docker Swarm**

❌ No **TLS Secrets** (secure credential storage).  
❌ No **RBAC** (fine-grained access control).  
❌ No **Readiness/Liveness Probes** (auto-recovery checks).

**5. Conclusion & Best Practices**

**Key Takeaways**

✔ **Use Swarm for simplicity** (testing, small teams).  
✔ **Always have an ODD number of managers** (3 recommended).  
✔ **Leverage**--constraint to control where services run.  
✔ **Monitor with tools like**docker visualizer.

**Next Steps**

* **Automate Swarm with Terraform**.
* **Explore Kubernetes** for larger deployments.