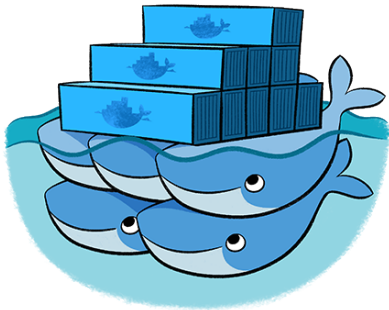


# How Nodes work



## What are Roles?

A **swarm** is a group of Docker hosts (servers running Docker) that are connected and work together to run containerized applications. Each host can play one of two roles:

1. **Manager:** A node that controls the swarm. It handles the cluster management tasks, such as assigning workloads (tasks) to worker nodes and maintaining the desired state of the services.
2. **Worker:** A node that does the actual work by running containers. The worker nodes execute the tasks assigned by the manager.

Any Docker host in the swarm can be a **manager**, a **worker**, or even perform both roles.

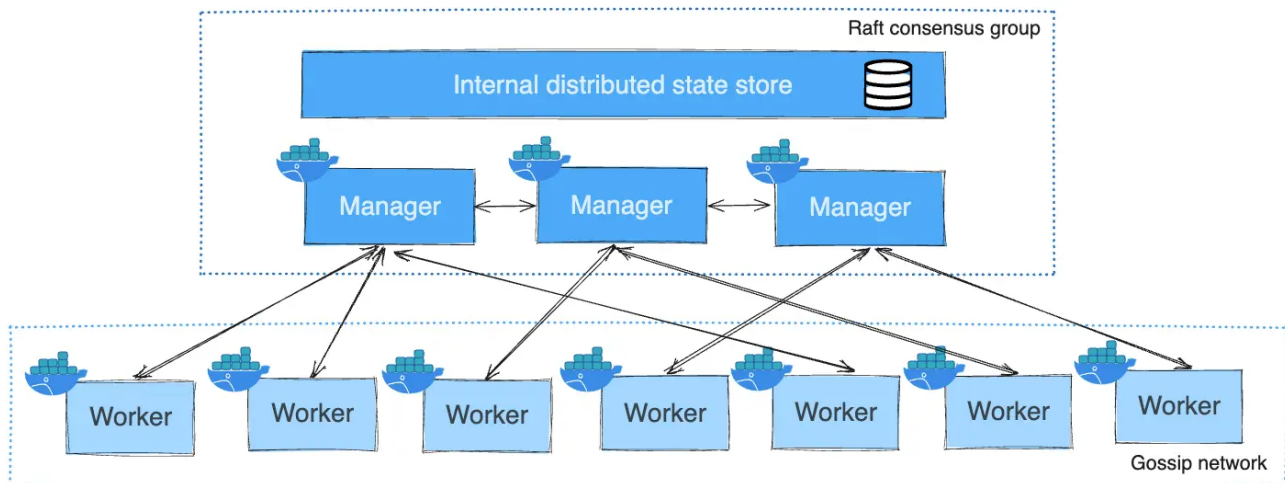
### “ Example: Creating a 3-Node Docker Swarm with All Manager Nodes

Let's walk through an example where we set up a **Docker Swarm** with three nodes, all acting as **manager nodes**. This scenario is useful when you want high availability and fault tolerance in your cluster, meaning if one or two manager nodes fail, the remaining nodes can continue managing the swarm.

### Why Make All Nodes Managers?

In a Docker Swarm, **manager nodes** are responsible for handling the cluster's state, scheduling tasks, and distributing containers to the **worker nodes**. By making all three nodes **managers**, you ensure that your swarm can tolerate failures of one or even two nodes and still function. This is known as **high availability** because the swarm can elect a new leader and continue operating without downtime.

If you haven't already, read through the [Swarm mode overview](#) and [key concepts](#).



## Manager Nodes

**Manager nodes** play a crucial role in maintaining and orchestrating the state of the swarm. They are responsible for:

- **Maintaining Cluster State:** Manager nodes keep track of the state of the swarm and all services running on it, ensuring that the desired state (e.g., the number of replicas of a service) is met.
- **Scheduling Services:** Managers are in charge of scheduling tasks (containers) across the worker nodes in the swarm.
- **Serving Swarm Mode HTTP API Endpoints:** Manager nodes expose the Swarm mode HTTP API, which is used to control the swarm via commands or automation.

## Fault Tolerance and High Availability

Docker Swarm uses a **Raft consensus algorithm** to ensure that the state of the swarm is consistent across all manager nodes. This is particularly important for high availability. The general rule is that an **odd number** of managers provides fault tolerance, allowing the swarm to continue functioning if some managers fail.

- A **three-manager swarm** can tolerate the loss of one manager node.
- A **five-manager swarm** can tolerate the loss of two manager nodes.

The formula for fault tolerance is that a swarm can tolerate the loss of at most  **$(N-1)/2$**  manager nodes, where **N** is the total number of managers.

## Best Practices for Manager Nodes

- **Odd Number of Managers:** To take full advantage of Docker's fault tolerance, always use an odd number of manager nodes. This ensures that the swarm can maintain quorum (i.e., the minimum number of nodes needed to keep the swarm functional).
- **Limit the Number of Managers:** Although adding more manager nodes increases fault tolerance, it does **not** improve performance or scalability. In fact, having too many manager nodes can slow down decision-making processes. Docker recommends a maximum of seven managers in a swarm.

## Manager Node Failure

If you're running a single-manager swarm and the manager node fails, the services on the worker nodes will continue to run, but you won't be able to control or update the swarm. You'd need to recreate the swarm to recover full control.

In contrast, when running a swarm with multiple managers, if one manager fails, the remaining managers can take over, ensuring that the swarm continues to operate without downtime.

---

## Worker Nodes

**Worker nodes** are simpler compared to manager nodes. Their primary purpose is to **execute containers**. Worker nodes don't participate in swarm management decisions and don't maintain the state of the swarm.

- **Executing Tasks:** Workers run the containers assigned to them by the manager nodes. They receive tasks (containers to run) and report back to the managers on the status of these tasks.

- **No Raft Participation:** Worker nodes do not store the cluster's state and don't participate in the Raft consensus. This allows them to focus purely on running workloads.

## Worker Node Setup

In any Docker Swarm, there must be at least one manager node, but you can have any number of worker nodes. By default, all manager nodes also act as workers, meaning they can schedule tasks and run containers.

However, if you want to prevent managers from running containers (e.g., to dedicate them solely to management tasks), you can adjust their availability. This brings us to the concept of **Drain Mode**.

---

## Manager Node Availability and Drain Mode

In a multi-node swarm, you may want to **prevent managers** from running any tasks or containers. For example, you might want to ensure that managers are purely dedicated to orchestration and scheduling tasks, leaving the heavy lifting of running containers to worker nodes.

You can change the availability of a manager node to **Drain** mode, which means the scheduler will not assign new tasks to that node, and existing tasks will be moved to other nodes.

To set a manager node to Drain mode, you can run the following command:

```
docker node update --availability drain <manager-node-name>
```

This will ensure that the manager node doesn't run any new tasks but continues its role as a swarm manager, making scheduling decisions and maintaining the cluster's state.

---

## Changing Roles: Promoting and Demoting Nodes

Swarm mode also provides flexibility when it comes to changing the roles of your nodes. You can **promote** a worker node to become a manager or **demote** a manager node to a worker if needed.

## Promoting a Worker to a Manager

If you want to add more fault tolerance to your swarm or need to take a manager node offline for maintenance, you can promote a worker node to a manager. This is done with the following command:

```
docker node promote <worker-node-name>
```

This is useful in situations where you want to ensure that the cluster remains highly available, even if one of your manager nodes needs to be taken down.

## Demoting a Manager to a Worker

If you no longer need a manager node or want to reduce the number of managers for better performance, you can demote a manager back to a worker. This can be done using:

```
docker node demote <manager-node-name>
```

---

## Conclusion

Understanding the roles of **manager** and **worker nodes** in Docker Swarm is essential for creating a stable, highly available, and scalable cluster. Manager nodes handle critical tasks such as maintaining the cluster state and scheduling services, while worker nodes focus purely on running containers.

In larger clusters, having multiple manager nodes ensures that your swarm can tolerate failures without disrupting service. However, it's important to maintain an odd number of managers for fault tolerance and to avoid adding too many managers, as this can slow down the swarm's performance.

By effectively using **Drain mode** and **node promotion/demotion**, you can adjust your swarm's architecture to meet your organization's needs, ensuring optimal performance and availability.