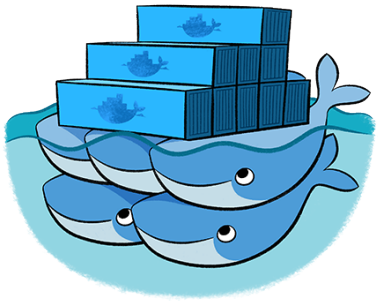
How Services work



When Docker Engine operates in **Swarm mode**, services become the fundamental units for deploying and managing applications. A service is essentially a set of instructions for running containerized applications across multiple Docker nodes. This might be part of a larger application, such as a microservice architecture, or it could be a stand-alone service like an HTTP server or a database.

What is a Service?

A **service** in Docker Swarm represents a task that you want to run, such as running a containerized application.
When you create a service, you specify a few essential options:

- The **container image** to use for the service.
- **Commands** to execute inside the running containers.
- The **port** to expose to make the service available outside the swarm.
- An **overlay network** to connect the service with other services within the swarm.
- **CPU and memory limits** for the resources allocated to each service instance.
- **Rolling update policies** to control how services are updated across the swarm.
- The number of **replicas** (i.e., how many copies of the service) you want to run.

Services, Tasks, and Containers

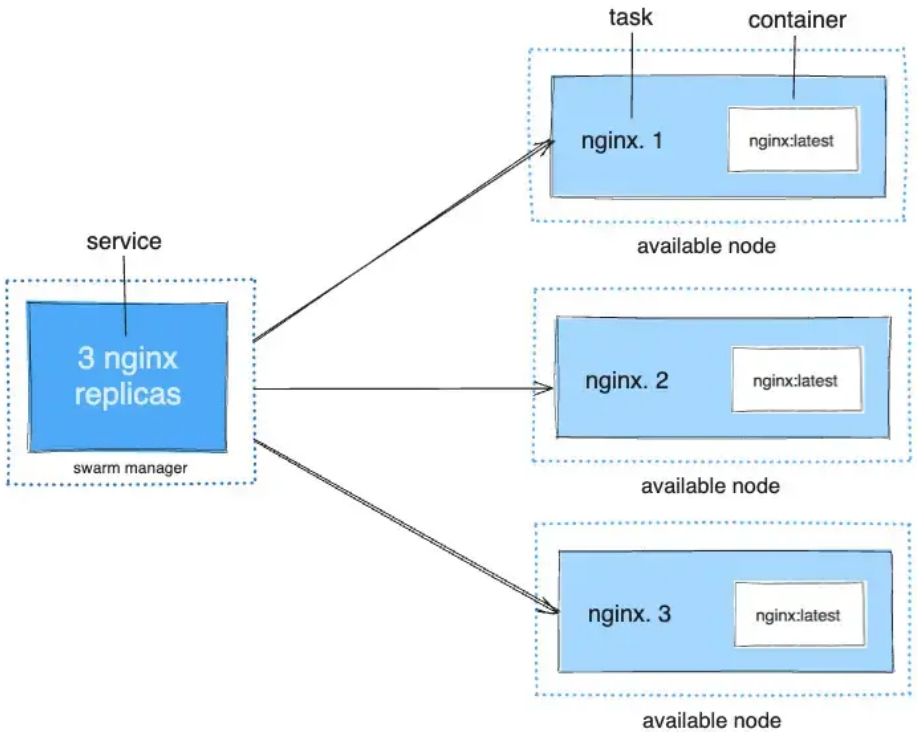
Once you define the service, the swarm **manager** takes your service definition and turns it into one or more **tasks**. A task is the smallest unit of work in a swarm and represents a single instance of a container running on a node.

For example, if you want to balance traffic between three instances of a web server, you might deploy a service with three replicas. Each replica is a task, and each task runs one container on a

different node.

- **Service**: Defines the application you want to run.
- **Task**: A unit of work managed by the swarm that runs a single container.
- **Container**: The actual application process that runs as part of a task on a node.

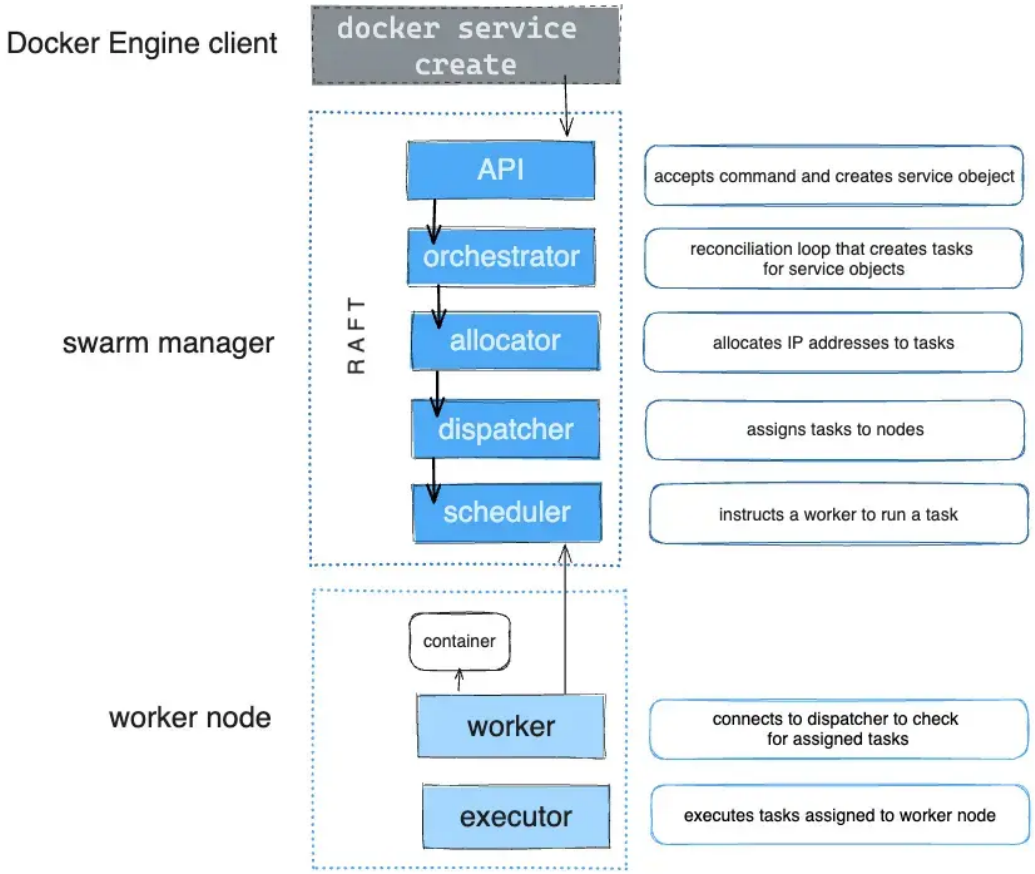
In short, when you deploy a service, you define the desired state, and Docker Swarm schedules that service across the available nodes.



Tasks and Scheduling

When you create or update a service, you declare a desired state (e.g., three instances of an HTTP service). The **orchestrator** (which is part of the swarm manager) ensures that this state is met by creating and scheduling tasks. For instance, if you define a service with three replicas, the swarm manager creates three tasks. Each task runs a container on a node.

If a container crashes or fails its health check, the orchestrator detects the failure and schedules a new task to replace it, ensuring that the desired state of the service is always maintained.



Pending Services

Sometimes, a service may remain in a **pending state**, meaning that it cannot be deployed yet. Here are a few scenarios where this might happen:

- **Node Availability**: If no nodes are available to run tasks (e.g., all nodes are in **paused** or **drained** mode), the service will remain in the pending state until nodes become available.
- **Resource Constraints**: If the service requires more memory or CPU than any node can provide (e.g., a service requiring 500GB of RAM but no node has that capacity), the service will stay pending.
- **Placement Constraints**: If the service is configured with specific placement constraints (e.g., to only run on certain nodes), it may stay pending until a suitable node is available.

> Tip:
> In these cases, it is often better to scale the service to zero replicas if your goal is to pause the service temporarily.
> If your only intention is to prevent a service from being deployed, scale the service to 0 instead of trying to configure it in such a way that it remains in `pending`.

Replicated and Global Services

There are two types of services in Docker Swarm: **replicated services** and **global services**.

Replicated Services

For replicated services, you specify the number of **identical tasks** you want to run. Each replica runs on a separate node, providing redundancy and load balancing. For example, if you create a service with three replicas, the swarm manager ensures that three identical instances (tasks) of the service are running on different nodes.

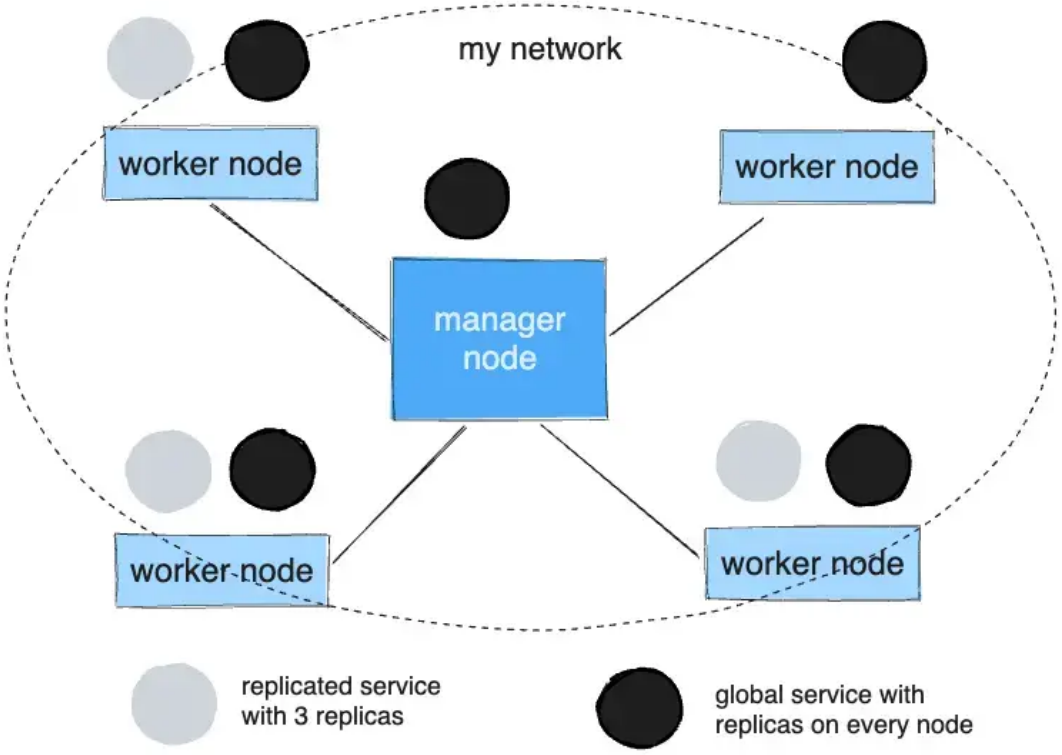
This is useful for applications like web servers, where you want multiple instances of the same service to handle requests simultaneously.

Global Services

A global service runs **one instance of the service on every node** in the swarm. You do not specify the number of replicas; instead, the swarm ensures that each node runs exactly one instance of the service.

Global services are ideal for tasks like monitoring or logging agents, where you want each node to run the same service. For example, you might use a global service to run an anti-virus scanner or a network monitoring agent on every node in your swarm.

> The diagram below shows a three-service replica in gray and a global service in black.



Revision #2
Created 11 September 2024 13:23:05 by aeoneros
Updated 11 September 2024 13:34:00 by aeoneros