Introduction to Docker Swarm

A little about DevOps with Docker

We'll cover the following

Till the last section, we were learning from the developer's perspective. This section will give you a little exposure to DevOps using Docker.

Setting our expectations

This course focuses on development using Docker. We have covered all the things required from a developer perspective. With this, you will be able to develop web apps using Docker easily.

However, you might want to understand a little bit about deployment to know what is going on in your project. So, this section will provide you with a high-level understanding of the swarm architecture to have a conceptual understanding of the production environment which is deployed using Docker swarm.

So, let's start.

What is container orchestration?

Suppose you have developed a wonderful application which will be used by millions of users. How will it be deployed efficiently without having a single point failure to handle a large number of requests?

Managing systems at large scale very difficult. Millions of containers run on different machines. How will you track which container has stopped or which service is no longer running?

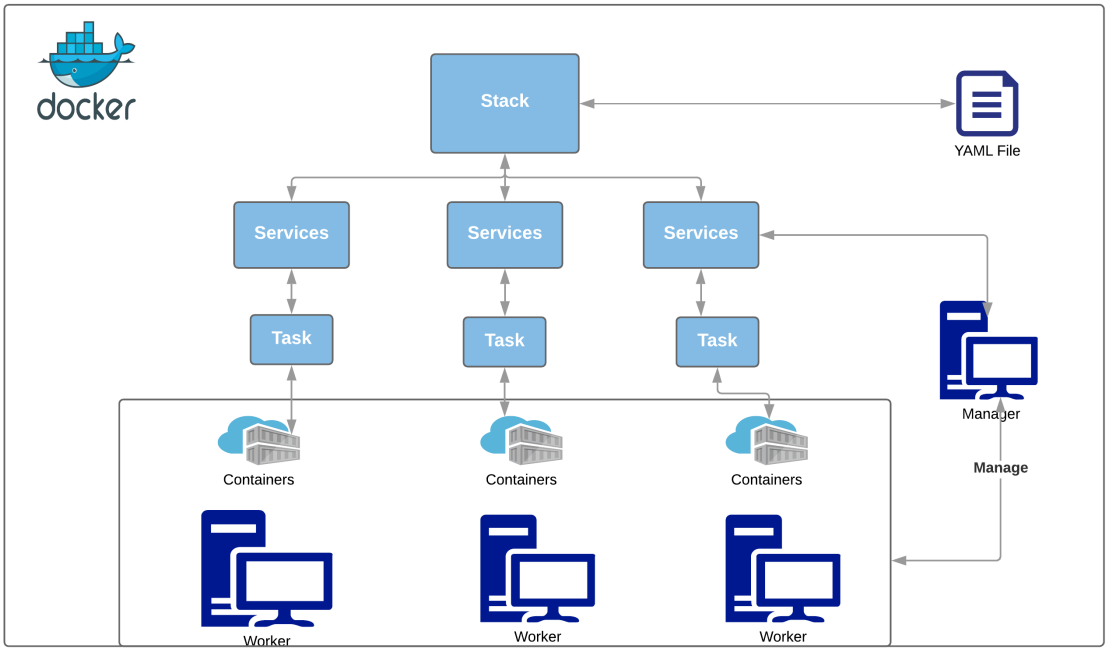
Managing this cluster of containers effectively is called container orchestration. At present, there are three major tools available that help in container orchestration namely, Docker swarm, Kubernetes, and Apache Mesos.

In the rest of the lessons, we will learn more about Docker swarm.

What is Docker swarm?

Docker swarm is a container orchestration tool that allows a user to manage multiple containers deployed across multiple host machines. Docker swarm has different components which helps it to manage the swarm effectively. Let's have a look at the architecture to understand how it works.

Docker swarm architecture



A worker node is something on which tasks are created by the manager. It will have services running on it to maintain the state defined in the YAML file.

In this architecture, there is only one YAML file. That's the power of the YAML file. A stack of services is created and deployed on each worker node using the YAML file.

> A node is a machine. It can act as a manager, worker, or both. We can also have multiple manager nodes but there will always be a primary node to

manage the swarm cluster.

Characteristics of the manager node include:

- It always knows the exact state of the cluster
- It is responsible for maintaining the state of the cluster described in the YAML file
- It creates the different tasks on worker nodes to maintain the desired service state.
- It keeps backups in case if a node fails and, starts new containers from backups on available nodes to maintain the count of containers

Let's understand each component of the architecture.

Building blocks of Docker swarm

A swarm cluster can have the following components:

- **Node:** The node is the host machine. A machine can act as a worker, manager, or both. We can have a swarm with one node as well. In that case, the respective machine acts as both a worker and manager nodes.

For example, In a cluster of 5 computers, each computer will be a node. And there will be a master computer which is responsible for the communication of the Cluster.

- **Stack:** A set of services combined is called a stack. Stack and Compose work somewhat similarly, except, there are some commands which will be ignored by docker-compose in a non-swarm mode such as `deploy`.

For example, when we want to deploy our whole project which is a collection of services (like web server, database and maybe a task queue) in a single command; we declare all the services in a YAML file and that YAML file is now a stack of services for the swarm manager. Swarm manager will deploy the stack and maintain the desired states of services defined in the YAML file.

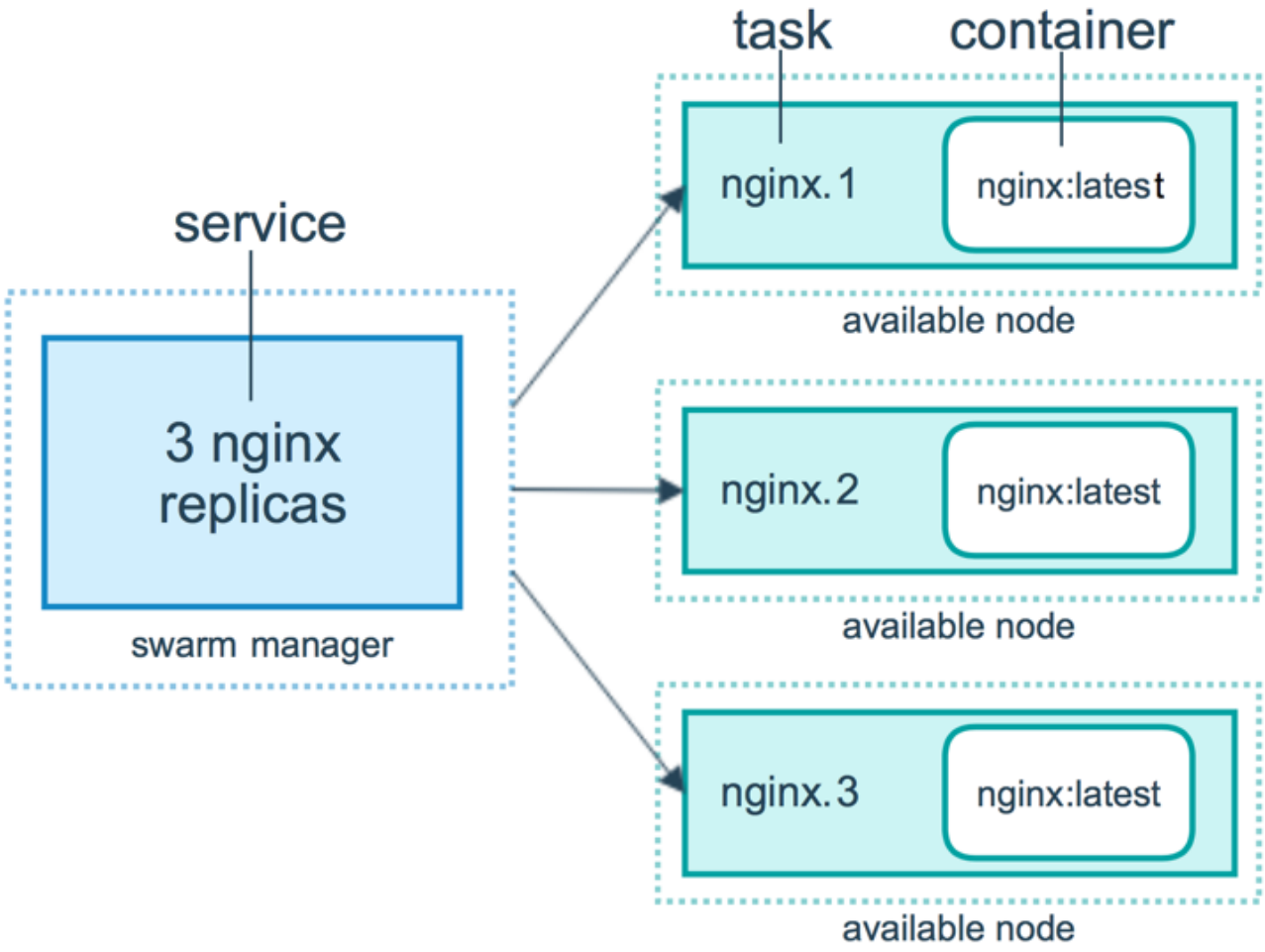
- **Service:** A service is the definition of the tasks to execute on the manager or worker nodes. It is the central structure of the swarm system and the primary root of user interaction with the swarm. When you create a service, you

specify which container image to use and which commands to execute inside running containers.

For example, The flask app we created earlier is a service. Why? Because we are using flask_app:2.0 docker image which will be running in a container with the command to run the WSGI server.

- **Task:** Task is responsible for maintaining the desired replica set of the service. To deploy a service on a worker node, swarm manager creates a task and starts a container in it.

For instance, you define a service that instructs the swarm manager to keep three instances of an HTTP listener running at all times. The orchestrator responds by creating three tasks. Each task is a slot that the scheduler fills by spawning a container. The container is the instantiation of the task. If an HTTP listener task subsequently fails its health check or crashes, the orchestrator creates a new replica task that spawns a new container.



A task is a one-directional mechanism. It progresses monotonically through a series of states: assigned, prepared, running, etc. If the task fails the orchestrator removes the task and its container and then creates a new task to replace it according to the desired state specified by the service.

In the next lesson, we will see how to create a small swarm cluster and also modify our docker-compose file to work with the swarm.