**DOCKER SWARM**

Docker is a go-to choice for application deployment for ALL the distributed systems. But, managing the application after deployment is as essential as deploying it. Docker Swarm is one such management tool developed by Docker Inc. I will explain what is Docker Swarm*,*but I will also walk you through the topics mentioned below.

**Table of Contents:**

* [What are Docker and Docker Container?](https://k21academy.com/docker-kubernetes/docker-swarm/#FAQs)
* [What is Docker Swarm?](https://k21academy.com/docker-kubernetes/docker-swarm/#But_just_the_Docker_containers_cannot_do_everything_independently_and_that_is_where_an_orchestrator_comes_in_So_let_s_check_out_how_Docker_Swarm_helps_better_management_of_Docker_containers)
* [Features of Docker Swarm](https://k21academy.com/docker-kubernetes/docker-swarm/#Docker_Swarm_is_a_small_scale_management_tool_that_provides_a_good_orchestration_system_for_Dockerized_applications_Docker_Swarm_turns_a_pool_of_Docker_hosts_into_a_single_virtual_server_allowing_clustering_with_the_built_in_Swarm_orchestration_In_essence_we_can_utilize_the_Docker_Swarm_model_to_efficiently_deploy_manage_and_scale_a_cluster_of_nodes_on_Docker)
* [Docker Swarm Mode Concepts](https://k21academy.com/docker-kubernetes/docker-swarm/#Autoload_balancing)
* [The architecture and Working of Docker Swarm Mode](https://k21academy.com/docker-kubernetes/docker-swarm/#A_node_in_Swarm_mode_is_an_instance_of_the_Docker_engine)
* [Demo on Docker Swarm for Beginners](https://k21academy.com/docker-kubernetes/docker-swarm/#The_worker_nodes_talk_to_the_manager_node_via_API_over_HTTP_As_said_earlier_services_can_be_deployed_and_accessed_by_any_node_of_the_same_cluster_So_while_creating_a_service_you_ll_have_to_specify_the_container_image_You_can_set_up_commands_and_services_to_either_global_or_replicated_a_global_service_will_run_on_every_Swarm_node_and_on_a_replicated_service_the_manager_node_assigns_tasks_to_worker_nodes)
* [What is the Difference Between Kubernetes and Docker Swarm?](https://k21academy.com/docker-kubernetes/docker-swarm/#Hurray_Mission_accomplished_We_have_successfully_created_a_Swarm_cluster_with_1_Manager_node_and_2_Worker_Nodes_Wait_no_more_and_start_deploying_your_applications_on_the_swarm_cluster)
* [Frequently Asked Questions](https://k21academy.com/docker-kubernetes/docker-swarm/#Logging_and_Monitoring)

Let me give a quick introduction about Docker to get everybody on the same page.

**What are Docker and Docker Container?**

Docker is an open-source tool used to automate application deployment in the form of lightweight containers. And Docker containers are not only lightweight but also make the application platform-independent. So, the application that runs on your computer will run the same way on your friend’s computer!

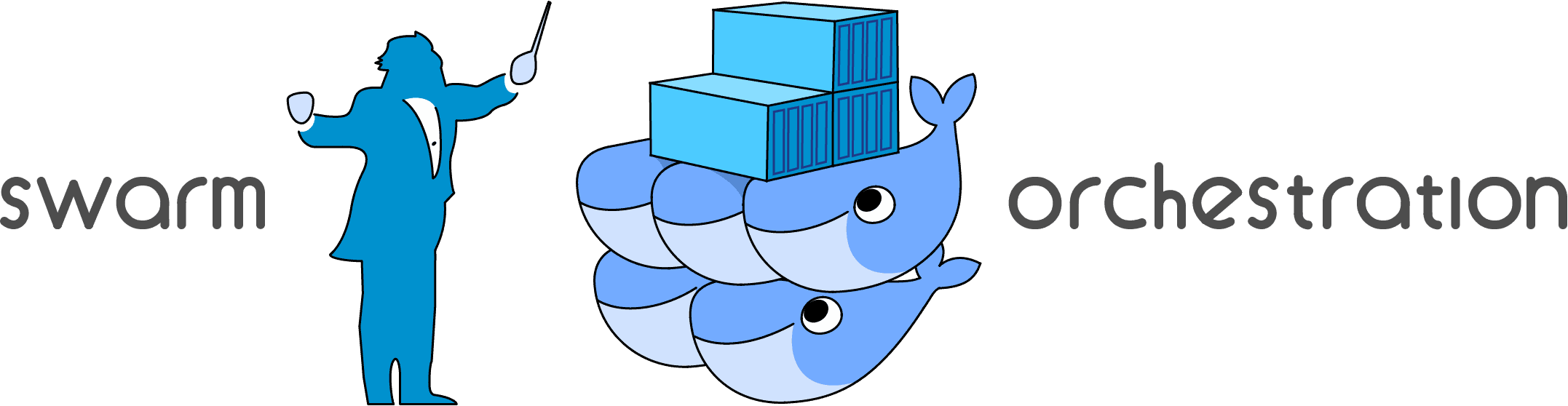
[](https://k21academy.com/wp-content/uploads/2021/08/vertical-logo-monochromatic.png)

The Docker Containers overcome a lot of problems faced by the virtual machines. Hence, the Docker Container wins the game of Docker Container vs. Virtual Machines any day! To enumerate, Docker Containers are faster, portable, provide isolation, use less memory, etc.

But just the Docker containers cannot do everything independently, and that is where an orchestrator comes in. So, let’s check out how Docker Swarm helps better management of Docker containers.

**What is Docker Swarm?**

Docker Swarm is native to the Docker platform. The Docker Swarm is integrated into Docker itself, and the Docker Swarm is installed when you install Docker.

[](https://k21academy.com/wp-content/uploads/2021/08/Docker-Swarm-Orchestration.png)

Docker Swarm is a small-scale management tool that provides a good orchestration system for Dockerized applications. Docker Swarm turns a pool of Docker hosts into a single virtual server allowing clustering with the built-in Swarm orchestration. In essence, we can utilize the Docker Swarm model to efficiently deploy, manage, and scale a cluster of nodes on Docker.

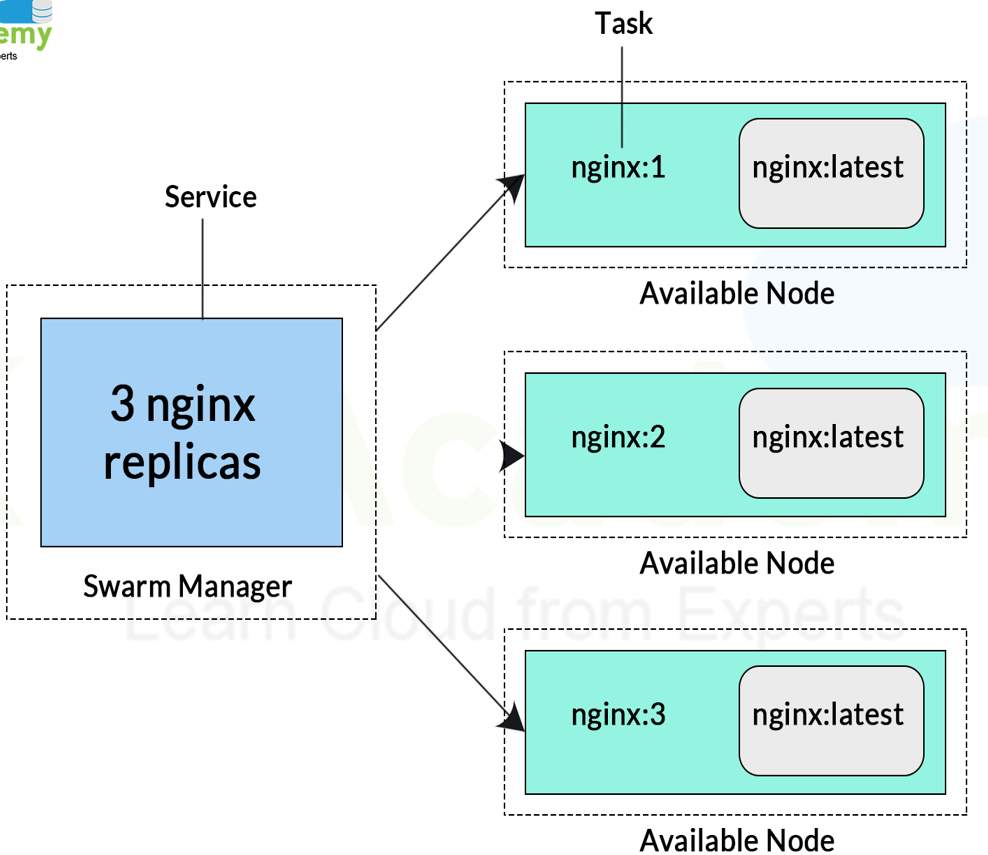
**Features of Docker Swarm**

Have a look at some of the prominent features of the Docker Swam mode, and why do we need it:

* **Simple and Fast:**The Swarm mode offers simplicity and speed for development and deployment.
* **Decentralized access:**Swarm makes it pretty easy for organizations to enter and manage the environment
* **High security:**The communication between the manager and a worker nodes is highly secure within the swarm
* **Autoload balancing:**We also have an autoload balancing option within your environment, and you can write your own script for this.
* **High scalability**: With load balancing comes high scalability.
* **Roll-back:** It also allows you to roll back environments to previous safe environments

**Docker Swarm Mode Concepts**

So, whenever I refer to ‘*Docker* Swarm’, I was actually referring to Docker Swarm Mode but not the deprecated Docker Swarm. Let’s look at some of the key concepts of swarm mode:

****

**Service and Tasks**

We use services to launch our Docker containers. And they can be deployed in either global or replicated ways.

* **Global services** are used to monitor containers that want to run on a swarm node.
* Whereas a **Replicated service** is to specify the number of similar tasks that we require on a host machine.
* Scaling becomes effortless with Docker Swarm mode.
* Services can be accessed by any of the nodes in the same cluster.
* A service is a description of a task, whereas a task performs the work.
* A developer creates a service with Docker, which can start tasks. However, when a task is allotted to a node, the same task can’t be assigned to another node.

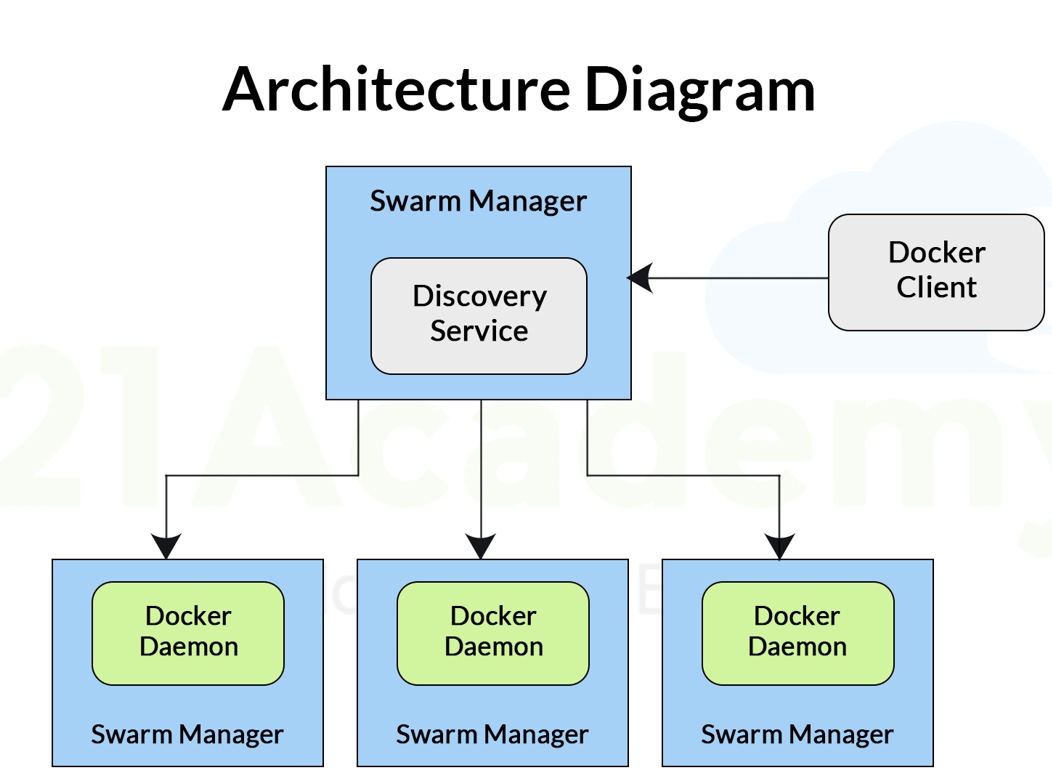
**Node**

* A node in Swarm mode is an instance of the Docker engine.
* It is possible to run multiple nodes on a single server. But in production, nodes are spread across various devices.

**The architecture and working of Docker Swarm Mode**

The idea of how you make use of Docker Swarm is simple: you declare the applications as stacks of services and let Docker handle the rest. The key components of a Docker Swarm are Docker Nodes, Docker Services, and Docker Tasks.

There are two kinds of Docker Nodes, the **Manager Node**, and the **Worker Node**. And a Swarm consists of at least one node, be it physical or virtual machines running Docker version 1.12 or later.

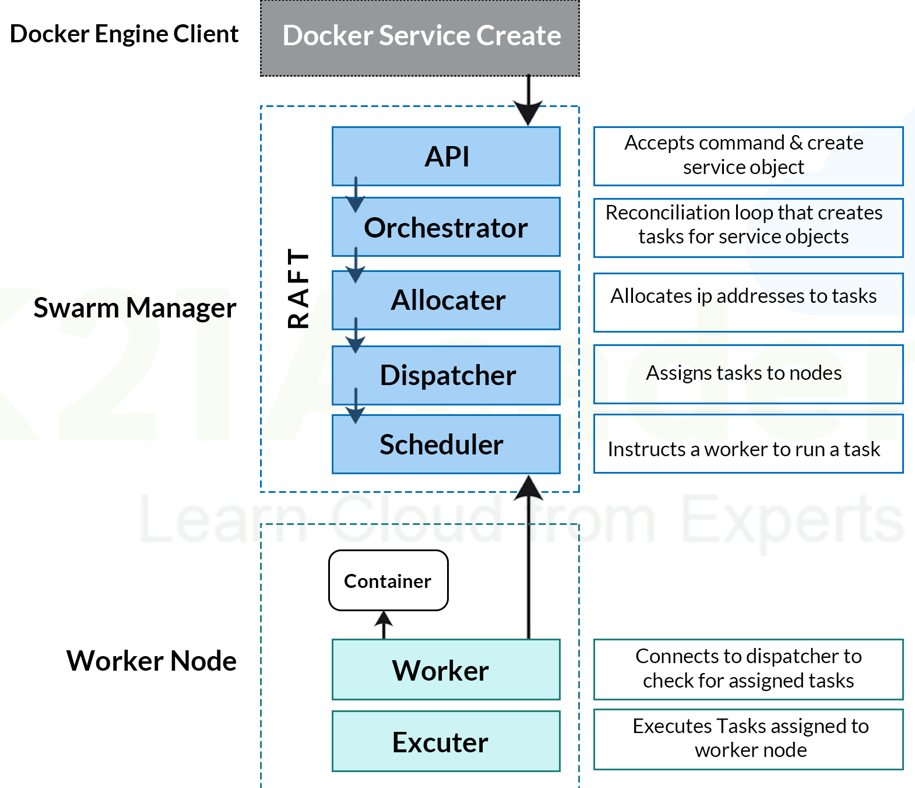
****

**Manager Node**

The manager node, as the name suggests, is responsible for the cluster management tasks:

* To maintain the cluster state
* Scheduling of Services
* And serving Swarm mode HTTP API endpoints.

But, for persistence, we use something called **Raft**implementation. The managers maintain a consistent internal state of services and the whole Swarm cluster itself. The best practice in production/non-testing purposes we should implement more than one manager node.



**Worker Node**

The Worker nodes are nothing but the instances of Docker Engine which are created to run Docker Containers. They are not involved in managing tasks. A manager node can exist without a worker node; a worker cannot exist without a manager. A manager node is also a worker node by default.

The worker nodes talk to the manager node via API over HTTP. As said earlier, services can be deployed and accessed by any node of the same cluster. So, while creating a service, you’ll have to specify the container image. You can set up commands and services to either global or replicated: a global service will run on every Swarm node, and on a replicated service, the manager node assigns tasks to worker nodes.

Now comes the most interesting section. I will show you how to create a cluster, upgrade nodes, start a service and, much more!

**Demo on Docker Swarm for Beginners**

Now comes the most interesting part, hands-on with Docker Swarm! I will show you how to set up a Swarm cluster with 1 manager node and 2 worker nodes. Check out the prerequisite to get started:

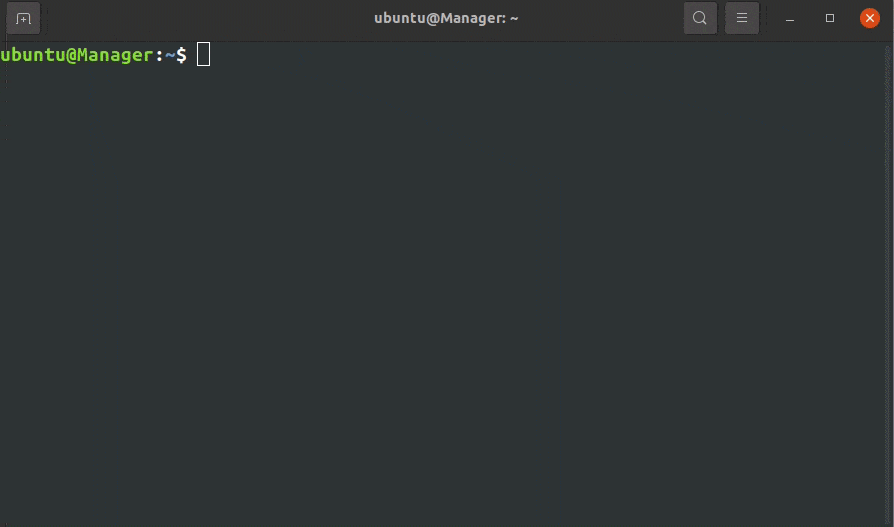
1. Three Ubuntu (or any other Linux distros) virtual machines or
2. Three Amazon AWS ec2 instances (I will be using ec2)  
   Know how to create a free [**AWS account**](https://k21academy.com/amazon-web-services/aws-solutions-architect/create-aws-free-tier-account/).
3. Interest to try out new things

**Step 1:  Install Docker**

Install Docker on all three instances or VMs with the following command

curl -fsSL https://get.docker.com -o get-docker.sh

sh get-docker.sh

[](https://k21academy.com/wp-content/uploads/2021/08/cut.gif)

You can check if it is installed by the below command:

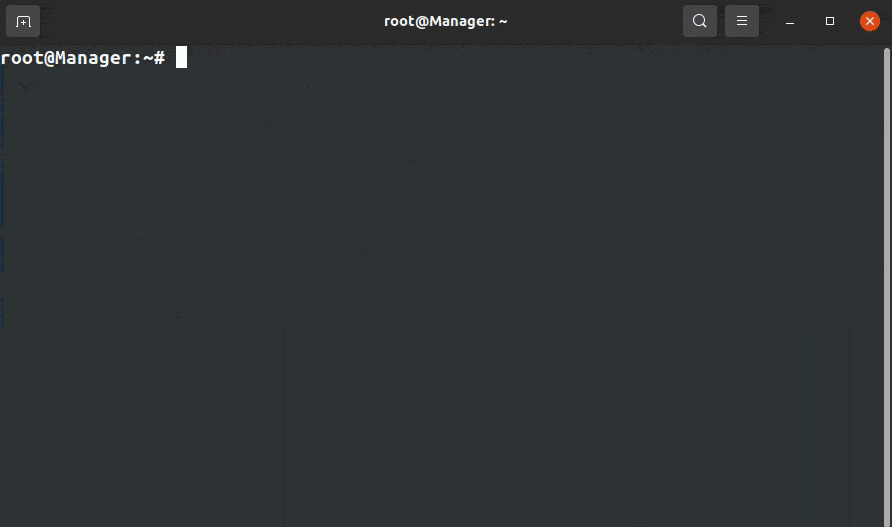
docker --version

**Step 2: Create a Swarm Manager**

We will have to create a manager to handle our worker nodes. And, you should also know that the manager is also a worker node with some special privileges.

docker swarm init --advertise-addr [private or public ip address]

Use private IP addresses if all the instances/VMs are in the same network, if not you should be using the pubic IP instead.

[](https://k21academy.com/wp-content/uploads/2021/08/ezgif.com-gif-maker-1.gif)

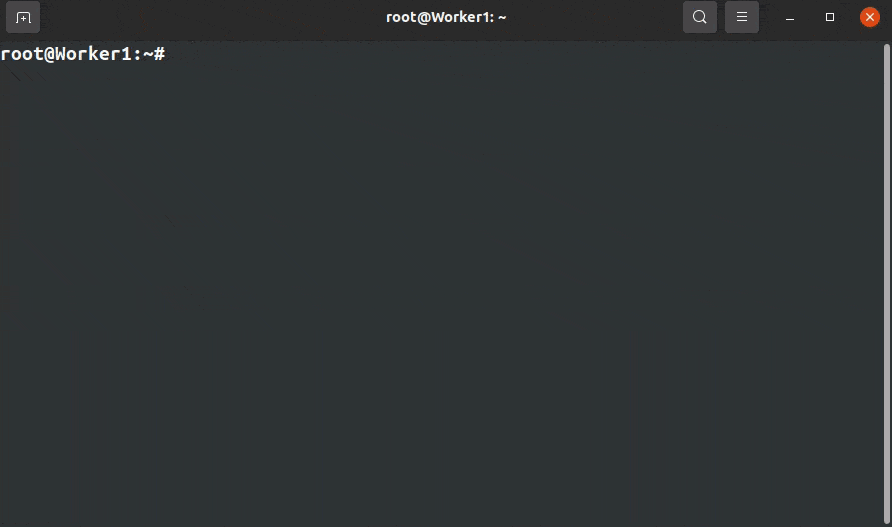
**Note:***Make sure you copy the above-selected command which will be required in the next step.*

**Step 3: Add Worker Nodes**

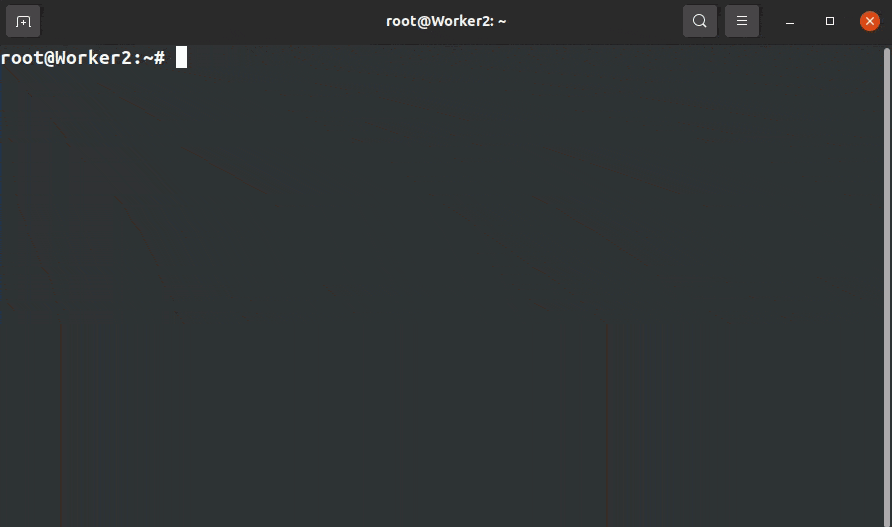
The next step is to join our two worker nodes to the Swarm cluster by using the token which was generated earlier.

docker swarm join --token <token-id> <ip:port>

**Worker 1:**

[](https://k21academy.com/wp-content/uploads/2021/08/ezgif.com-gif-maker-2.gif)

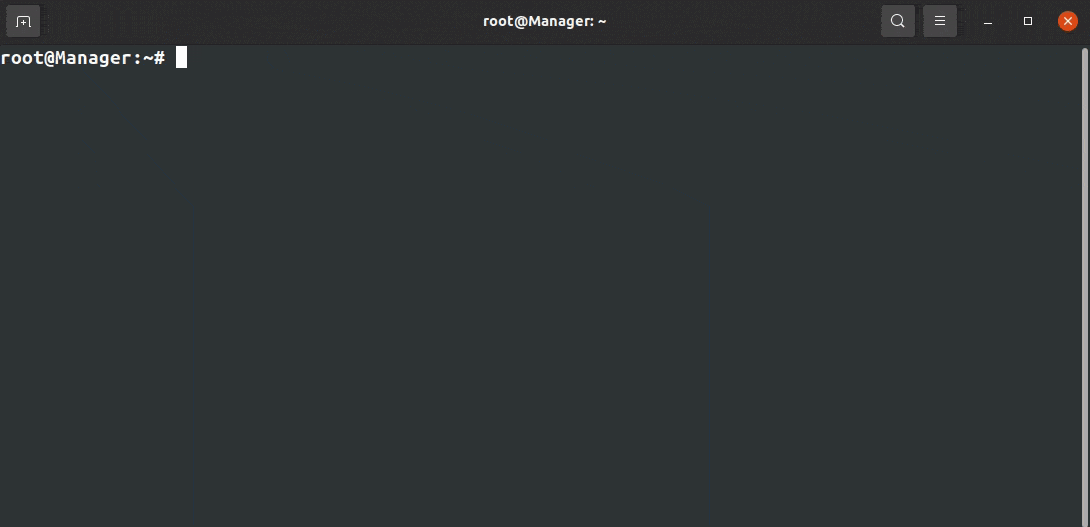
**Worker 2:**

[](https://k21academy.com/wp-content/uploads/2021/08/ezgif.com-gif-maker-3.gif)

**Step 5: Check the Nodes**

Now, let us check the status of our Swarm cluster. Perform the command below.

docker node ls

[](https://k21academy.com/wp-content/uploads/2021/08/Peek-2021-08-12-19-58.gif)

Hurray! Mission accomplished. We have successfully created a Swarm cluster with 1 Manager node and 2 Worker Nodes. Wait no more and start deploying your applications on the swarm cluster.

**What is the Difference Between Kubernetes and Docker Swarm?**

Kubernetes and Docker Swarm are the two most used open-source platforms, that have similar features. But, on closer review, various fundamental differences can be seen between how these two functions. Refer to the table below for a better picture.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Kubernetes** | **Docker Swarm** |
| **Application Deployment** | Applications can be deployed in K8s using innumerable microservices, deployments, and pods. | Applications can be used only as microservices in a swarm cluster. |
| **Scalability** | Kubernetes acts like more of an all-in-one framework when working with distributed systems. It is thus significantly more complicated as it provides strong guarantees in terms of the cluster state and a unified set of APIs. Container scaling and deployment are therefore slowed down. | Docker Swarm can deploy containers much faster than Kubernetes, which allows faster reaction times for scaling on demand. |
| **Container Setup** | By utilizing its own YAML, API, and client definitions, Kubernetes differs from other standard docker equivalents. Thus, Docker Compose or Docker CLI cannot be used to define containers. Also, YAML definitions and commands must be rewritten when switching platforms. | The Docker Swarm API offers much of the familiar functionality from Docker, supporting most of the tools that run with Docker. However, Swarm cannot be utilized if the Docker API is deficient in a particular operation. |
| **Networking** | Kubernetes has a flat network model, allowing all the pods to communicate with each other. Network policies are in place to define how the pods interact with one another. The network is implemented typically as an overlay, requiring two CIDRS for the services and the pods. | When a node joins a swarm cluster, it creates an overlay network for services for each host in the docker swarm. It also creates a host-only docker bridge network for containers. This gives users a choice while encrypting the container data traffic to create its own overlay network. |
| **Availability** | Kubernetes offers significantly high availability as it distributes all the pods among the nodes. This is achieved by absorbing the failure of an application. Unhealthy pods are detected by load balancing services, which subsequently deactivate them. | Docker also offers high availability architecture since all the services can be cloned in Swarm nodes. The Swarm manager Nodes manage the worker’s node resources and the whole cluster. |
| **Load Balancing** | In Kubernetes, pods are exposed via service, allowing them to be implemented as a load balancer inside a cluster. An ingress is generally used for load balancing. | Swarm Mode comes with a DNS element that can be used for distributing incoming requests to a service name. Thus services can be assigned automatically or function on ports that are pre-specified by the user. |
| **Logging and Monitoring** | It includes built-in tools for managing both processes. | It does not require using any tools for logging and monitoring. |
| **GUI** | Kubernetes has detailed dashboards to allow even non-technical users to control the clusters effectively. | ocker Swarm, alternatively, requires a third-party tool such as Portainer.io to manage the UI conveniently. |

**Frequently Asked Questions**

**What is Docker Swarm used for?**

Docker Swarm is used for managing and orchestrating Docker containers, enabling efficient deployment, scaling, and management of applications.

**Can Docker Swarm work with Kubernetes?**

While Docker Swarm and Kubernetes are separate orchestration tools, it is possible to use them together in some scenarios, though it is not common practice.

**Is Docker Swarm suitable for production use?**

Yes, Docker Swarm is suitable for production use, especially for small to medium-sized applications.

**How does Docker Swarm handle scaling?**

Docker Swarm handles scaling by allowing you to specify the number of replicas for a service, which can be distributed across available nodes.

**What is the role of a manager node in Docker Swarm?**

The manager node is responsible for maintaining the cluster state, scheduling services, and serving Swarm mode HTTP API endpoints.