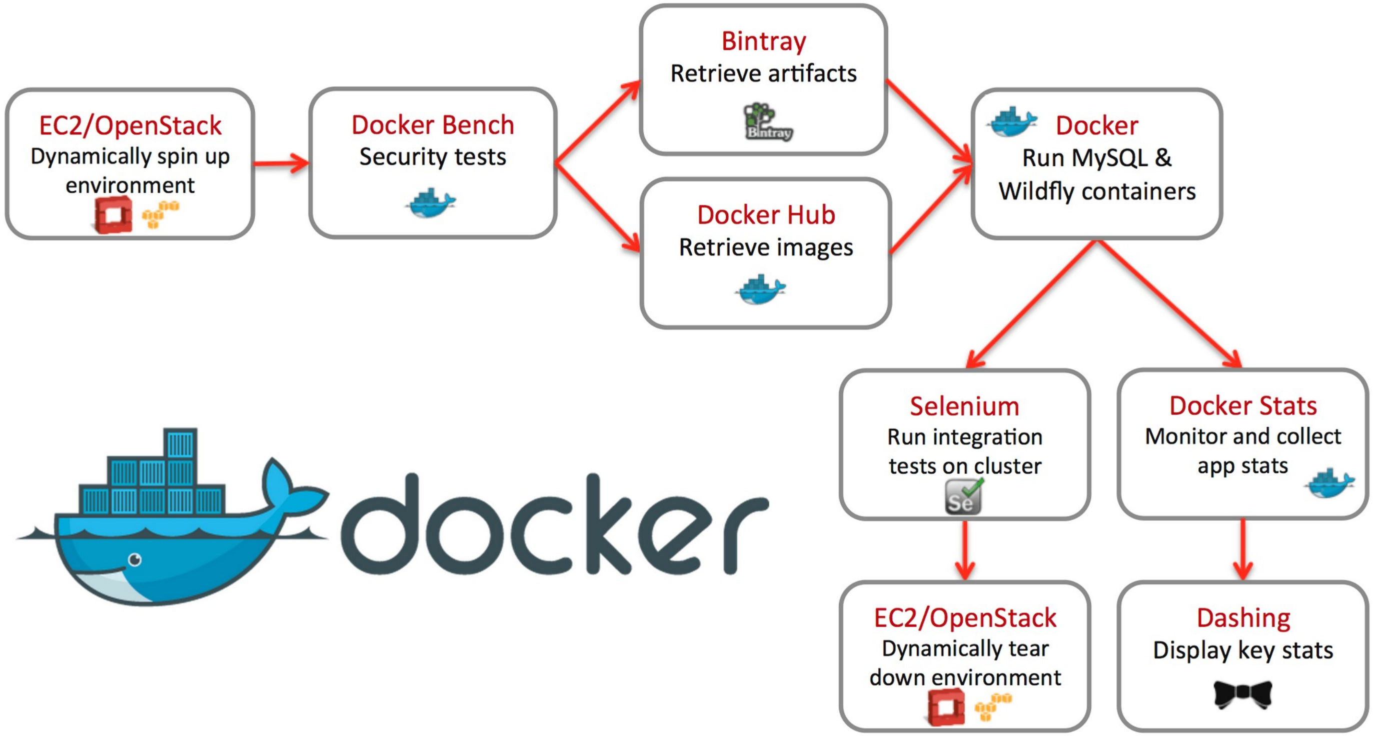
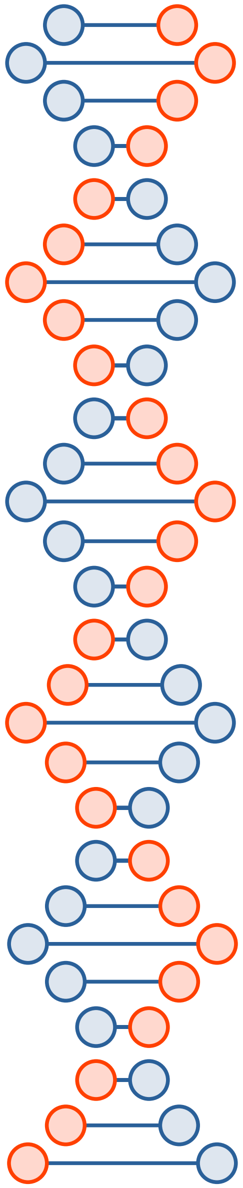
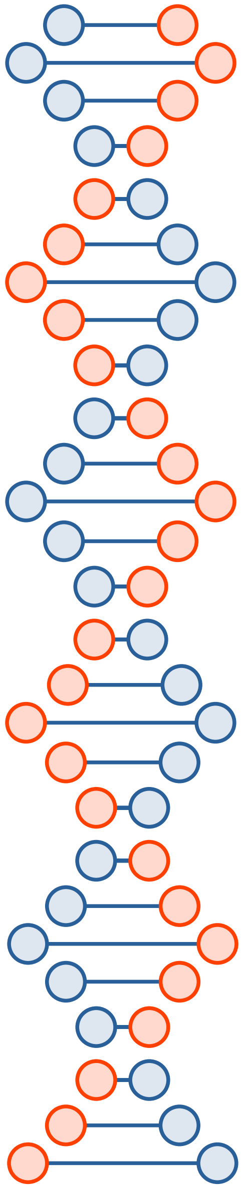
# Docker Presentatiton


EC2/OpenStack
Dynamically spin up environment
Docker Bench
Security tests
Bintray
Retrieve artifacts
Docker Hub
Retrieve images
Docker
Run MySQL & Wildfly containers
Selenium
Run integration tests on cluster
Docker Stats
Monitor and collect app stats
EC2/OpenStack
Dynamically tear down environment
Dashing
Display key stats

Present By : Ayush Sharma



## Table of Contents:



# What is Docker

Docker is a popular platform that allows developers to package, distribute, and run applications within containers. These containers are lightweight, portable, and self-sufficient units that can encapsulate all the necessary components (such as code, runtime, system tools, libraries, and settings) required to run an application.

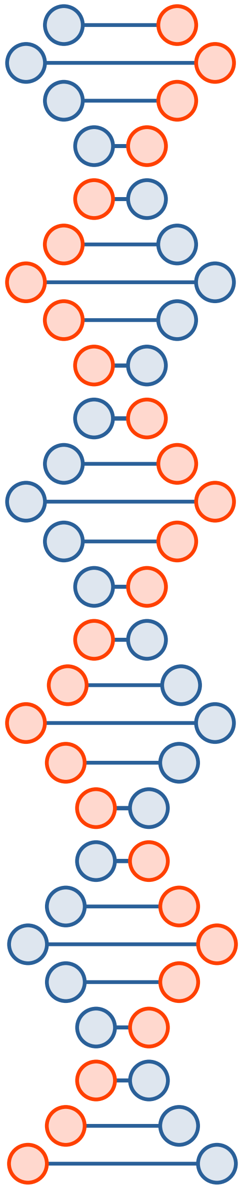
**Key components of Docker include:**

**Docker Engine**: The core of Docker, responsible for building, running, and managing containers. It consists of a server (daemon) and a command-line interface (CLI) tool.

**Docker Image**: A lightweight, standalone, executable package that contains all the necessary components to run a piece of software. Images are the building blocks used to create containers.

**Docker Container**: An instance of a Docker image that runs as a process on the host machine's operating system. Containers are isolated from each other and from the host system, providing consistency and portability across different environments.

**Dockerfile**: A text file that contains instructions for building a Docker image. It specifies the base image, dependencies, environment variables, and other configurations required for the application.



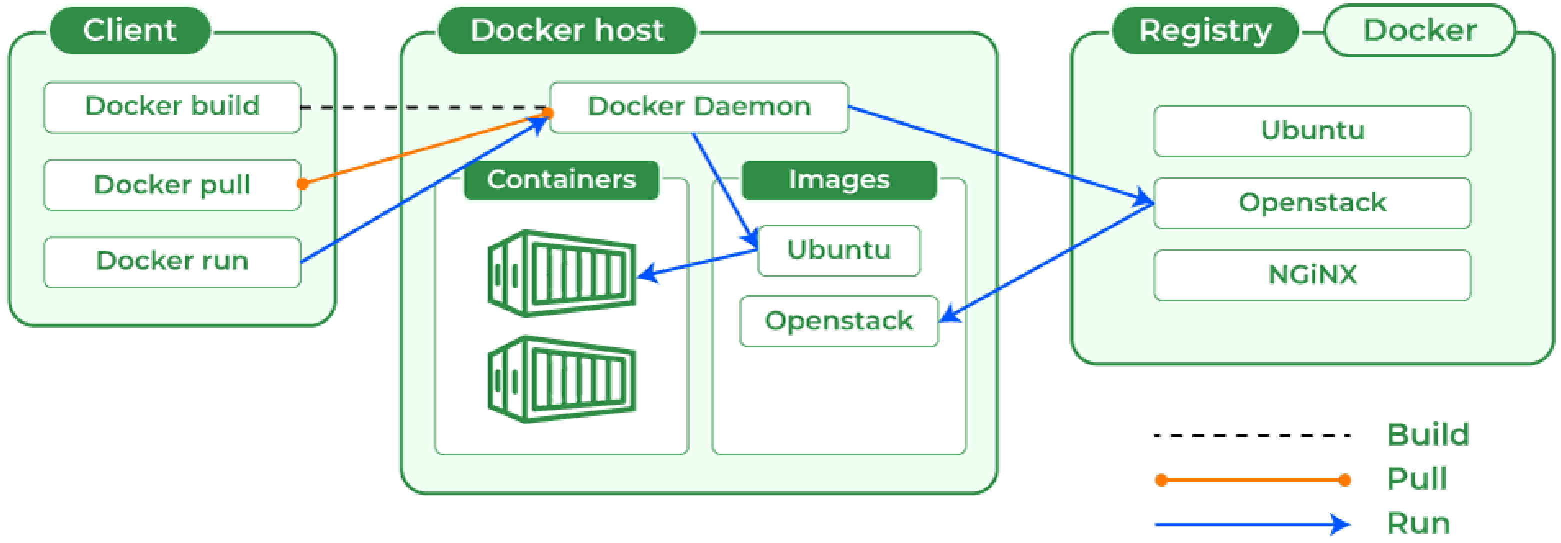
**Docker revolutionizes the way applications are developed, shipped, and deployed by offering several benefits:**

Consistency: Docker ensures that applications run consistently across different environments, from development to production.

Portability: Docker containers can be easily moved and deployed across various platforms, including on-premises servers, cloud environments, and developer workstations.
Resource Efficiency: Containers share the host system's kernel and utilize resources more efficiently compared to traditional virtual machines.

Scalability: Docker enables horizontal scaling by allowing applications to be easily replicated and distributed across multiple containers.
Rapid Deployment: With Docker, developers can quickly build, test, and deploy applications, reducing time-to-market and improving agility.

## Architecture of Docker



Docker makes use of a client-server architecture. The Docker client talks with the docker daemon which helps in building, running, and distributing the docker containers.

The Docker client runs with the daemon on the same system or we can connect the Docker client with the Docker daemon remotely. With the help of REST API over a UNIX socket or a network, the docker client and daemon interact with each other.

## What is Docker Daemon?

Docker daemon manages all the services by communicating with other daemons. It manages docker objects such as images, containers, networks, and volumes with the help of the API requests of Docker.

## Docker Client

With the help of the docker client, the docker users can interact with the docker. The docker command uses the Docker API. The Docker client can communicate with multiple daemons. When a docker client runs any docker command on the docker terminal then the terminal sends instructions to the daemon.

The Docker daemon gets those instructions from the docker client withinside the shape of the command and REST API's request.

## Docker Host

A Docker host is a type of machine that is responsible for running more than one container. It comprises the Docker daemon, Images, Containers, Networks, and Storage.

## Docker Registry

All the docker images are stored in the docker registry. There is a public registry which is known as a docker hub that can be used by anyone. We can run our private registry also. With the help of docker run or docker pull commands, we can pull the required images from our configured registry. Images are pushed into configured registry with the help of the docker push command.

## Docker Images

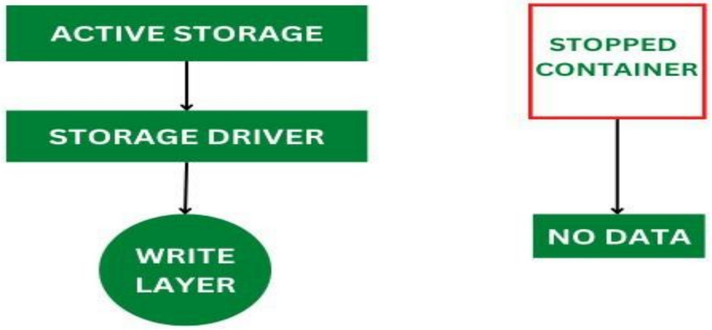
An image contains instructions for creating a docker container. It is just a read-only template. It is used to store and ship applications. Images are an important part of the docker experience as they enable collaboration between developers in any way which is not possible earlier.

## Docker Containers

Containers are created from docker images as they are ready applications. With the help of Docker API or CLI, we can start, stop, delete, or move a container. A container can access only those resources which are defined in the image unless additional access is defined during the building of an image in the container.

## Docker Storage

We can store data within the writable layer of the container but it requires a storage driver. Storage driver controls and manages the images and containers on our docker host.



## Types of Docker Storage

**Data Volumes**: Data Volumes can be mounted directly into the filesystem of the container and are essentially directories or files on the Docker Host filesystem.

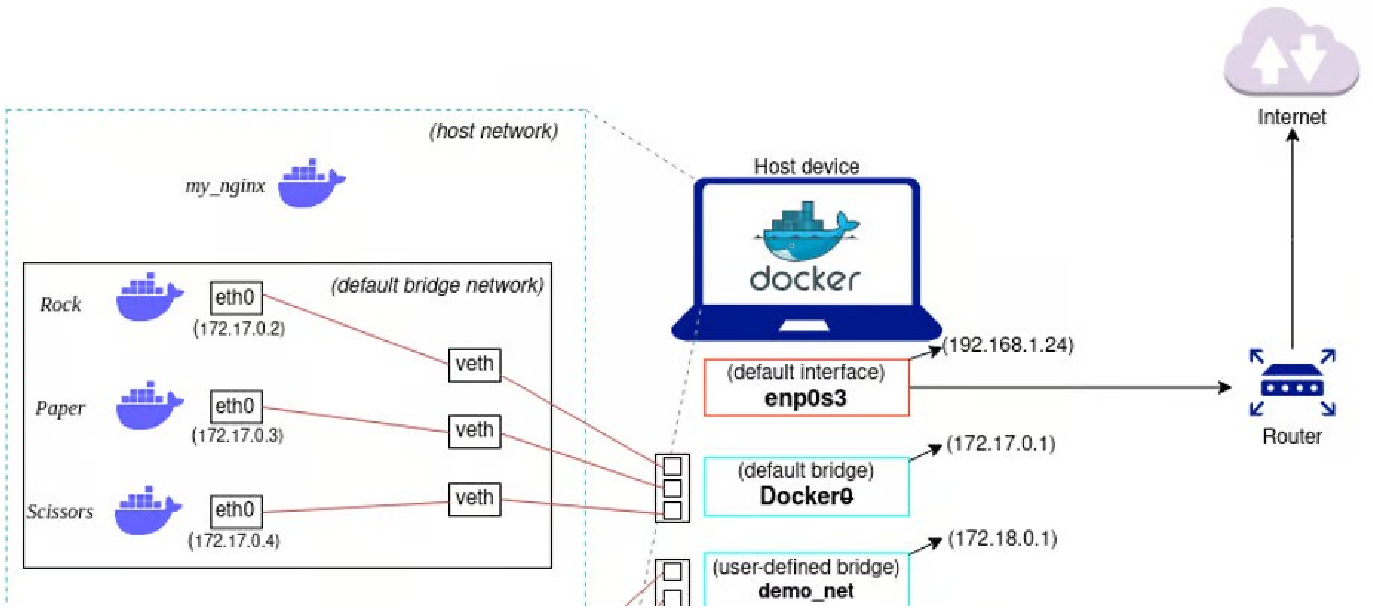
**Volume Container**: In order to maintain the state of the containers (data) produced by the running container, Docker volumes file systems are mounted on Docker containers. independent container life cycle, the volumes are stored on the host. This makes it simple for users to exchange file systems among containers and backup data.

**Directory Mounts**: A host directory that is mounted as a volume in your container might be specified.

**Storage Plugins**: Docker volume plugins enable us to integrate the Docker containers with external volumes like Amazon EBS by this we can maintain the state of the container.

## Docker Networking

Docker Networking provides complete isolation for docker containers. It means a user can link a docker container to many networks. It requires very less OS instances to run the workload.



## Types of Docker Network

**Bridge**: It is the default network driver. We can use this when different containers communicate with the same docker host.

**Host**: When you don't need any isolation between the container and host then it is used.

**Overlay**: For communication with each other, it will enable the swarm services.

None: It disables all networking.

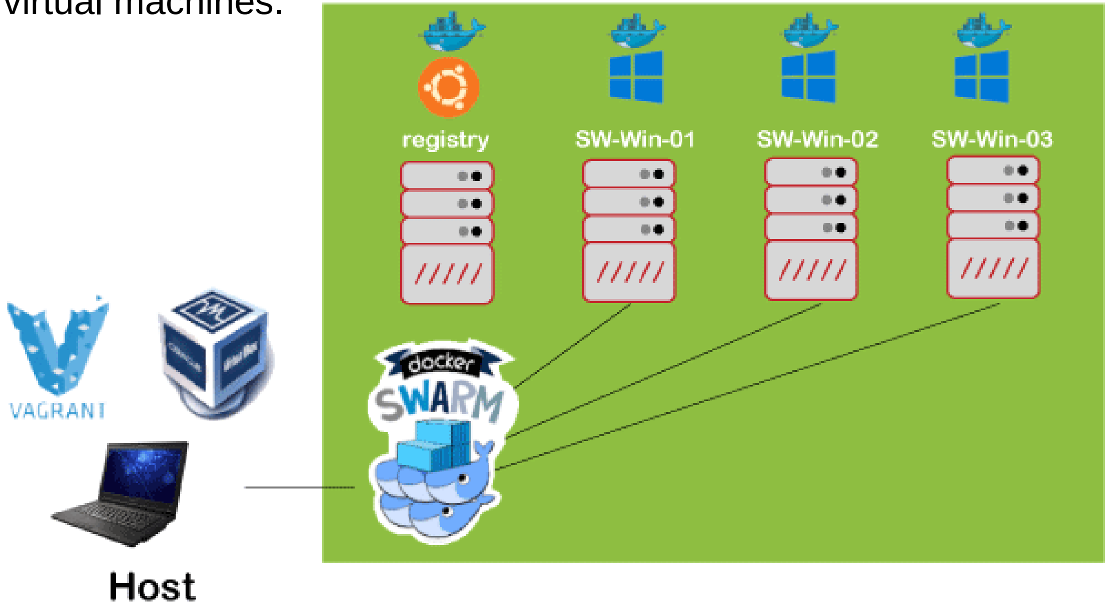
**macvlan**: This network assigns MAC(Media Access control) address to the containers which look like a physical address.

## Why Choose Docker Swarm?

Developers are vouching for Docker Swarm due to its ease of container orchestration management and automated load-balancing feature. Along with offering high scalability and security, it is the best and most reliable tool to use for your next project. Why?

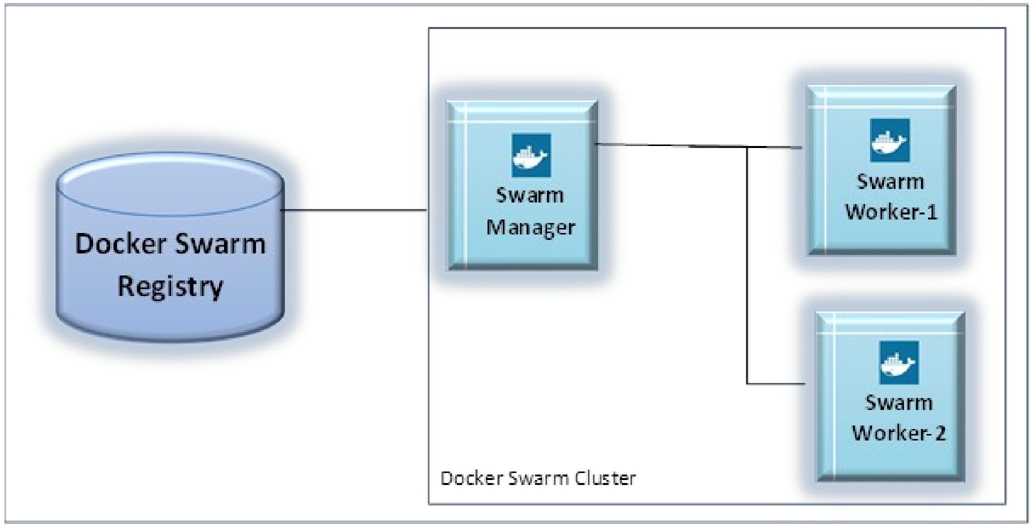
This blog explains it all. It is a comprehensive guide to Docker Swarm. By the end, you will have a better understanding of Docker swarm, its key concepts, its service nodes, advanced features, key benefits, and steps of implementation. We have also added some interesting comparisons of the docker swarm with the latest container technologies.

But before we start with docker swarm, we must understand important aspects of Docker and how its implementation is different from virtual machines.



## What is a Swarm Cluster? Which are the types of Swarm Nodes?

Swarm means the formation of a group or a cluster of nodes working together in a docker environment.



**There are three types of nodes in a docker swarm:**

**Leader Node:**

Once the cluster is formed, a leader node is rendered by using the 'Raft Consensus' algorithm. It takes all the decisions related to task orchestration and management.In any case, if the leader node becomes unavailable, its responsibility is transformed to another node by rendering the same algorithm.

**Manager Node:**

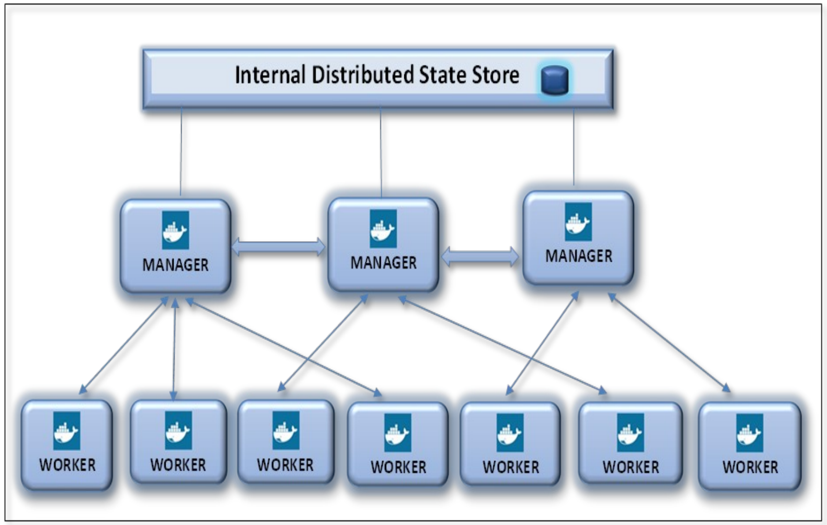
A manager node allocates the task to worker nodes and makes sure that tasks are executed well.

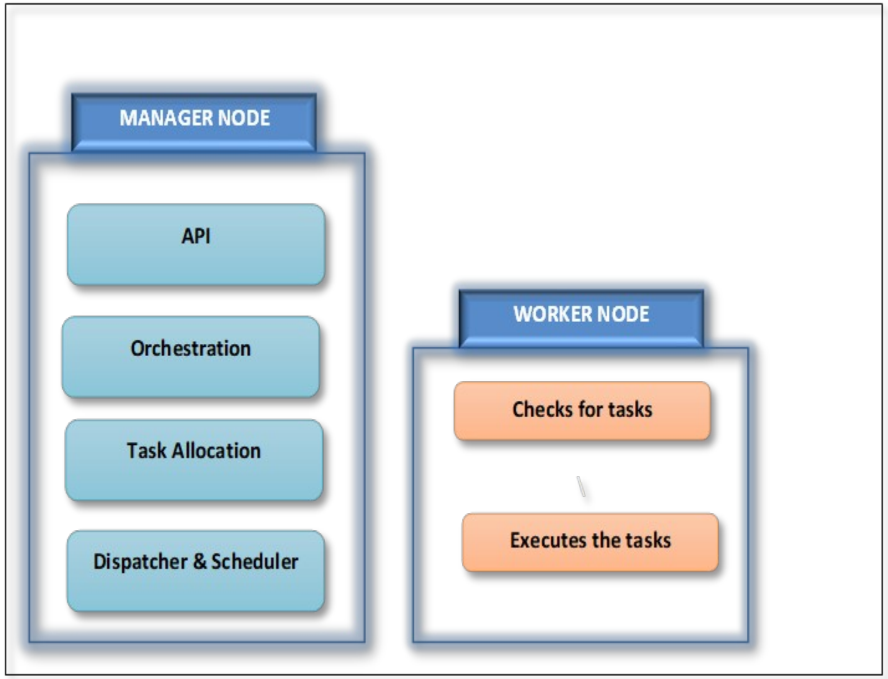
**Worker Node:**

A worker node checks for an assigned task and executes it. You need a manager node to create a worker node.

## How does Docker Swarm Work?

The containers are launched using services. A service refers to a collection of tasks that are to be executed by worker nodes. It is created by using Command-Line Interface (CLI).



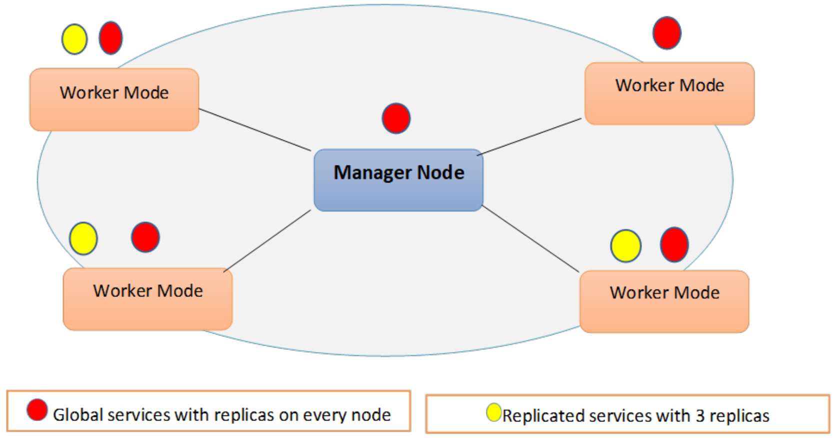


## What are Docker Swarm Modes?

By now you are aware that docker containers are launched using services. There are two docker swarm modes to deploy these services:

### Global Service Mode:

In the global mode, a manager node schedules a single task to all the available worker nodes that fulfill resource requirements and service constraints. Global services monitor the containers that want to run on swarm mode.


Worker Mode
Worker Mode
Manager Node
Worker Mode
Worker Mode
● Global services with replicas on every node
● Replicated services with 3 replicas

### Replicated Service Mode:

Here users define a specific number of replica tasks for manager nodes to allocate. Manager nodes are commanded to allocate these tasks to specific worker nodes that meet the requirements and service constraints. Replicated services decide the number of replicated tasks that a developer requires to implement on the host machine.

### What are the Key features of Docker Swarm?

Some key features offered by docker swarm are:

Decentralized Access:

With docker swarm, developers can easily access and manage multiple computing environments.

## Setting up Docker Swarm cluster.

Now let us set up a Docker Swarm manager node and a service on Ubuntu 16.04. The prerequisite for this setup is we should have the latest version of Docker Engine installed on the host. So let us begin with the steps.

### Download and run the container.

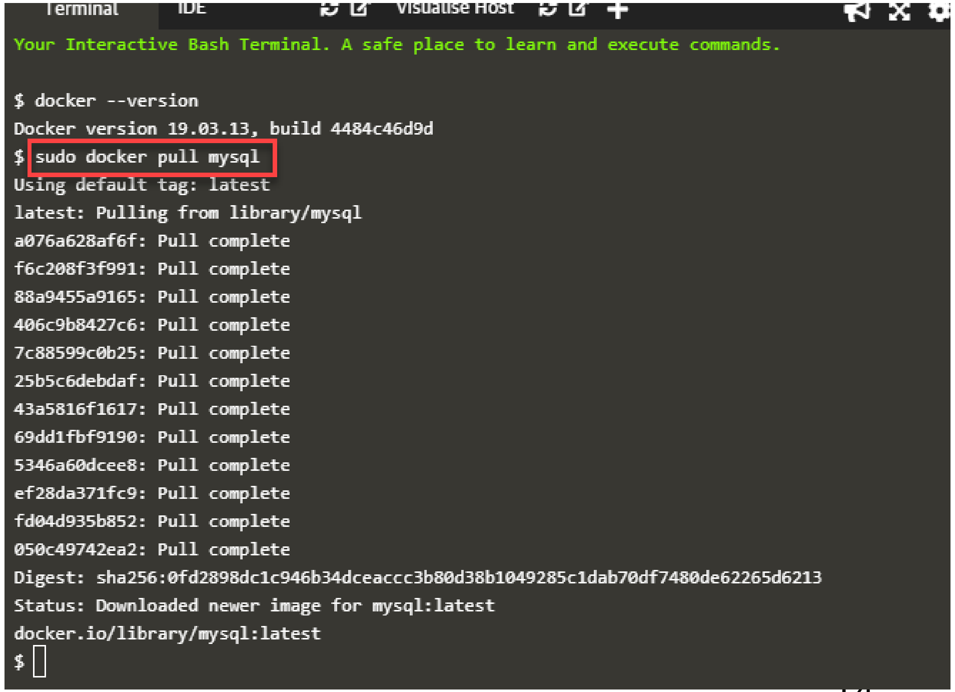
First, we can verify the docker version on the host. Then we download the container, say MySQL using the following command.

```
docker pull mysql
```

```
This command will download the
latest version MySQL container as
shown in the output below.
```

**Next, we run this container using the following command.**

```
docker run -it -d -p 80:80 mysql:latest
```



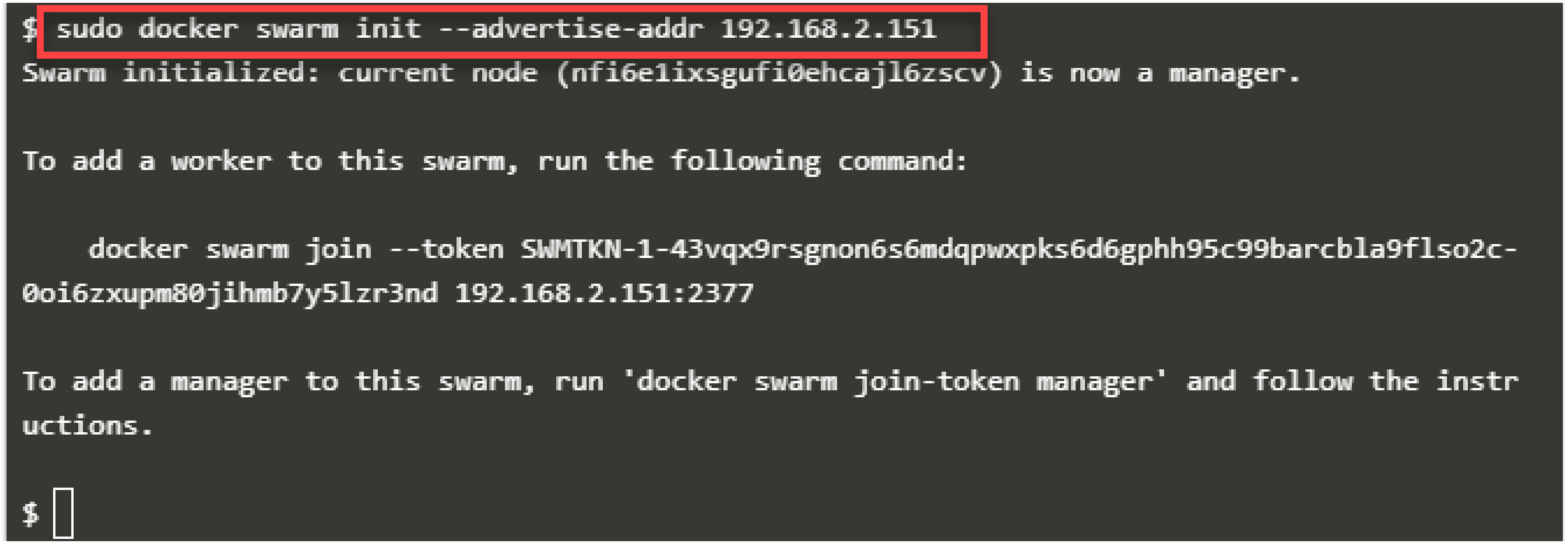
```
Your Interactive Bash Terminal. A safe place to learn and execute commands.

$ docker --version
Docker version 19.03.13, build 4484c46d9d
$ sudo docker pull mysql
Using default tag: latest
latest: Pulling from library/mysql
a076a628af6f: Pull complete
f6c208f3f991: Pull complete
88a9455a9165: Pull complete
406c9b8427c6: Pull complete
7c88599c0b25: Pull complete
25b5c6debdaf: Pull complete
43a5816f1617: Pull complete
69dd1fbf9190: Pull complete
5346a60dcee8: Pull complete
ef28da371fc9: Pull complete
fd04d935b852: Pull complete
050c49742ea2: Pull complete
Digest: sha256:0fd2898dc1c946b34dceaccc3b80d38b1049285c1dab70df7480de62265d6213
Status: Downloaded newer image for mysql:latest
docker.io/library/mysql:latest
$ 
```

As seen in the above screenshot, we can verify the running of the MySQL container using the 'docker ps -a' command that shows an entry of the above container. Once the container is running now, we go ahead and create Docker Swarm.

**Create Swarm : docker using swarm init command**

Here first, we create a Swarm cluster by giving the IP address of the manager node. For this, issue the following command.

```
docker swarm init --advertise-addr 192.168.2.151
```



```
$ sudo docker swarm init --advertise-addr 192.168.2.151
Swarm initialized: current node (nfi6e1ixsgufi0ehcajl6zscv) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-43vqx9rsgnon6s6mdqpwxpks6d6gphh95c99barcbla9flso2c-
0oi6zxupm80jihmb7y5lzr3nd 192.168.2.151:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instr
uctions.

$ 
```

As you can see from the above output, the manager node created.

Note:

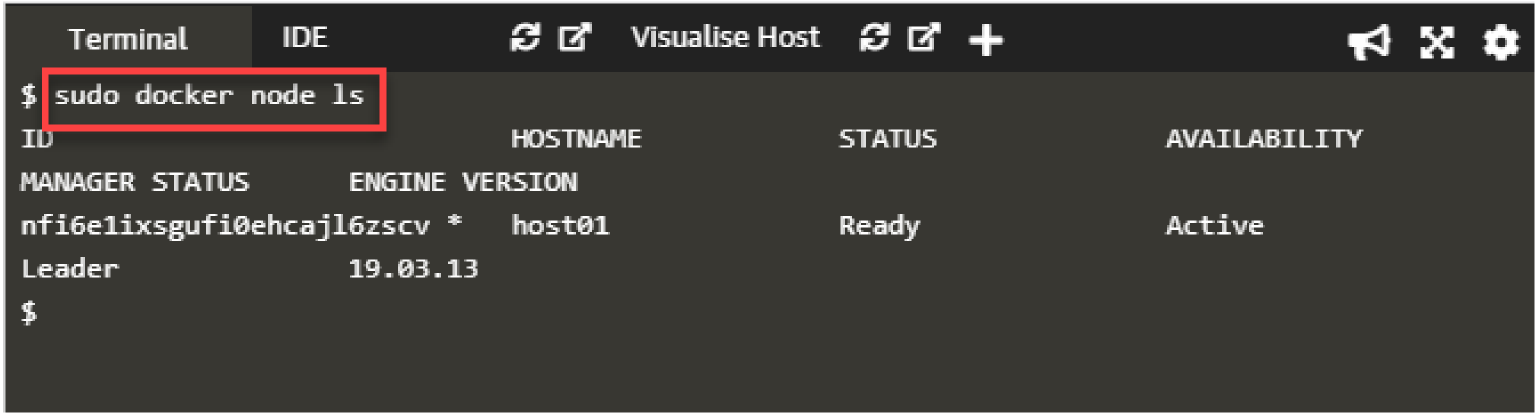
We can similarly create a worker node by copying the command of the “swarm init” (above command) and pasting the output onto the worker node

**"sudo Docker Swarm join --token SWMTKN-1- xxxxx".** It has to execute on a different host.

So to create a complete swarm cluster, we require two hosts with the latest version of Docker Engine installed.

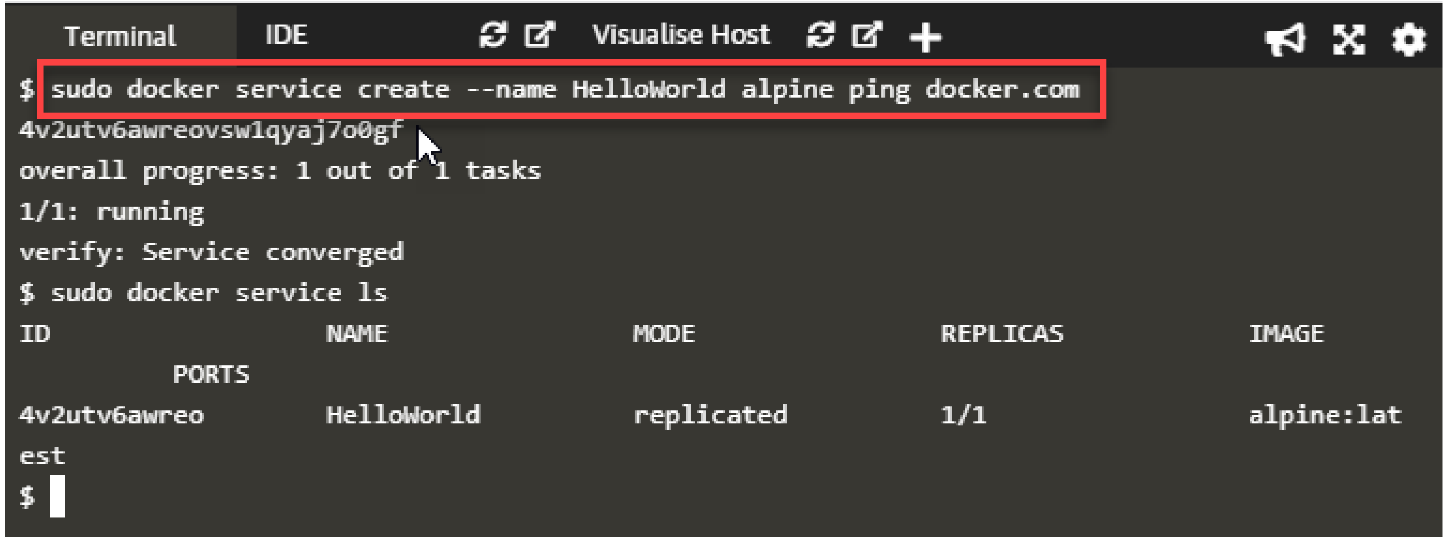
Once we do this, we proceed further and list the nodes from the manager node with the following command.

**`docker node ls`**



```
$ sudo docker node ls
ID                          HOSTNAME            STATUS              AVAILABILITY
MANAGER STATUS      ENGINE VERSION
nfi6e1ixsgufi0ehcajl6zscv *   host01              Ready               Active
Leader              19.03.13
$
```

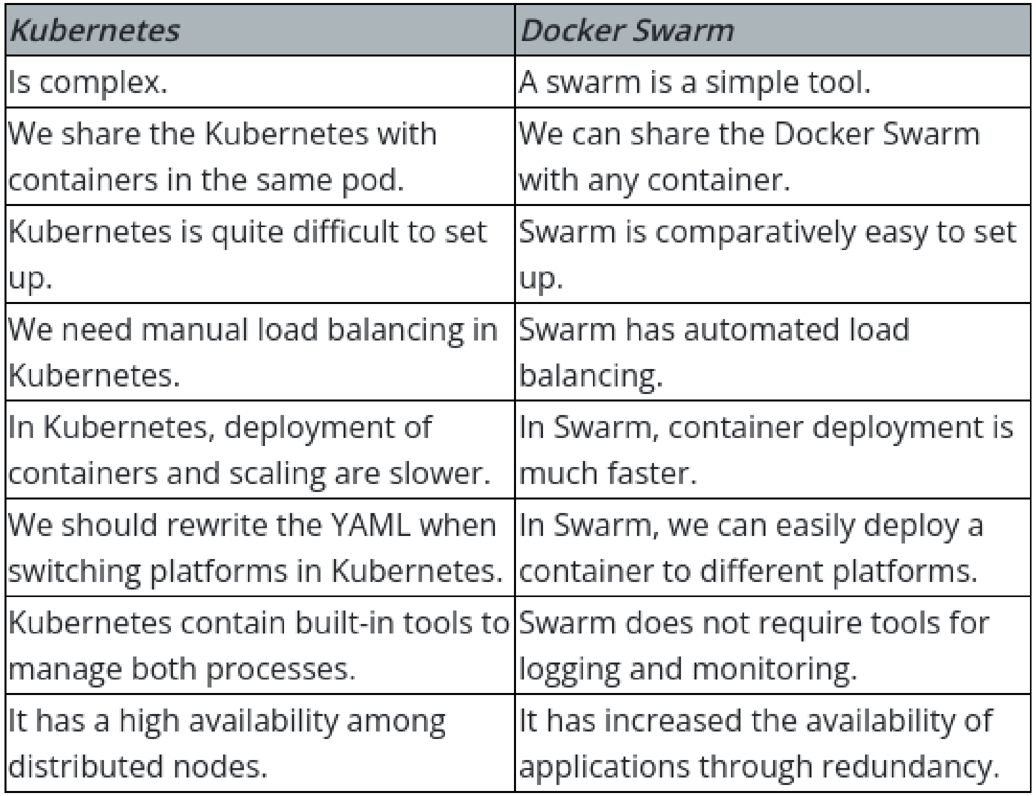
Next, we launch the service in Swarm mode. We have to do it in the manager node. We will deploy the simple service 'HelloWorld' using the following command.



```
$ sudo docker service create --name HelloWorld alpine ping docker.com
4v2utv6awreovsw1qyaj7o0gf
overall progress: 1 out of 1 tasks
1/1: running
verify: Service converged
$ sudo docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE
          PORTS
4v2utv6awreo        HelloWorld          replicated          1/1                 alpine:lat
est
$
```

## Kubernetes vs. Docker Swarm

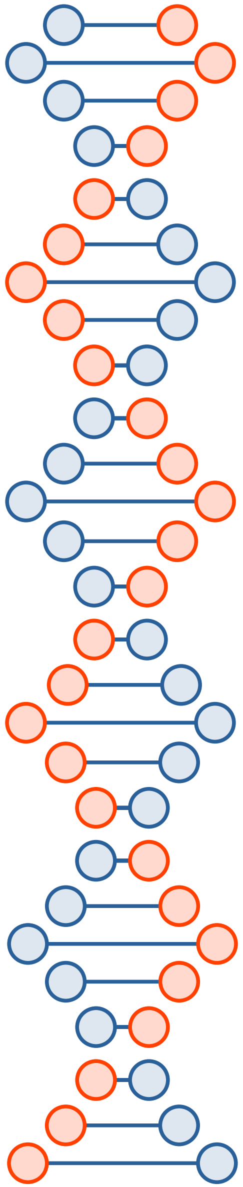
The below table shows the significant differences between Kubernetes and Docker Swarm.



| *Kubernetes* | *Docker Swarm* |
|---|---|
| Is complex. | A swarm is a simple tool. |
| We share the Kubernetes with containers in the same pod. | We can share the Docker Swarm with any container. |
| Kubernetes is quite difficult to set up. | Swarm is comparatively easy to set up. |
| We need manual load balancing in Kubernetes. | Swarm has automated load balancing. |
| In Kubernetes, deployment of containers and scaling are slower. | In Swarm, container deployment is much faster. |
| We should rewrite the YAML when switching platforms in Kubernetes. | In Swarm, we can easily deploy a container to different platforms. |
| Kubernetes contain built-in tools to manage both processes. | Swarm does not require tools for logging and monitoring. |
| It has a high availability among distributed nodes. | It has increased the availability of applications through redundancy. |

## Key TakeAways

The Docker Swarm is an orchestration tool. Additionally, it has two significant nodes, namely, the manager node and the worker node. The manager node is responsible for the management of the Swarm cluster and distributing tasks to worker nodes. Worker node checks for tasks and then completes them.



Thank You