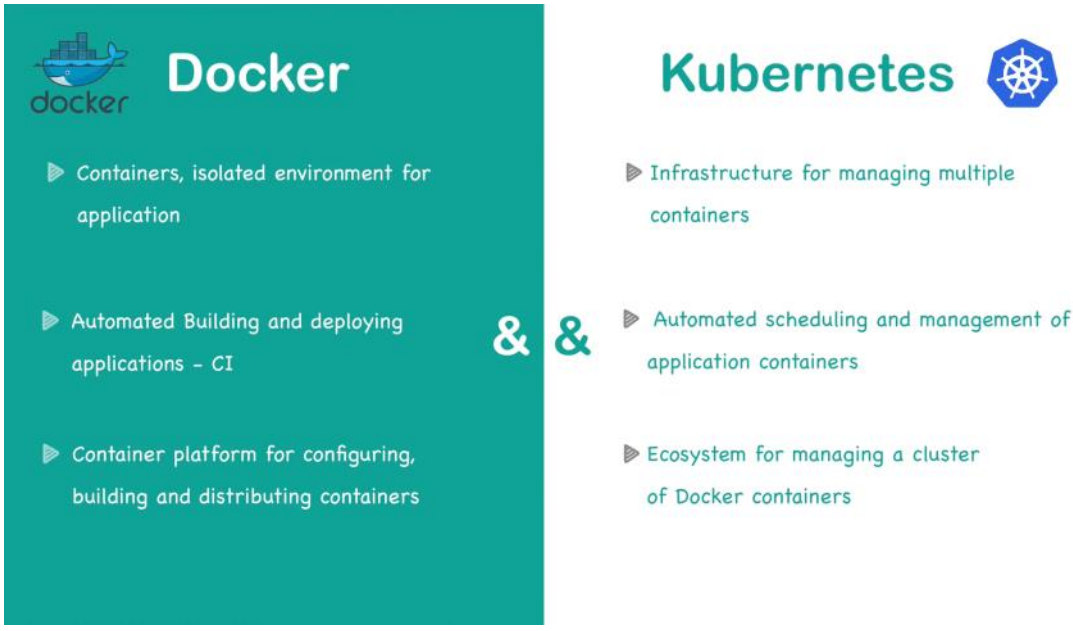
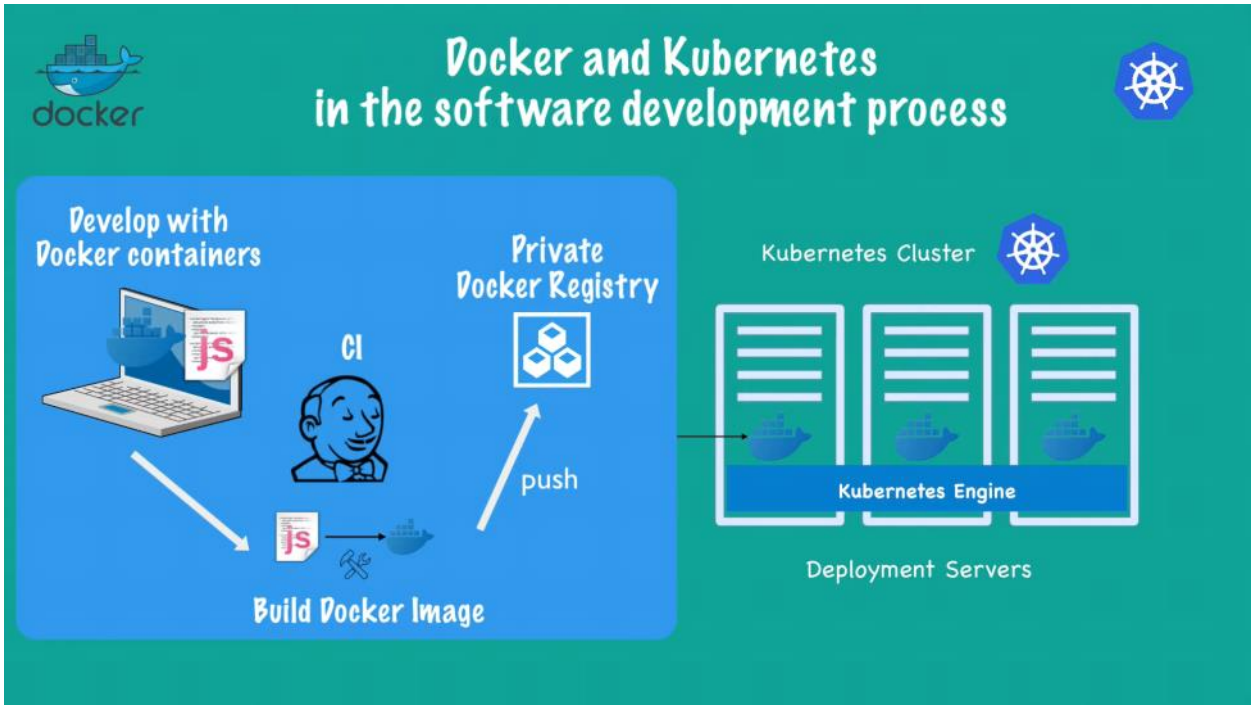
Docker-swarm Introduction

24 April 2023 17:03



Docker: Basically helps to ***create the containers*** , which helps to create an ***isolated environment*** for the application to work on , this helps us to automate the ***building and deploying the applications*** , this is a ***part of continuous integration*** process. We use the container platform for ***configuring , building and distributing*** containers.

Kubernetes: This provides an infrastructure for ***managing multiple containers*** , this does the ***automated scheduling and management of application containers***.

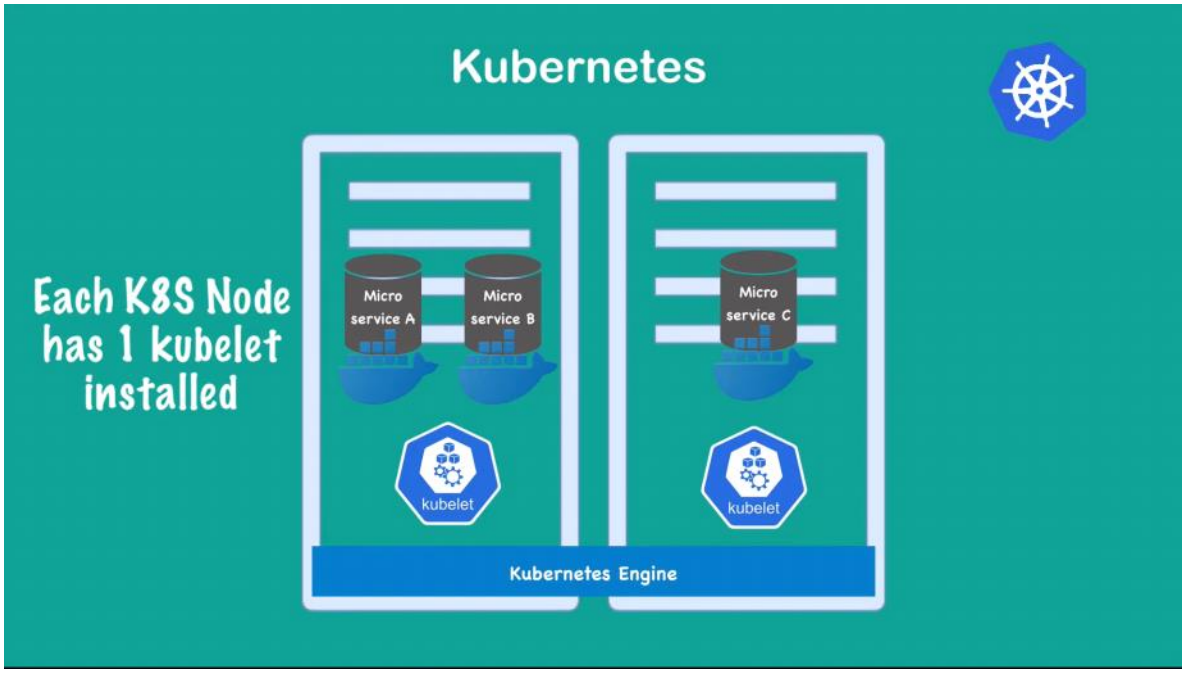


Docker and Kubernetes in the software development process

Part of docker is only up to create an environment in docker image , so that we can run the application in docker containers. This process is continuously Integrated , either we generate docker images dynamically to stay up to date , or we also generate artifacts , which is used to run the application from the container. And store the images inside the private docker registry.

Now Kubernetes use the images , which are published into artifactory and host them on the docker

servers above the Kubernetes engine , which is hosted on deployment servers , on the Kubernetes cluster.

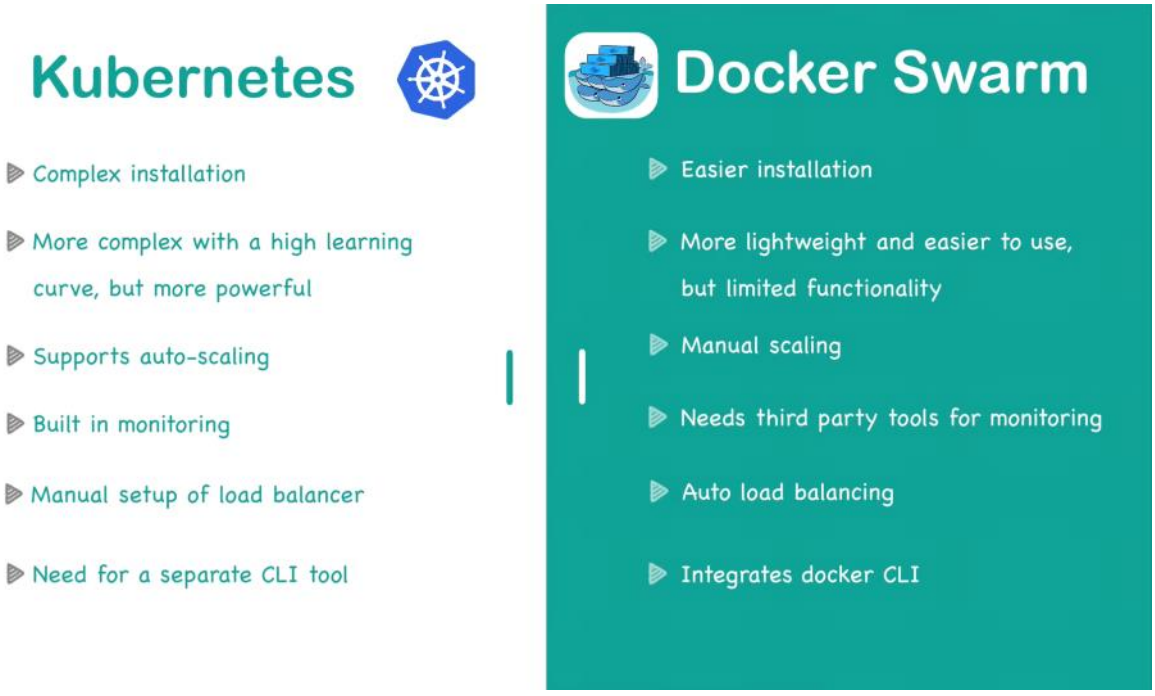


Each Kubernetes Node have 1 kubelet installed on it , which helps to manage the Kubernetes nodes , as the application docker images are hosted on the docker clients which are installed on these individual kube nodes , and these Kubernetes nodes will be created by the Kubernetes engine.

Each docker client have the application docker image running on that , running multiple instance of the same docker image , helps us to load balancing the application , this can be part of scaling up / down.

Docker swarm is also a container orchestration tool , in here , we will be having docker daemons , instead of kubectl. And replacing Kubernetes engine , we will be having docker installed on the deployment servers.

Differences between docker swarm and Kubernetes

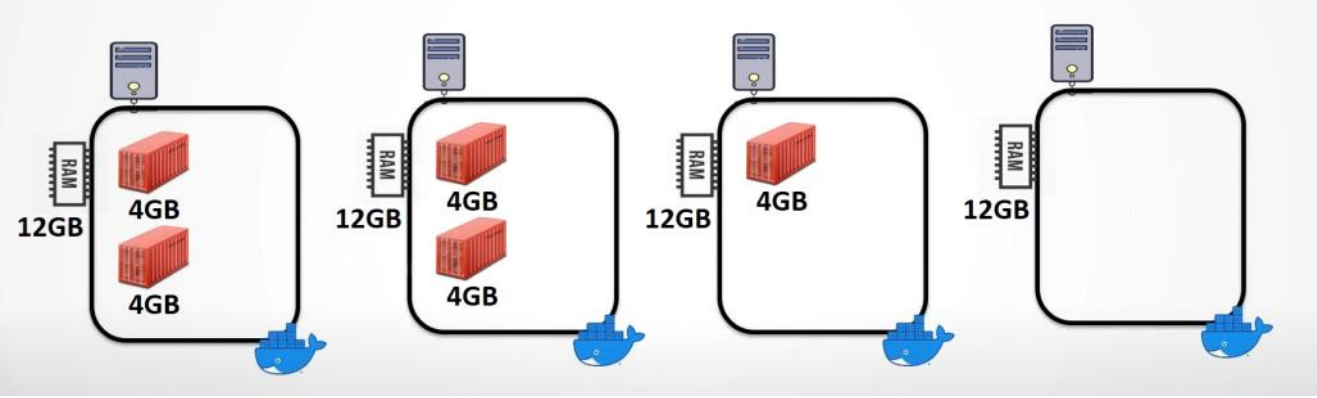


25 April 2023 12:53

Docker-swarm is nothing about orchestrating.

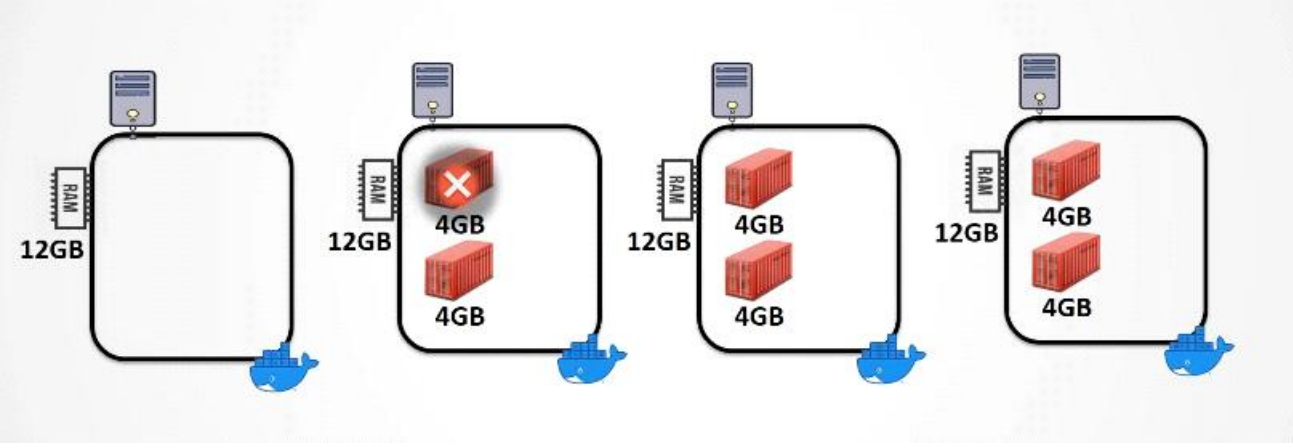
What are the problems that we would face , if we use the regular docker daemon.
When we use docker-compose for the development , we can up n number of containers and manually , monitor , but this isn't a case on QA , Beta , Production , because , there will be many containers to be created , and in there , we couldn't go manually go and perform the creation of the container on all the machines manually. So , we will have all the available systems , on which , we will be having specified number of containers to be installed , the number of containers to be installed in each one of them is indifferent from each other , because , it depends on the space / resource availability of the RAM on the system.

For , example , it would be like this , we need to install five containers , on four system , and we will install in the following way.



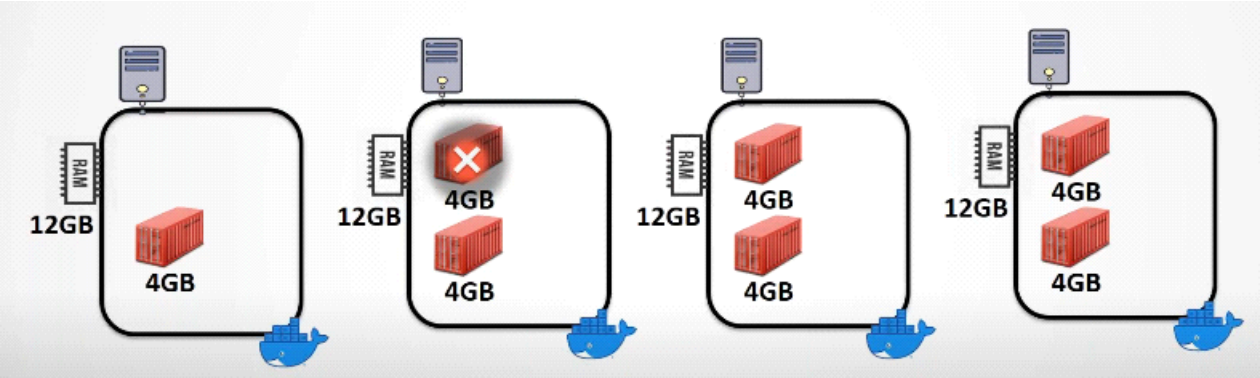
If we does this installation manually , it is very difficult to do , that , so for this purpose , we use the docker swarm for orchestrating the installation of containers. Some cases , where one of our system gets down , we need to shift our lost resources to our rest of our systems. So, in general cases , we must do them manually , but that will be difficult , if we have n number of systems. So , for that , we need an orchestrator , here Kubernetes and docker swarm does the job , it auto scales up and down.

Docker swarm and Kubernetes does the auto scaling and monitoring of the resource nodes, and if any of the resource node is dropped , then the orchestrator does the scaling. And if there are many resource available , then we need to shift the load from the systems.



There will be some cases , where we have our resource nodes , in which containers are running on , and among them , one of the container may not be working properly , so now , we not only monitor the number of containers working on , we also need to check the health status of those containers.

So , for that , these orchestrator's help and add another container , instead of the container , that isn't working.



Suppose , consider we are having multiple containers , running on multiple systems , then it is possible to have multiple containers trying to use the same port , so for maintaining such kind of issues on a system , we use "Load Balancer" , but when coming to the systems (servers) , where we run multiple containers on that , we use "Load balancers" by the Orchestrator.

Why do we want to Container Orchestration System:
Suppose , if we had to run hundreds of containers. We will need from a management angle to make sure that the cluster is up and running.
It supports following options to us ,

1) Process the health checks on the containers , so if it is not sound healthy , we will create one more container based on the resources.
2) Deploying a fixed set of containers , and make sure fixed set of containers , stays up and healthy.
3) If we want to scale the containers either up or down , the orchestrator does that.
4) And if we want to update the containers , all at a time in all the resource nodes , we can do that.
5) It also helps us in load balancing of containers. If we want to up the containers , the orchestrator balances the load , between the many resource nodes of the cluster.

There are two famous orchestration tools available in the market ,

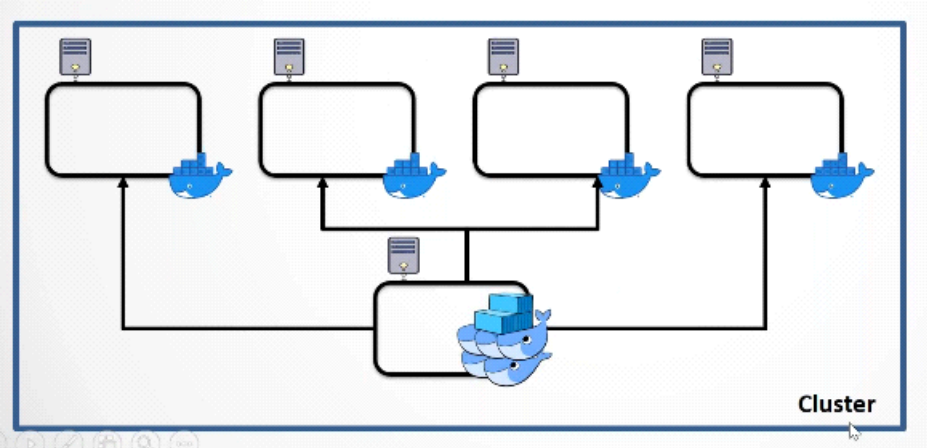
1) Docker Swarm
2) Kubernetes

Docker Swarm comes with default with docker , if we install docker in the machine , with docker daemon , we will get docker swarm , pre-installed.

Docker Swarm is a technique in which we join multiple Docker Engines running on different hosts and use them together as cluster.

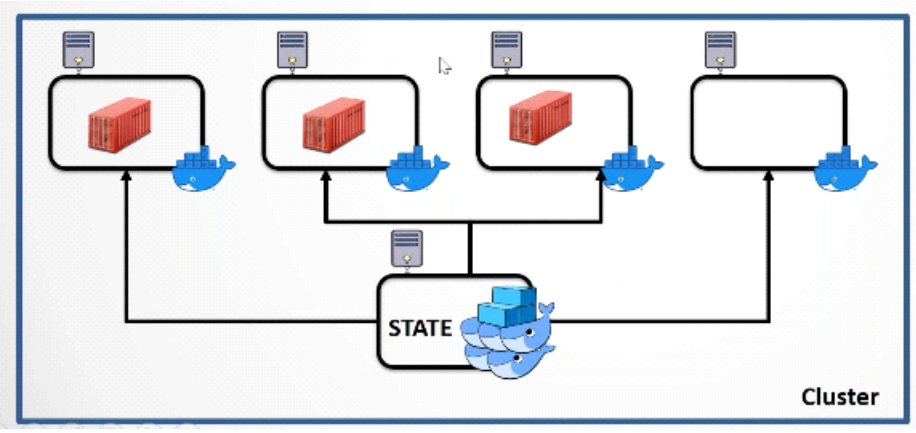
Docker Swarm connects multiple machines in which docker engine is installed.

Docker swarm , manages different systems , and launch the containers as required , docker swarm creates a cluster of systems , where we can use them as the resource and launch containers in them dynamically , as required , that process , is called , upscaling and load balancing. Upscaling and load balancing both are different.



As said , we will have multiple machines in which docker is installed and as said the main disadvantage of native docker management is , we need to go to each and every server and either up or down or create or do whatever operation manually , so to orchestrate that , we are creating a cluster of these individual machines and manage them by some maintaining system , so whatever instruction , like "I need this application to launch 6 containers in the cluster" , we need to tell it to the maintainer system , so this orchestrate the operations in here.

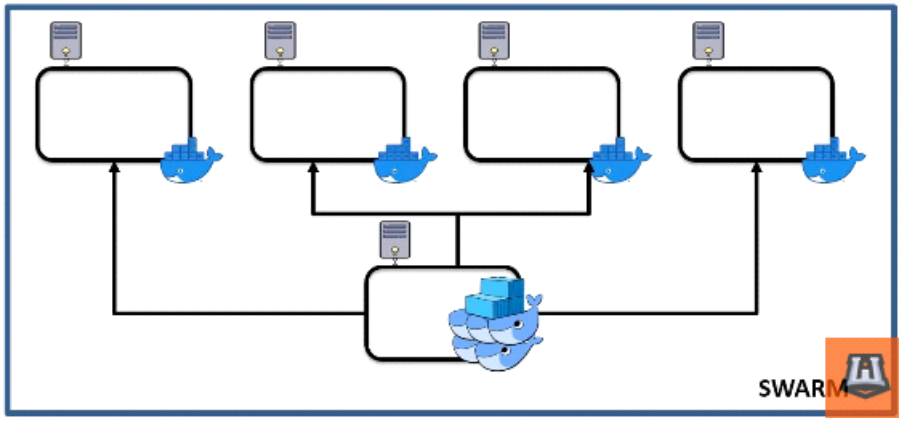
Docker swarm comes by default with docker , but it will be disabled. Until now all these individual machines can't connect to one and other because there is no network established between them , but when we are having docker swarm enabled , we are creating a cluster , so the machines communicate with each other through the cluster as the cluster establishes the network that is required. And the maintainer system that we spoke about is the orchestrator.



As said in the previous example , where our moto is to create 6 containers , and in this case we need to tell the requirement or the state to the , so called , maintainer system , and this maintainer node , which is connected to all other systems through the cluster , manages and creates the containers based on the space availability present under the individual systems in which docker is installed. Docker Swarm does the maintenance.

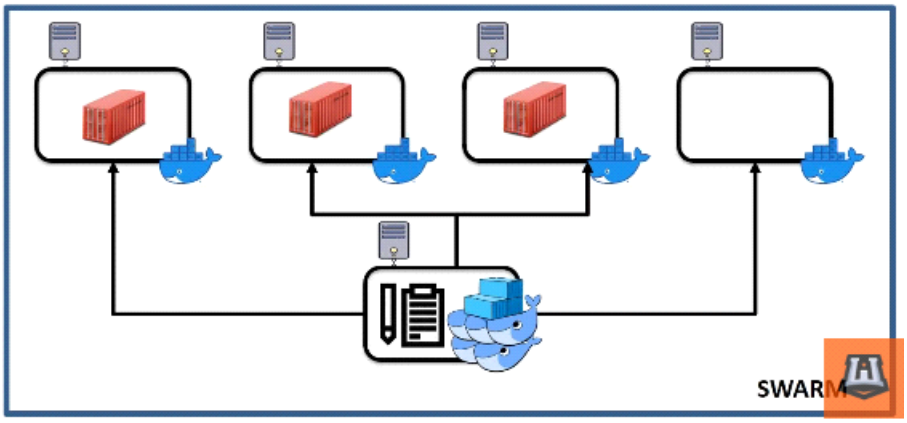
Docker Swarm Components:

- **Node:** A node is an individual Docker Engine participating in the swarm or it can also be said as each and every system on the cluster in which docker engine is installed.

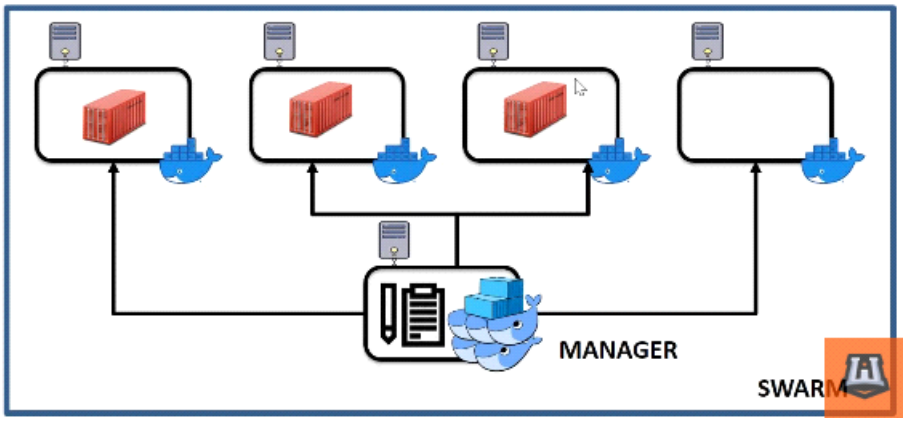


Here in this image , we will be having 5 nodes in total.

- **Swarm:** It is a mode in which we consists of multiple docker hosts which run in a cluster. Swarm is basically a key , in which if we enable it , we are going to use the network among the docker host nodes in the cluster.
- **Service:** A service is the definition of the tasks to execute on the swarm nodes , it is basically the rules that we are going to define in the maintenance node , which pose the directions to its peer nodes in the same network.
    - The following operations could be define in the service on the master node.
        - Which image to use
        - How many containers to run
        - Which commands to execute inside the containers
        - Which port , volumes , network etc. to use



- **Task:** A task carries a Docker Container and the commands to run inside the container. Basically , the service defines number of containers / replicas of the images to be launched as containers , and the task is noting but the work that is happening inside the container. Based on task is running or not , the docker swarm is able to know that , how many replicas are up and running , and if swarm feels , if any of the task isn't running mean , the container is not healthy or not running.
    - Once a task is assigned to a node , it cannot move to another node
    - It can only run on the assigned node or fail.
- **Manager Nodes:** This is the one , which we communicate with in the cluster , we declare the service in here , so , this node manages whatever replicas needed to be created among these available resource , and check the health checks and status of the container ,and even create the containers depending on the resource availability.
    - The main task of the manager node is to assign tasks to other nodes.
    - Manager nodes handle cluster management tasks like maintaining cluster state , scheduling services , and serving swarm mode HTTP API endpoints.
    - Managers maintain a consistent state of the swarm and services running on it.



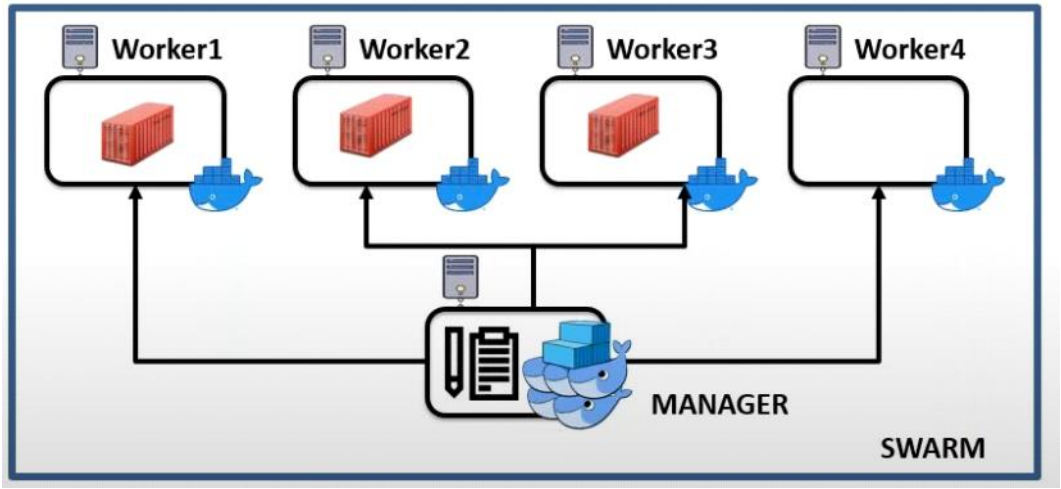
- **Worker nodes** are those where actual containers run , we don't communicate with the worker nodes , we communicate with the manager node , and the manager node provides the instructions to the worker nodes.

Creating Docker Swarm

28 April 2023 00:30

- As said , whatever machine , which is installed with the docker , will also have the docker swarm pre-installed with it. But as we are running in default in standalone mode , we need to enable it manually.
- Docker engine runs with swarm mode disabled by default.
- To run Docker in swarm mode , you can either create a new swarm or have the node join an existing swarm.
- Pre-requisite: We need to have four machines (four servers) , with which docker is installed on them , and then if we check
    - *docker info | grep -i swarm*
    - We would get the status of swarm , if we get swarm as inactive , it shows that swarm is installed with docker , but it is just inactive as all the systems (servers) run in standalone mode.
- **Creating Docker Swarm:**
    - *docker swarm init*
        - This is the command , which initializes the docker swarm on to the system, and the command on which we run this , will be the manager node. And we need to add systems further to this cluster , and the manager node is the one from which we proceed the communication.
        - On the system that , we are running this command , we will see such prompt on that system , this prompt will only be visible on the manager node system.

        - The docker swarm join command is the one , which we must use on the child nodes , so that , we can add new nodes , which are also similar to the child nodes into the cluster , communication for those nodes , happen through the manager node.
        - Here in this command , we can see an ip address , it is the IP address of the manager node.
        - Ensure all the systems are under the same network , so they can or possible to communicate with each other under the same cluster.

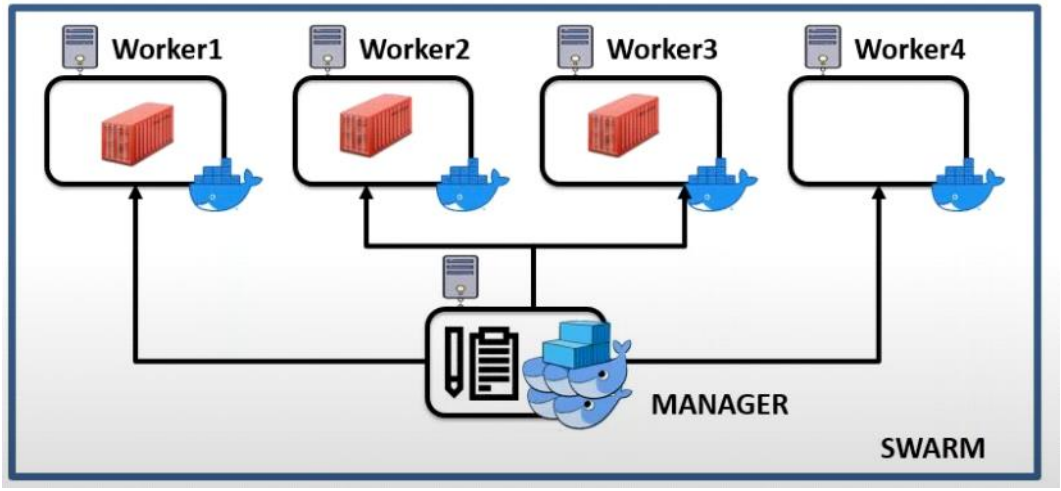


And all these nodes communicate through the manager node , and it will be communicate through specified port , ensure , that we give , enough permissions in the inbound rules.

```
root@ip-172-31-3-68:/home/ubuntu# docker swarm init
Swarm initialized: current node (paewblzss8egii7sonu59qrrq) is now a manager.

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

    docker swarm join --token SWMTKN-1-632dm7njp61ze8my5h43dziwq1cbqz141d75x101fo37whf4jr-3im4xxfcm
6l0ky7otj8bu3q35 172.31.3.68:2377

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

- Here 2377 is the internal port , through which communication proceed.
- Earlier we have seen , in all the systems , the swarm is present but inactive , but now , after adding that system / worker node into the cluster , the docker swarm will become active.

Based on this image , we can say that



We are having a docker swarm cluster , in which we are having three worker nodes which have container deployed in it and one manager node.

Adding a node worker to the swarm cluster:

- docker swarm join --token <TOKEN> <MANAGER-IP>

Viewing the current swarm nodes

- docker node ls

Execute on docker swarm cluster

28 April 2023 11:32

```
root@ip-172-31-3-68:/home/ubuntu# docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER S
TATUS      ENGINE VERSION
paewblzss8egii7sonu59qrrq *   ip-172-31-3-68      Ready               Active              Leader
           19.03.6
rqq5mhzdir23hwq0xm9ox67om     ip-172-31-5-46      Ready               Active
           19.03.6
yxttv47xuwnrt6qy8kl1zzzu9     ip-172-31-8-14      Ready               Active
           19.03.6
vfbcihovh7bq12rxvdk24fgfe     ip-172-31-14-104    Ready               Active
           19.03.6
```

If we wanted to know what are the nodes or systems that are present on the swarm cluster , we needed to use

docker node ls

All the machine details , that are visible under this "docker node ls" are of the same swarm cluster.

We basically , have 4 machines , and in where we initialize the swarm in one of the machine , which will be of leader and where it also generates one command with the token.

docker swarm join --token <TOKEN> <MANAGER-IP>

Which we use it to activate the child nodes , which will be in the control of the manager. We communicate with the manager , if we wanted do any operations on the cluster.

We are going to give the list of commands to run through a service to a cluster.

docker service create --name <NAME> --replicas <#> -p <HP:CP> IMAGE

The above syntax is of imperative method , where we pass the commands through command line.

Where we are also having , another method which is called declarative method , where we pass the commands through a file called docker-compose.yml , to the service.

docker service create --name <NAME> --replicas <#> -p <HP:CP> IMAGE

For example ,
If we want to create a container , based on nginx image , naming the container as the web , which will be mapped , to the port 80:80. We use ,
docker service create -p 80:80 --name web nginx
If we wanted to create the same containers with 3 replicas
docker service create --replicas 3 -p 80:80 --name web nginx