Docker Swarm

Requirements:

- Hosts (2+): Docker Installed

+ Best Practices:

- Have more than one manager node in a single cluster to prevent fault tolerant

- Have one than one manager nodes will be leads to conflict of interest. Solution is to make one of the manager nodes to handle the manager decision (leader). *It does not mean that the leader can decide on its own, it must be agreed by the majority of managers in the cluster.

- Recommended by Docker to have only 7 managers at most.

- Manager nodes should be in odd number: 3,5,7

- If all the nodes master failed at the same time, the worker node still performs its operation as normal but it could not receive any modification or getting new nodes. Solution: try to bring at least one master node back to normal. If one of the 3 master nodes, and 2 are failed to bring online, the only way to recover is to force create new cluster. { $ docker swarm init --force--new--cluster} on node manager that could online (only 1 of 3)

- To promote worker into master, ${$docker node promote}

- Master node can also work as a normal node such as hosting web by default. We can also disable it to only make it handle management tasks only with cmd {$docker node update --availability drain <Node>}.

docker swarm init

Host:

$both_node: docker node ls: list of nodes in a cluster

$worker_node: worker node: docker swarm leave = leave cluster as a node worker

$master_node: docker node rm {hostname} = remove a worker node out of cluster

$master_node: docker swarm join-token manager = generate a manager token from a leader node for promote other to a manager

$master_node: docker swarm join-token worker = generate a manager token for a node to become a worker in a cluster

$master_node: docker node promote {hostname}

Demonstrate:

There are 3 manager nodes in the cluster, if 2 of the 3 are down, the cluster will down too because it does not meet the quorum.

Solution #1: Try to bring the down manager back to online

Solution #2: Try to create a new cluster with the existing online node manager with cmd:

> docker swarm init --force--new--cluster
>
> Docker Service: a service is a definition for the tasks to be executed on the nodes of a Docker Swarm. Docker Swarm is a clustering
>
> and scheduling tool for Docker containers.
>
> $master_node: docker service create --replicas=3 --name web_service my-web-server = create 3 instances of
>
> my-web-server and distribute across worker nodes
>
> $master_node: docker service update --replicas=4 web_service = when create a new worker node and update
>
> instances of service to 4.
>
> -optional = -p 8080:80, --network frontend

***Replica vs Global

$master_node: docker service create --mode global monitoring = create an instance of service to every worker nodes.

Docker service demo

Swarm clustor info:

+ 1 master node

+ 2 worker nodes

+ create service from nginx

$master: docker service create nginx

$master: docker service ps == to check which node is disire to run

$worker_node2: docker ps = to check whether the service is up or no

*let assume that we run a nginx but when we create a service, we forgot to expose the port mapping:

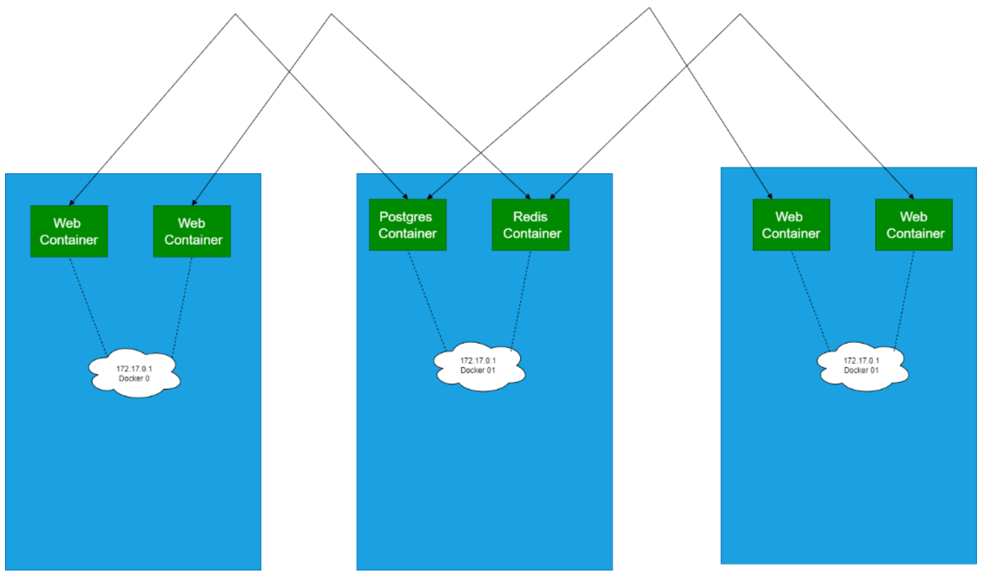
$master: docker service update $service_id --publish--add 5000:80

Docker network:

3 types of networks:

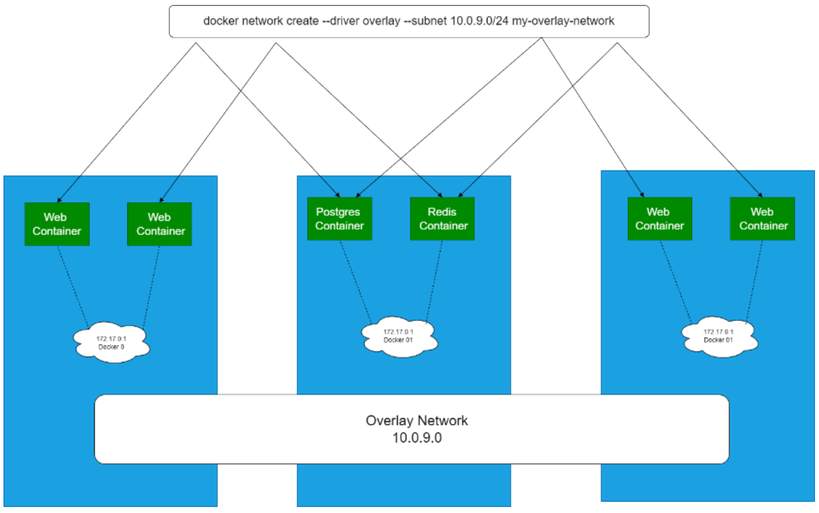
-Bridge = is a default one that is a private internal network on the host. To connect the container from the outside world, we must map the port container to the host on the docker host. ex: [-p 8080:80].

For example, let's say we have multiple Docker hosts running containers. Each Docker host has its own internal private Bridge network in the 172.17 series allowing the containers running on each host to communicate with each other. However, containers across the host have no way of communicating with each other unless you publish the ports on those containers and set up some kind of routing yourself.



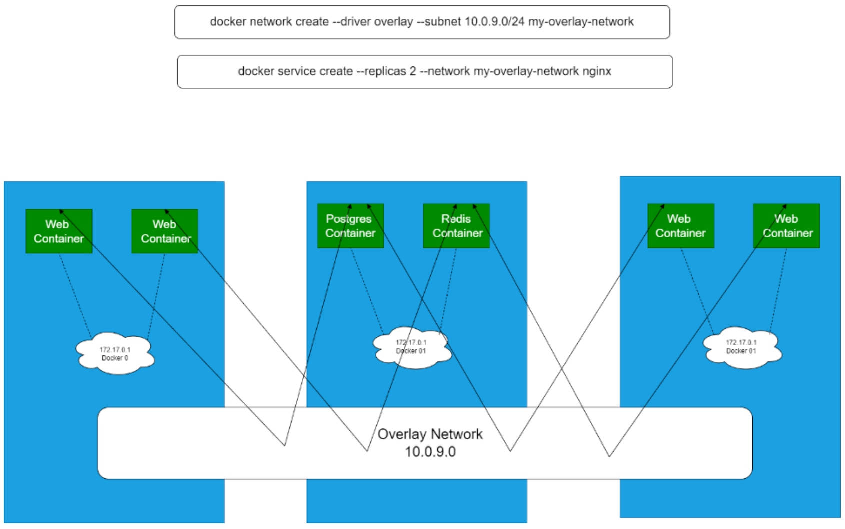
Solution: Docker Network Overlay

$ docker network create --driver overlay --subnet 10.0.9.0/24 my-overlay-network = It will create an internal private network that spread across all the nodes in the swarm cluster.



Then we can attach the containers or services to this network using:

$ docker service create --replicas 2 --network my-overlay-network nginx



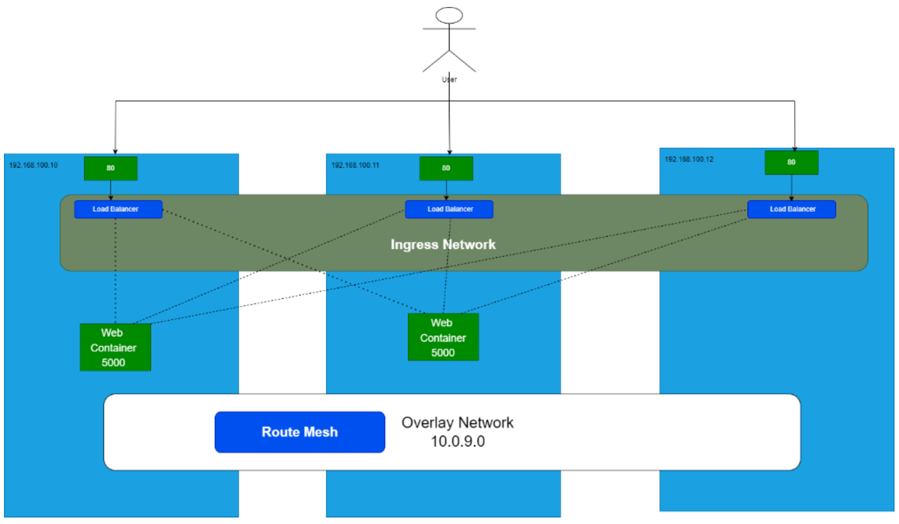
Docker Ingress Network (Type of Overlay Network)

When using single container, we usually map port {-p 80:5000} and it works fine. However, what will happen if we use it in swarm cluster?

Let's assume that we only have 1 node in the cluster, and we create a service with 2 replicas of web-service that mapped to port 80:5000. In a single node, we cannot map the same port.

By default, ingress network is created when we create docker swarm. It has a built-in load balancer that redirect traffic port to all mapped ports on each container, so we don't have to config ingress network if we only have 1 node in a single cluster.

In another case, let's say we have 3 nodes in the single cluster, and we only create 2 replicas which is on node 1 and node 2. *Node 3 has no instance in this case. If we use access to container via ip address of the node 192.168.100.11:80 and 192.168.100.12:80, it works fine, but if the use access to the node 3 192.168.100.12:80 that has no instance, it will not response, so all of this are default configuration of docker swam and we don't need to do anything.



What we will do:

- Create service
- Define number of replicas
- Publish the port