Docker Swarm: 18-12-2019

# Docker Swarm:

It is a cluster based orchestrator

Current docker application can be moved to swarm mode by running docker swarm –init –advertise-addr <MANAGER-IP> 🡪 The machine will act as a manager in swarm mode.

## To add workers under manager:

Docker swarm join –token <TOKEN> <IPADDRESS: PORT>

## To get the token for adding either manager or worker

For manager:

Docker swarm join-token manager

For worker:

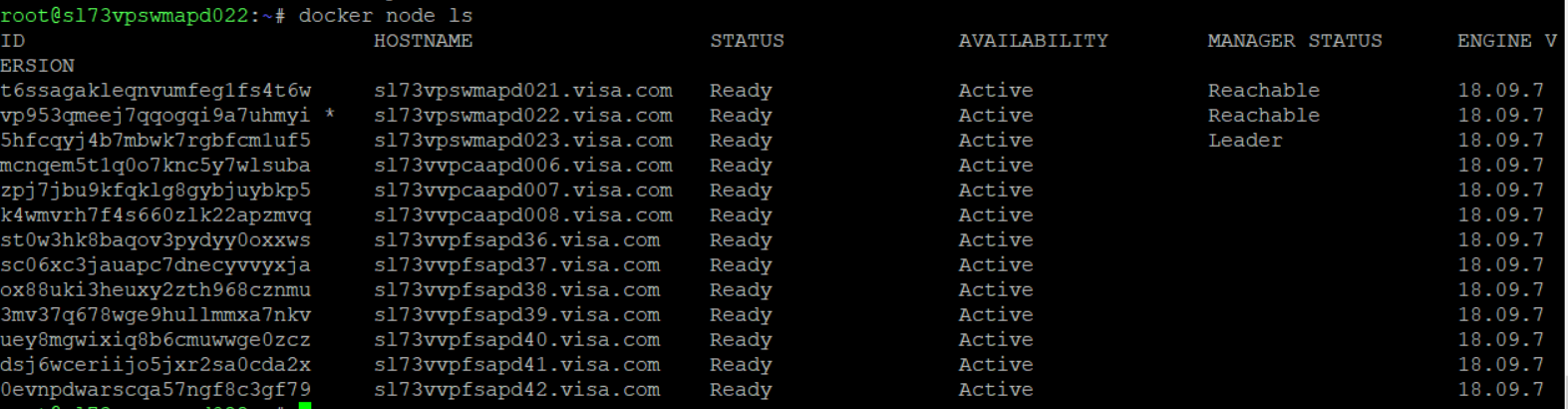
Docker swarm join-token worker

Docker info:

Gives the information about the current swarm.

Docker node ls:

To view information about the nodes.



## To deploy a service to swarm:

Docker service create –replicas 2 –name helloworld Ubuntu ping google.com

Docker service create 🡪 command to create the service

Replicas 🡪 number of instances

Name 🡪 Name of the service

Run ping google.com on Ubuntu machine

## To inspect a service on the swarm

Docker service inspect –pretty <name of the service>

Note: without –pretty it gives output in the form of json.

## To check which node is running the service:

Docker service ps helloworld

## To change the service scale in swarm:

Docker service scale <name of the service > = <number of tasks>

Eg : docker service scale helloworld=5

## Deleting a service from swarm:

Docker service rm <name of the service >

Eg: docker service rm helloworld

## Updates to images/application:

Docker service update –image redis:3.0.7 redis

Above command will update the container image of redis.

## Drain a node during maintenance:

Docker node update –availability drain <name of the node>

## Routing Mesh (Ingress network):

Docker swarm is made it easy to publish a ports for make them available outside swarm.

--publish 🡪 to create a port to publish.

--target 🡪 port inside the container.

Docker service create –name <SERVICE NAME > --publish published=<PUBLISHING PORT> --target=<port inside the container> <IMAGE>

--publish is equivalent to –p 8080:80

If you omit providing publish port, docker swarm picks some high value number by itlsef. Later you can get that high value port number via inspect command.

Note:

When you access port 8080 from any node, Docker routes you to active container. Routing Mesh takes care of routing without creating any traffic or port conflict.

## We can publish a port for running service via:

Docker service update –publish-add published=<PORT NUMBER> , target=<CONATINER PORT> <service name>

## To view running service’s published port:

Docker service inspect –format=”{{json .Endpoint.Spec.Ports}}” <SERVICE NAME>

## Publish a port for TCP or UDP:

**TBD**

## Docker Cluster:

Options:

* --dry-run - Skip provisioning resources
* --log-level warn|error|info|debug etc

## Backup a running cluster:

Docker cluster backup [option] cluster.

Eg: Docker cluster backup –file backup.tar.gz cluster

## To Create a Docker cluster:

Docker cluster create –name Dayananda\_Cluster

## Docker Cluster Inspect:

Docker cluster inspect [options] Dayananda\_Cluster

Eg:Docker cluster inspect –all Dayananda\_Cluster

## List all the Available Clusters:

Docker cluster ls

## Cluster Restore:

Docker cluster restore –file backup.tar.gz new\_Dayananda\_Cluster

## Remove a cluster:

Docker cluster rm [options] Dayananda\_Cluster

Eg : Docker cluster –force Dayananda\_Cluster

## Update a running cluster’s state:

Docker cluster update [options] Dayananda\_Cluster

Eg : Docker cluster update –file Daya\_cluster\_defn Dayananda\_Cluster

## Docker Cluster Version:

Docker cluster version

# Docker Networking:

* Portability
* Extensibility

## Overview:

* Container Network Management <🡪 LibNetwork.
* LibNetwork is scalable and pluggable foe other network drives as well
* The CNM is an open-source container networking specification contributed to the community by Docker Inc.
* Libnetwork is Docker’s implementation of the CNM.
* Libnetwork is extensible via pluggable drives.
* Libnetwork is opensource.
* CNM defines sandbox, endpoints and networks.

## To check all the available network drives:

Docker info

You can see all the available network drives under plugins section.

Plugins:

Volume: Local

Network: bridge host ipvlan macvlan null overlay (Native drives)

## Creating our own network:

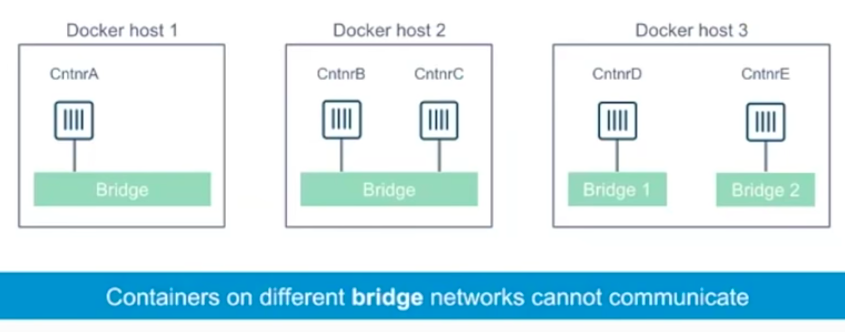
Docker network create –d <driver> <Name of the network>

Docker run –network <Name of the network>

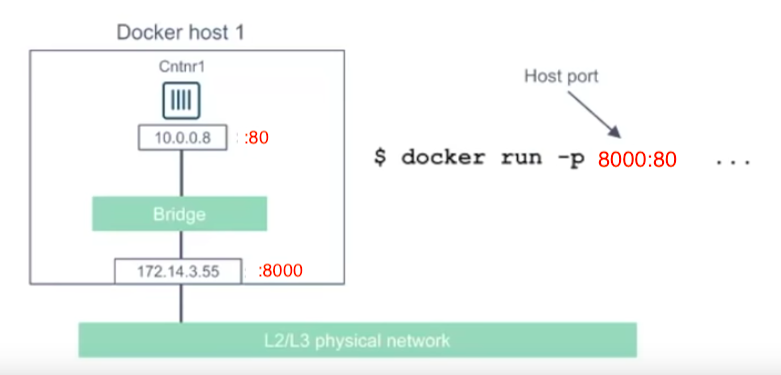
Attach the containers of your choice to this network.

## Bridge Drive:

* It is the default driver.
* Single Host networking.
* Useful for local development.
* Very robust and used in many production deployments.
* Bridge is a private network restricted to a single Docker host.

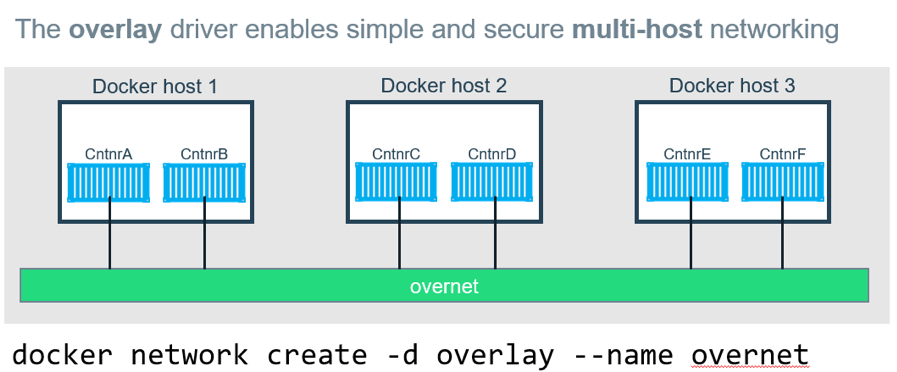


To overcome from restricting to a single Docker host, we expose the service via port mapping



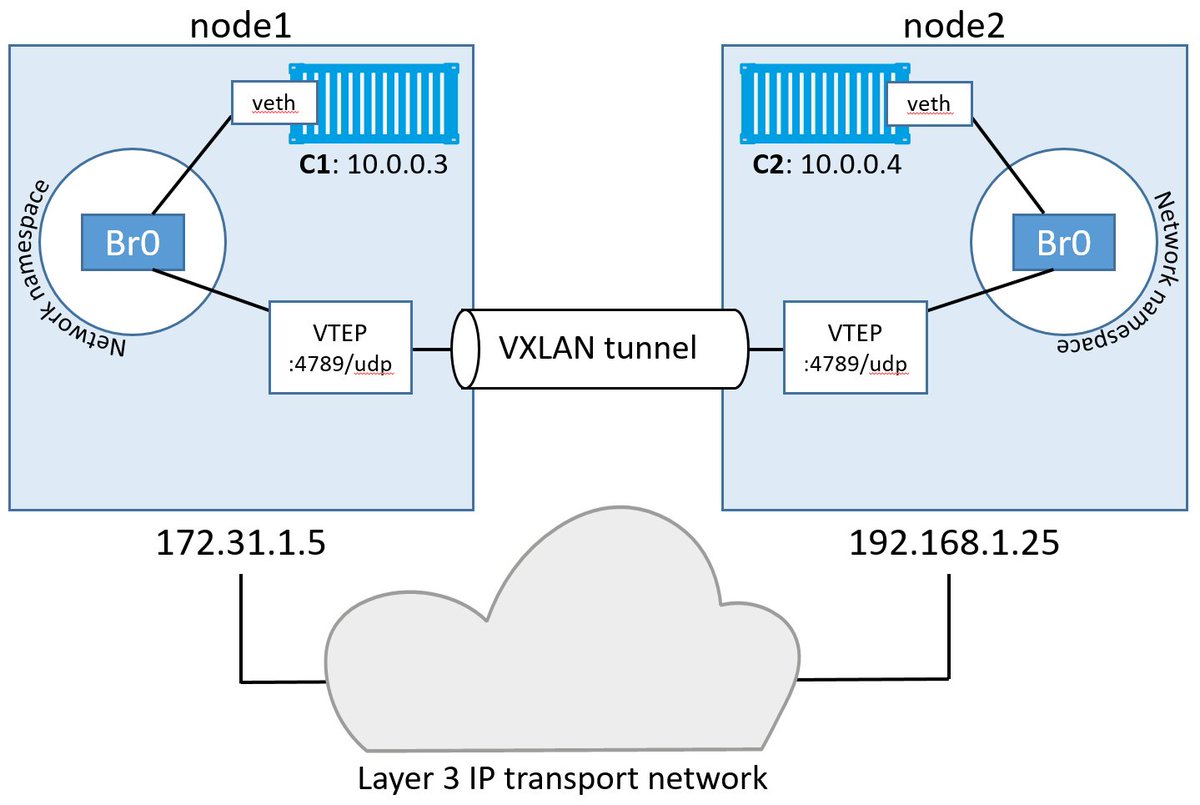
## Overlay Networking:

Overlay driver enables simple and secure multi-host networking.



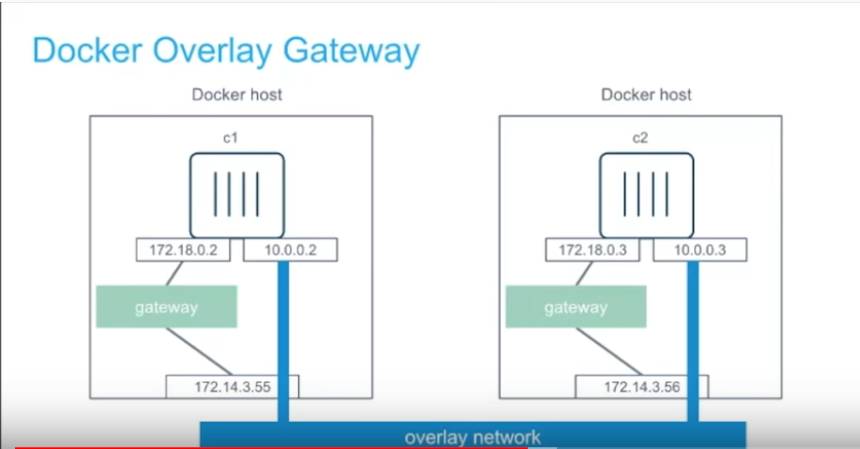
* Portable
  + Works in any cloud/on premise environments with little reconfiguration of existing physical network.
* Simple to setup
  + Docker handles all control plane configuration and management.
  + Network self-configuration.
* Secure
  + Control plane is encrypted by default.
  + Data traffic between containers is encrypted.

## Docker Overlay Networks and VXLAN



* The Overlay driver uses VXLAN technology to build the network.
* A VXLAN tunnel is created through the underlay network.
* At each end of the tunnel is a “VXLAN tunnel end point (VTEP)”
* VTEP performs encapsulation and de-encapsulation.
* The container doesn’t know it’s an overlay.

The picture bellows tells us that for each container there will be two IPs associated one IP for communicating internally on the same host and another IP for communicating with another host via overlay network.



Separation of Networks:

We have mechanism to divert the traffic between the interfaces during docker swarm init

Docker swarm intit –advertise-addr eth0 –data-path-addr eth1

## MACVLAN Driver:

TBD

# Docker Base Commands:

## Docker attach:

To attach stdin.stderr or stdout to a running container

Docker run –d –name topdemo Ubuntu /usr/bin/top –b

Docker attach topdemo

This displays top command output on the console.

## Docker build:

Docker build [options ] path|url

Options and use cases:

To limit number of cpus for a running container:

* Docker build –cpuset-cpus=”0-2” redis:latest

Fractional limit on CPUs:

* Docker run –cpu 2.5 redis:latest
* 2.5 % cores on the host.

To specify the file location of Dockerfile

Docker build –t myapp –filename /root/daya/dockerfile

Always remove intermediate containers

Docker build . –t daya –force-rm

Memory limit

Docker build –t daya . –memory 1g

Network

Docker build –t daya –network brigde|overlay|Custom etc

Tag

Docker build –t daya –f dockerfile –tag “example:one”

## Docker builder prune:

Docker builder prune

--all deletes all dangling images including unused images

--filter provide filter like –unused-for 24h

--force do not prompt for confirmation

## Commit a container changes to a new image:

Docker commit <container id> <new-imagename>

Commit with some changes in EXPOSE and CMD

Docker commit –change=’CMD [“apache”,”DFOREGROUND”]’ –c “EXPOSE 80” <container id> <new-image\_name>

Docker container:

Committing container changes to a new image:

Docker container commit [options] CONTAINER [Repository:[TAG]]

Options:

--author : [daydr@visa.com](mailto:daydr@visa.com)

--message: commit message

--pause True : Pause container during commit

Copying files from host to container and vice versa

Docker container cp containerId:sourcepath destination path

Docker cp source path destination path

Docker create:

Docker run = docker create + docker start

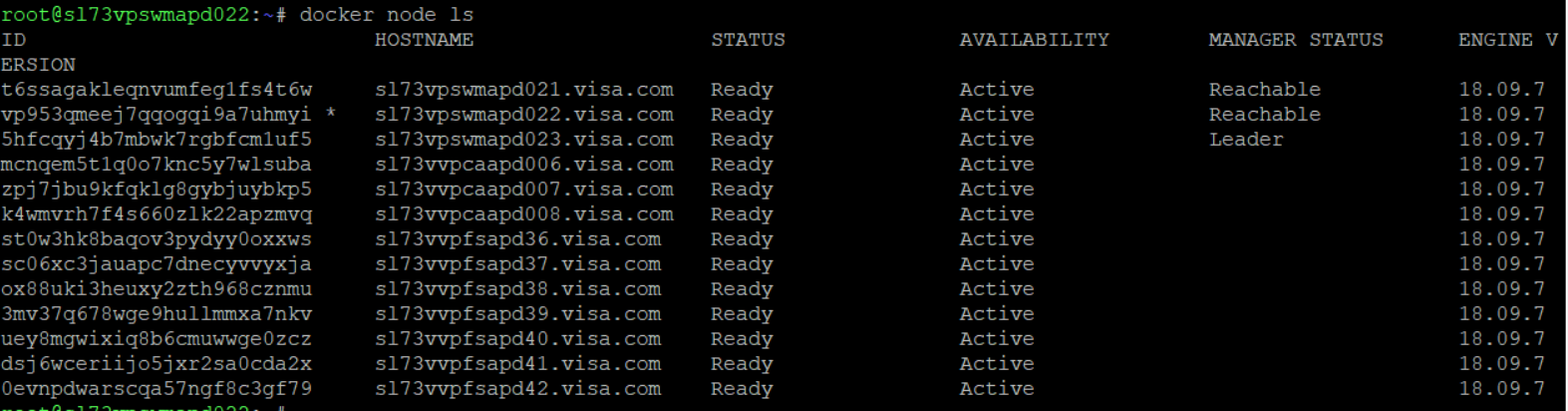
Docker Exec:

Docker exec –it <container id>

# Most Used Commands in VISA:

## Docker node ls

To list all the nodes in a swarm



## Docker service ps auth\_BLR-DAYANANDA-DEMO2

To check on which server particular service is running

