**Basics of Docker Networking**

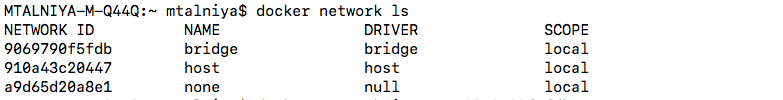
Docker Networking allows you to create a Network of Docker Containers managed by a master node called the manager. Containers inside the Docker Network can talk to each other by sharing packets of information.

Docker networking utilizes already existing Linux Kernel Networking features like (iptables, namespaces, bridges etc.).

With Docker Networking, we can **connect various docker images**running on same host or across multiple hosts.

**By default, three network modes are active in Docker.**

1. **Bridge**
2. **Host**
3. **Null**



To extract more information about each of the networks:



## 1. Host Networking:

Shares TCP/IP stack, namespace etc of host OS. That is, all of the network interfaces defined on the host will be accessible to the container. Below command connects the microservice docker image to the host network.

$ docker run --net=host microservice-demo

## 2. Bridge Network:

**Bridge Network driver provides single host networking capabilities.**By default containers connect to Bridge Network. Whenever container starts, it is provided an internal IP address. All the containers connected to the internal bridge can now communicate with one another. But they can’t communicate outside the bridge network.

With -p flag however, we can map the docker port to the native port.

$ docker run -p 4000:80 microservice-demo

**Both Host and Bridge Networking are contained within a single host.**

## 3. Overlay Networking:

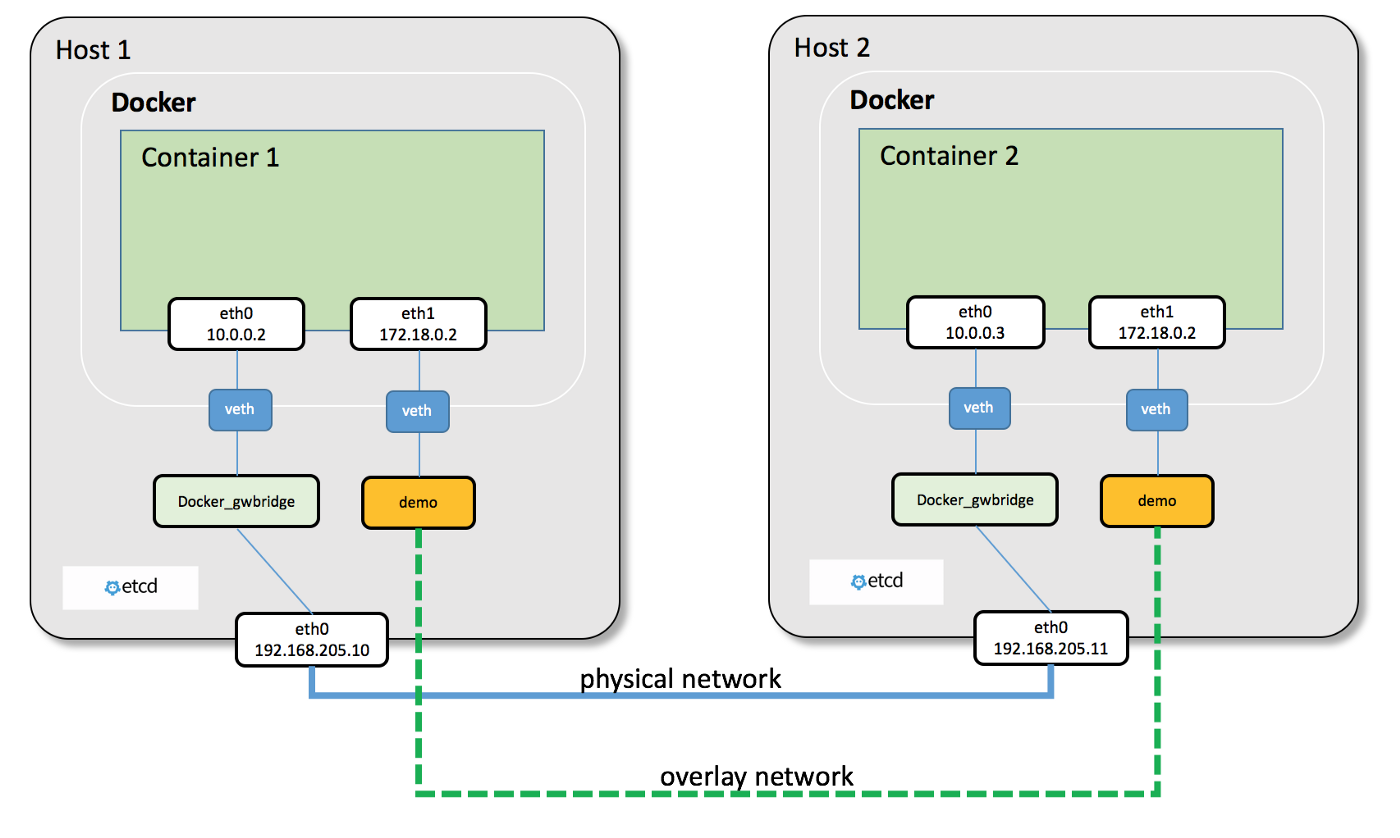
**Overlay Networking provides simple and secure multi host networking.**The overlay network makes use of **VXLAN** over underlying network.

## [VXLAN Overview: Cisco Nexus 9000 Series Switches](https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-729383.html" \t "_blank)

### [What You Will Learn Traditional network segmentation has been provided by VLANs that are standardized under the IEEE…](https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-729383.html" \t "_blank)

[www.cisco.com](https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-729383.html" \t "_blank)

Containers that are part of Overlay Network can communicate with containers regardless of the host. **Containers part of Overlay network see each other if they are on same L2 network.**



<http://docker-k8s-lab.readthedocs.io/en/latest/_images/docker-overlay.png>

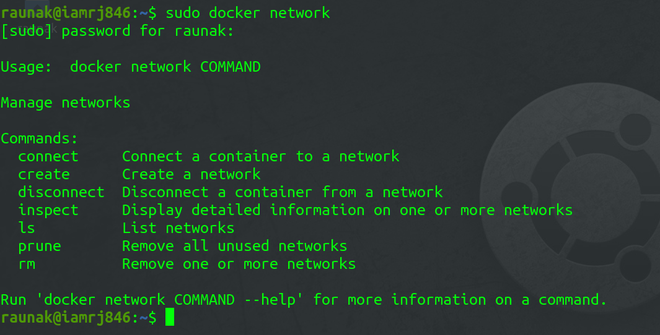
**Each container in the Overlay network receives two IP address.**

First IP allows for communication between the containers across host. While second IP maps to **VXLAN-VTEP**endpoint and contains all the actual data between hosts.

**1. Understanding the Docker Network Command**

The Docker **Network** command is the main command that would allow you to create, manage, and configure your Docker Network. Let’s see what are the sub-commands that can be used with the Docker Network command.

sudo docker network

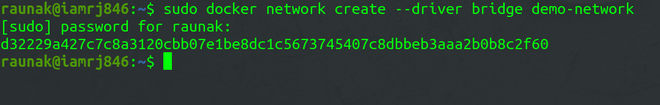


We will see all the Network sub-commands one by one.

**2. Using Docker Network Create sub-command**

The **Create** sub-command allows you to create a Docker Network.

sudo docker network create --driver <driver-name> <bridge-name>



**3. Using Docker Network Connect sub-command**

Using **Connect** sub-command, you can connect a running Docker Container to an existing Network.

sudo docker network connect <network-name> <container-name or id>

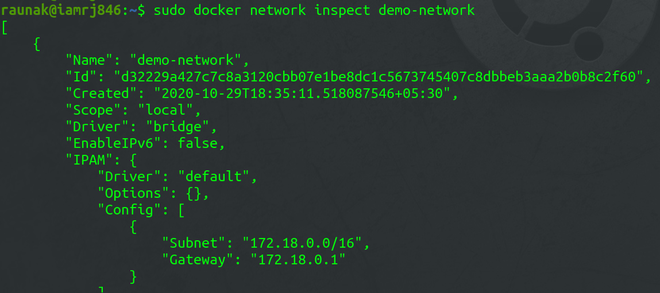
In this example, we will connect an Ubuntu Container to the Bridge Network we created in the last step.

Docker Network Connect sub-command

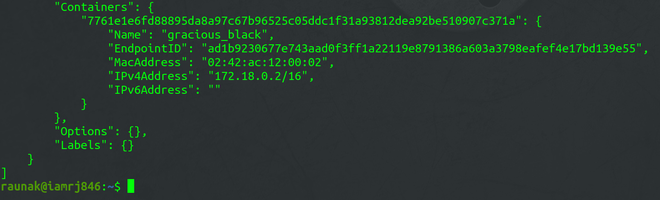
**4. Using Docker Network Inspect sub-command**

Using the Network Inspect command, you can find out the details of a Docker Network.

sudo docker network inspect <network-name>



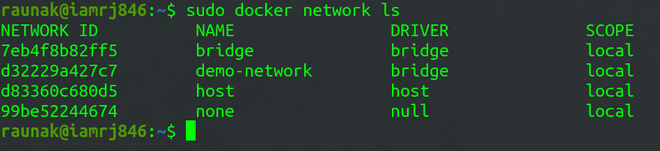
You can also find the list of Containers that are connected to the Network.



**5. Using Docker Network ls sub-command**

To list all the Docker Networks, you can use the **list** sub-command.

sudo docker network ls



**6. Using Docker Network Disconnect sub-command**

The **disconnect** sub-command can be used to remove a Container from the Network.

sudo docker network disconnect <network-name> <container-name>

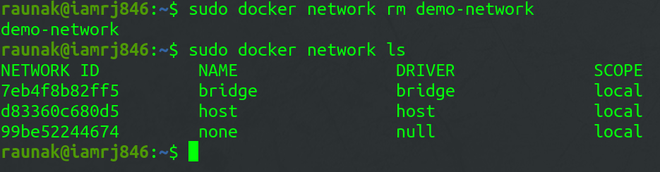
Docker Network Disconnect sub-command

**7. Using Docker Network rm sub-command**

You can remove a Docker Network using the **rm** sub-command.

sudo docker network rm <network-name>

Note that if you want to remove a network, you need to make sure that no container is currently referencing the network.

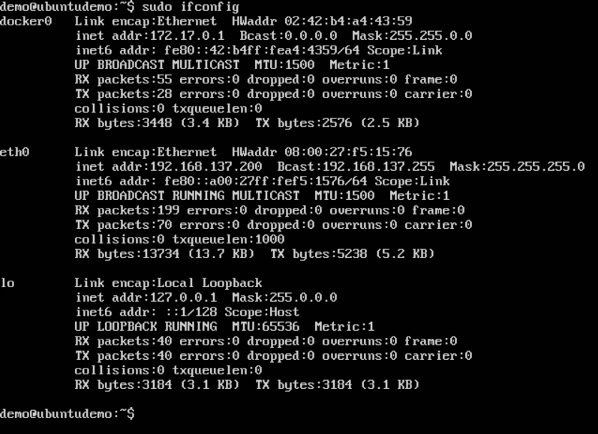


**8. Using Docker Network prune sub-command**

To remove all the unused Docker Networks, you can use the **prune** sub-command.

sudo docker network prune

Docker takes care of the networking aspects so that the containers can communicate with other containers and also with the Docker Host. If you do an **ifconfig** on the Docker Host, you will see the Docker Ethernet adapter. This adapter is created when Docker is installed on the Docker Host.



This is a bridge between the Docker Host and the Linux Host. Now let’s look at some commands associated with networking in Docker.

## Listing All Docker Networks

This command can be used to list all the networks associated with Docker on the host.

### Syntax

docker network ls

### Options

None

### Return Value

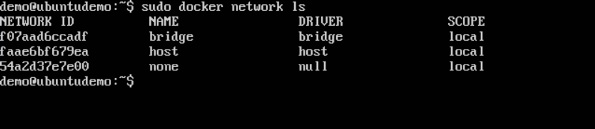
The command will output all the networks on the Docker Host.

### Example

sudo docker network ls

### Output

The output of the above command is shown below



## Inspecting a Docker network

If you want to see more details on the network associated with Docker, you can use the Docker **network inspect** command.

### Syntax

docker network inspect networkname

### Options

* **networkname** − This is the name of the network you need to inspect.

### Return Value

The command will output all the details about the network.

### Example

sudo docker network inspect bridge

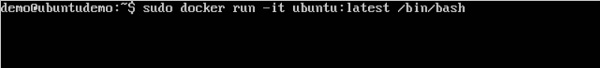
### Output

The output of the above command is shown below −



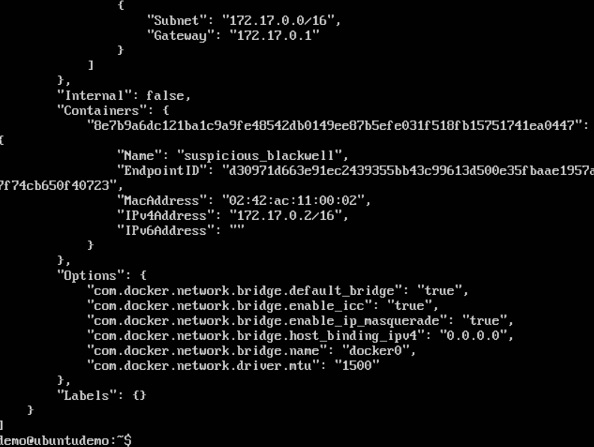
Now let’s run a container and see what happens when we inspect the network again. Let’s spin up an Ubuntu container with the following command −

sudo docker run –it ubuntu:latest /bin/bash



Now if we inspect our network name via the following command, you will now see that the container is attached to the bridge.

sudo docker network inspect bridge



## Creating Your Own New Network

One can create a network in Docker before launching containers. This can be done with the following command −

### Syntax

docker network create –-driver drivername name

### Options

* **drivername** − This is the name used for the network driver.
* **name** − This is the name given to the network.

### Return Value

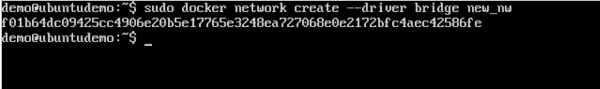
The command will output the long ID for the new network.

### Example

sudo docker network create –-driver bridge new\_nw

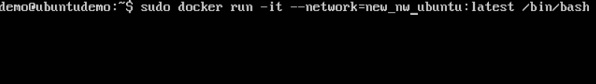
### Output

The output of the above command is shown below −



You can now attach the new network when launching the container. So let’s spin up an Ubuntu container with the following command −

sudo docker run –it –network=new\_nw ubuntu:latest /bin/bash



And now when you inspect the network via the following command, you will see the container attached to the network.

sudo docker network inspect new\_nw

