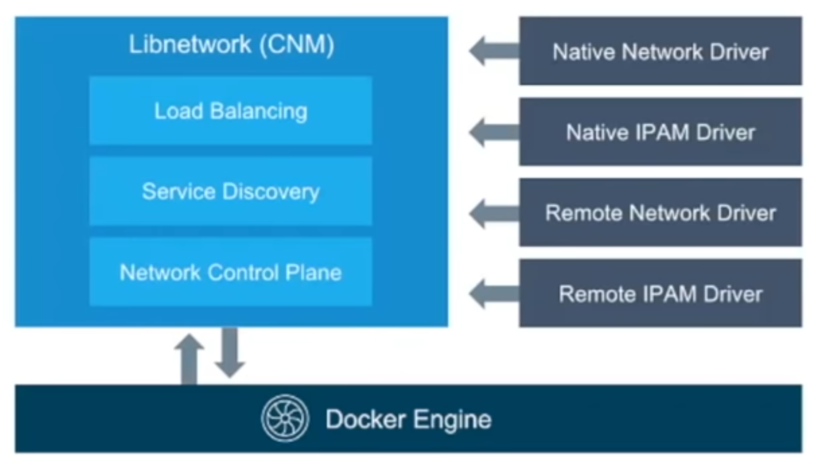
**Docker Networking**:- **Docker networking** is basically used to establish communication between the docker containers and the outside world via host machine or you can say it is a communication passage through which all the isolated containers communicate with each other in various situations to perform the required actions.

Three pillars of docker networking

CNM (container network model) Libnework() Drivers



**CNM**- it’s documentation of networking.

**Libnetwork**- it’s implementation of CNM.

**Driver(s)-** Protocol implementation, such as NAT driver.

**Note-** 1) Whenever docker client want to communicate they use Libnetwork concept. 2) Volume mapping is possible because of Libnetwork.

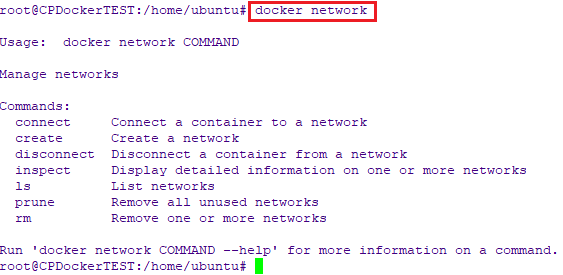
**How to check connectivity between host?** – Ping command.

**Explaining Docker Networking Concepts**

All commands listed below are tested with **root** privileges on **Ubuntu**.

To manage network operations, like creating a new network, connecting a container to a network, disconnect a container from the network, listing available networks and removing networks etc., we use the following command:

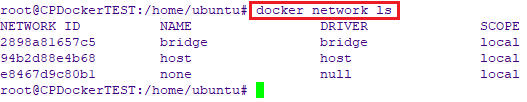
# docker network

[](https://www.ostechnix.com/wp-content/uploads/2019/10/docker-network-command.png)

**Types of docker network drivers**

To list all your networks, run:

# docker network ls

[](https://www.ostechnix.com/wp-content/uploads/2019/10/List-docker-networks.png)

Let’s have some short introduction on all of them.

1. **Bridge network:** When you start Docker, a default bridge network is created automatically. A newly started containers will connect automatically to it. You can also create user-defined custom bridge networks. User-defined bridge networks are superior to the default bridge network.
2. **Host network:** It remove network isolation between the container and the Docker host, and use the host’s networking directly. If you run a container which binds to port 80 and you use host networking, the container’s application is available on port 80 on the host’s IP address. Means you will not be able to run multiple web containers on the same host, on the same port as the port is now common to all containers in the host network.
3. **None network:** In this kind of network, containers are not attached to any network and do not have any access to the external network or other containers. So, this network is used when you want to completely disable the networking stack on a container.
4. **Overlay network:** Creates an internal private network that spans across all the nodes participating in the swarm cluster. So, overlay networks facilitate communication between a docker swarm service and a standalone container, or between two standalone containers on different Docker Daemons.
5. **Macvlan network :** Some applications, especially legacy applications or applications which monitor network traffic, expect to be directly connected to the physical network. In this type of situation, you can use the Macvlan network driver to assign a MAC address to each container’s virtual network interface, making it appear to be a physical network interface directly connected to the physical network.

**Casestudy: Enabling communication between container in single host**

root@ip-172-31-23-238:~# docker run -d --name c1 --network mynet jenkins sleep 1d 07c7122bb75528efd9468a5e5268bc250b362be68dec953547f120b8e697716f

root@ip-172-31-23-238:~# docker run -d --name c2 --network mynet jenkins sleep 1d cd6168d5fd4cc6e2c3c987ac252ca323e11c559ca37e4552b6e6c02593109c3b

root@ip-172-31-23-238:~# docker exec -it p1 /bin/bash

jenkins@07c7122bb755:/$ ping p2 Unable to connect

root@ip-172-31-23-238:~# docker network create mynet 1ddc99f170ac1b2acc6e12be97475533a0e41b4c0dd12d04213b12bebffee4c8

root@ip-172-31-23-238:~# docker run -d --name p1 --network mynet jenkins sleep 1d 07c7122bb75528efd9468a5e5268bc250b362be68dec953547f120b8e697716f

root@ip-172-31-23-238:~# docker run -d --name p2 --network mynet jenkins sleep 1d cd6168d5fd4cc6e2c3c987ac252ca323e11c559ca37e4552b6e6c02593109c3b

root@ip-172-31-23-238:~# docker exec -it p1 /bin/bash

jenkins@07c7122bb755:/$ ping p2 PING p2 (172.18.0.3) 56(84) bytes of data.

64 bytes from p2.mynet (172.18.0.3): icmp\_seq=1 ttl=64 time=0.066 ms 64 bytes from p2.mynet (172.18.0.3): icmp\_seq=2 ttl=64 time=0.062 ms 64 bytes from p2.mynet (172.18.0.3): icmp\_seq=3 ttl=64 time=0.060 ms

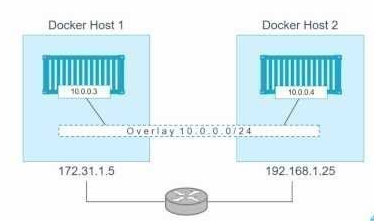
**Why we need to enable communication by name:** This concept is called service discover by name, if we do such things then it will be easy for us to work with driver.

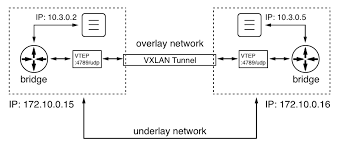
Multihost networking: in Multihost networking we are managing containers on serval machine.

Prework: effected system needs to have ping connectivity.

**Driver name for single host networking**: Bridge.

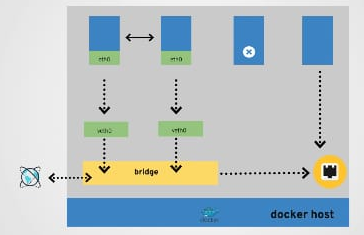
**Driver name for Multihost networking**: overlay.



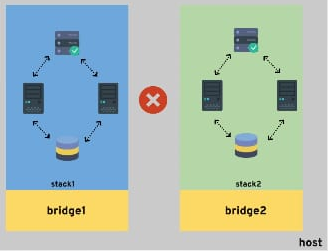


docker swarm: docker Kubernetes: google.

Master salve communication Docker swarm init, and docker swarn join.



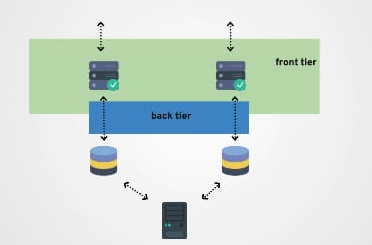
In above diagram, two containers are communicating as they are connected by bridge network, container can use host networking, and no networking.



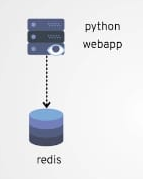
Connecting to microservices by Docker

We have example below where we are going to connect to online botting application, and it has internally redis as db.

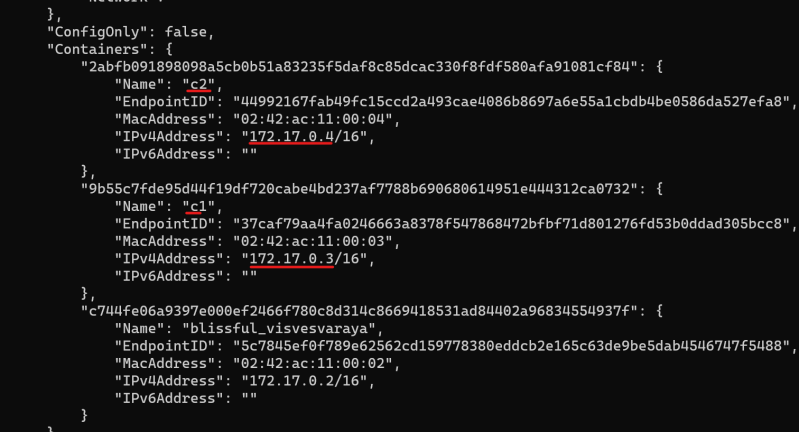
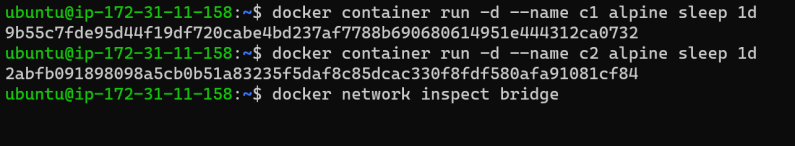
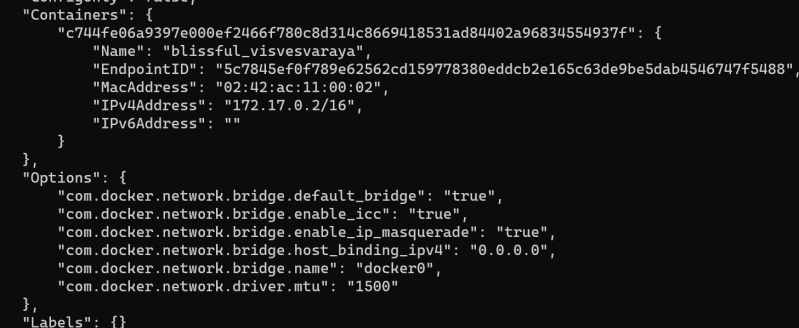
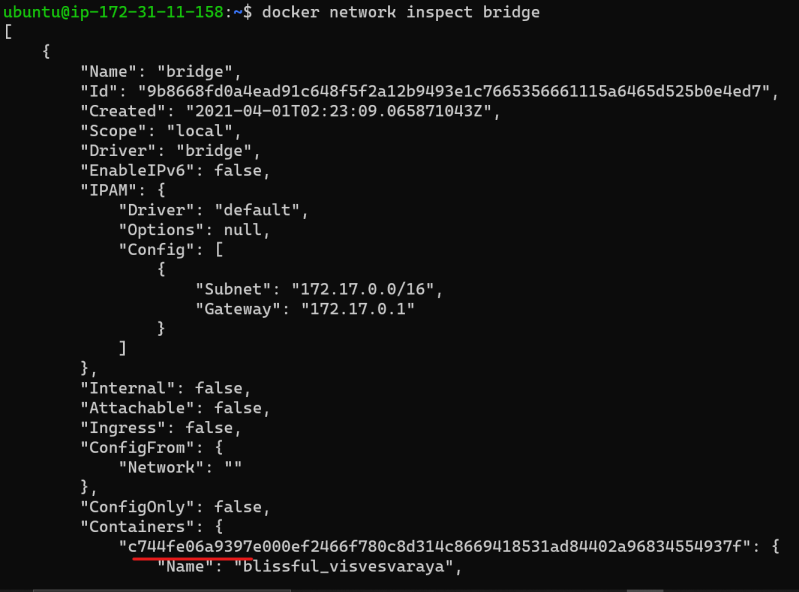
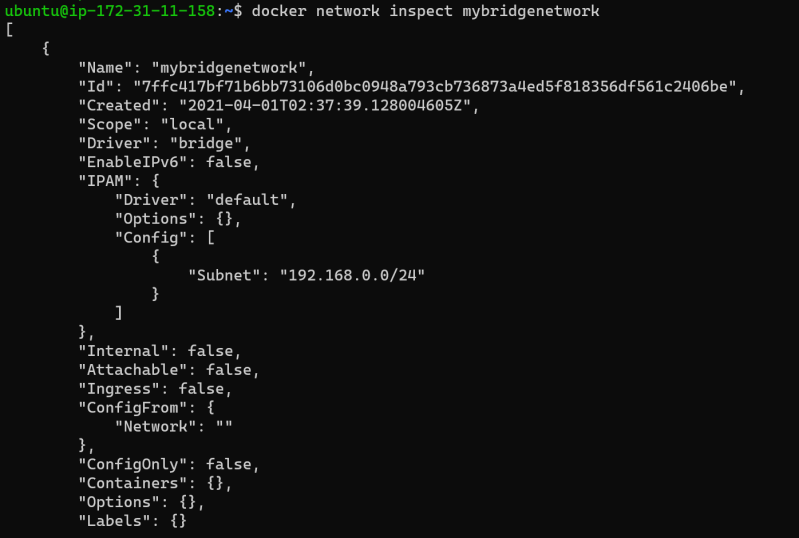
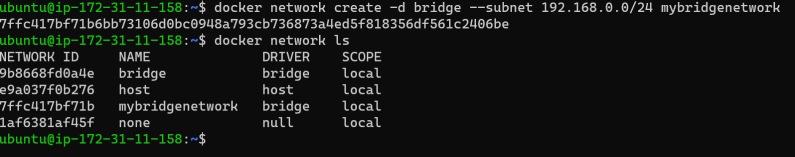
Redis: R**edis** is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker

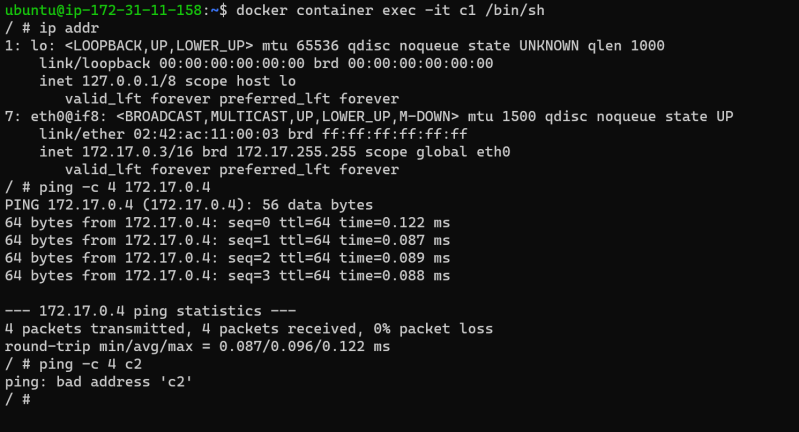


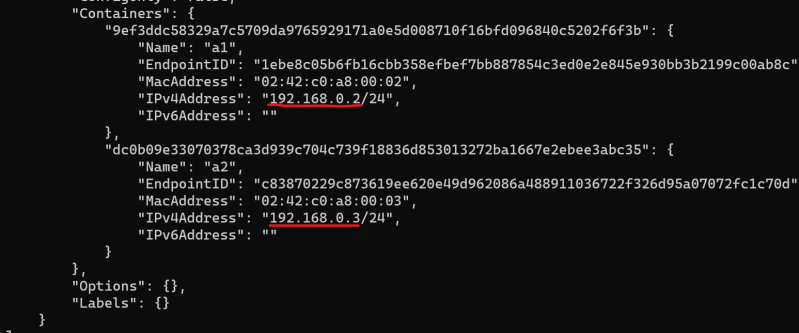
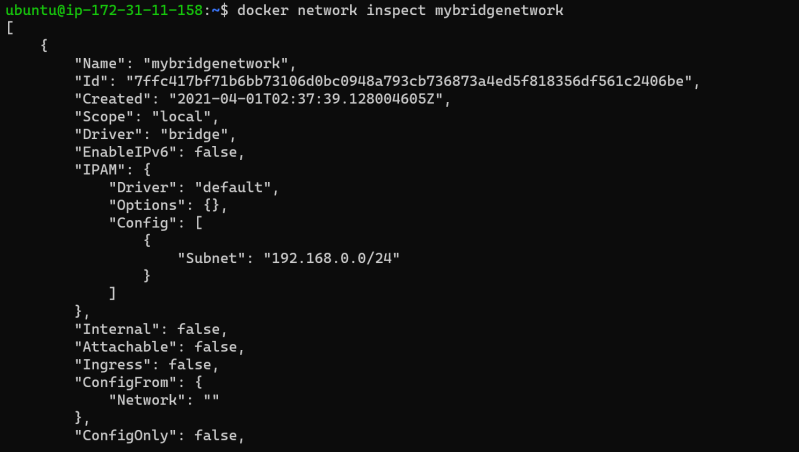
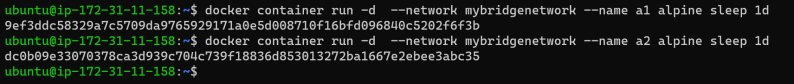
1) Create a python vote container. And try to hit the URL, it should give internal server error, as redis in not configured.

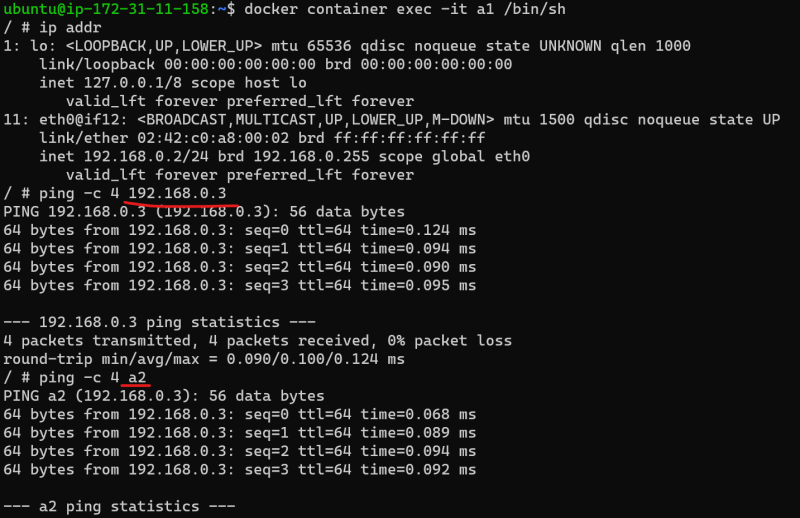


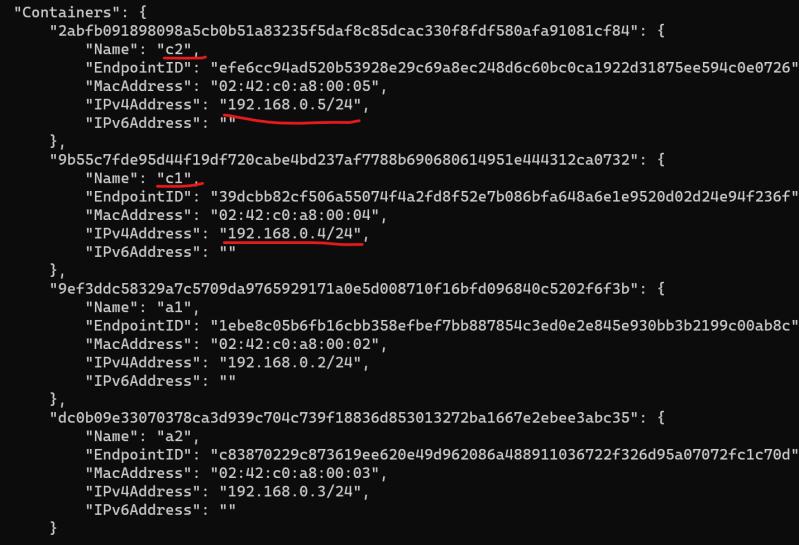
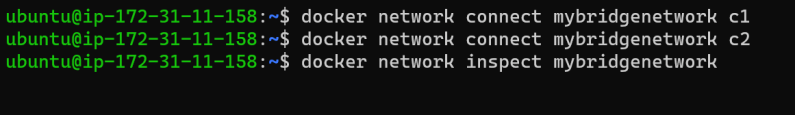
Note: We need to link containers, so that they can communicate.

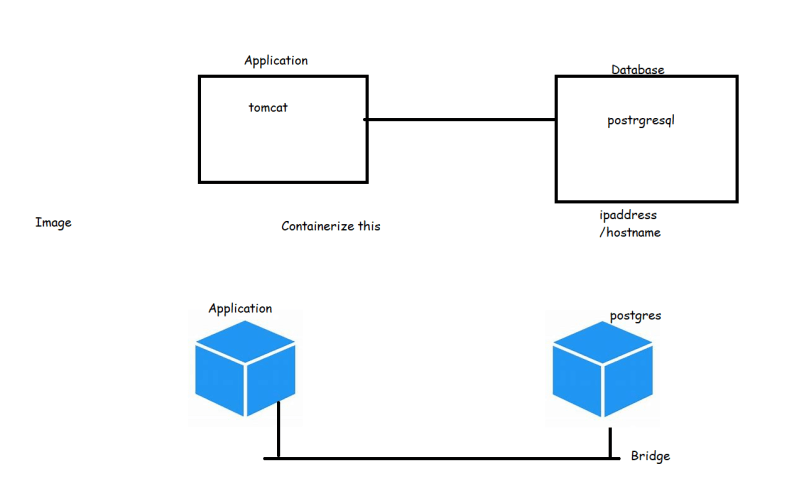
Images of the commands used in class 

 On the default bridge n/w we can establish communication b/w two containers using ip addresses but not by names 

 Now lets create two more containers in our user-defined bridge etwork 

 Now lets login into a1 container and check the connectivity with a2 

 We can connect the running containers to a different network 

* All the containers in the same network will have n/w communication but the containers in different n/ws will not have connectivity
* In the case of windows for single host networking we have nat driver which is similar to bridge in linux and we have transparent driver which is similar to host driver in linux
* Application and database containers on single host 
* Now when we run our containers on a single host and if this host is down then our application also will be down. So we need a solution where we can run docker containers on different hosts and there should be n/w connectivity b/w them
* The bridge/nat/host/transperent drivers can create a network which has a scope of single host (local)