**120A3051**

**Shreya Idate**

**Batch: E3**

**Assignment**

**AIM**: Assignment on Dockerizing a Python Flask App / Nodejs App

**THEORY**:

temp

**CONCLUSION:** temp

**120A3051**

**Shreya Idate**

**Batch: E3**

**BMW Assignment**

**AIM**: Study of Docker Networking.

**THEORY**:

One of the reasons Docker containers and services are so powerful is that you can connect them together, or connect them to non-Docker workloads. Docker containers and services do not even need to be aware that they are deployed on Docker, or whether their peers are also Docker workloads or not. Whether your Docker hosts run Linux, Windows, or a mix of the two, you can use Docker to manage them in a platformagnostic way.

This topic defines some basic Docker networking concepts and prepares you to design and deploy your applications to take full advantage of these capabilities.

Network drivers

Docker’s networking subsystem is pluggable, using drivers. Several drivers exist by default, and provide core networking functionality:

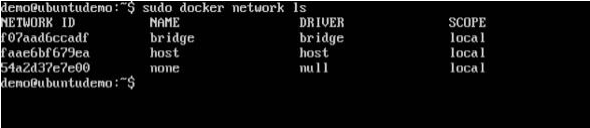
* bridge: The default network driver. If you don’t specify a driver, this is the type of network you are creating. Bridge networks are usually used when your applications run in standalone containers that need to communicate.
* host: For standalone containers, remove network isolation between the container and the Docker host, and use the host’s networking directly.
* overlay: Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons. This strategy removes the need to do OS-level routing between these containers.
* ipvlan: IPvlan networks give users total control over both IPv4 and IPv6 addressing. The VLAN driver builds on top of that in giving operators complete control of layer 2 VLAN tagging and even IPvlan L3 routing for users interested in underlay network integration.
* macvlan: Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. The Docker daemon routes traffic to containers by their MAC addresses. Using the macvlan driver is sometimes the best choice when dealing with legacy applications that expect to be directly connected to the physical network, rather than routed through the Docker host’s network stack.
* none: For this container, disable all networking. Usually used in conjunction with a custom network driver. none is not available for swarm services.
* Network plugins: You can install and use third-party network plugins with Docker. These plugins are available from Docker Hub or from third-party vendors.

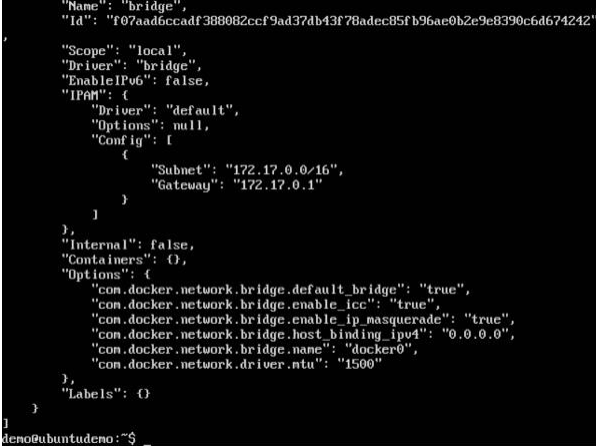
Network driver summary

* User-defined bridge networks are best when you need multiple containers to communicate on the same Docker host.
* Host networks are best when the network stack should not be isolated from the Docker host, but you want other aspects of the container to be isolated.
* Overlay networks are best when you need containers running on different Docker hosts to communicate, or when multiple applications work together using swarm services.
* Macvlan networks are best when you are migrating from a VM setup or need your containers to look like physical hosts on your network, each with a unique MAC address.
* Third-party network plugins allow you to integrate Docker with specialized network stacks.

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
  + 

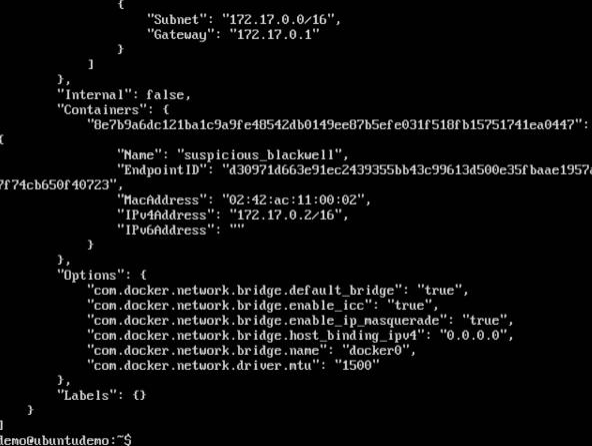
Now let’s run a container and see what happens when we inspect the network again. Let’s spin up an Ubuntu container with the following command –

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

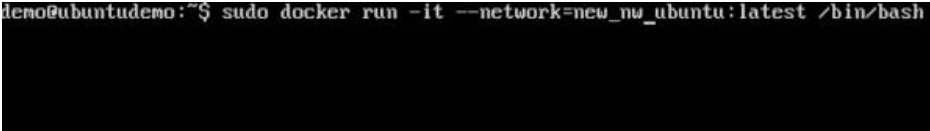


* Creating Your Own New Network: One can create a network in Docker before launching containers. This can be done with the following command −
  + Syntax: docker network create –-driver drivername name
  + Options:
    - drivername − This is the name used for the network driver.
    - name − This is 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



**CONCLUSION:** Successfully studied Docker Networking.