**DOCKER**

**Docker Networks:**

Docker Network has some types of drivers:

1. Bridge
2. Overlay
3. Host
4. Mcvlan
5. None
6. IPvlan
7. Third Party Network Plugins

**1.Bridge:**

Bridge networks are usually used when your applications run in standalone containers that need to communicate. In terms of Docker, a bridge network uses a software bridge which allows containers connected to the same bridge network to communicate, while providing isolation from containers which are not connected to that bridge network. Containers on the **default bridge network** can only access each other by IP addresses, unless you use the --link option, which is considered legacy. All containers without a --network specified, are attached to the default bridge network. This can be a risk, as unrelated stacks/services/containers are then able to communicate.

Using a user-defined network provides a scoped network in which only containers attached to that network are able to communicate.

During a container’s lifetime, you can connect or disconnect it from user-defined networks on the fly. To remove a container from the default bridge network, you need to stop the container and recreate it with different network options.

Originally, the only way to share environment variables between two containers was to link them using the --link flag. This type of variable sharing is not possible with user-defined networks. However, there are superior ways to share environment variables. A few ideas:

Multiple containers can mount a file or directory containing the shared information, using a Docker volume.

Multiple containers can be started together using docker-compose and the compose file can define the shared variables.

You can use swarm services instead of standalone containers, and take advantage of shared secrets and configs.

**Commands:**

- docker network ls

- docker network create -d bridge mynet1

- docker network create -d bridge --subnet=172.18.0.0/16 --ip-range=172.18.5.0/24 --gateway=172.18.0.254 net1

- docker network create –driver=bridge --subnet= <subnet ip> --ip-range=<ip’s range> --gateway=<gateway ip> (network name)

- docker network inspect <network name>

(In default bridge network we cannot ping the another container in it with hostname, but in custom created bridge network we can ping the containers with hostnames also )

- docker network connect <network name> <container id> (To connect one network containers to another network containers)

- docker network connect net2 cent2

- docker inspect <container id or image id> (To see full details of you want)

- docker network rm my-net

- docker create --name my-nginx --network my-net --publish 8080:80 nginx:latest

It also publishes port 80 in the container to port 8080 on the Docker host, so external clients can access that port. Any other container connected to the my-net network has access to all ports on the my-nginx container, and vice versa.

To connect a running container to an existing user-defined bridge, use the docker network connect command.

* docker network connect my-net my-nginx

To disconnect a running container from a user-defined bridge, use the docker network disconnect command.

* docker network disconnect my-net my-nginx

By default, traffic from containers connected to the default bridge network is not forwarded to the outside world. To enable forwarding, you need to change two settings. These are not Docker commands and they affect the Docker host’s kernel.

Configure the Linux kernel to allow IP forwarding.

sysctl net.ipv4.conf.all.forwarding=1

Change the policy for the iptables FORWARD policy from DROP to ACCEPT.

sudo iptables -P FORWARD ACCEPT

Docker has a known issue where it seems to ignore the network's maximum packet size MTU (**Maximum Transmission Unit**) and set a default of 1500.

**2. Overlay:**

The overlay network driver creates a distributed network among multiple Docker daemon hosts. This network sits on top of (overlays) the host-specific networks, allowing containers connected to it (including swarm service containers) to communicate securely when encryption is enabled. Docker transparently handles routing of each packet to and from the correct Docker daemon host and the correct destination container. The overlay network driver creates a distributed network among multiple Docker daemon hosts. This network sits on top of (overlays) the host-specific networks, allowing containers connected to it (including swarm service containers) to communicate securely when encryption is enabled. Docker transparently handles routing of each packet to and from the correct Docker daemon host and the correct destination container.

When you initialize a swarm or join a Docker host to an existing swarm, two new networks are created on that Docker host:

* an overlay network called ingress, which handles the control and data traffic related to swarm services. When you create a swarm service and do not connect it to a user-defined overlay network, it connects to the ingress network by default.
* a bridge network called docker\_gwbridge, which connects the individual Docker daemon to the other daemons participating in the swarm.

**Create an overlay network & Prerequisites:**

Firewall rules for Docker daemons using overlay networks, You need the following ports open to traffic to and from each Docker host participating on an overlay network:

* TCP port 2377 for cluster management communications & TCP and UDP port 7946 for communication among nodes
* UDP port 4789 for overlay network traffic

Before you can create an overlay network, you need to either initialize your Docker daemon as a swarm manager using docker swarm init or join it to an existing swarm using docker swarm join. Either of these creates the default ingress overlay network which is used by swarm services by default.

**Host networking:**

If you use the host network mode for a container, that container’s network stack is not isolated from the Docker host (the container shares the host’s networking namespace), and the container does not get its own IP-address allocated.

For instance, 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. The host networking driver only works on Linux hosts, and is not supported on Docker Desktop for Mac, Docker Desktop for Windows, or Docker EE for Windows Server. You can also use a host network for a swarm service, by passing --network host to the docker service create command.

**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. See Macvlan networks.

**none**:

For this container, disable all networking. Usually used in conjunction with a custom network driver. none is not available for swarm services. See disable container networking.

**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. See the vendor’s documentation for installing and using a given network plugin.

**Docker and Iptables:**

Docker manipulates iptables rules to provide network isolation. If you’re running Docker on a host that is exposed to the Internet, you will probably want to have iptables policies in place that prevent unauthorized access to containers or other services running on your host.

**Add iptables policies before Docker’s rules**

Docker installs two IP tables chains and it ensures that incoming packets are always checked by these two chains first.

1. DOCKER
2. DOCKER-USER

All of Docker’s iptables rules are added to the **DOCKER** chain. Do not manipulate this chain manually. If you need to add rules which load before Docker’s rules, add them to the **DOCKER-USER** chain. These rules are applied before any rules Docker creates automatically.

Rules added to the **FORWARD** chain either manually, or by another iptables-based firewall are evaluated after these chains. This means that if you expose a port through Docker, this port gets exposed no matter what rules your firewall has configured. If you want those rules to apply even when a port gets exposed through Docker, you must add these rules to the **DOCKER-USER** chain.All external source IPs are allowed to connect to the Docker host. To allow only a specific IP or network to access the containers, insert a negated rule at the top of the **DOCKER-USER** filter chain.For example, the following rule restricts external access from all IP addresses except 192.168.1.1:

iptables -I DOCKER-USER -i ext\_if ! -s 192.168.1.1 -j DROP

The following rule only allows access from the subnet 192.168.1.0/24:

iptables -I DOCKER-USER -i ext\_if ! -s 192.168.1.0/24 -j DROP

Finally, you can specify a range of IP addresses to accept using --src-range (Remember to also add -m iprange when using --src-range or --dst-range):

iptables -I DOCKER-USER -m iprange -i ext\_if ! --src-range 192.168.1.1-192.168.1.3 -j DROP

**Docker Volumes**