Networking overview :

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 platform-agnostic 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. See bridge networks.

host:

For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly. See use the host network.

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. See overlay networks.

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.

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.

Differences between user-defined bridges and the default bridge :

User-defined bridges provide automatic DNS resolution between containers.

Containers on the default bridge network can only access each other by IP addresses, unless you use the --link option, which is considered legacy. On a user-defined bridge network, containers can resolve each other by name or alias.

Imagine an application with a web front-end and a database back-end. If you call your containers web and db, the web container can connect to the db container at db, no matter which Docker host the application stack is running on.

If you run the same application stack on the default bridge network, you need to manually create links between the containers (using the legacy --link flag). These links need to be created in both directions, so you can see this gets complex with more than two containers which need to communicate. Alternatively, you can manipulate the /etc/hosts files within the containers, but this creates problems that are difficult to debug.

User-defined bridges provide better isolation.

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.

Containers can be attached and detached from user-defined networks on the fly.

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.

Each user-defined network creates a configurable bridge.

If your containers use the default bridge network, you can configure it, but all the containers use the same settings, such as MTU and iptables rules. In addition, configuring the default bridge network happens outside of Docker itself, and requires a restart of Docker.

User-defined bridge networks are created and configured using docker network create. If different groups of applications have different network requirements, you can configure each user-defined bridge separately, as you create it.

Linked containers on the default bridge network share environment variables.

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.

Containers connected to the same user-defined bridge network effectively expose all ports to each other. For a port to be accessible to containers or non-Docker hosts on different networks, that port must be published using the -p or --publish flag.

#docker network ls **(Lists all the available Networks on Docker Host)**

#docker network create --driver bridge --subnet 172.18.0.0/16 bridge1 **(Creating new user/custom defined bridge network)**

#docker network ls

#docker network inspect <network-Name> **(Inspecting a specific network)**

Creating Container with Specific Network :

#docker run -d --name web1 --network bridge1 nginx

#docker network inspect bridge

#docker network connect bridge1 web9 **(Connecting Existing container to another new Network , Assume we have container web9)**

#docker network inspect bridge1
#docker container inspect web9

Disconnecting Container from the Network :
#docker network disconnect <N/W Name> web9
#docker container inspect web9

Connecting Container from the Network :
#docker network connect <N/W Name> web9
#docker container inspect web9

Deleting Network :
#docker network rm bridge1

Removing all the unused Networks :
#docker network prune
#docker network ls