Docker networks allow containers to communicate with each other, the Docker host, and external networks. Each container can be attached to different networks, which control how they connect with other containers or the outside world.

Docker Network Types

1. **Bridge Network (default)**
   - **Use case**: Allows containers on the same host to communicate.
   - Docker creates a default bridge network when you start Docker, and containers are automatically attached to this network unless specified otherwise.
2. **Host Network**
   - **Use case**: Removes network isolation between the container and the Docker host.
   - The container shares the host's IP address, which can be useful for performance improvements.
3. **Overlay Network**
   - **Use case**: Allows containers running on different Docker hosts to communicate.
   - Overlay networks are mostly used with Docker Swarm or Kubernetes for multi-host communication.
4. **Macvlan Network**
   - **Use case**: Allows containers to have their own MAC address and appear on the network like physical devices.
   - Useful for network segregation and when working with existing infrastructure that uses MAC-based filtering.
5. **None Network**
   - **Use case**: Disables networking completely for the container.
   - Containers attached to this network type have no network interfaces apart from the loopback.

Detailed Examples

1. Bridge Network Example

Let's create a custom bridge network and run containers within it.

```
# Create a custom bridge network
docker network create my_bridge_network
```

```
# Run two containers in the custom network
docker run -d --name container1 --network my_bridge_network alpine
sleep 1000
docker run -d --name container2 --network my_bridge_network alpine
sleep 1000
```

Container Communication:

Once the containers are connected to the same network, they can communicate by name.

```
# Exec into container1 and ping container2 by name
docker exec -it container1 sh
ping container2
```

2. Host Network Example

In some scenarios, using the host network can improve network performance.

```
# Run a container with the host network
docker run -d --network host nginx
```

Here, NGINX will be running directly on the host's IP address, meaning if your host IP is `192.168.1.10`, NGINX will be accessible via `192.168.1.10:80`.

3. Overlay Network Example (Swarm)

An overlay network allows containers on different Docker hosts to communicate, often used with Docker Swarm.

```
# Initialize Docker Swarm on a manager node
docker swarm init

# Create an overlay network
docker network create -d overlay my_overlay_network

# Run a service using the overlay network
```

```
docker service create --name web --network my_overlay_network -p 80:80
nginx
```

With this setup, any worker nodes added to the swarm will connect to the overlay network, and containers on different nodes can communicate seamlessly.

4. Macvlan Network Example

With Macvlan, containers can have their own IP addresses on the local network.

```
# Create a Macvlan network with a specific subnet and gateway
docker network create -d macvlan \
  --subnet=192.168.1.0/24 \
  --gateway=192.168.1.1 \
  -o parent=eth0 my_macvlan_network

# Run a container with an IP address from the specified subnet
docker run -d --network my_macvlan_network --ip 192.168.1.100 nginx
```

The container will appear on the network as a physical device with the IP `192.168.1.100`, and it can be accessed from other devices on the network.

5. None Network Example

You may want to run a container without any network interfaces.

```
# Run a container with no network
docker run -d --network none alpine sleep 1000
```

This container won't be able to access any external network, which can be useful in isolated or testing environments.

Inspecting Networks

You can inspect network configurations using the `docker network inspect` command to view detailed information about networks and their connected containers.

```
docker network inspect my_bridge_network
```

This provides information like IP addresses, connected containers, and more.

Docker DNS

Docker provides a built-in DNS server that containers use to resolve hostnames. For example, if two containers are connected to the same user-defined bridge network, you can resolve the other container’s name to its internal IP address.

```
# In container1
ping container2
```

Key Concepts

- **DNS-Based Container Communication**: Containers can communicate by their names when attached to the same custom network.
- **Isolation**: Containers attached to different networks are isolated from each other.
- **Multi-Host Networking**: Overlay networks allow communication between containers across multiple hosts.