Handout: Container Networking

Docker containers are isolated by default, but they can communicate through configurable **networks**. Docker's networking options are essential for building multi-container applications (like frontend-backend setups or service stacks).

1. Default Networks

Docker creates several built-in networks:

- `bridge` : Default for standalone containers on a single host.
- `host` : Shares the host's network stack (Linux only).
- `none` : No networking (fully isolated).

To see available networks:

```
docker network ls
```

2. The Bridge Network (Default)

If you don't specify a network, Docker connects containers to the default `bridge` network.

```
docker run -d --name app nginx
```

This container is isolated but can reach the internet (e.g. for updates) via NAT. **NAT (Network Address Translation)** allows multiple containers to share the host's IP address for outbound traffic — Docker translates internal container IPs to the host's IP when making requests. This enables internet access, but **inbound traffic is blocked** unless you explicitly publish ports. Other containers can't access it by name unless you **create a custom network**.

3. Creating a Custom Bridge Network

Custom bridge networks enable automatic DNS-based container discovery. The general command to create a network is:

```
docker network create <network-name>
```

For example, to create a network called `my-net` :

```
docker network create my-net
```

Once the network exists, you can start containers and attach them using the `--network` flag:

```
docker run -d --name <container-name> --network <network-name> <image>
```

Example:

```
docker run -d --name backend --network my-net backend-image
docker run -d --name frontend --network my-net frontend-image
```

Inside the frontend container, you can access `backend` using its container name as a hostname:

```
ping backend
```

📦 Custom networks must be created **before** you can attach containers to them using `--network`. Use them for multi-container setups to enable name-based service discovery and isolate app components.

4. Exposing Ports

To make a container accessible from the host machine (e.g. browser or CLI), you must publish a port:

```
docker run -p <host-port>:<container-port> <image>
```

Example:

```
docker run -d -p 8080:80 nginx
```

This maps port `80` in the container to port `8080` on the host.

📦 Port publishing is needed **only for host access**. Containers on the same network can talk via internal ports directly.

5. Host Network (Linux Only)

Use the host network to remove isolation and make the container share the host's network stack:

```
docker run --network host <image>
```

This gives the container direct access to host services (e.g. `localhost:5432`), and the container itself is accessible via the host's IP/hostname.

📦 Not available on Docker Desktop for macOS/Windows due to VM-based architecture.

6. None Network

To start a completely isolated container:

```
docker run --network none <image>
```

Useful for containers that should not have any external network access.

📦 Use the `none` network when you want complete network isolation, e.g. for security audits, air-gapped containers, or when you manually configure networking.

7. Connecting to Networks After Start

You can connect a running container to an existing network using:

```
docker network connect <network-name> <container-name>
```

For example,

```
docker network connect my-net my-container
```

To disconnect:

```
docker network disconnect <network-name> <container-name>
```

For example,

```
docker network disconnect my-net my-container
```

📦 A container can be attached to **multiple networks**, but you should **plan your network layout** to avoid unnecessary complexity.

8. Cleaning Up

To remove unused networks:

```
docker network prune
```

To remove a specific network:

```
docker network rm my-net
```

📦 A network cannot be removed while containers are still connected to it.

9. Summary: Network Types

Network Type	**Own Network Stack**	**Host Access**	**Container Discovery**
bridge (default)	✅	Needs port mapping	❌
custom bridge	✅	Needs port mapping	✅
host	❌	Direct access	❌
none	✅	❌	❌

📦 Use **custom bridge networks** for most containerized applications to enable DNS-based service discovery. Use **host** for performance-sensitive cases on Linux, and **none** for full network isolation or manual setups.