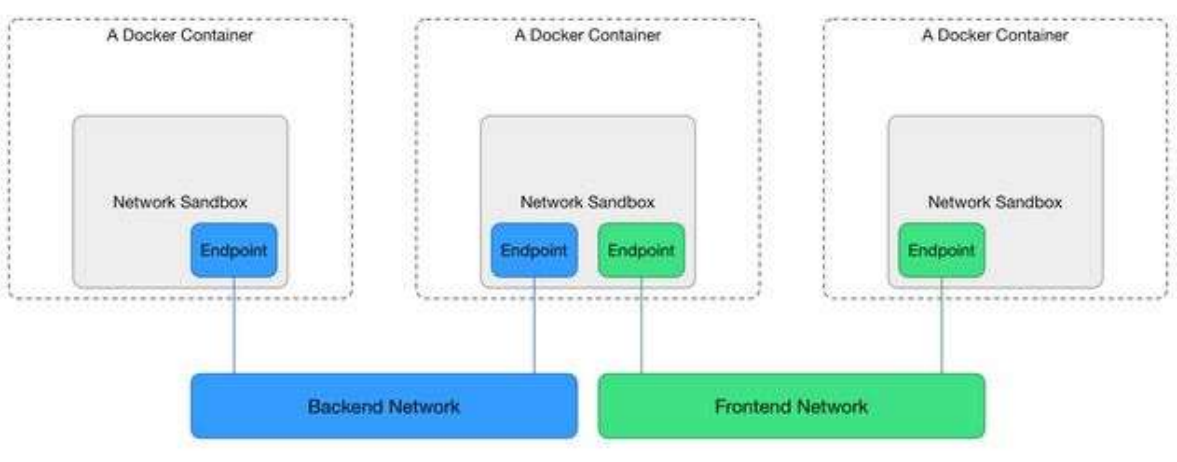
CLOUD COMPUTING APPLICATIONS

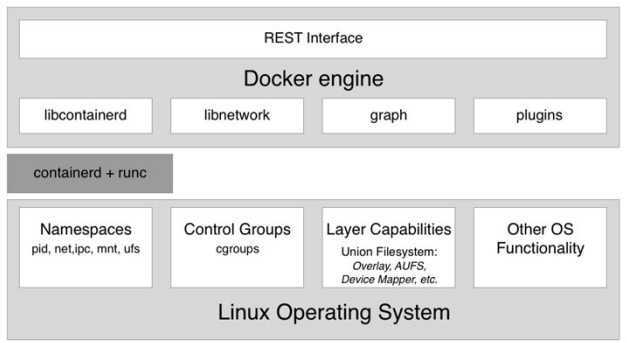
Containers: Networking

Prof. Reza Farivar

Container Network Model

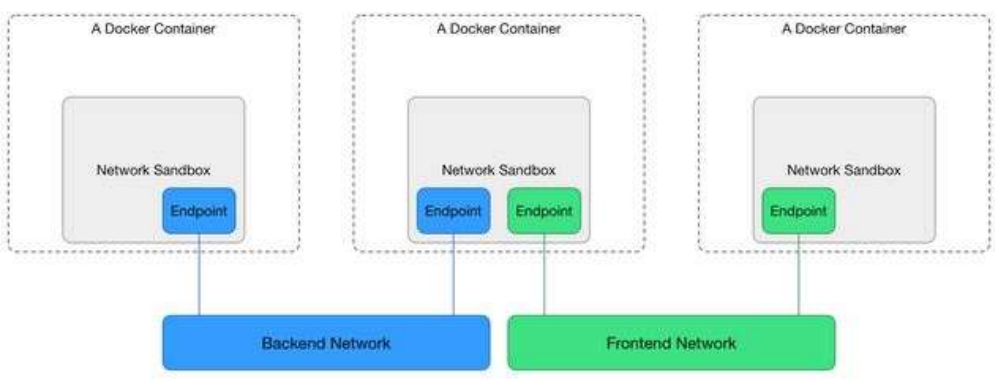
- Libnetwork implements Container Network Model (CNM)
- Formalizes the steps required to provide networking for containers while providing an abstraction that can be used to support multiple network drivers





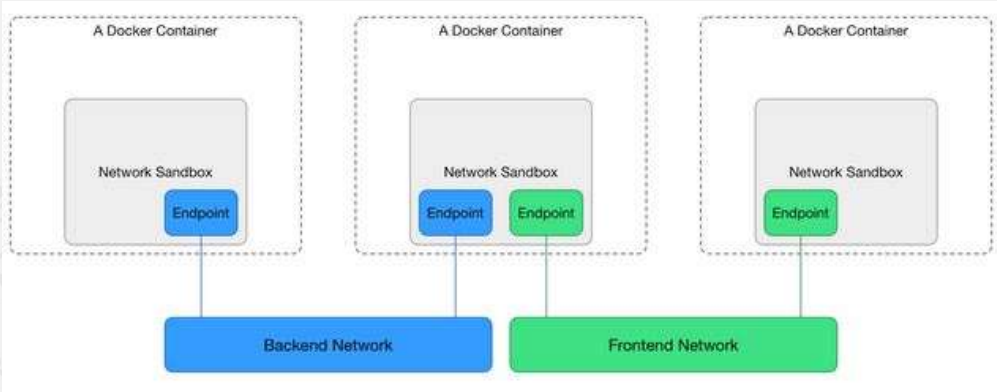
Container Network Model: Sandbox

- A Sandbox contains the configuration of a container's network stack.
- This includes management of the container's interfaces, routing table and DNS settings.
- A Sandbox may contain *many* endpoints from *multiple* networks.
- An implementation of a Sandbox could be a Linux Network Namespace, a FreeBSD Jail or other similar concept.
- Libnetwork implements sandbox in Linux through network namespace
- It creates a Network Namespace for each sandbox which is uniquely identified by a path on the host filesystem.



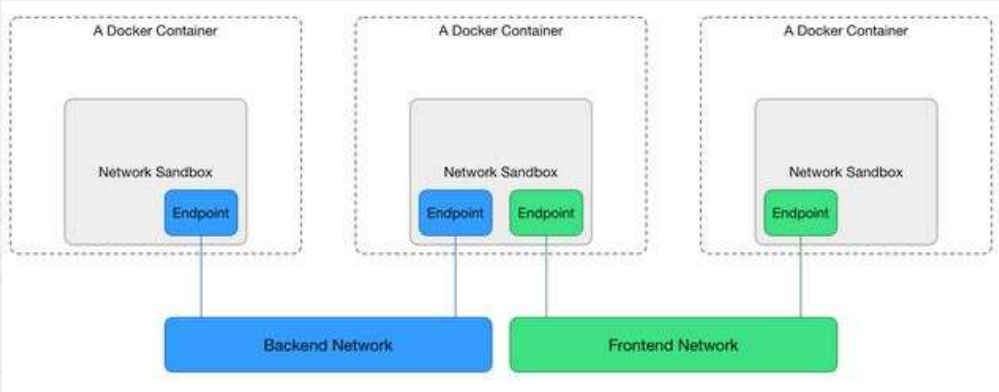
Container Network Model: Endpoint

- An Endpoint joins a Sandbox to a Network.
- An implementation of an Endpoint could be a `veth pair`, an Open vSwitch internal port or similar.
  - The **veth** devices are virtual Ethernet devices.
  - They can act as tunnels between network namespaces to create a bridge to a physical network device in another namespace
  - Can also be used as standalone network devices
  - **veth** devices are always created in interconnected pairs
    - One end is paced in one network namespace, and the other end in another namespace
- An Endpoint can belong to only one network and it can belong to only one Sandbox, if connected.
- Libnetwork delegates the actual implementation to the drivers which realize the functionality



Container Network Model: Network

- A Network is a group of Endpoints that are able to communicate with each-other directly.
- An implementation of a Network could be a Linux bridge, a VLAN, etc.
- Networks consist of *many* endpoints.
- Libnetwork delegates the actual implementation to the drivers which realize the functionality



Driver packages

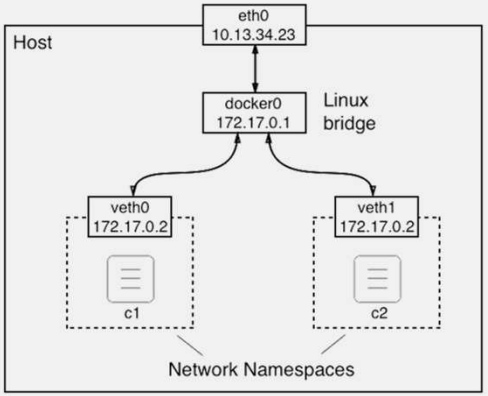
- Extension of libnetwork and provide the actual implementation of API
  - driver.Config
  - driver.CreateNetwork
  - driver.DeleteNetwork
  - driver.CreateEndpoint
  - driver.DeleteEndpoint
  - driver.Join
  - driver.Leave

Default Drivers in Docker Libnetwork

- Bridge: uses Linux Bridging and iptables to provide connectivity for containers
  - It creates a single bridge, called `docker0` by default, and attaches a `veth pair` between the bridge and every endpoint.
- host: For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly
- Overlay: networking that can span multiple hosts using overlay network encapsulations such as VXLAN
  - Enable swarm services to communicate with each other
- macvlan: Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network.
  - Docker daemon routes traffic to containers by their MAC addresses
- None
- Note: The type of network a container uses is transparent from within the container

Bridge Networks

- Bridge networks are usually link layer devices that forward traffic between networks
- In Docker, bridge network uses a software bridge allowing containers connected to the same bridge network on the same host
  - Isolating containers from other containers not connected to the bridge
  - For communicating with containers in other hosts, use overlay network
- The Docker bridge driver automatically installs rules in the host machine so that containers on different bridge networks cannot communicate directly with each other
  - `iptables` rules on Linux



Default Bridge Network

- When you start Docker, a default bridge network (also called `bridge`) is created automatically, and newly-started containers connect to it unless otherwise specified.
- Containers on the default bridge network can only access each other by IP addresses
  - User-defined bridges provide automatic DNS resolution between containers.
- The default bridge network is considered a legacy detail of Docker and is **NOT recommended** for production use

User-defined Networks

- Use --network to attach a container to a specific network
- Better isolation
- DNS resolution
  - On a user-defined bridge network, containers can resolve each other by the *container name* or alias
  - Much better than messing with /etc/hosts
- Containers can be attached and detached from user-defined networks on the fly
- Containers connected to the same user-defined bridge network effectively expose all ports to each other

Publishing ports

- From the container's point of view, it has a network interface with an IP address, a gateway, a routing table, DNS services, and other networking details
- 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 create --name my-nginx  --network my-net --publish 8080:80 nginx:latest`

Host : Container

IPAM: IP Address Management

- IPAM tracks and manages IP addresses for each network
  - Subnet
    - E.g. 172.17.0.0/16
    - All containers attached to this network will get an IP address taken from this CIDR range
  - Gateway
    - E.g. 172.17.0.1
    - Router for this network
- By default only egress traffic is allowed
  - Containerized applications can reach the internet, but they cannot be reached by any outside traffic

RFC 1918

```
~ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "232cda23c82c663ab4ea5297eb5e1e7a57b91cdd37d8ee63eb217
        "Created": "2021-02-28T03:04:28.953451723Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
```

Containers in the same namespace

- We can have multiple containers in the same namespace
- Processes in two containers in the same namespace can communicate through localhost
  - Compare to bridge networking, with two containers connected to the same network, where each host gets its own IP address
- Note that a sandbox (aka the Linux namespace) is connected to a network
  - We typically run each container in its own sandbox
  - But multiple containers can run in the same sandbox