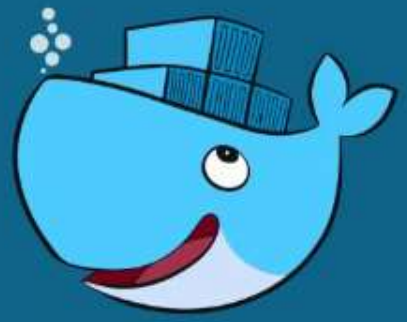


# Docker Configuration, Developing, and Networking

# Agenda

- Managing and Configuring Docker Daemon
- Securing Docker Daemon for Remote Access
- Introduction to nsenter
- Introduction to runc
- Introduction to Docker Networking
- Network Types

# MANAGING AND CONFIGURING DOCKER DAEMON

# Managing Docker Daemon

- Docker Daemon is responsible for creating, running, and monitoring containers. It is also used for building and storing images.
- Use the docker init script to manage the Docker daemon.
- On most Ubuntu systems, it will be located in the `/etc/init.d/docker` file.
- Like most other init services, it can be managed via the service command.
- The Docker daemon runs as root:

```
root@test01:~# service docker status
docker start/running, process 1553
root@test01:~# service docker stop
docker stop/waiting
root@test01:~# service docker start
docker start/running, process 3212
root@test01:~#
```

# Configuring Docker Daemon

- The daemon includes many configuration options, which you can either
  - Pass flags when starting Docker manually,
  - Set in the *daemon.json* configuration file.
- Second method is recommended as these configuration changes persist when you restart Docker.
- Click dockerd for a full list of configuration options.
- The configuration file is located in `/etc/default/docker`.

# Configuring Docker Daemon : Flags

- Here is an example of starting the Docker daemon manually with some configuration options:

```
$ dockerd -D --tls --tlscert/var/docker/server.pem --tlskey/var/docker/serverkey.pem -H
tcp://192.168.59.3:2376
```

- This command enables debugging (-D), enables TLS (-tls), specifies the server certificate and key (--tlscert and --tlskey), and specifies the network interface where the daemon listens for connections (-H).

# Configuring Docker Daemon : daemon.json

- A better approach is to put these options into the *daemon.json* file and restart Docker.
- This method works for every Docker platform.
- The following daemon.json example sets all the same options as the above command:

```
{
  "debug": true,
  "tls": true,
  "tlscert": "/var/docker/server.pem",
  "tlskey": "/var/docker/serverkey.pem",
  "hosts": ["tcp://192.168.59.3:2376"]
}
```

# Default Docker Daemon

- The default Docker daemon listens on a local Unix socket, `/var/run/docker.sock`, which is accessible only locally.
- However, you would like to access your Docker host remotely, calling the Docker API from a different machine. Let's see how?

# Accessing Docker Daemon Remotely – Facts

- The Docker daemon can listen for Docker Remote API requests via three different types of Socket: *unix, tcp, and fd.*
- If you need to access the Docker daemon remotely, you need to enable the *tcp* Socket.
- Default setup provides un-encrypted and un-authenticated direct access to the Docker daemon - and should be secured either using the built in HTTPS encrypted socket, or by putting a secure web proxy in front of it.
- You can listen on port 2375 on all network interfaces with -H tcp://0.0.0.0:2375, or on a particular network interface using its IP address: -H tcp://192.168.59.103:2375.
- It is conventional to use port 2375 for un-encrypted, and port 2376 for encrypted communication with the daemon.

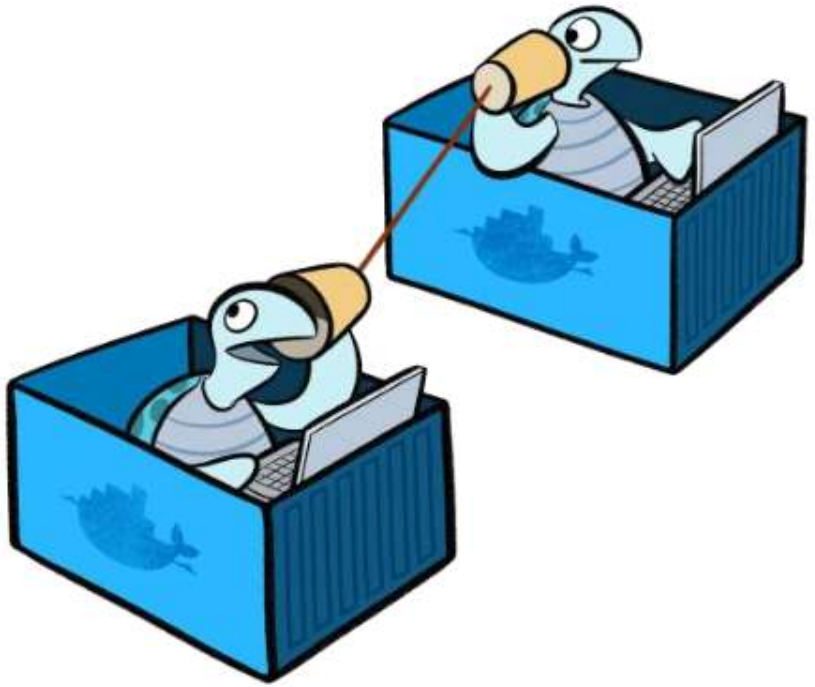
# How to Secure Docker Daemon for Remote Access?

- Set up TLS-based access to your Docker daemon.
- This will use public-key cryptography to encrypt and authenticate communication between a Docker client and the Docker daemon that you have set up with TLS.
- The basic steps to test this security feature are described on the Docker website.
  - This includes how to create your own certificate authority (CA) and sign server and client certificates using the CA. In a properly set up infrastructure, you need to contact the CA that you use routinely and ask for server certificates.

# Introduction to Networking

# Docker Network

- Networks in docker can be configured to provide complete isolation for containers.
- This enables building web applications that work together *securely*.



# Docker Network

| Concept | Description |
| --- | --- |
| Network Namespace | Provides a way of having separate network stack for each instance of a container. |
| Docker0 Bridge | Default bridge created by Docker to provide communication across Docker containers and external world including the host. |
| Port Mapping | Mechanism to map a port in the host machine with the Docker container's networking stack. |
| Veth Pair | Veth is a special, logical, virtual interface which is similar to a link / pipe. It has two ends, which are logical interfaces and provide connectivity across two different network elements. |

# Docker – Network Types

# Docker Network : Viewing the Types

- When Docker is installed, it creates three networks automatically, which can be listed using the docker network Command: `docker network ls`

```
root@docker:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
6aca5b47a886        bridge              bridge              local
1fe323c16250        host                host                local
8b0b57f18efa        none                null                local
root@docker:~# _
```

- These three networks are built into Docker. When you run a container, you can use the `--network` flag to specify which networks your container should connect to.

# Docker Network Types – Details

## Bridge Network

- The bridge network represents the docker0 network present in all Docker installations
- Docker daemon connects containers to this network by default.

## Host Network

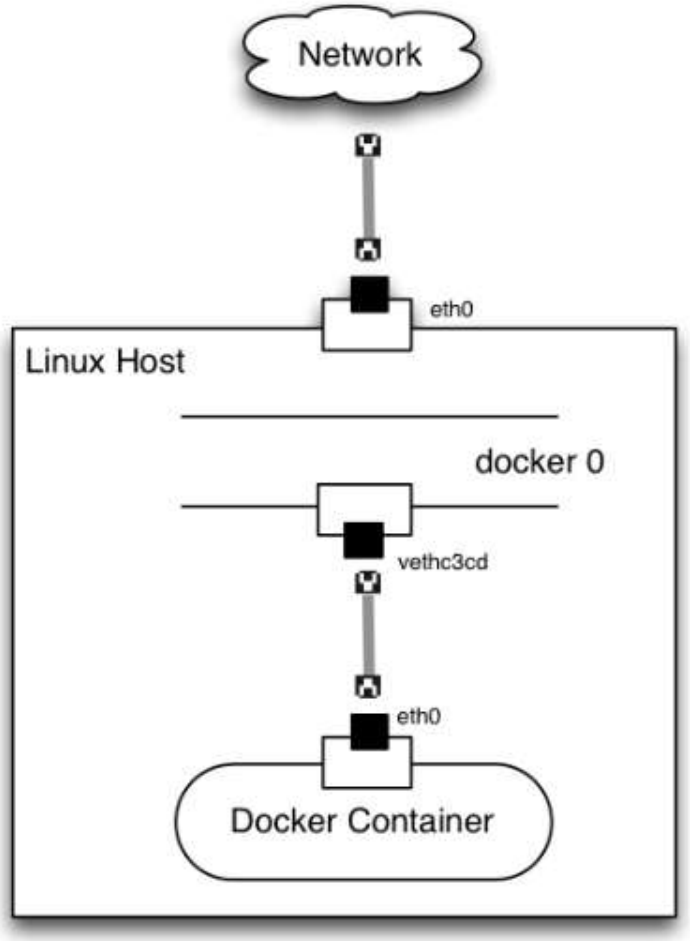
- Host is basically Docker app that uses the Host's IP address
- In docker terms it means that the docker container has unrestricted access to all the ports available to the host. It will also be using the host's IP address.

## None Network

- The none network adds a container to a container-specific network stack. That container lacks a network interface.

# Bridge Network

- **Docker0 bridge**
  - Virtual bridge is similar to linux bridge
  - Created in the host machine during the creation of Docker container
- **Veth Pair**
  - Is created during the creation of Docker container
  - One end of the *veth* pair is attached to the eth0 interface of Docker container
  - Another end is attached to the docker0 bridge with interface name starts with *vethc3cd*

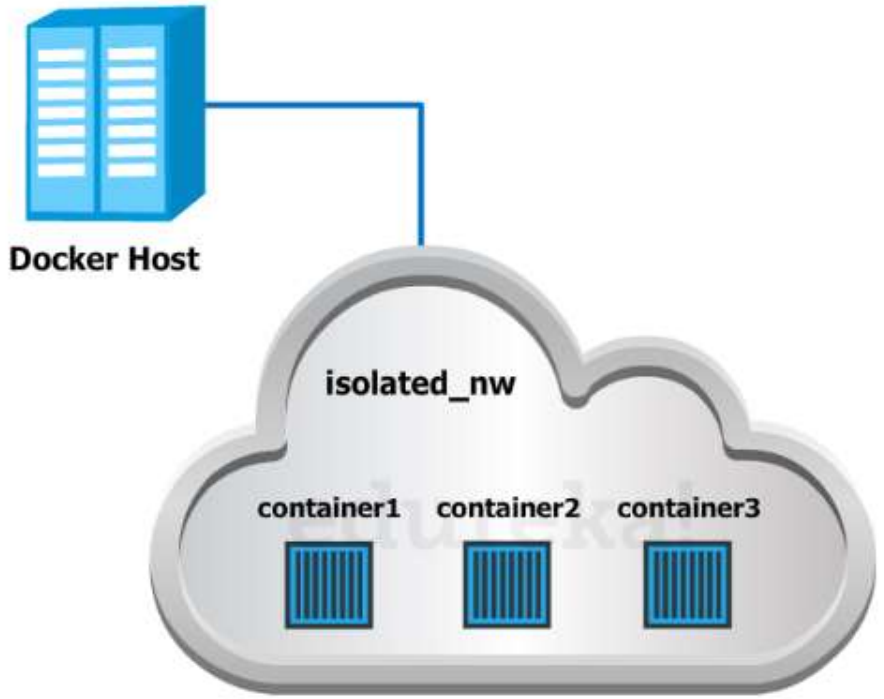


# When to use the bridge Network?

- If you have a set of containers,
  - Each providing micro-services to the other (and)
  - Should not be exposed to the external world

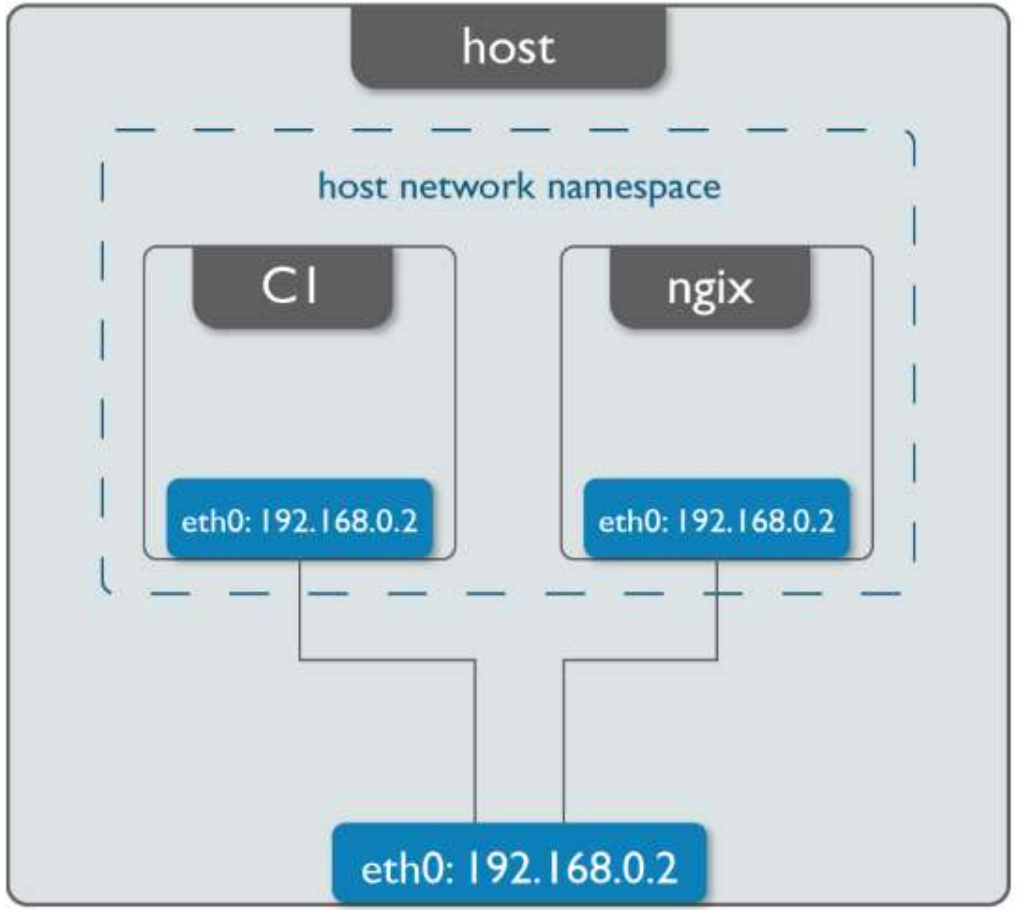
*The bridge network is your best choice.*

- Bridge network is typically used with layered architectures.
- Be aware that services in these containers are not exposed to outside networks.



# Host Network

- A port in the host machine is bound to a port in Docker.
- It is a simple way of running a service in Docker container and exposing the service to external world.
- Simple use case can be, running a webserver in the docker container and forwarding all the web traffic, which is coming to the HTTP port in host machine to Docker container.



# Host Network

Think of host mode like setting DMZ on your firewall to one of your computers. Basically you are allowing all the container to do whatever it wants using the hosts' IP / ports.