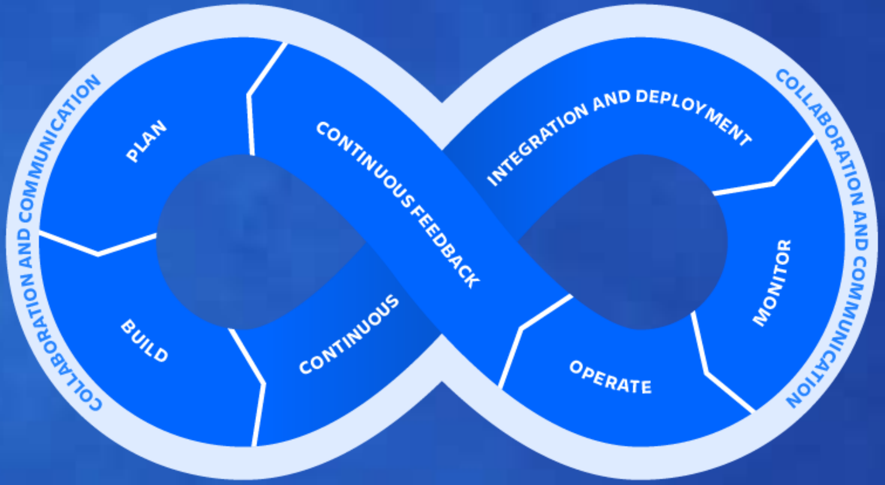

COLLABORATION AND COMMUNICATION
PLAN
CONTINUOUS FEEDBACK
INTEGRATION AND DEPLOYMENT
COLLABORATION AND COMMUNICATION
MONITOR
OPERATE
CONTINUOUS
BUILD

# Session 14. Docker Networking

Ram N Sangwan

## Agenda

- Accessing containers
- Linking containers
- Exposing container ports
- Container Routing

# Accessing Containers

- If we have a container running in detached mode and exposing the port 3070 to the outside world.

  ```
  # docker run -it -d -p 3070:3070 nodewebapp:v1
  ```

  *We can access it at http://<ip address>:3070*

- To have shell access to container (e.g MySQL), use ***docker exec -it*** command to start a bash shell inside the container:

  ```
  # docker exec -it mysql1 bash
  bash-4.2#
  ```

## Linking Containers

- By linking containers, you provide a secure channel via which Docker containers can communicate to each other.
- Think of a sample web application. You might have a Web Server and a Database Server. When we talk about linking Docker Containers, what we are talking about here is the following:
    - We can launch one Docker container that will be running the Database Server.
    - We will launch the second Docker container (Web Server) with a link flag to the container launched in Step 1. This way, it will be able to talk to the Database Server via the link name.

## Linking Containers - Step 1

- Download the images of mysql and wordpress from dockerhub with your account by running these simple commands-

```
$ docker pull mysql
$ docker pull wordpress
```

## Linking Containers - Step 2

- After downloading both images successfully we will run our back-end mysql image by running this command-

```
$ docker run --name demo-mysql -e MYSQL_ROOT_PASSWORD=your-password -d mysql:latest
```

- Here ‘demo-mysql’ is the name of our container and ‘your-password’ is the password of our mysql database. You can change it if you want.

# Linking Containers - Step 3

- Now we are going to run our wordpress image and also we will link our wordpress container to mysql container by running this command-

```
$ docker run --name demo-wordpress --link demo-mysql:mysql -p 8082:80 -d wordpress
```

- Here ‘demo-wordpress’ is the name of our wordpress container. Here we also used –link tag to link wordpress container to mysql container.

# Linking Containers - Step 4

- Now we can check that both the containers are running correctly by running this command-

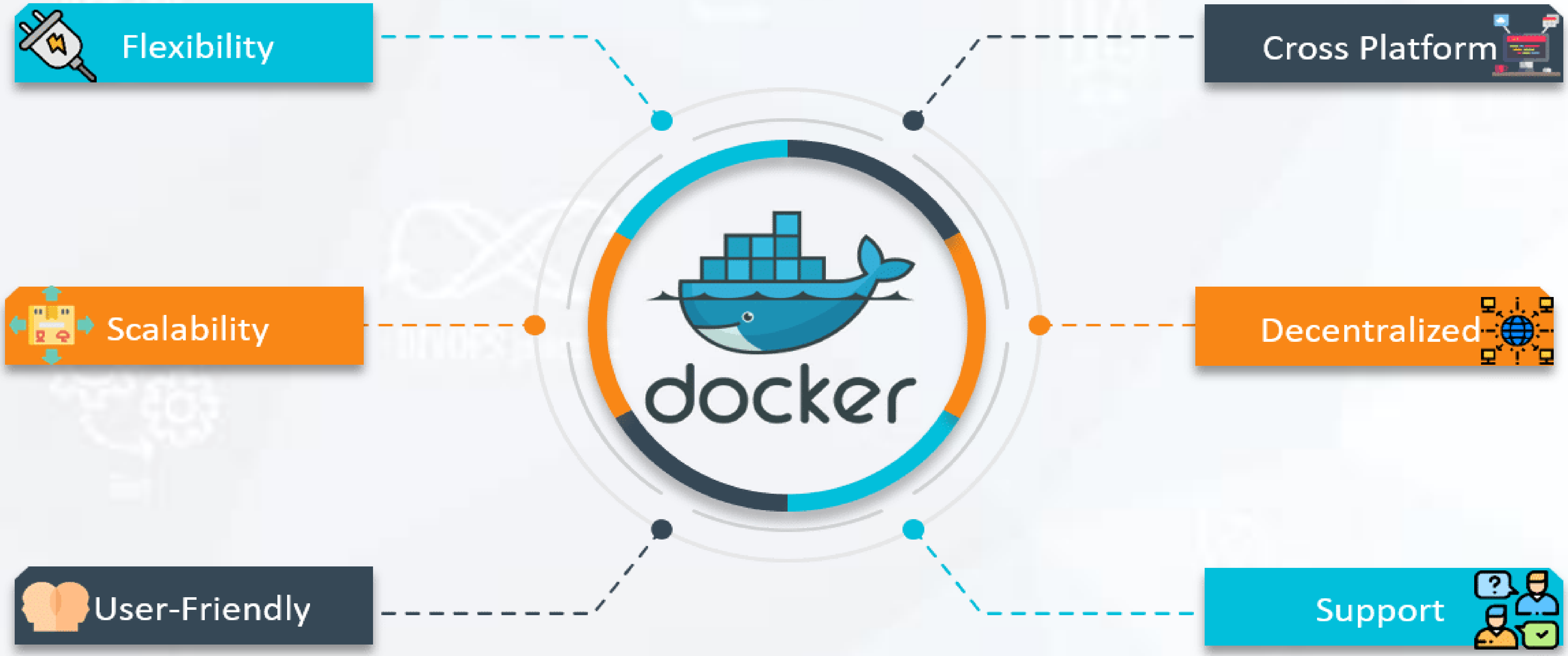
  *$ docker ps*

- Now just open your browser and go to the address localhost:8080. Here you will find the worpress app runnnig there.
- Now you can configure your wordpress app.

## Why Docker Networking

- Containers need to talk to external world.
- Reach Containers from external world to use the service that containers provides.
- Allows Containers to talk to host machine.
- Inter-container connectivity in same host and across hosts.
- Discover services provided by containers automatically.

## Goals of Docker Networking?



# Container Network Model (CNM)

- It standardizes the steps required to provide networking for containers using multiple network drivers.
- The CNM has interfaces for IPAM plugins and network plugins.
- The IPAM plugin APIs are used to create/delete address pools and allocate/deallocate container IP addresses, whereas the network plugin APIs are used to create/delete networks and add/remove containers from networks.
- A CNM has mainly built on 5 objects: Network Controller, Driver, Network, Endpoint, and Sandbox.

# Exposing Container ports

- Add an **EXPOSE** instruction in the Dockerfile
- Use the **–expose** flag at runtime to expose a port
- Use the **-p** flag or **-P** flag in the Docker run string to publish a port

# Exposing Docker ports via EXPOSE or –expose

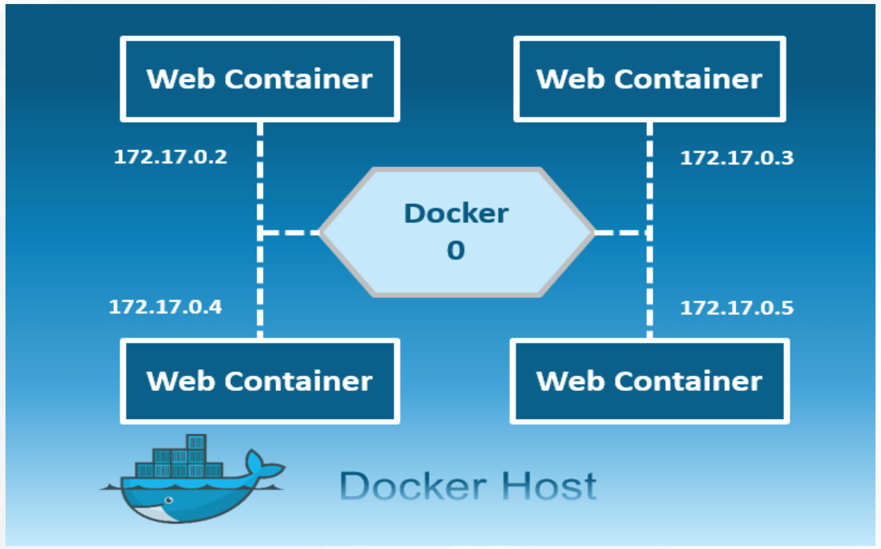
- There are two ways of exposing ports in Docker:
- Including an **EXPOSE** instruction in the Dockerfile
- Using the **–expose** flag at runtime

# Publishing Docker ports via -P or -p

- There are two ways of publishing ports in Docker:
- Using the **-P** flag - lets you publish all exposed ports to random ports on the host interfaces. It’s short for –publish-all
- Using the **-p** flag - lets you publish a container’s specific port(s) to the Docker host. It’s short for –publish

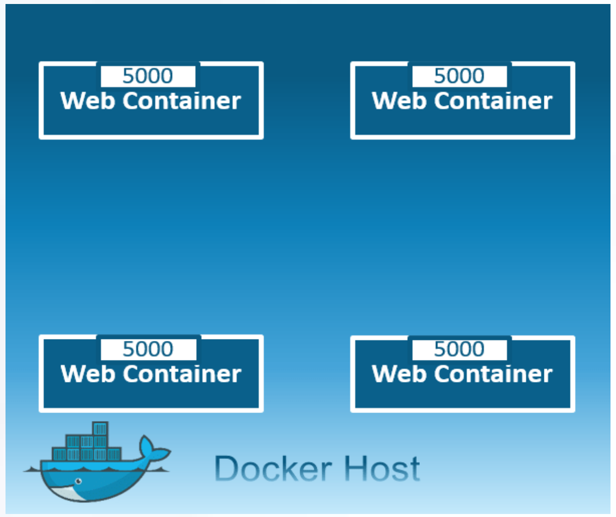
# Container Routing - Network Drivers

- There are mainly 5 network drivers: Bridge, Host, None, Overlay, Macvlan
- **Bridge:** A private default internal network created by docker on the host.
- So, all containers get an internal IP address to access each other.
- Used when your applications run in standalone containers that need to communicate.

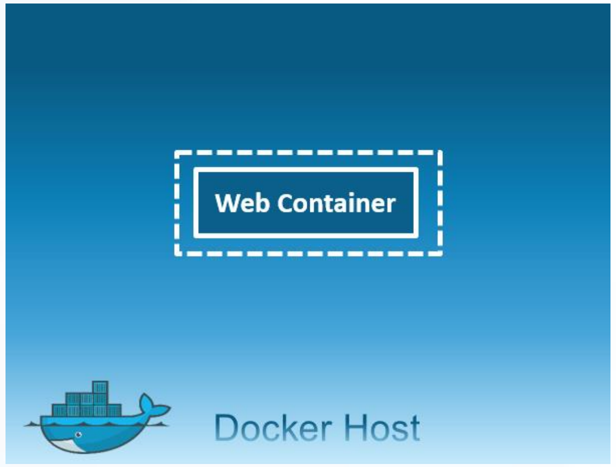


# Container Routing - Network Drivers

- **Host**: Removes the network isolation between the docker host and the containers to use the host's networking directly.

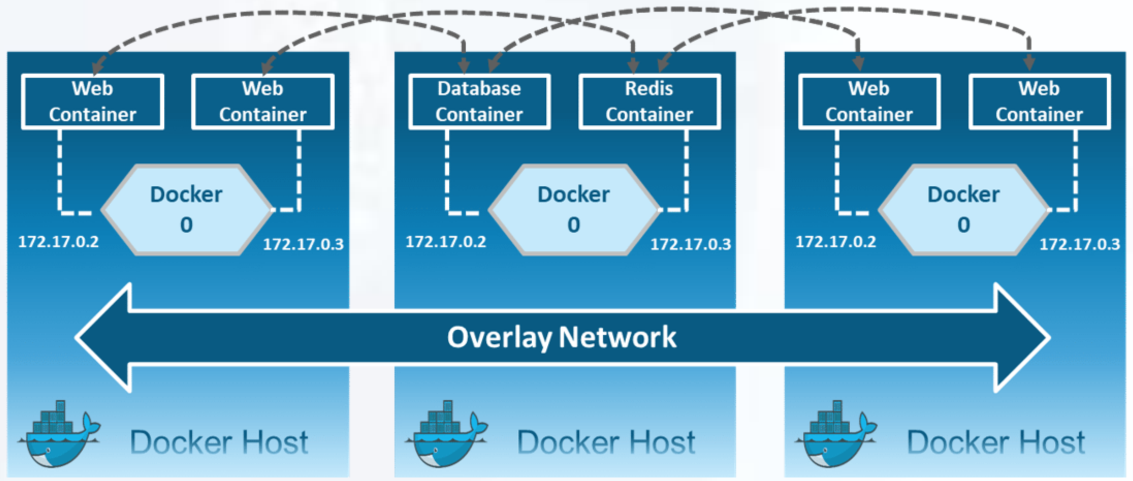


- **None**: Containers are not attached to any network and do not have any access to the external network or other containers.

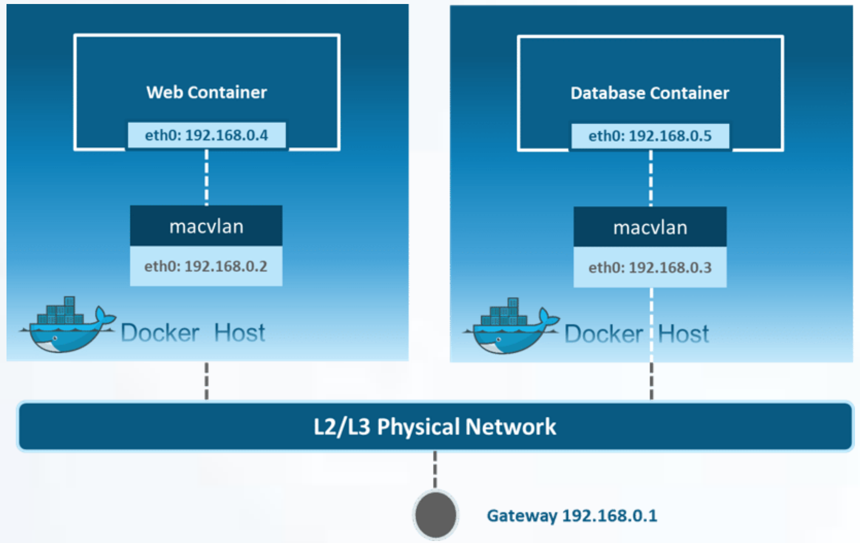


# Container Routing - Network Drivers

- **Overlay**: Creates an internal private network that spans across all the nodes participating in the swarm cluster.



- **Macvlan:** 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.



# Listing All Docker Networks

- This command can be used to list all the networks associated with Docker on the host.

  *docker network ls*

- The command will output all the networks on the Docker Host.

  *sudo docker network ls*

## Inspecting a Docker network

- If you want to see more details on the network associated with Docker, you can use the Docker network inspect command.

  *docker network inspect networkname*

  networkname − is the name of the network you need to inspect.

- The command will output all the details about the network.

  *sudo docker network inspect bridge*

## Creating Your Own New Network

- One can create a network in Docker before launching containers.

  *docker network create –-driver drivername name*

  - drivername − is the name used for the network driver.
  - name − is the name given to the network.
- The command will output the long ID for the new network.

  *sudo docker network create –-driver bridge new_nw*

# Thank You