**Docker:**

**# To install Docker**

$ yum install docker

**# To download docker image**

$ docker pull Ubuntu

**# Check docker images**

$ docker images

**# Build docker image/app**

$ docker build -t IMAGE\_NAME ( Path of the app file )

**# Check container**

$ docker ps –l

$ docker container ls

**# Run container**

$ docker run –it CONTAINER\_NAME /bin/bash

$ docker run –itd Image\_Name /bin/bash ( It makes the container and run it, also container runs in background)

$ docker attach container\_name

**# Attach running docker** ( Need to start Docker container)

$ docker attach CONTAINER\_ID

**# Login into Docker hub repo**

$ docker login

**# Tag the image**

$ docker tag image username/repository:tag

Ex: docker tag friendlyhello 37318918/get-started:part2

**# Start/Stop docker container**

$ docker container start CONTAINER\_ID

$ docker container stop CONTAINER\_ID

**# Publish the image**

$docker push username/repository:tag

**# Pull and run the image from remote repository**

$ docker run –p 4000:80 37318918/get-started:part2

**Service:**

**# Launch stack**

$ docker stack deploy –c docker-compose.yml getstartedlab

**# Check docker services**

$ docker service ls

**# Inspect the task**

$ docker inspect

**# Command to list container**

$ docker container ls –q

**# Remove stack**

$ docker stack rm getstartedlab

**Swarm:**

**# Swarm initialization**

$ docker swarm init --advertise-addr 10.0.0.xx

**# Deploy the app on swarm manager**

$ docker stack deploy –c docker-compose.yml getstartedlab

$ docker stack ps getstartedlab

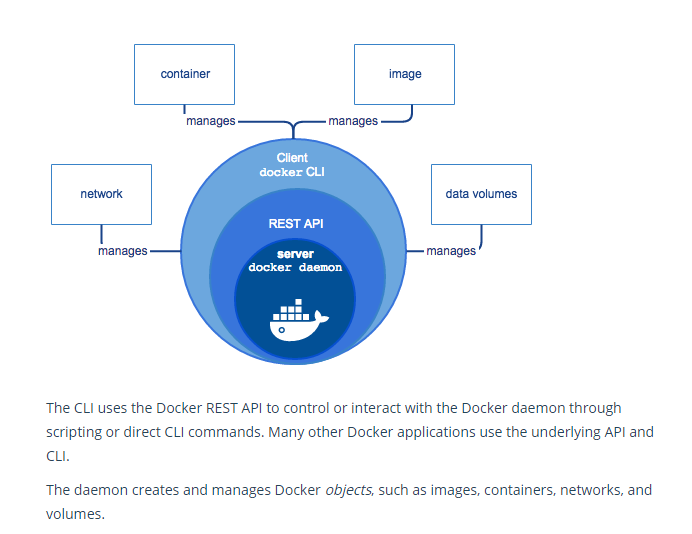
**# take don node sarm from the manager**

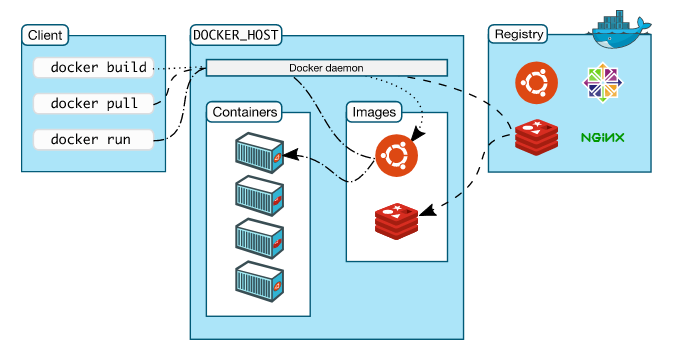
$ docker swarm leave –force

**# Command to check swarm node**

$ docker node ls

**Docker Engine:**



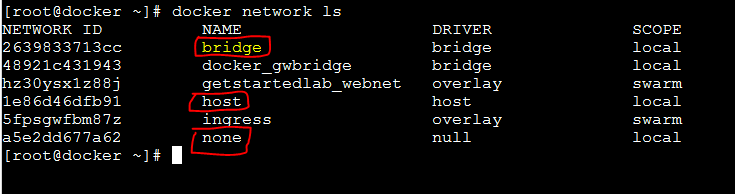
**Docker Architecture:**  
  


Note: Docker written in **GO** language.

**Docker container Networking:**

Docker creates three network automatically.

$ docker network ls

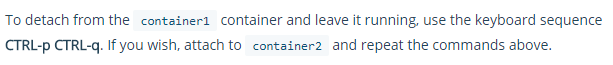


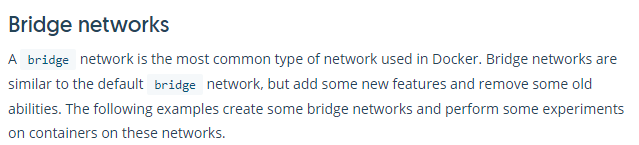
Note: By default Docker container connects Docker bridge network.

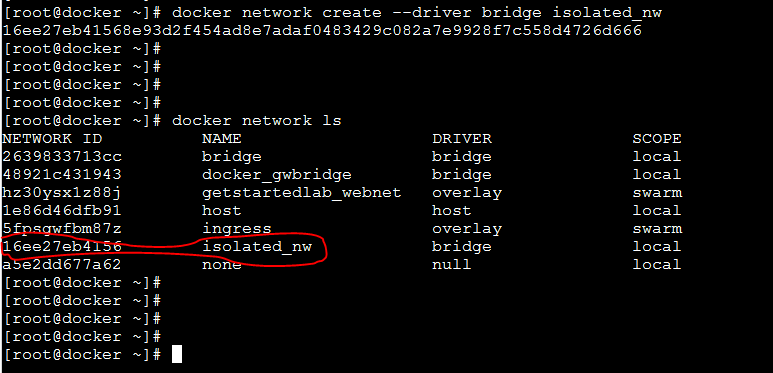
**Default bridge network:**

**# Check default bridge network, also shows containers network**

$ docker network inspect bridge

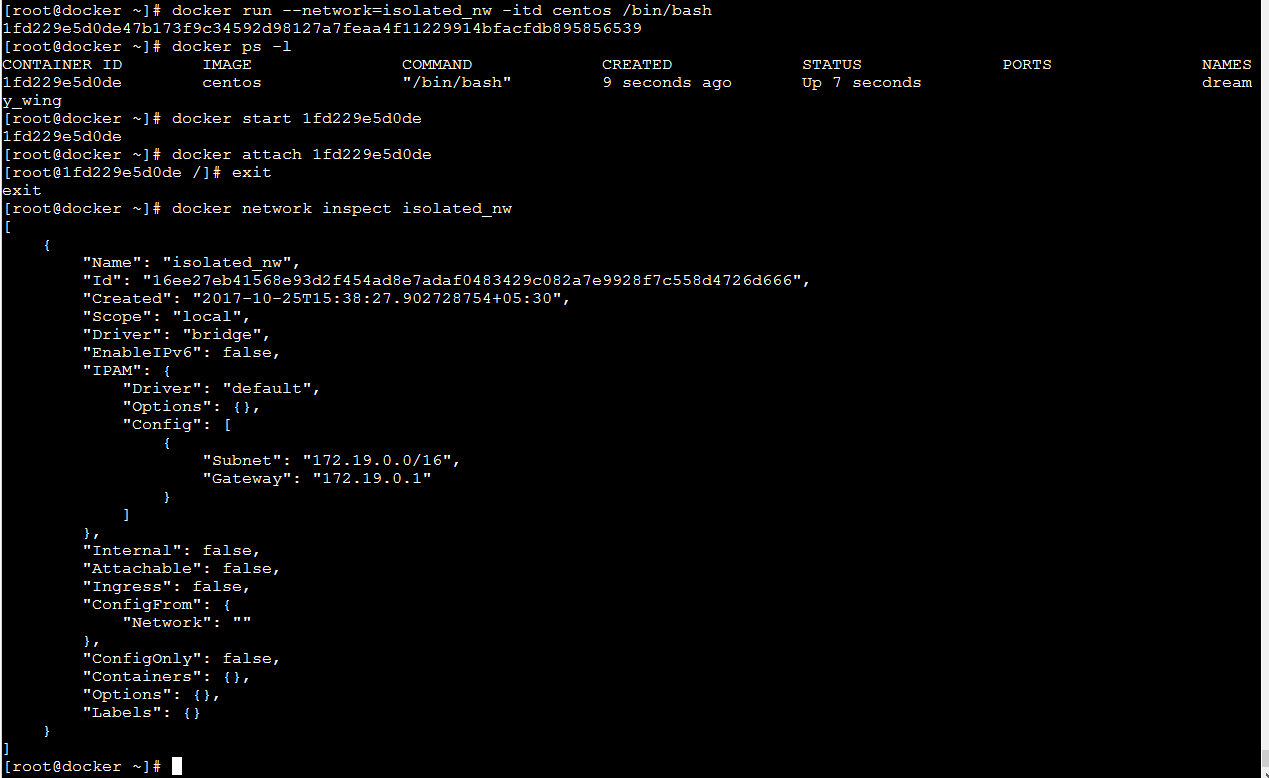






After you create the network, you can launch container using it.

$ docker run --network=isolated\_nw –itd –name=Container\_Name or ID /bin/bash



**# Exposing and publishing ports:**

EXPOSE:

You expose ports using EXPOSE. Exposing ports is a way of documenting which ports are used, but does not actually map or open any ports.  
  
PUBLISH:

You publish ports using –publish or –publish-all flag to docker run. This tells Docker which ports to open on the container’s network interface.

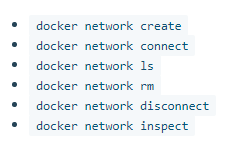
EX:  
  
$ docker run –it –d –p 80 nginx

The next example specifies that port 80 should be mapped to port 8080 on the host machine. It will fail if port 8080 is not available.

$ docker run –it –d –p 8080:80 nginx

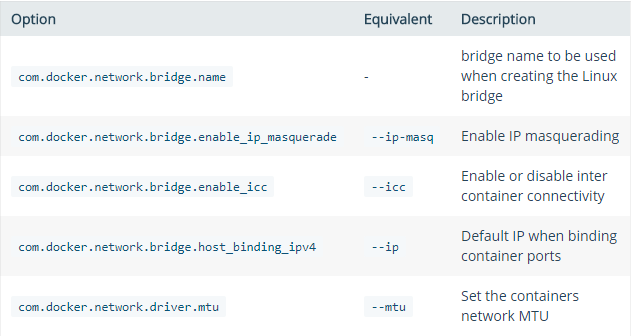
Note: It is highly recommended to define subnet while creating network otherwise Docker assigns itself subnet, which may be cause of connection overlapping and due to this failure occurs.

**Catch all command in one short:**



**Work with network commands:**





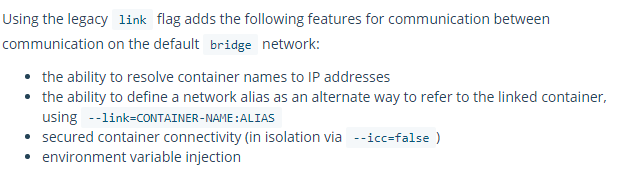
The below given example uses –o to bind to a specific address when binding ports, then uses

Docker network inspect to inspect the network, and finally attach a new container to new network.

$ docker network create -o "com.docker.network.bridge.host\_binding\_ipv4"="172.23.0.1" my-network



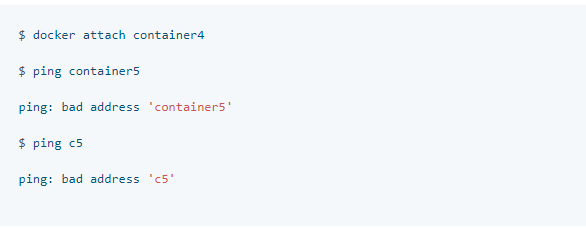
**Link container without using user-defined network.**



**# Create a container “container4 and connect to the network “isolated\_nw”. In addition, link to container5 (Which doesn’t exceed yet).**

$ docker run --network=isolated\_nw -itd --name=container4 –link container5:c5 centos

(Note: C% is alias name)

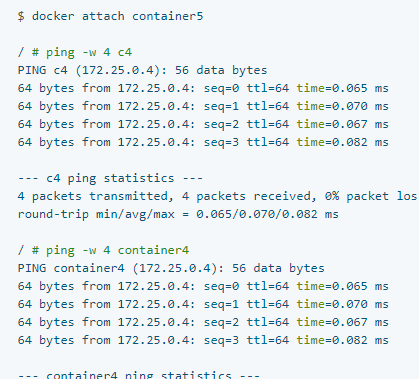


Create another container named container5, and linke it to container4 using alias c4.

$ docker run --network=isolated\_nw -itd --name=container5 --link container4:c4 contos

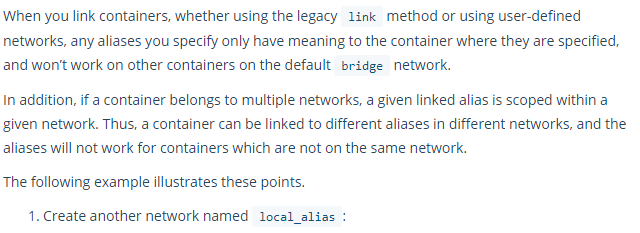
Now attach to container4 and try to ping c5 and container5.

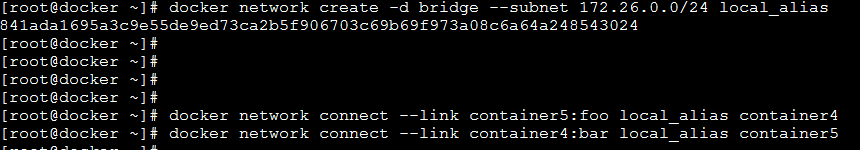




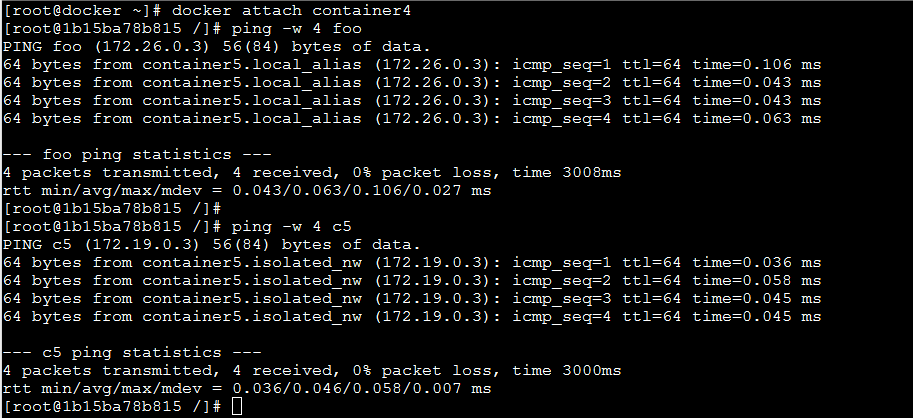


**Network alias scoping example:**





Note: foo and bar are aliases for container5 & container4

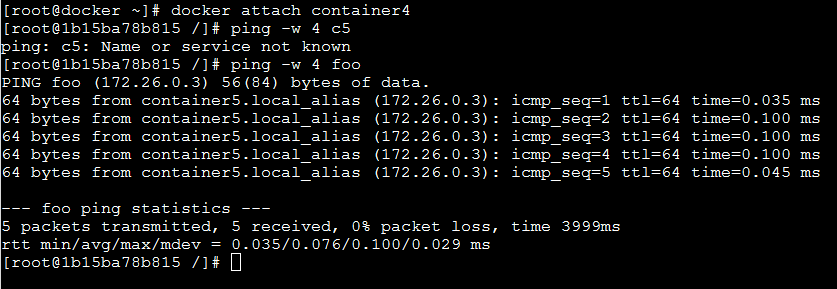


Both ping succeed, but the subnet are different, which means that the network are different.

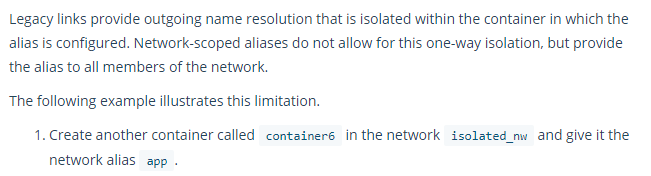
Detach from container4 and leave it running using CTRL-p CTRL-q.

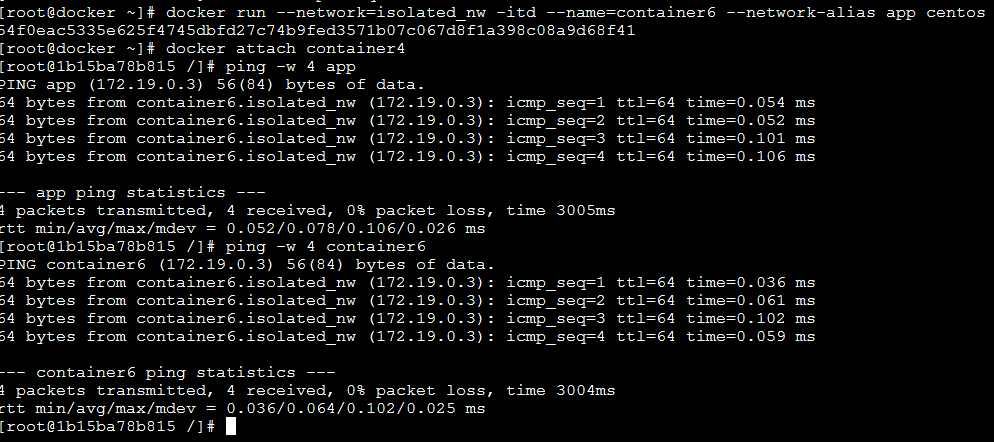
Disconnect container5 from the isolated\_nw network. Attach to container4 and try pinging c5 and foo.



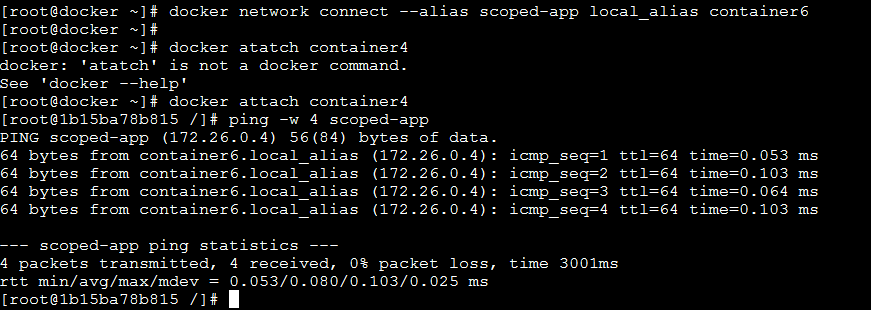


**Use Network-Scoped Aliases:**









Detach from container4 and leave it running.

