**The Seven Different Networks in Docker By Using Ubuntu VirtualBox**

**Project Description**

In this project I've used docker in Ubuntu to experience the six different Docker networks which are 1-Defualt bridge 2-User defined bridge 3 Host-Network 4-MACVLAN 5-MACVLAN 802.1q 6-IPVLAN L2 7-The NON-Network

**Project requirements**

Ubuntu OS Either in VirtualBox or in desktop locally, however, I used VirtualBox for this Project.

Note: in case of using VirtualBox you should go to settings and enable bridge network that will allow your machine to get an ip address from your home/local network.

**Phase One (Default Bridge)**

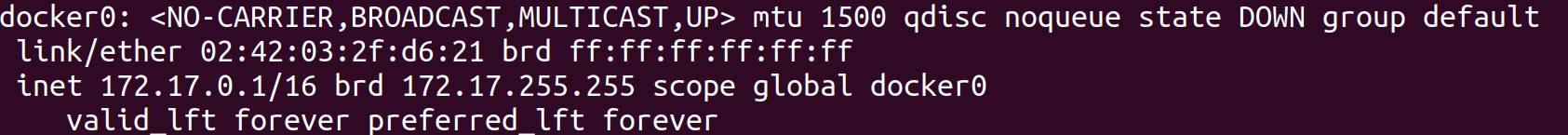
First You must install docker by typing the following command.

**sudo apt install docker.io -y**

To check if docker installed

**ip address show**

you should get the same output or similar.



docker0 is the new virtual bridge interface also it is the default interface and network for the default bridge, the default network in docker

To list docker networks type the following command

**sudo docker network ls**

A computer screen shot of white text

Description automatically generated

Drive means the network type and here for example we have bridge and type is bridge.

Now we will deploy some containers inside docker.

**sudo docker run -itd –rm – - name thor busybox**

itd = make it interactable and detached running in the background

rm = to clean it after we done with it

busybox = is the image that is used in this container

To make another container

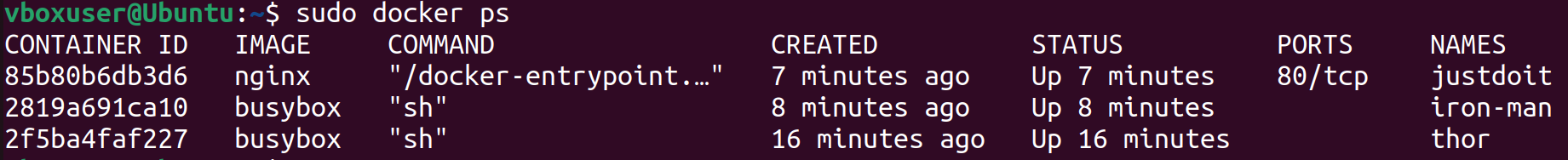
**sudo docker run -itd --rm --name iron-man busybox**

to make another container with different image

**sudo docker run -itd --rm --name justdoit nginx**

to make sure that they are all up type the following command

**sudo docker ps**

****

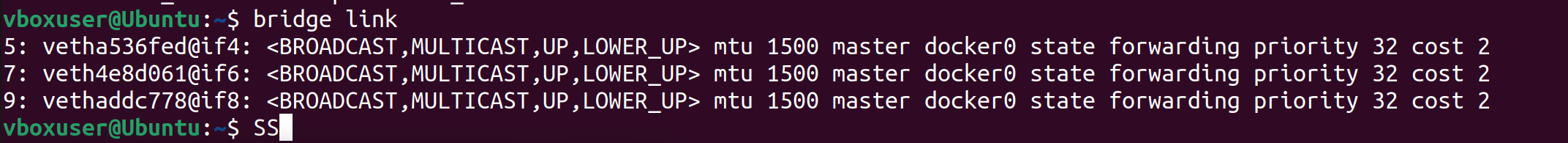
When we created the containers, we didn’t specify anything about networking which means by default it got thrown into our bridge network, thus docker automatically created three virtual ethernet interfaces and connected it or linked it to docker zero bridge it kind of acts like a switch and there is network interface for each container.

If we type ip address show it will show three new interfaces

A computer screen shot of a program

Description automatically generated

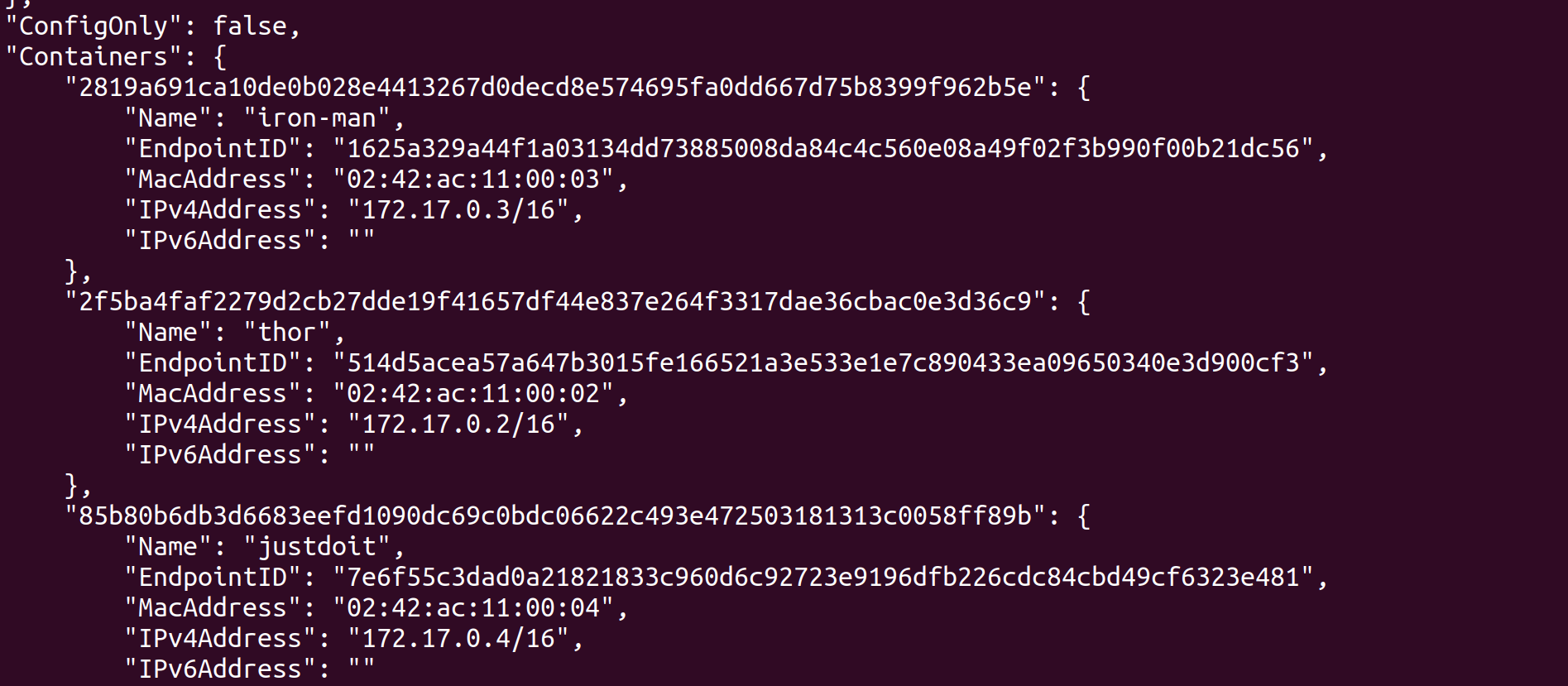
And by typing command bridge link



It shows that they are connected to docker zero.

Docker also run a DHCP server between these containers to inspect it type the following command

|  |
| --- |
| **sudo docker inspect bridge** |

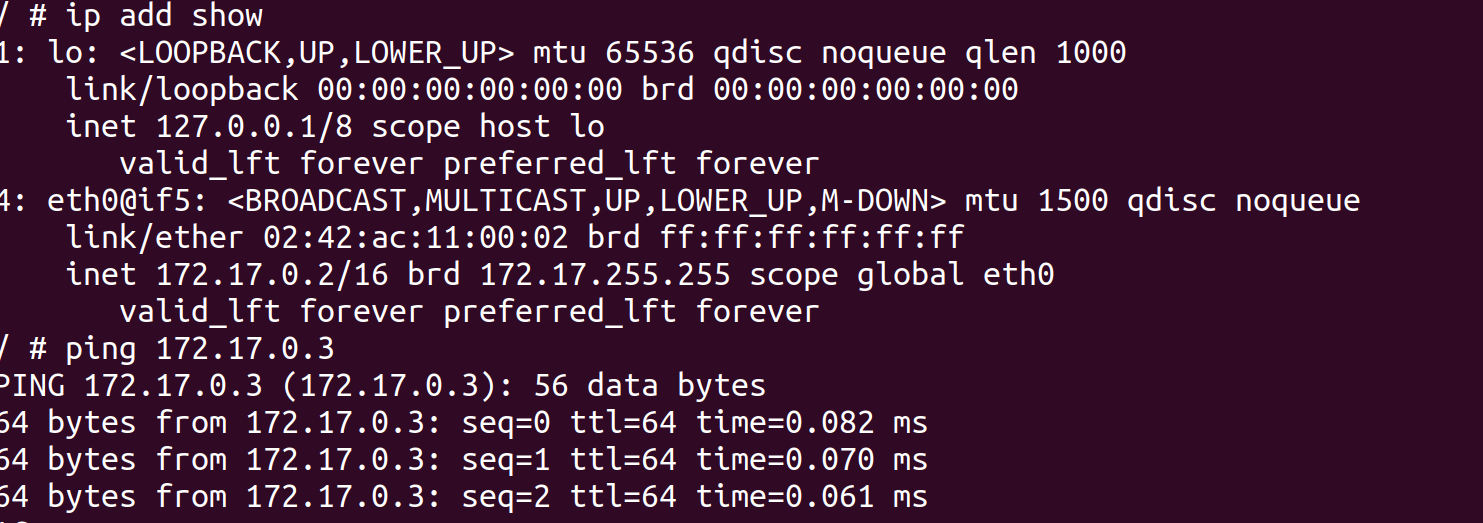


They each ip address inside network , also like any network it has DNS it takes the copy of the etc/resolve.conf from the host and put into the container so the containers can talk to each other lets try it out!!

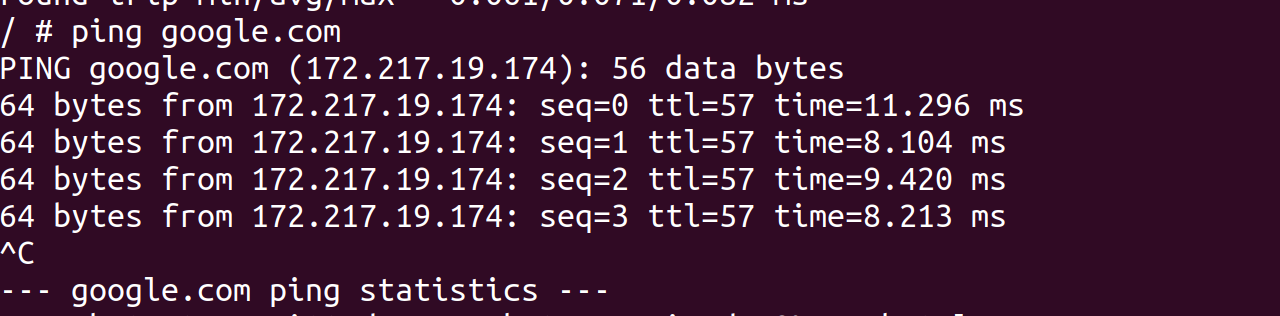
To jump into the container type the following command

**sudo docker exec -it thor sh**

and ping one of the containers for example



And probably it can connect to the internet



Because the default route and gateway for the container is docker0

A black background with white text

Description automatically generated

But how docker0 get the container ‘thor’ out to the internet because of something called NAT-Masquerade

“*by IBM Masquerade (hide) network address translation (NAT) enables you the actual address of a personal computer private. NAT routes traffic from your personal computer to your system, which essentially makes the system the gateway for your personal computer. Masquerade NAT allows you to translate multiple IP addresses to another single IP address. You can use masquerade NAT to hide one or more IP addresses on your internal network behind an IP address that you want to make public. This public address is the address to which the private addresses are translated and has to be a defined interface on your system. To be a defined interface, you must define the public address as a BORDER address*.” (IBM, n.d.)

For nignx container ’justdoit’ that is created it’s a webserver but the question this website this website is accessible in port 80 , the answer is no because in bridge network it won’t work by default , you to manually expose those ports in order to make it work.

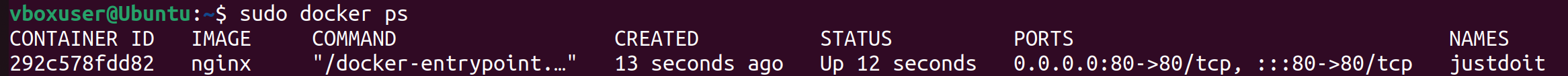
To do this we have to stop the container first by typing the following command

**sudo docker stop justdoit**

and then type the following command by telling docker to expose port 80 to host port 80

**sudo docker run -itd --rm -p 80:80 --name justdoit nginx**

to check it out



Type your bridge network Ip address on the web browser that is attached to **enp0s3** interface (in my case) and this page should appear, this webpage indicate that nginx successfully installed and running.

A black screen with white text

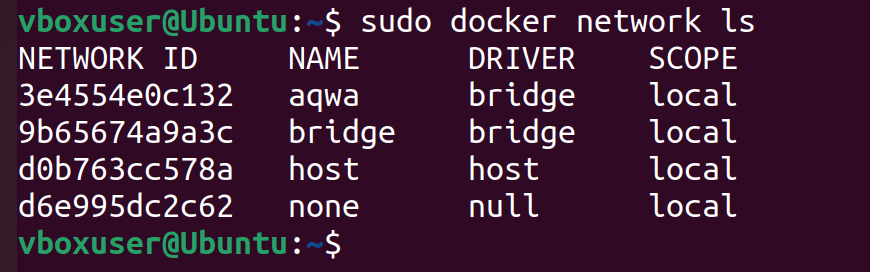
Description automatically generated

**Phase two (User-Defined Bridge network)**

Basically, it’s the same idea about the bridge but simply the user define it.

To create network by using the following command

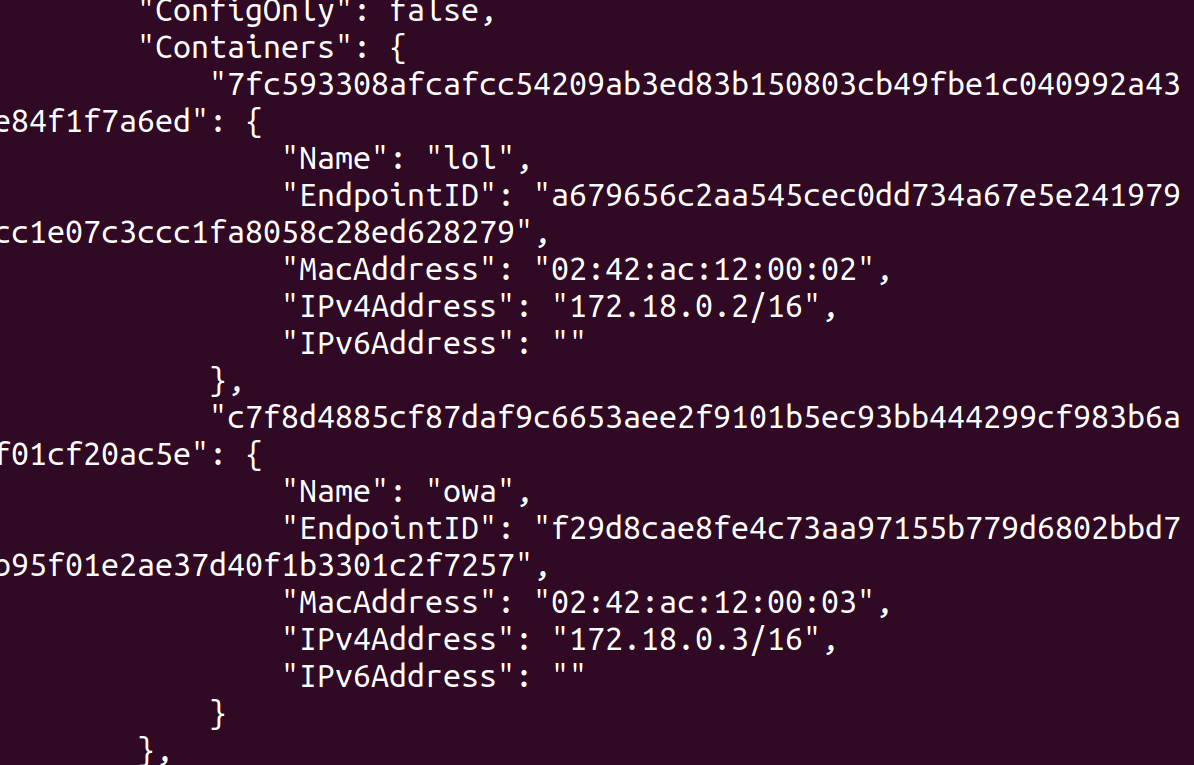
**sudo docker network create aqwa**

****

Now we create some containers inside ‘aqwa’ by typing the following command.

**sudo docker run -itd --rm --network aqwa --name owa busybox**

we just added the name of the network.



By inspecting the network lol and owa containers got different ip address than the other containers which is 172.18.0.3 (in my case ) which means that the two containers are isolated , the network aqwa is isolated from the default network which means they can’t talk to each other but in user-defined network in docker it has cool container to container DNS action.

A screenshot of a computer code

Description automatically generated

Again, by using default network can’t ping container by its name.

**Phase three (Host-Network)**

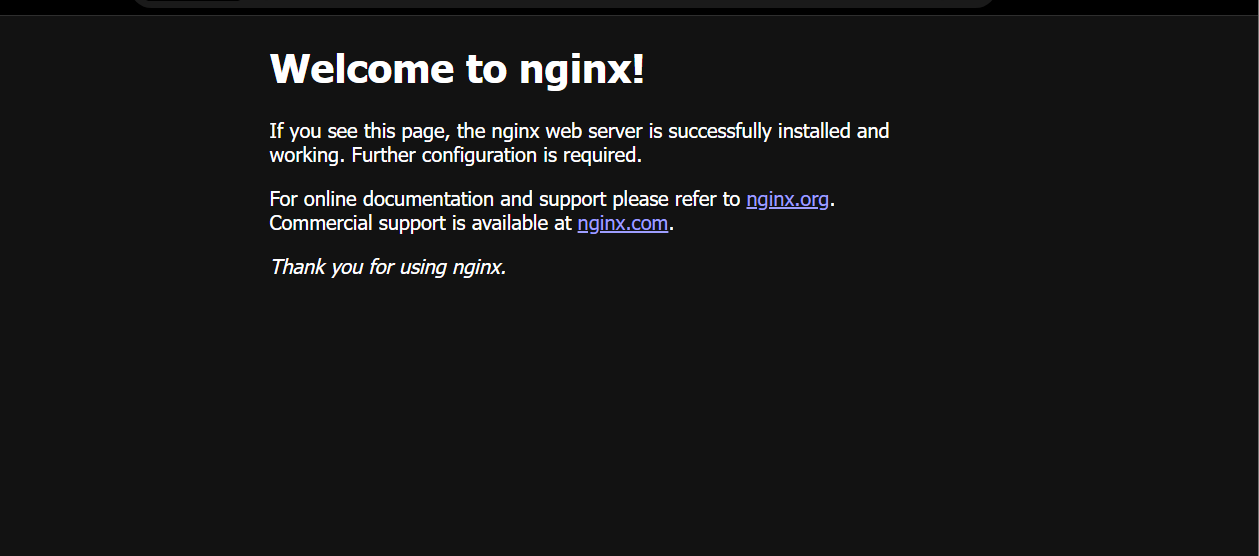
First let’s stop justdoit container by typing the following command.

**sudo docker stop justdoit**

And deploy it into the host network by typing following command.

**sudo docker run -itd --rm --network host --name justdoit nginx**

once this done it just bums off the host it shares the ip address and ports so now the container can be reached without exposing any ports



It’s running because it’s running as a regular application in the host even though it’s a container. The issue resides that is no isolation in this network because it’s directly connected to the host.

**Phase Four (MacVlan Network)**

This network allows to connect the docker containers into the physical network. Thus, the containers will be connected directly to the network to the local network.

So let’s move into the commands

Now we need to specify network as the pervious but with driver type MacVlan and defining the subnet and the default gateway by the following command. (type yours instead of x’s)

**sudo docker network create -d macvlan --subnet x.x.x.x/24 \**

**> --gateway x.x.x.x \**

**> -o parent=enp0s3 \**

**> newaqwa**

Deleting existing containers

**sudo docker stop thor iron-man**

Throwing containers inside newaqwa network

**sudo docker run -itd --rm --network newaqwa --ip x.x.x.x name thor busybox**

here we have to specify the ip address manually make sure it’s not used in your network

the downside of this network is that it won’t ping even the default gateway. Because each docker container getting it’s own Mac Address so the network may not able to have multiple mac addresses on one switch port. It may have port security that can have only one or two mac addresses on the one port so we need to enable promiscuous mode in order to make by command and in virtual box.

So basically, this network has all the benefits of default gateways except it’s connected to the local network

Let’s try out if we deployed nginx server inside we can reach it without exposing ports

**sudo docker run -itd --rm --network newaqwa --ip x.x.x.x --name spider-man nginx**

A screenshot of a computer screen

Description automatically generated

**Phase Five (MacVlan(802.1q) Network )**

In this phase is very similar to the pervious phase but here we can define sub-interfaces.

Let’s how to do it.

First we have to stop the all containers containers

**sudo docker stop thor iron-man spider-man**

and remove the network ‘newqwa’

**sudo docker network rm newaqwa**

create the network again

**sudo docker network create -d macvlan --subnet 192.168.20.0/24 --gateway 192.168.20.1 -o parent=enp0s3.20 macvlan20**

**A screen shot of a computer

Description automatically generated**

After trunking setup everything would set-up

**Phase six (IPVLAN Network)**

It solves the problem with MacVlan that every container has its own mac address it sounds good in theory, but it messes with promiscuous mode in the switch. But IPVALNS enable the containers to connect directly to the network and getting a real ip address but they allow the host to share it’s mac address with containers so the container will have the mac address match exactly to the host but it still have ip addresses on the network.

To create IPVLAN network

**sudo docker network create -d ipvlan --subnet 192.168.0.0/24 --gateway 192.168.0.1 -o parent=enp0s3 newearth**

throw container inside it

**sudo docker run -itd --rm --network newearth \**

**> --ip 192.168.0.222 \**

**> --name thor busybox**

A black background with white text

Description automatically generated

A black screen with white text

Description automatically generated

They do share the same mac address

**Also we have IPVLAN L3 which is in layer three (Network layer) so each container will connect to the host as the host acting like a router , so the containers cannot reach the internet unless we allow them to do so , but here is the powerful of this phase is control is the name of the game, so by modifying router traffic management we can control those network. Also it’s not relying on layer2 so avoiding a lot of problem in there.**

This how to do it

**sudo docker network create -d ipvlan --subnet 192.168.94.0/24 -o parent=enp0s3 -o ipvlan\_mode=l3 \**

**> --subnet 192.168.95.0/24 \**

**> newasgard**

So here is the difference that we didn’t define default gateway because the gateway is the parent interface and we have to define ipvlan\_mode which is layer 3 so

Throw containers inside the network

**sudo docker network create -d ipvlan --subnet 192.168.94.0/24 -o parent=enp0s3 -o ipvlan\_mode=l3 \**

**> --subnet 192.168.95.0/24 \]**

**> newasgard**

**sudo docker run -itd --rm --network newasgard --ip 192.168.94.8 --name thor busybox**

**sudo docker run -itd --rm --network newasgard --ip 192.168.94.7 --name mjonir busybox**

**sudo docker run -itd --rm --network newasgard --ip 192.168.95.7 --name loki busybox**

**sudo docker run -itd --rm --network newasgard --ip 192.168.95.8 --name odin busybox**

we just created four containers with two different subnets, but they can ping each other because they share same parent interface, however they can’t ping the outside network and this should be solved by modifying router settings by adding static routing.

This one of the best practice of networks.

**Phase Seven (The Non-network)**

It’s nothing basically. And by default, it’s created.



Also, we can throw a container inside it

A computer screen shot of a program

Description automatically generated

Nothing to show all it has is loopback.