Kindly complete the pre-test from your RPS Lab machine

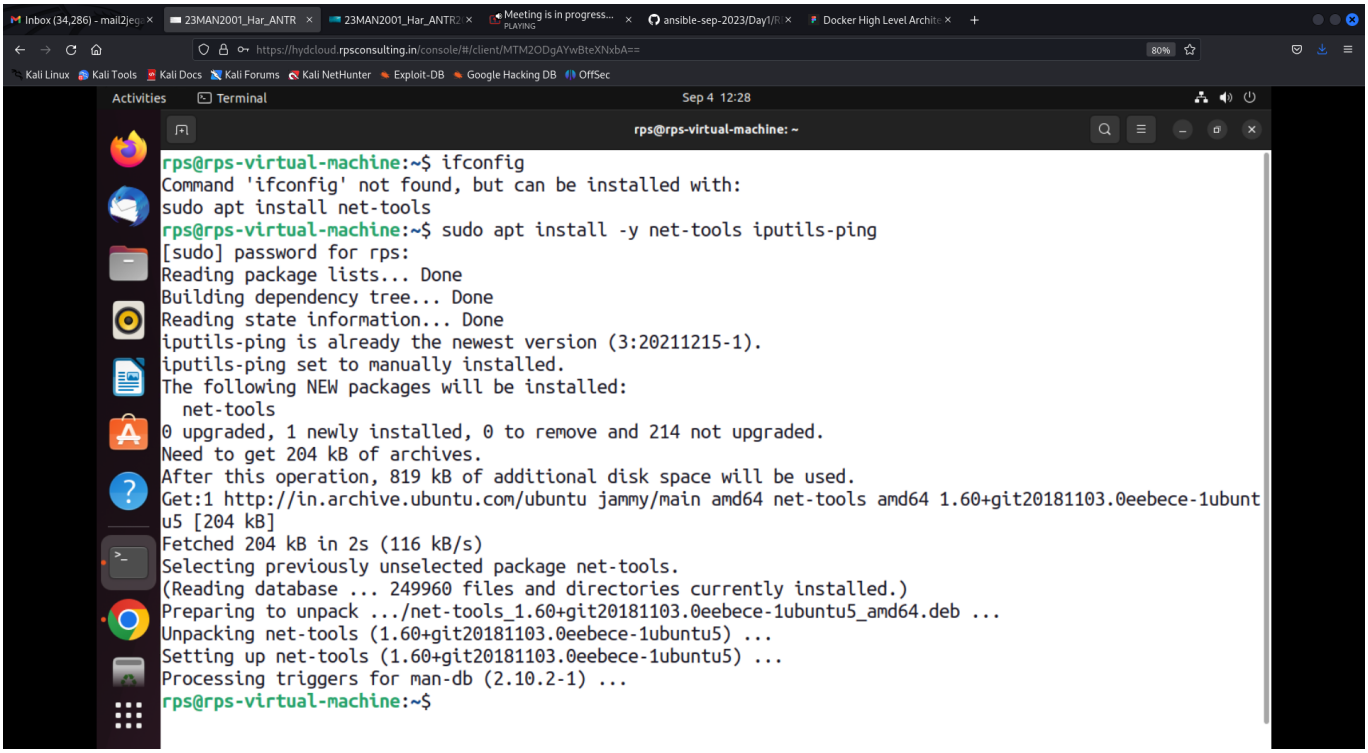
- Once you login to your RPS Lab machine, you will see a web browser with the below URL kept open
- Copy/Paste from your work/personal machine to/from RPS Lab machine is disabled as per your company policy
- Kindly provide your full name and personal email id while registering for the pre-test
- Once you complete the pre-test, kindly confirm me either orally or leave a message via WebEx chat if you have access to WebEx chat
- Once everyone completes the pre-test we can start the training until then I'll be on mute
- For any lab acces issues, please write to the RPS Tech Team user logged in via WebEx chat, you may have to share the RPS cloud user name and describe the technical challenge to

```
https://app.mymapit.in/code4/tiny/VxOr50
```

Installing network tools in Ubuntu lab machine

```
sudo apt install -y net-tools iputils-ping
```

Expected output



Docker Overview

What are Bootloaders?

- are system utilities that helps in setting up dual/multi boot in your laptop/desktop/workstation
- i.e you can install 3~4 Operating System on your laptop/desktop/workstation
- but only one OS can be active at any point of time
- If you switch on your laptop with once the BIOS POST(Power On Self Test) completes, the BIOS will instruct the CPU to execute the system utility at Sector 0, Byte 0 in your Hard Disk. The Sector 0, Byte 0 is 512 bytes and it is referred as Mater Boot Record (MBR)
- the Boot Loader System utility is installed on the MBR of your Hard disk
- Boot Loader will scan your Hard disk searching for Operating Systems, if it detects more than one Operating System, then it gives a menu for you to choose to select which OS you wish to boot into
- Let's say you booted machine into Windows, but you need Ubuntu, you first have to shutdown Windows and boot into Ubuntu and vice versa
- Only one OS can be active at any point of time
- Examples
    - LILO (Linux Loader - opensource)
    - GRUB1 - opensource
    - GRUB2 - opensource
    - For Mac users, BootCamp you can boot into windows on a Mac OS-X

What is Hypervisor?

- many OS can be running side-by-side on the same laptop/desktop/workstation/servers
- i.e many OS can be active at the same time
- Virtualization technology
- this is a combination of Hardware + Software technology
- Processors
    - AMD
        - AMD-V - Virtualization Feature set supported in the Processor
    - Intel
        - VT-X - Virtualization Feature set supported in the Processor
- there are two types of Hypervisors
    1. Type1 - used in Servers (Bare-metal Hypervisors - can be installed directly on a HW)
    2. Type2 - used in Laptops/Desktops/Workstations (requires Host OS - Unix/Linux/Windows/Mac)
- Examples VMWare
    - Workstation - Linux & Windows(type2)
    - Fusion - Mac OS-X (type2)
    - vSphere/vCenter - Bare-metal Hypervisor (type 1) Oracle virtualbox (type2 - Free) Parallels (type2 - Mac OS-X) Microsoft Hyper-V
- each Virtual Machine(VM)/Guest OS represents one fully functional Operating System
- each Virtual Machine(VM) gets its own IP address
- each Virtual Machine has its own shell, port range and Network stack

In the absence of Virtualization technology, how many minimal physical servers are required to support 1000 active OS?

1000 Physical servers are required

With Virtualization technology, how many minimal physical servers are required to support 1000 OS?

250 Physical Servers - each supporting 4 Virtual Machines

Processors with multiple CPU Cores

- AMD Processors it comes with 128 CPU Cores

Processors comes in 2 form factors

- SCM (Single Chip Module) - One IC will host one Processor
- MCM (Multi Chip Module) - Once IC will host multiple Processors

Server Motherboards with 4 Sockets

Let's say MCM Processor with 4 Processor per IC Each Processor supporting 128 CPU Cores 1 Socket = 4 Processors with 128 CPU Cores on each Processor = 4 x 128 x 4 = 2048

Hyperthreading - each Physical CPU Core is capable of running 2 threads parallely Hence, Hypervisor softwares will see each Physical CPU Cores as 2 Virtual Cores

2048 x 2 = 4096 virtual cores

MacBook Pro with Quad Core Processor, 16 GB RAM and 1 TB HDD

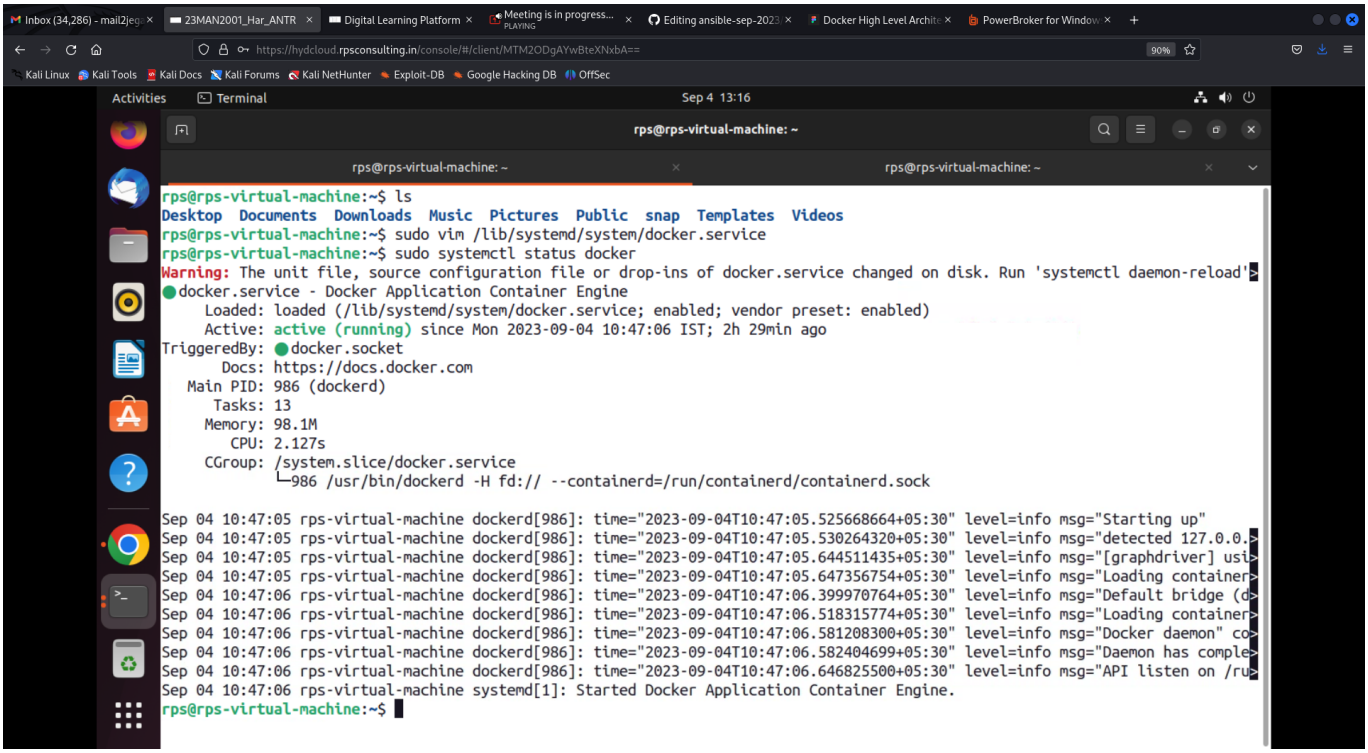
- I used to 2 VMS + 1 Host OS(Mac)

Enable the Docker Server REST API or TCP Socket

Check the status of docker service to find the path of docker service configuration file

```
sudo systemctl status docker
```

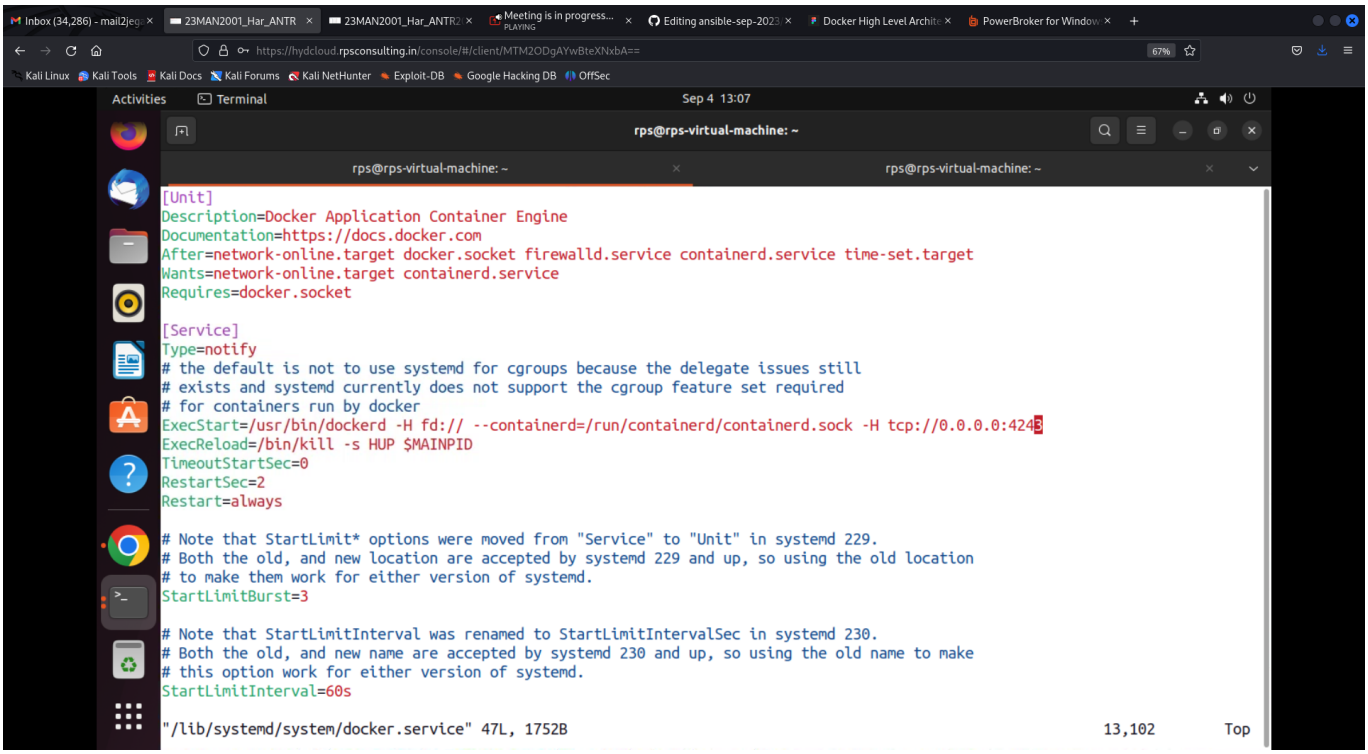
Expected output



You can edit the /lib/systemd/system/docker.service file as Administrator and append tcp://0.0.0.0:4243 at line 13 as shown below

```
sudo vim /lib/systemd/system/docker.service
```

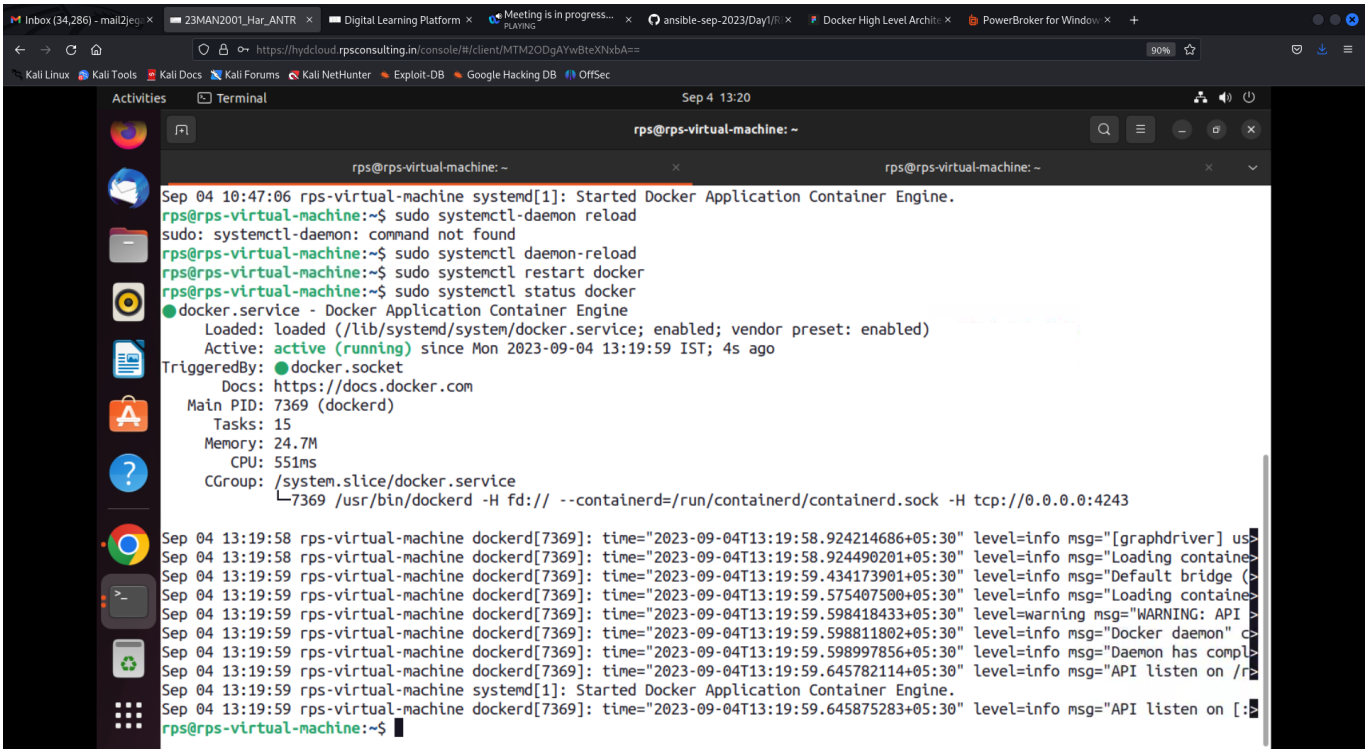
Expected output



You need to restart docker service to apply the config changes

```
sudo systemctl daemon-reload
sudo systemctl restart docker
sudo systemctl status docker
```

Expected output



What is Docker?

- is an application virtualization technology
- each container represents one application
- container doesn't OS Kernel
- container doesn't get its own dedicated Hardware resources
- each container has its own shell, port range and Network stack
- each container gets an IP address
- each container has a file system

Hypervisor vs Docker

- Virtual Machine represents an Operating System
- Container represents an application process
- Virtual machine gets its own hardware resources (CPU,RAM,Disk,etc)
- Containers running on the same system shares the Hardware resources on the underlying OS
- Container technolgy is an application virtualization technology
- container is light-weight virtualization technology
- Virtualization is heavy-weight - because each VM requires dedicated hardware resources
- Containers need Operating System
- containers are not a replacement for Virtualization(Virtual Machines)
- they both can be used together, they are complementing technology

Container Images

- specification of docker containers
- any software tools that you need on the containers must be installed on the Container image
- any number of containers can be creating using a Docker Image

Containers

- is a running instance of Docker Image
- containers get IP address, file system, network stack

Linux Kernal Features that enables the Container technology

1. Namespace - containers running on same machines are isolated by using namespace
2. Control Groups (CGroups)
    - we can apply resource quota restricts for each containers
    - we can restrict
        - how many CPU cores a container can use at the max
        - how much RAM a container can use at the max
        - how much storage a container can use at the max

How containers are different from normal application process

- containers also are normal application process, the only thing is it runs in a separate network namespace
-

Why Docker?

- high-level user-friendly software that makes things easier to manage images and containers
- internally docker depends on containerd
- containerd internally depends on container runtime called runC

What is a Container Engine?

- a user-friendly high-level software that uses other tools to manage container images/containers
- Examples
    - Docker
    - Podman

What is a Container Runtime?

- is a softwares that manages containers
    - create a container
    - list containers
    - delete containers
    - stop/start/restart/kill/abort containers
- container runtimes are low-level software utility which are not user-friendly, hence they are not normally used by end-users

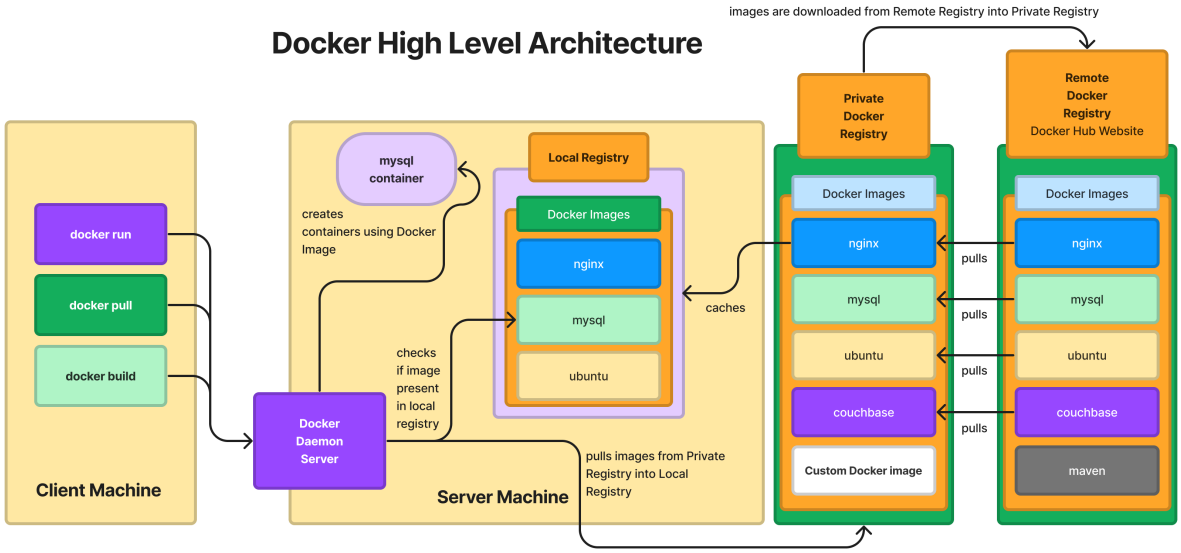
- these software are used by Container engines to manage containers Examples
- runc
- CRI-O

What type of applications can be containerized?

- Web Servers
- Application Servers
- DB Servers
- REST API
- SOAP API
- Microservices

Hypervisor High-Level Architecture

Docker High-Level Architecture



Docker Commands

Demo - Connecting your local Docker client to a remote Docker server

You need to do this on your local machine

```
export DOCKER_HOST=tcp://<ip-address-of-machine-where-docker-server-is-running>:4243
docker images
```

Lab - Finding docker version

```
docker --version
```

Expected output

```
┌──(jegan㉿tektutor.org)-[~/ansible-sep-2023]
└─$ docker --version
Docker version 20.10.25+dfsg1, build b82b9f3
```

Lab - Listing docker images from Local Docker Registry

```
docker images
```

Expected output

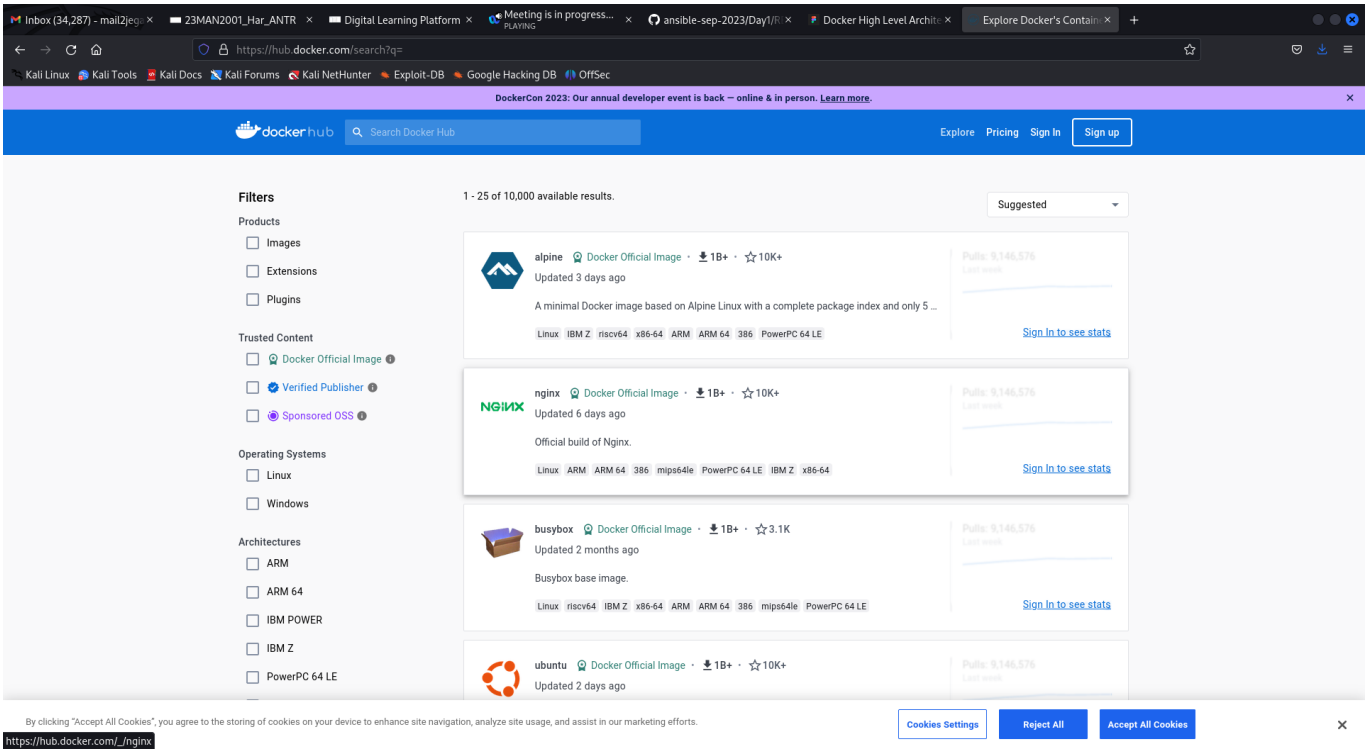
```
┌──(jegan㉿tektutor.org)-[~/ansible-sep-2023]
└─$ docker images
REPOSITORY                     TAG         IMAGE ID       CREATED
SIZE
tektutor/ansible-centos-node   latest      67e2c2ee6256   2 days ago
428MB
tektutor/ansible-ubuntu-node   latest      a3f39aea7460   2 days ago
220MB
centos                         7.9.2009    eeb6ee3f44bd   23 months ago
204MB
ubuntu                         16.04       b6f507652425   2 years ago
135MB
```

Lab - Docker Remote Registry

Open the below URL on your Ubuntu Cloud machine chrome web browser, click on Exlore

```
https://hub.docker.com
```

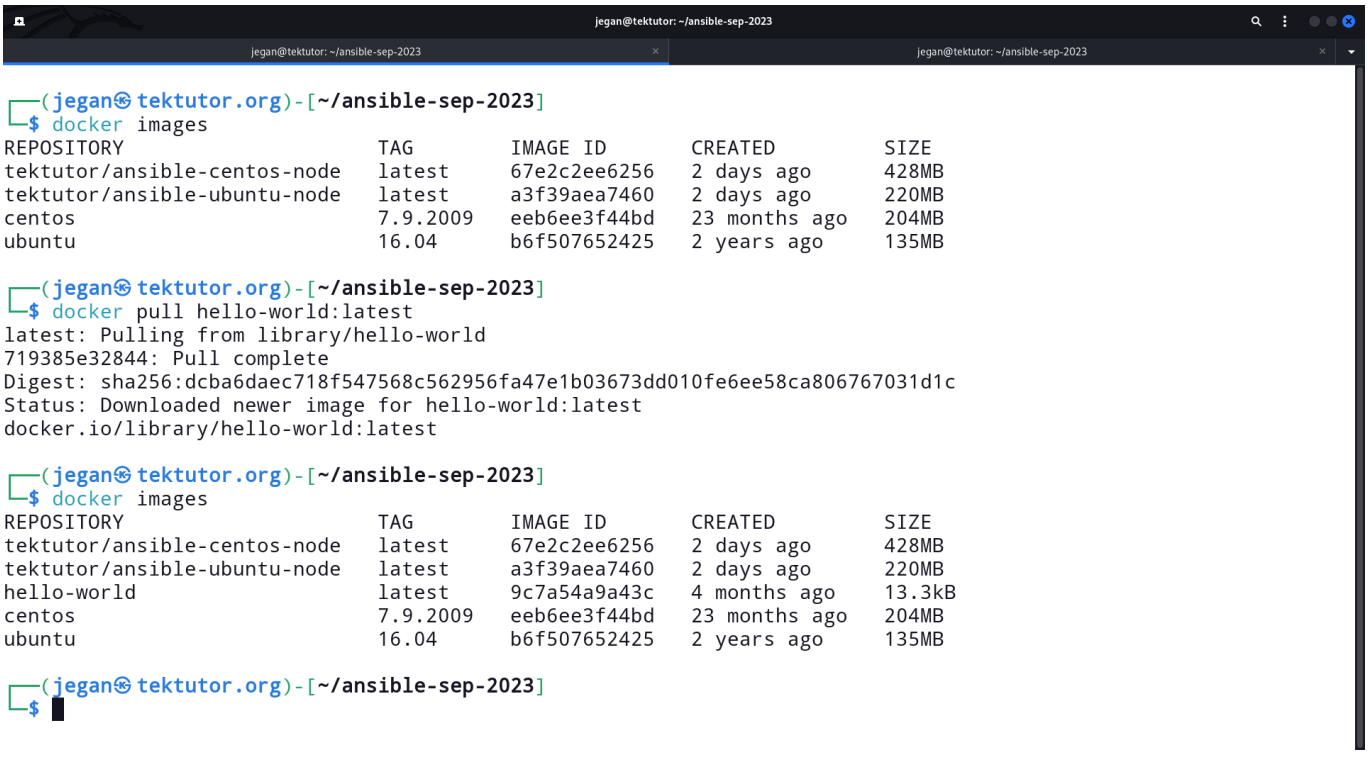
Expected output



Lab - Downloading the hello-world docker image from Docker Hub Remote Registry to your local docker registry

```
docker pull hello-world:latest
```

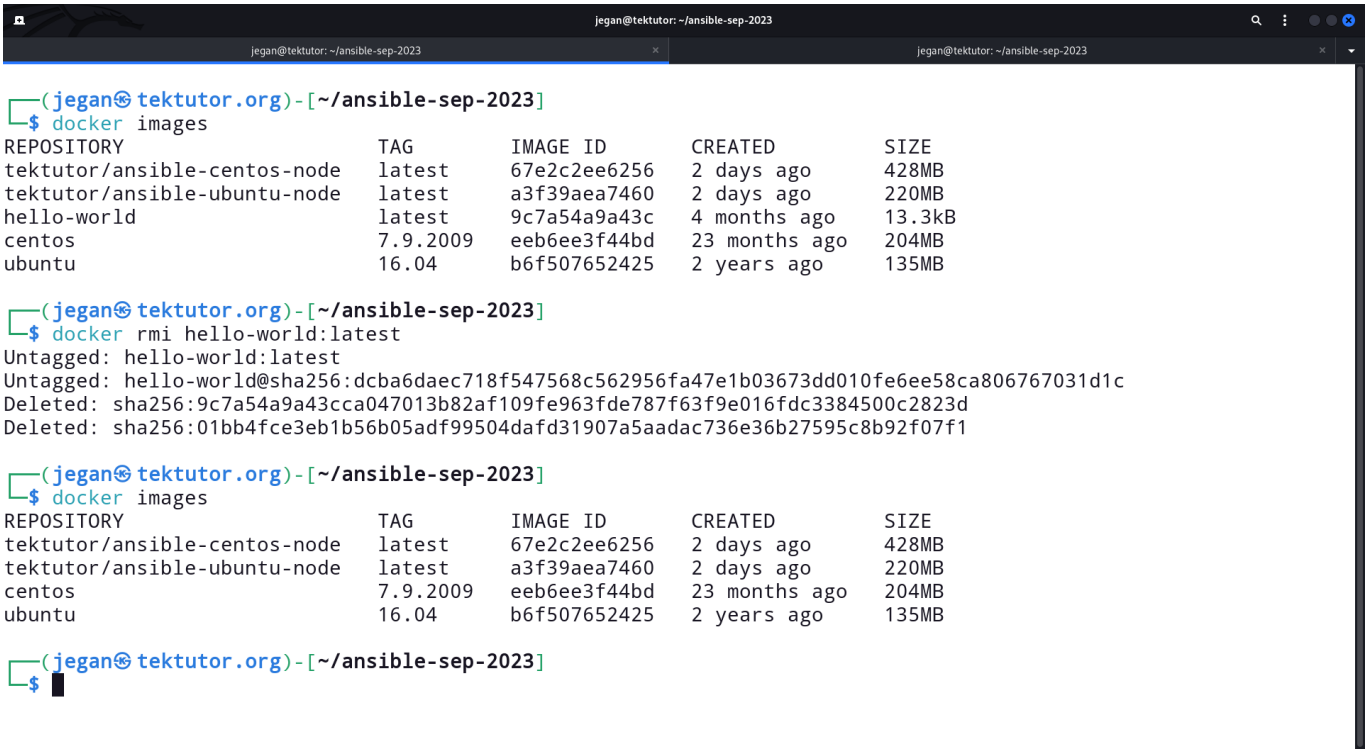
Expected output



Lab - Deleting a docker image from Local Docker Registry

```
docker rmi hello-world:latest
```

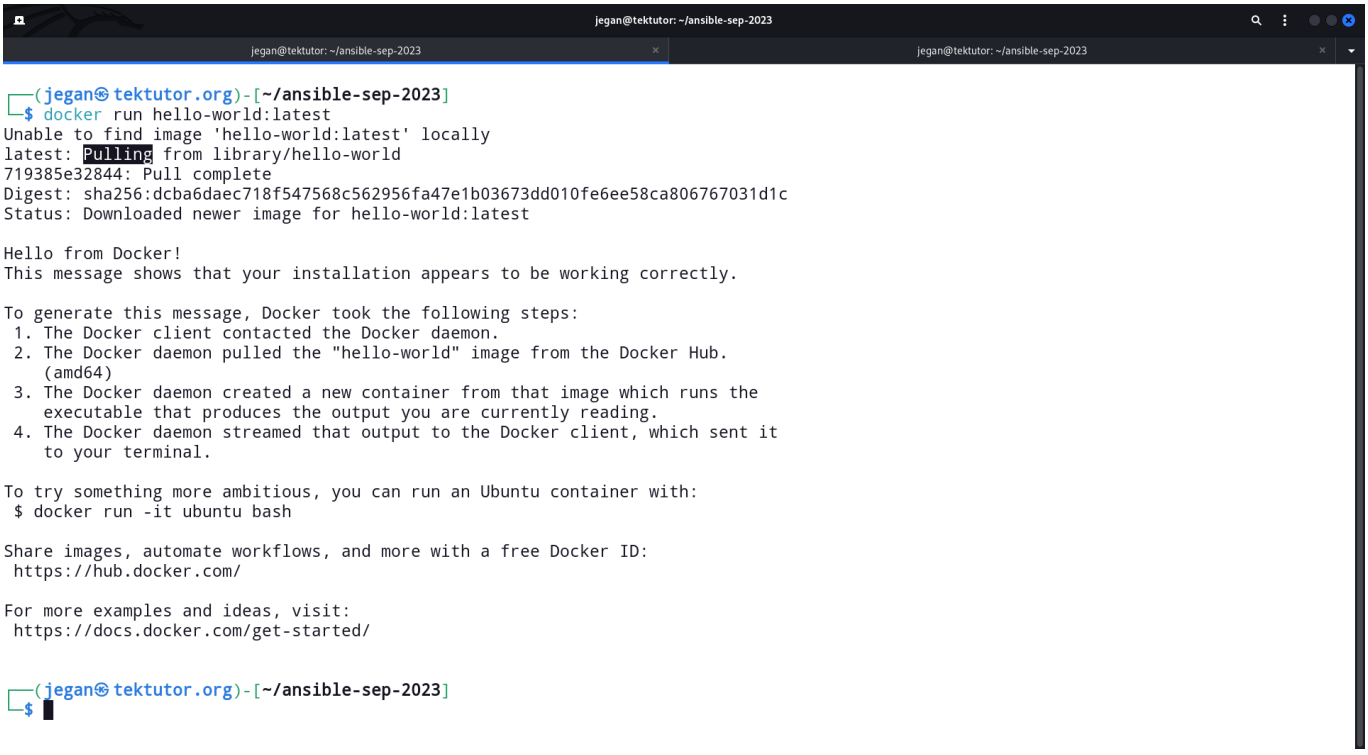
Expected output



Lab - Creating your first container

```
docker run hello-world:latest
```

Expected output



Lab - Listing the docker containers

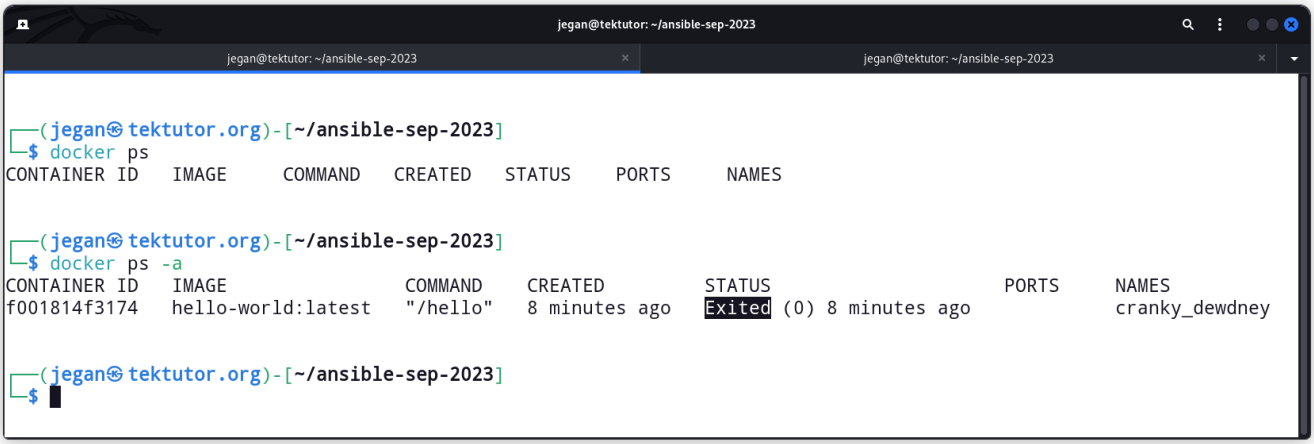
List only running containers

```
docker ps
```

List all containers

```
docker ps -a
```

Expected output



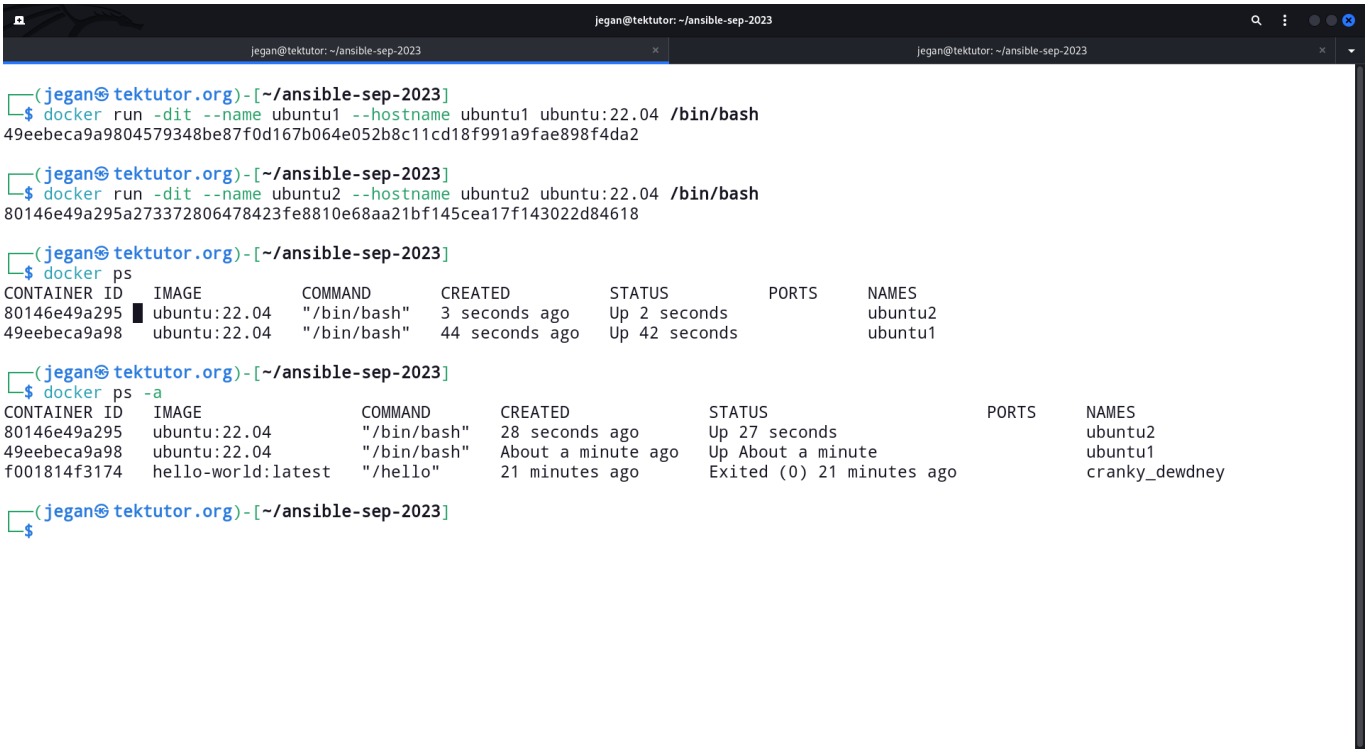
Lab - Creating container and run them in background

```
docker run -dit --name ubuntu1 --hostname ubuntu1 ubuntu:22.04 /bin/bash
docker run -dit --name ubuntu1 --hostname ubuntu1 ubuntu:22.04 /bin/bash
docker ps
docker ps -a
```

Let's understand the above run command

```
run - creates and start the container
dit - means deattached/backgorund and it stands for interactive terminal
```

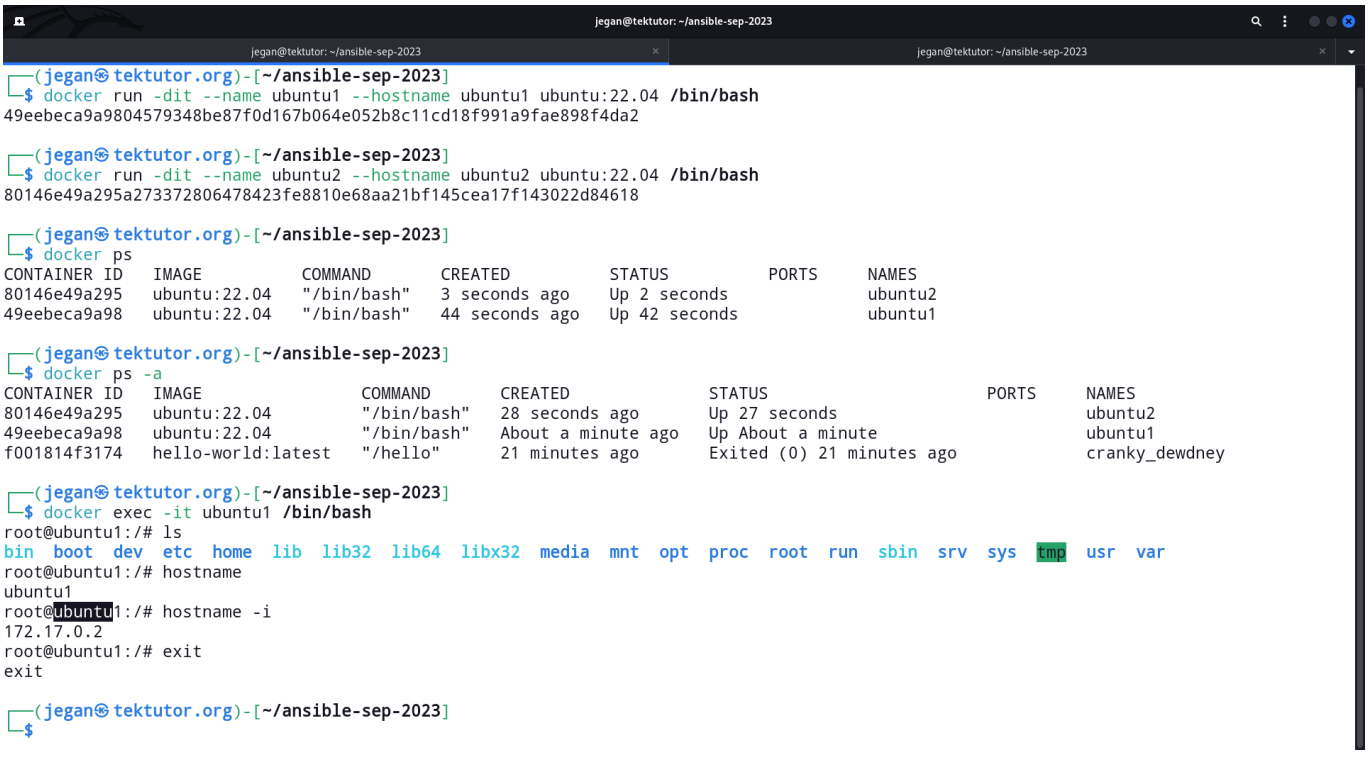
Expected output



Lab - Getting inside a container shell

```
docker exec -it ubuntu1 /bin/bash
ls
hostname
hostname -i
exit
```

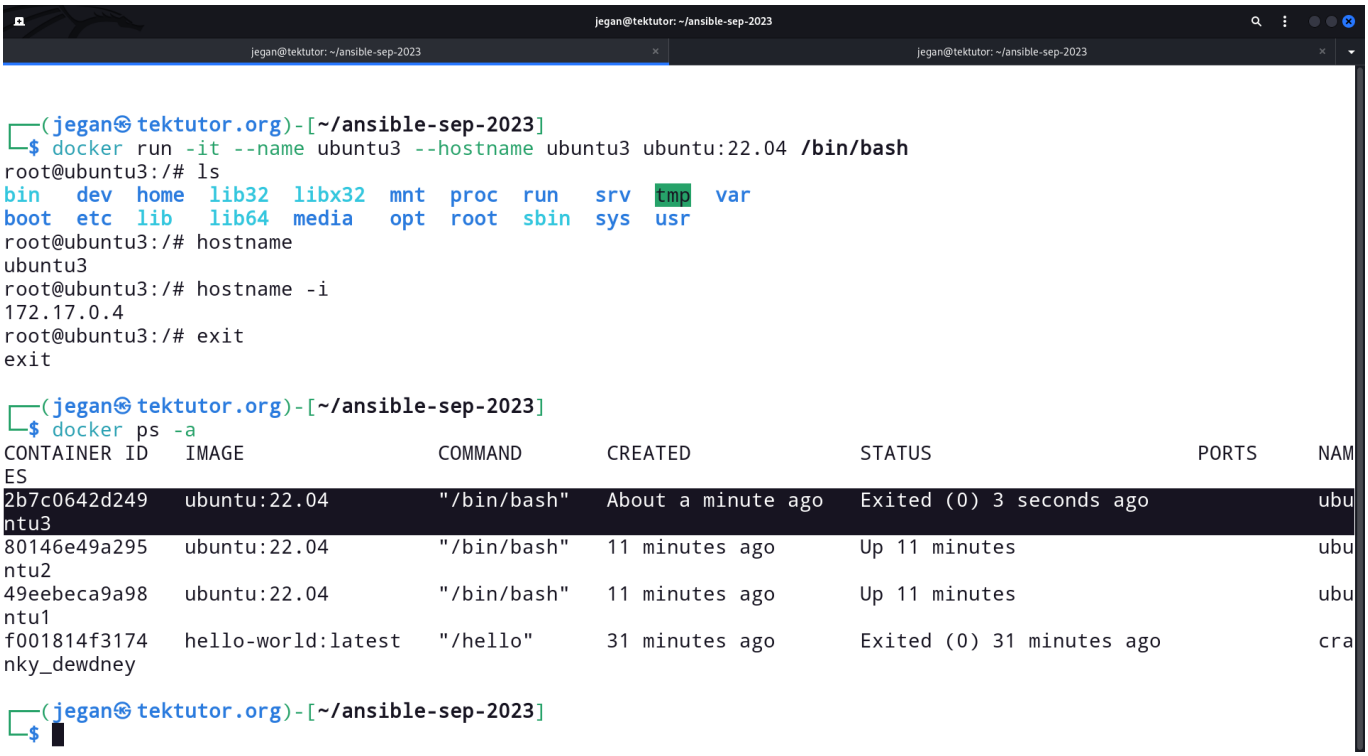
Expected output



Lab - Creating a container in the foreground/interactive mode

```
docker run -it --name ubuntu3 --hostname ubuntu3 ubuntu:22.04 /bin/bash
ls
hostname
hostname -i
exit
```

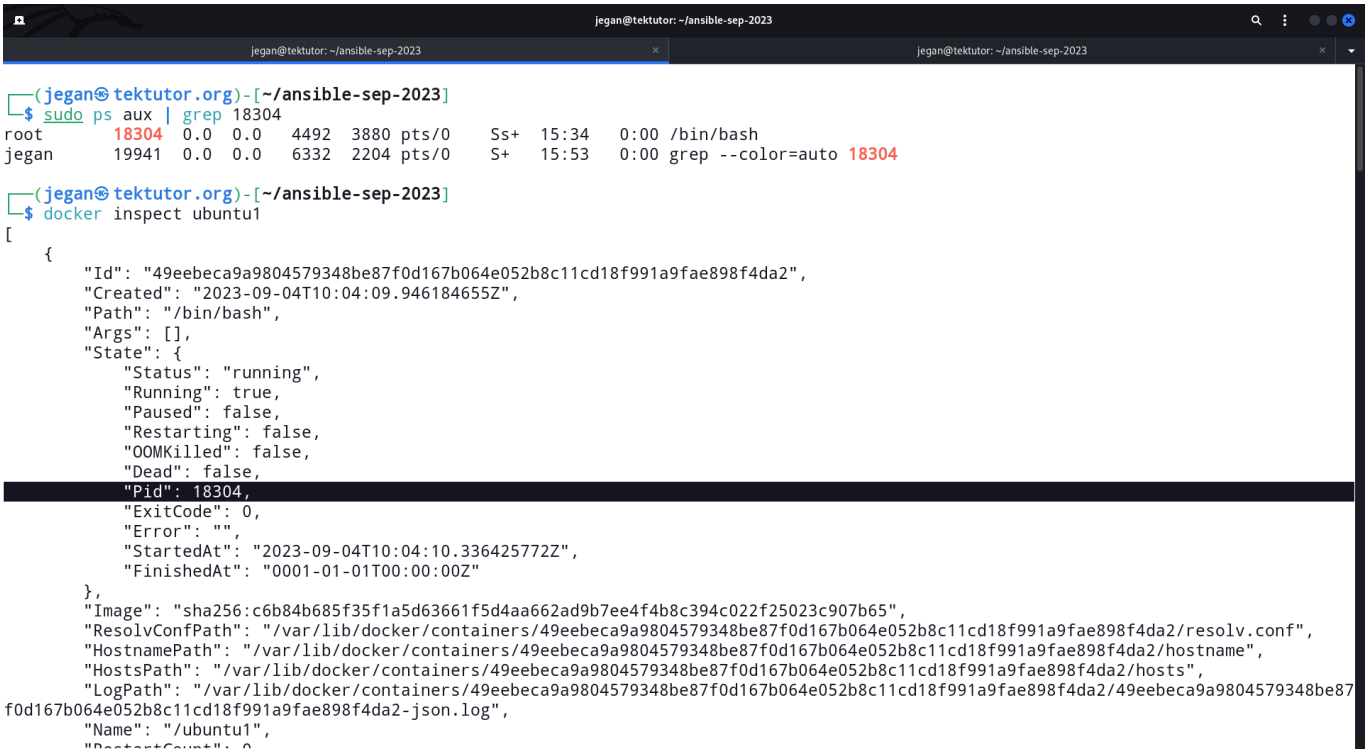
Expected output



Lab - Finding more details about a container

```
docker inspect ubuntu1
```

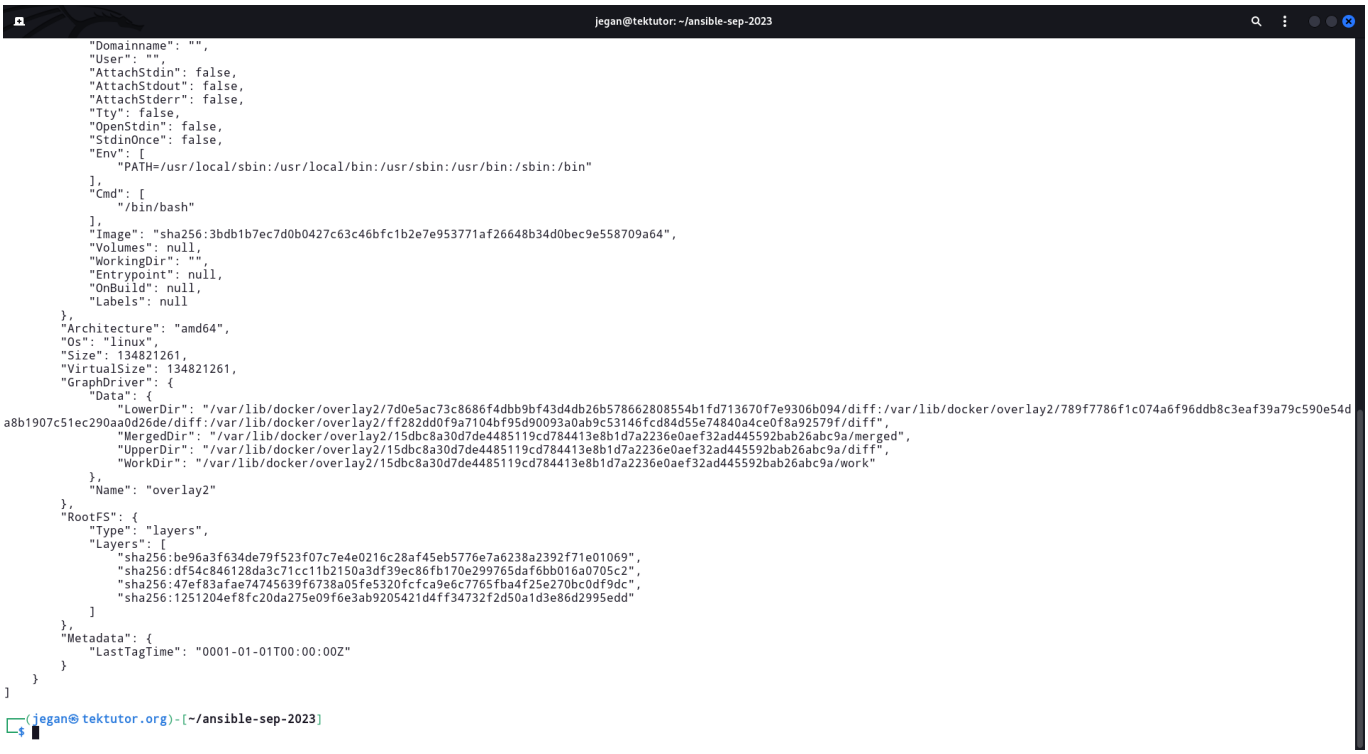
Expected output



Lab - Finding details of Docker image

```
docker image inspect ubuntu:16.04
```

Expected output

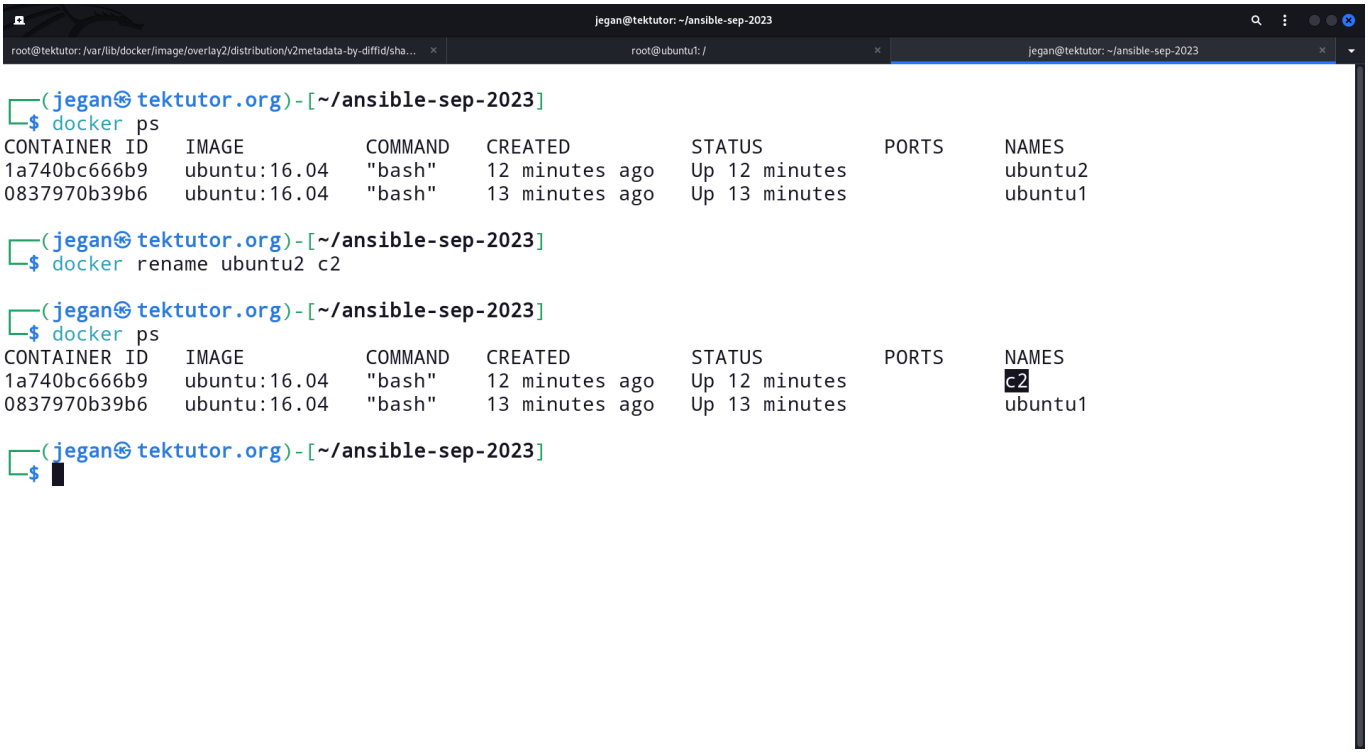


Lab - Renaming a container

```
docker rename <current-container-name> <new-name>
docker ps
docker rename ubuntu2 c25a18dbd4922c   nginx:latest   "/docker-entrypoint.
…"   9 minutes ago    Up 1 second     0.0.0.0:8001->80/tcp, :::8001->80/tcp
lb

docker ps
```

Expected output

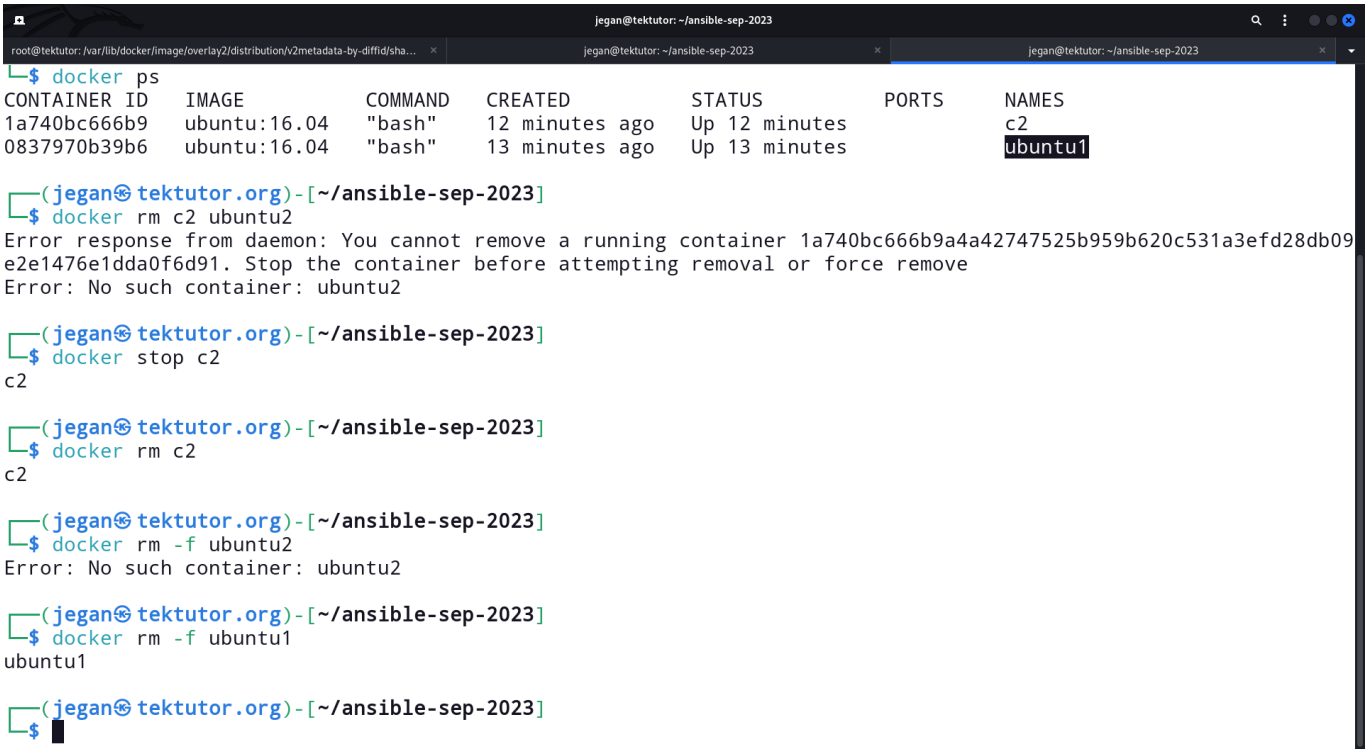


Lab - Deleting containers

```
docker ps
docker stop c2
docker rm c2

docker rm -f ubuntu1
docker ps
```

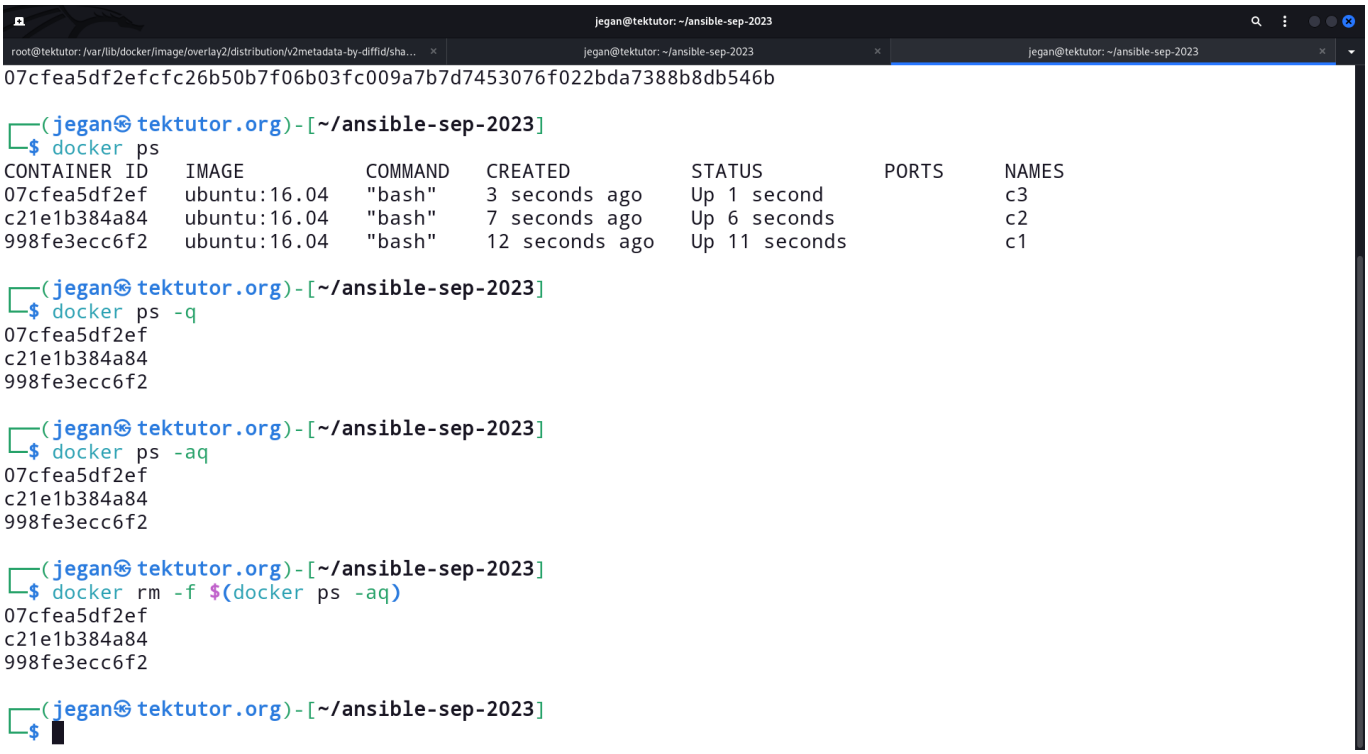
Expected output



Lab - Deleting multiple containers without calling out their names individually

```
docker ps
docker ps -q
docker ps -aq
docker rm -f $(docker ps -aq)
```

Expected output

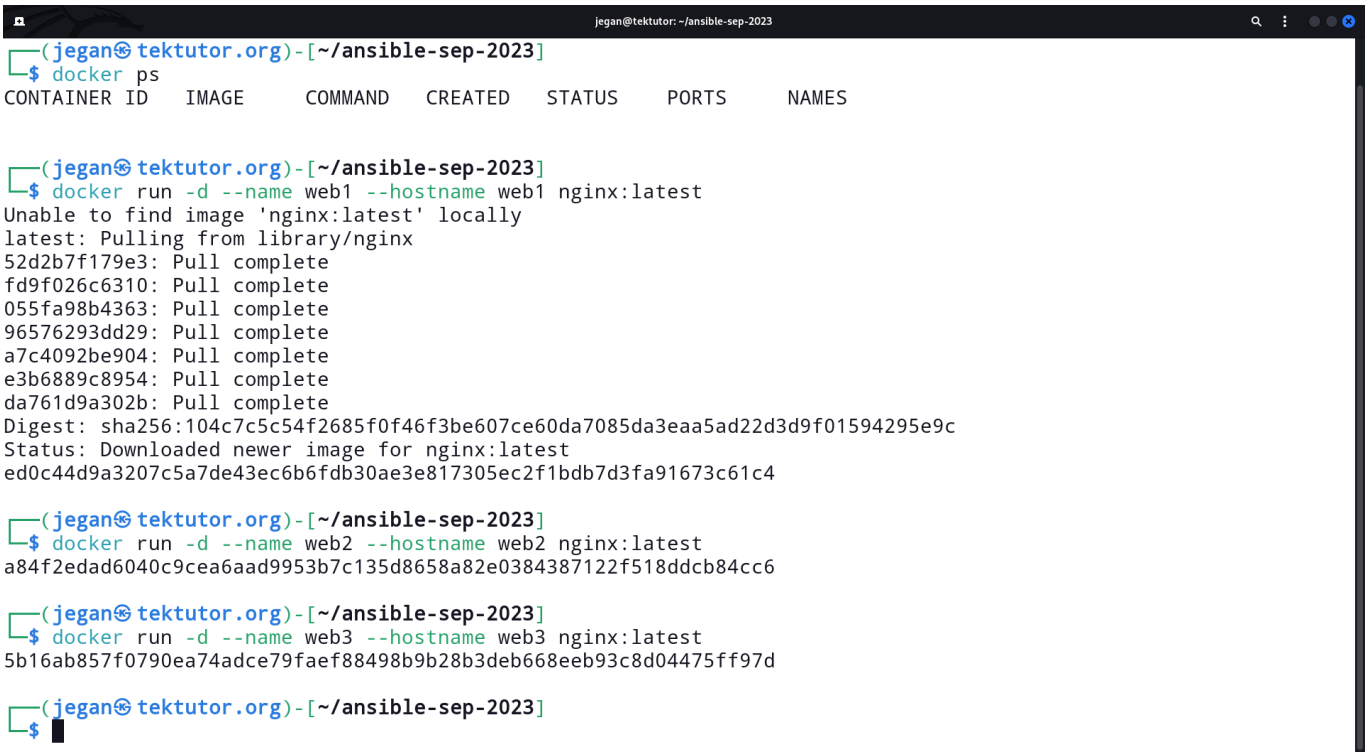


Lab - Setup a LoadBalancer with 3 nginx web servers - Port Forwarding

First let's create 3 web server containers uging nginx:latest docker image

```
docker ps
docker run -d --name web1 --hostname web1 nginx:latest
docker run -d --name web2 --hostname web2 nginx:latest
docker run -d --name web3 --hostname web3 nginx:latest
docker ps
```

Expected output



Let's find the IP addresses of web1, web2 and web3

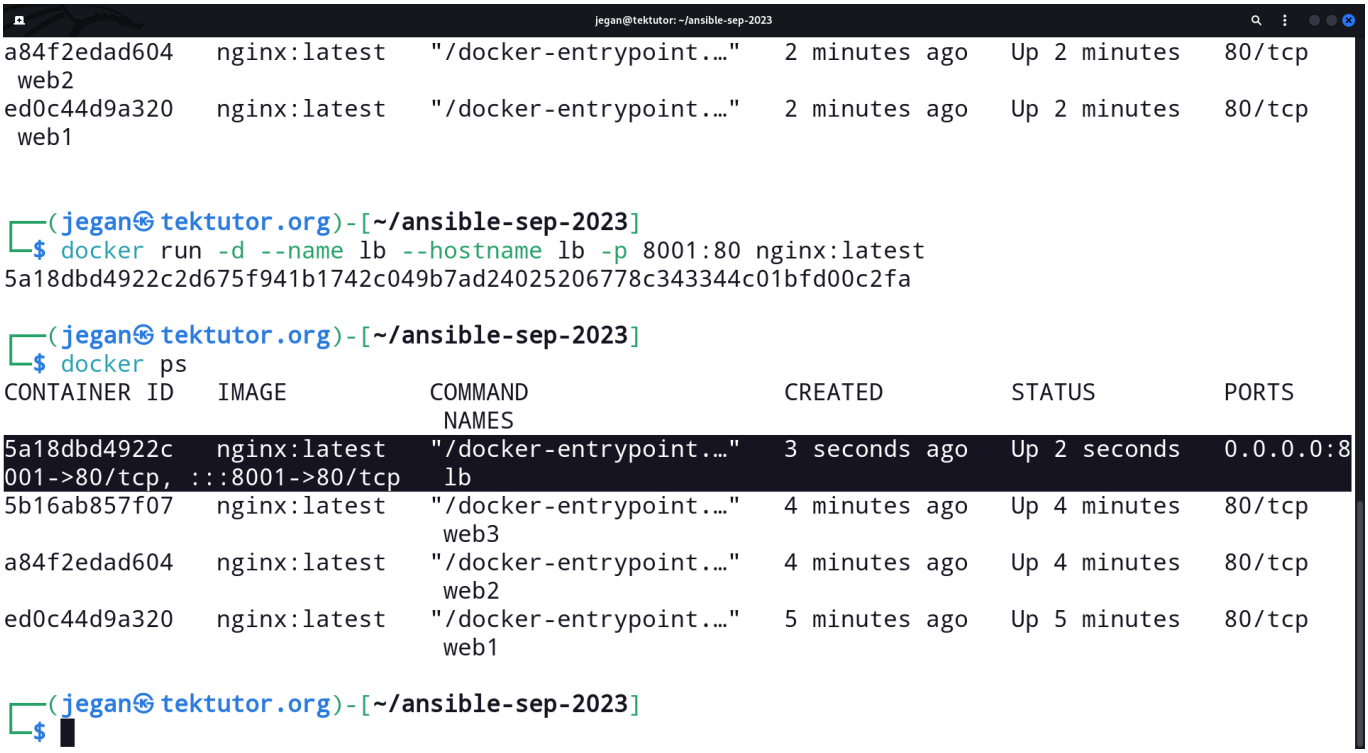
```
docker inspect web1 | grep IPA
docker inspect -f {{.NetworkSettings.IPAddress}} web1
docker inspect -f {{.NetworkSettings.IPAddress}} web2
docker inspect -f {{.NetworkSettings.IPAddress}} web3
```

Expected output

Let's create the load balancer container with port forward to make it accessible from other machines in the same network

```
docker run -d --name lb --hostname lb -p 8001:80 nginx:latest
```

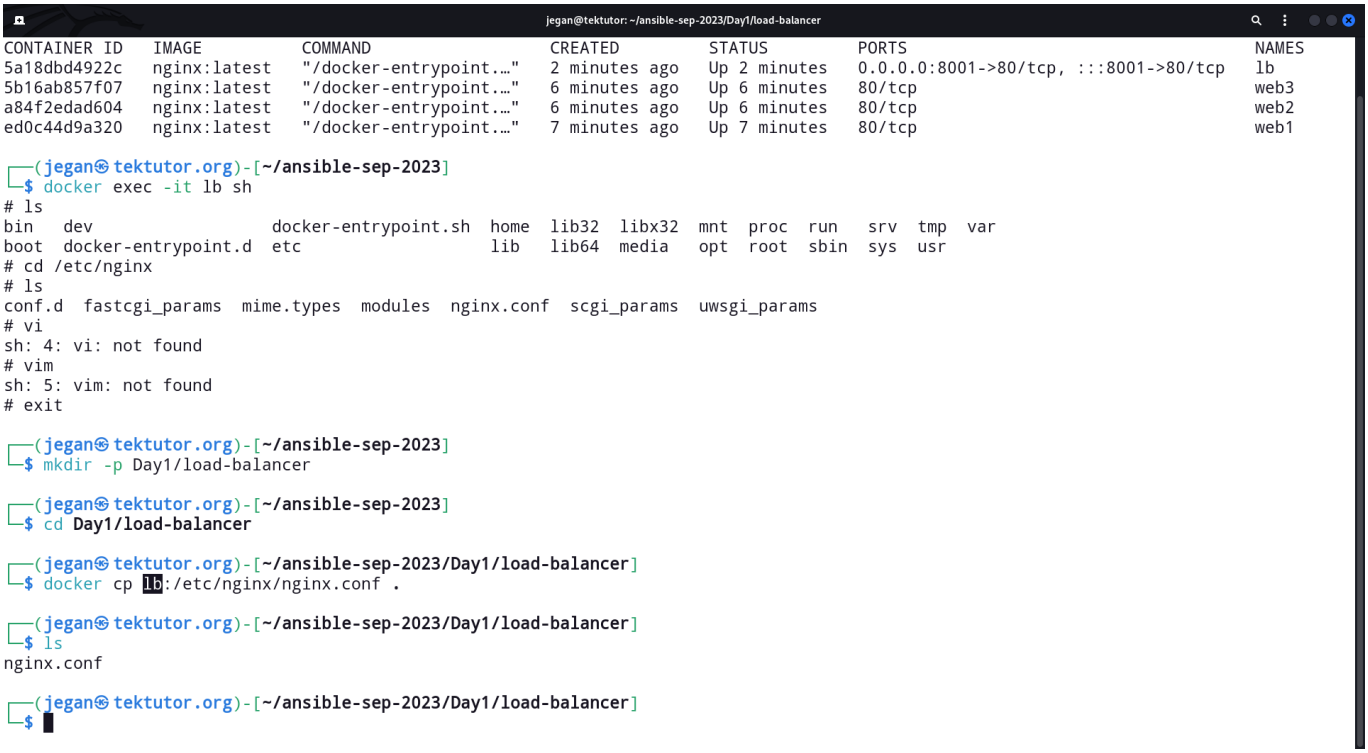
Expected output



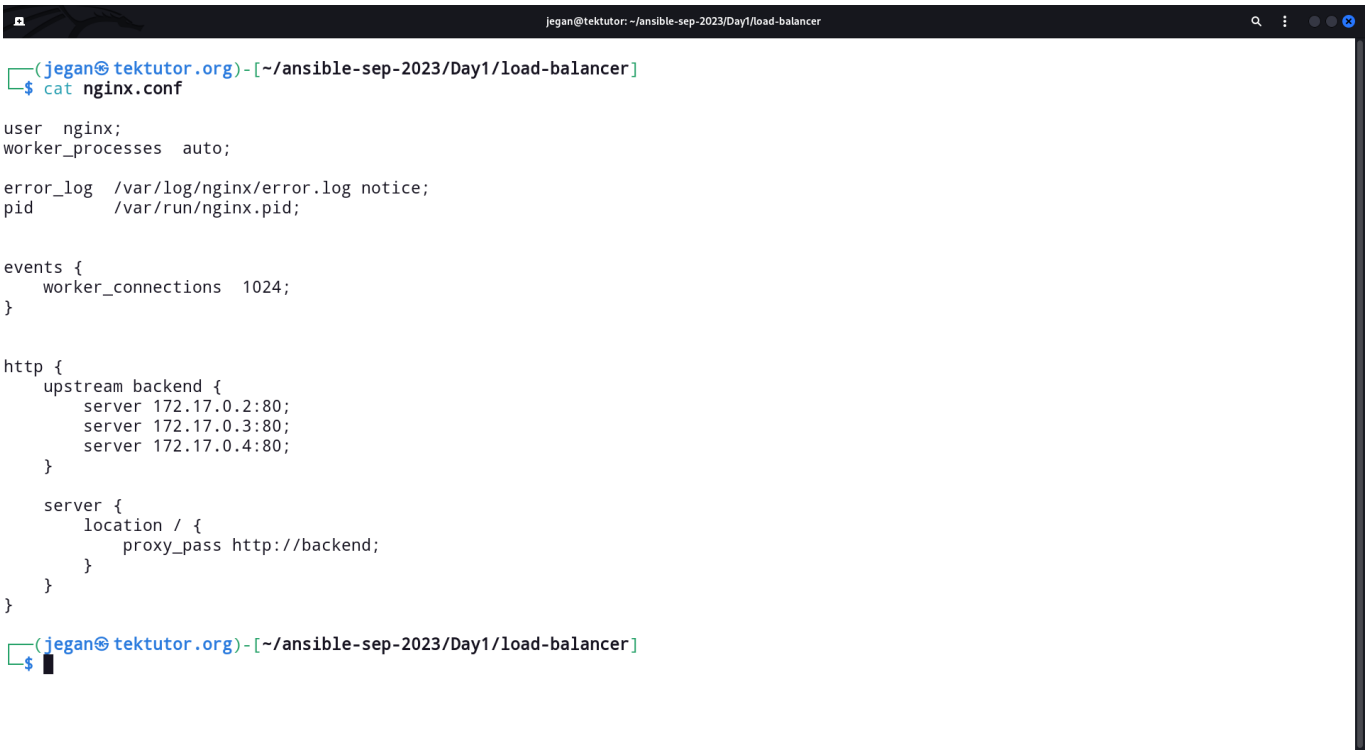
We need to configure the lb container to work like a load balancer, hence let's copy its config file to local machine to edit it and put it back inside.

```
cd ~/ansible-sep-2023/Day1/load-balancer
docker cp lb:/etc/nginx/nginx.conf .
ls
```

Expected output



We need to edit the nginx.conf as shown below updating your web1, web2 and web3 container IPs as shown below



We need to copy the nginx.conf file from our lab machine to lb container and restart lb container to apply config changes

```
cd ~/ansible-sep-2023
git pull
cd Day1/load-balancer
docker cp nginx.conf lb:/etc/nginx/nginx.conf
docker restart lb
docker ps
```

Expected output

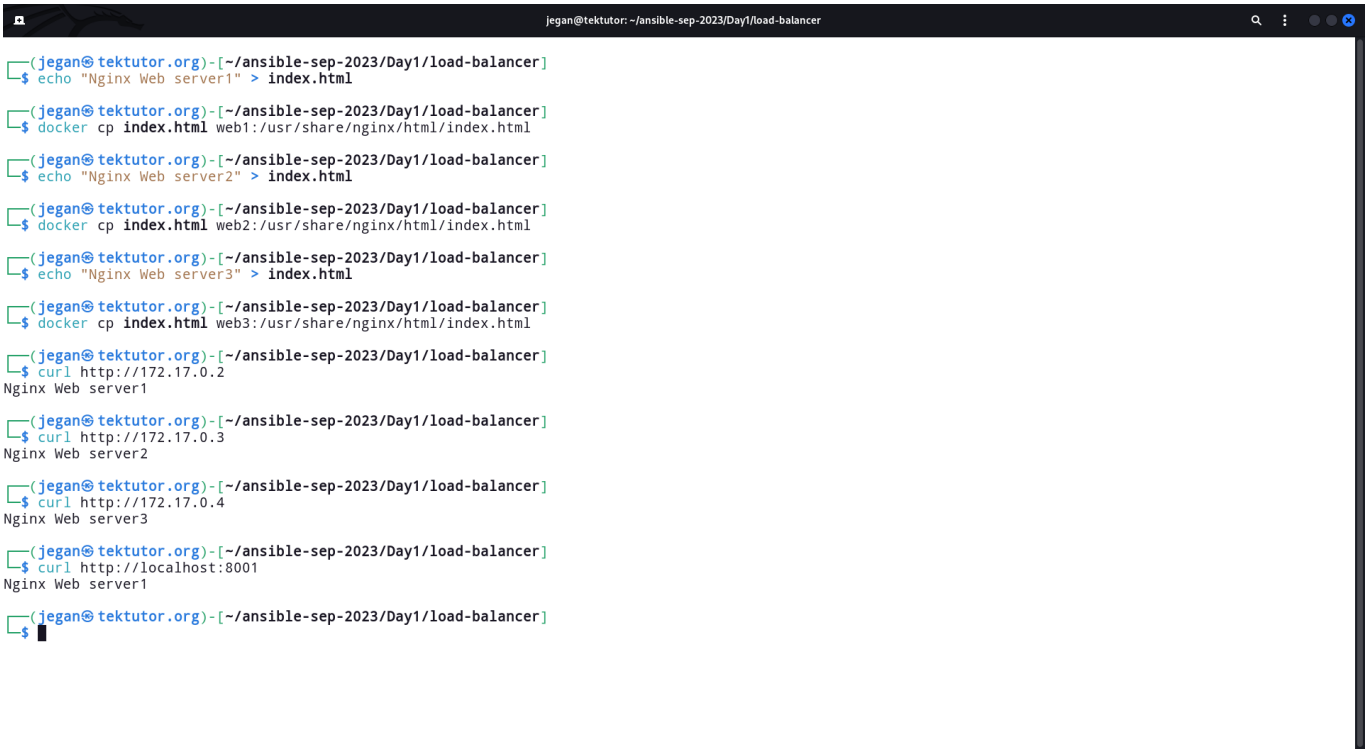
Let's create a custom html page and copy them to web1, web2 and web3 to differentiate which web server is responding to our request from lb

```
echo "Nginx Web Server1" > index.html
docker cp index.html web1:/usr/share/nginx/html/index.html

echo "Nginx Web Server2" > index.html
docker cp index.html web2:/usr/share/nginx/html/index.html

echo "Nginx Web Server3" > index.html
docker cp index.html web3:/usr/share/nginx/html/index.html
```

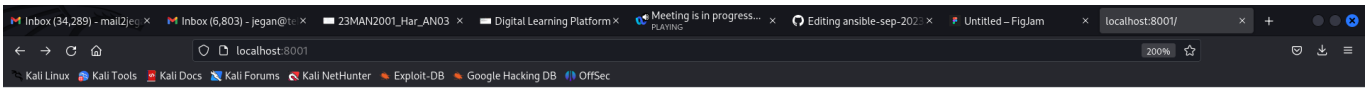
Expected output



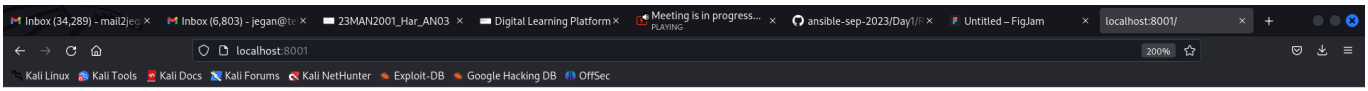
You should able to test the loadbalancer by access the below URL from ubuntu web browser

```
http://localhost:8001
http://localhost:8001
http://localhost:8001
```

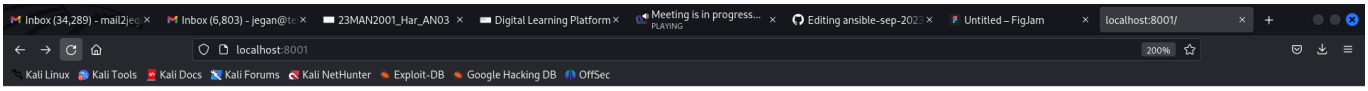
Expected output is each time you browse, it route the calls in round robin fashion



Nginx Web server1



Nginx Web server2

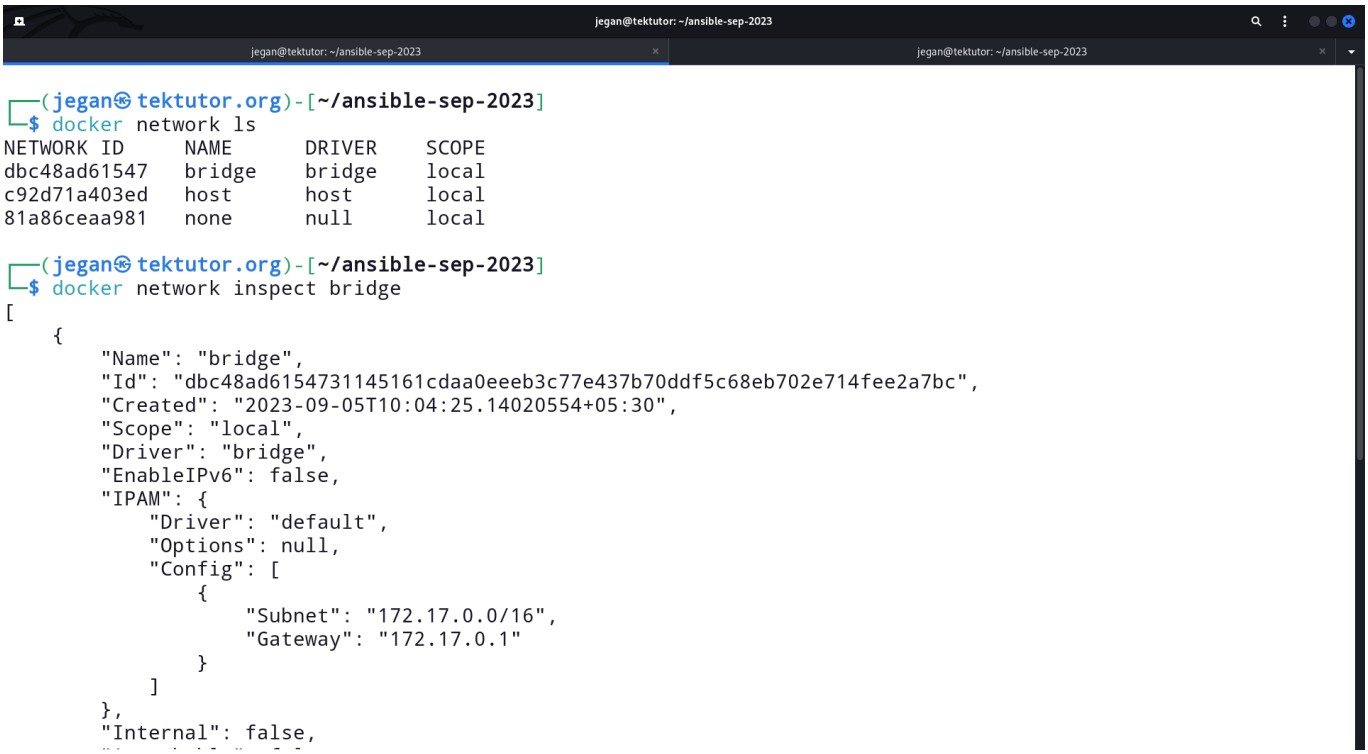


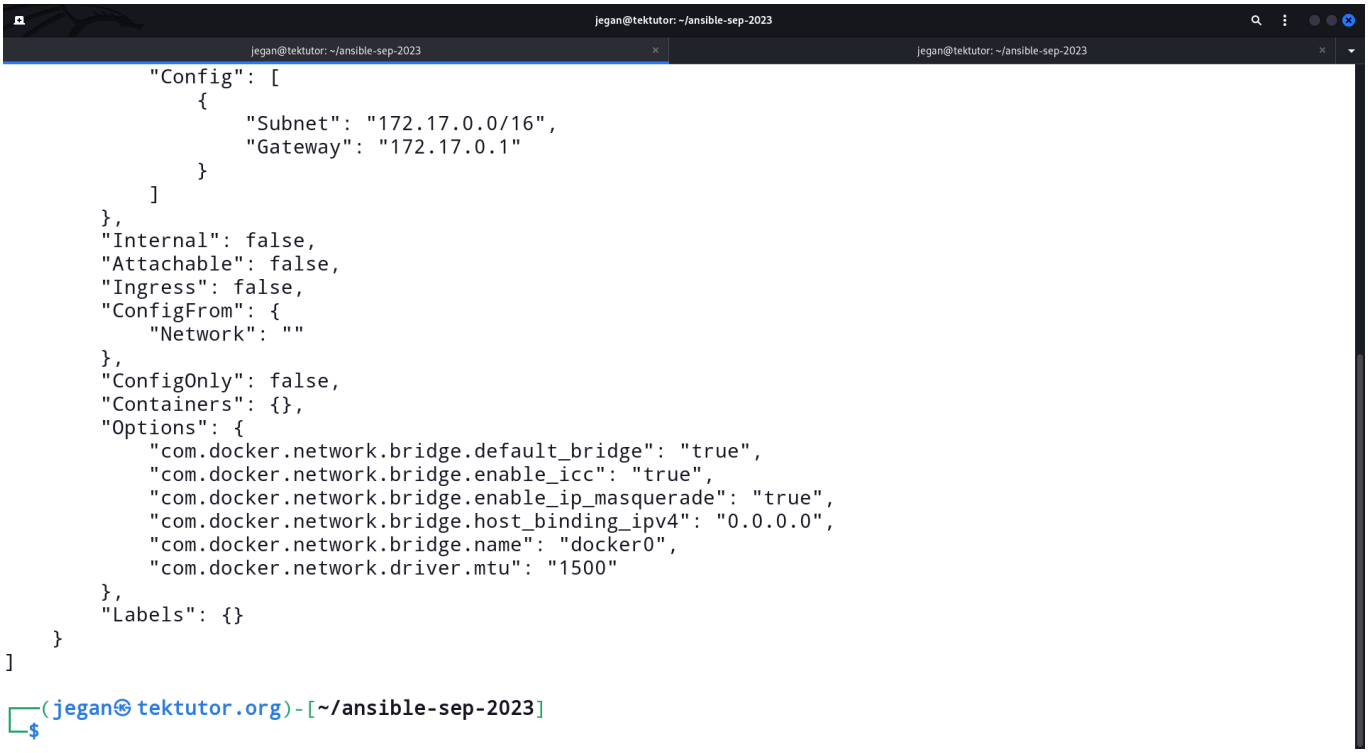
Nginx Web server3

Lab - Understanding Docker networking

Docker supports 3 types of network out of the box

```
docker network ls
docker network inspect bridge
```

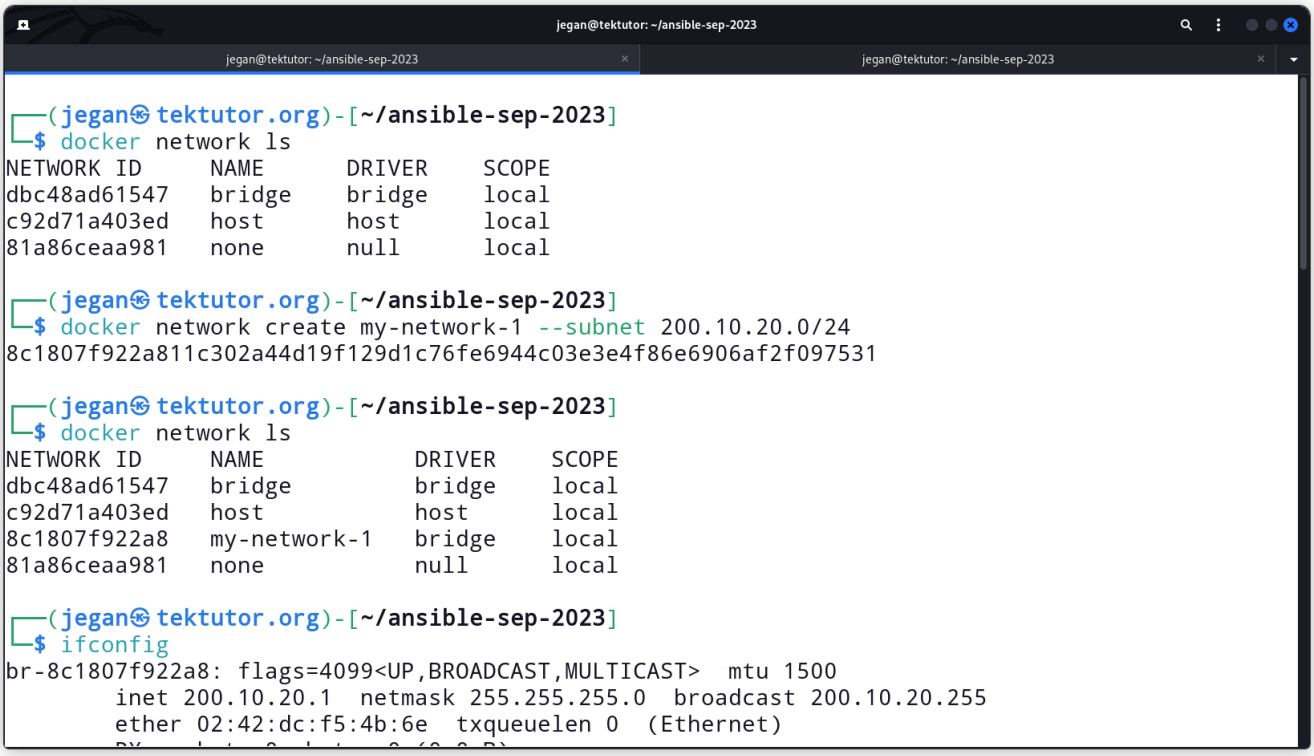




Creating a custom network in docker

```
docker network ls
docker network create my-network-1 --subnet 200.10.20.0/24
docker network ls
```

Expected output



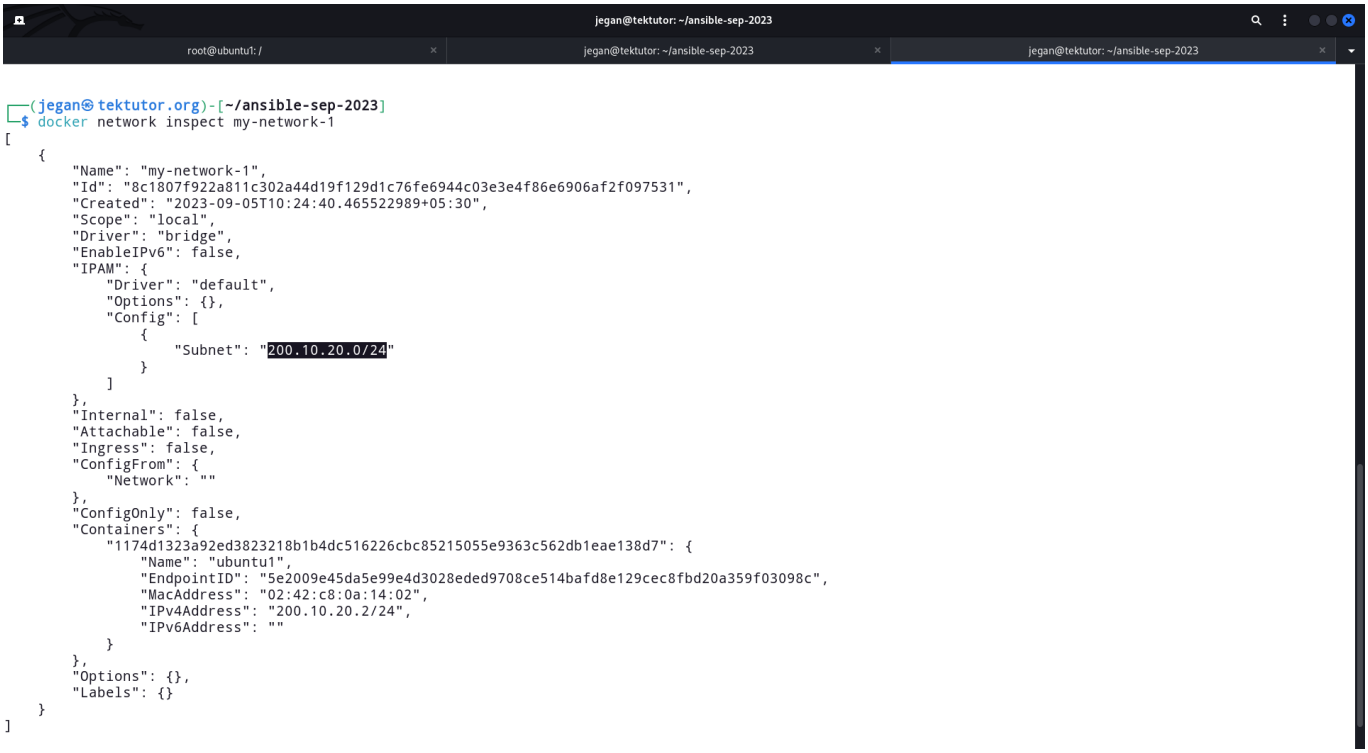
Let's create a new container and attach the new container to our custom network

```
docker run -dit --name ubuntu1 --hostname ubuntu1 --network=my-network-1
ubuntu:16.04 /bin/bash
```

Let's inspect the my-network-1 to see if the ubuntu1 container

```
docker network inspect my-network-1
```

Expected output



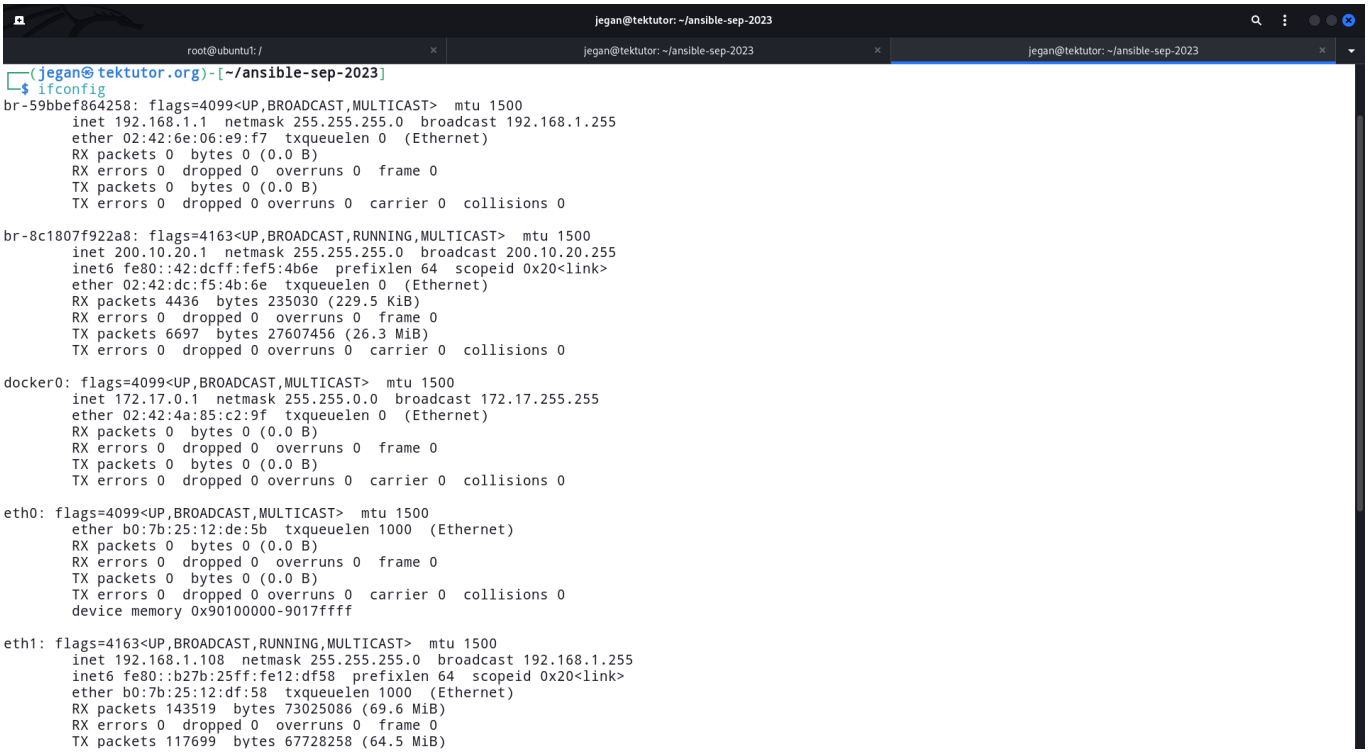
Each time a new container is created, docker creates a pair of veth devices. One of the veth (virtual ethernet) stays in the local machine where docker container is running and the other veth device is used within the docker container. The veth device within container acts as the virtual network card/interface for the container i.e eth0, this helps the container communicate with the outside world, the other veth device in the local machine machine helps the local machine communicate with the container.

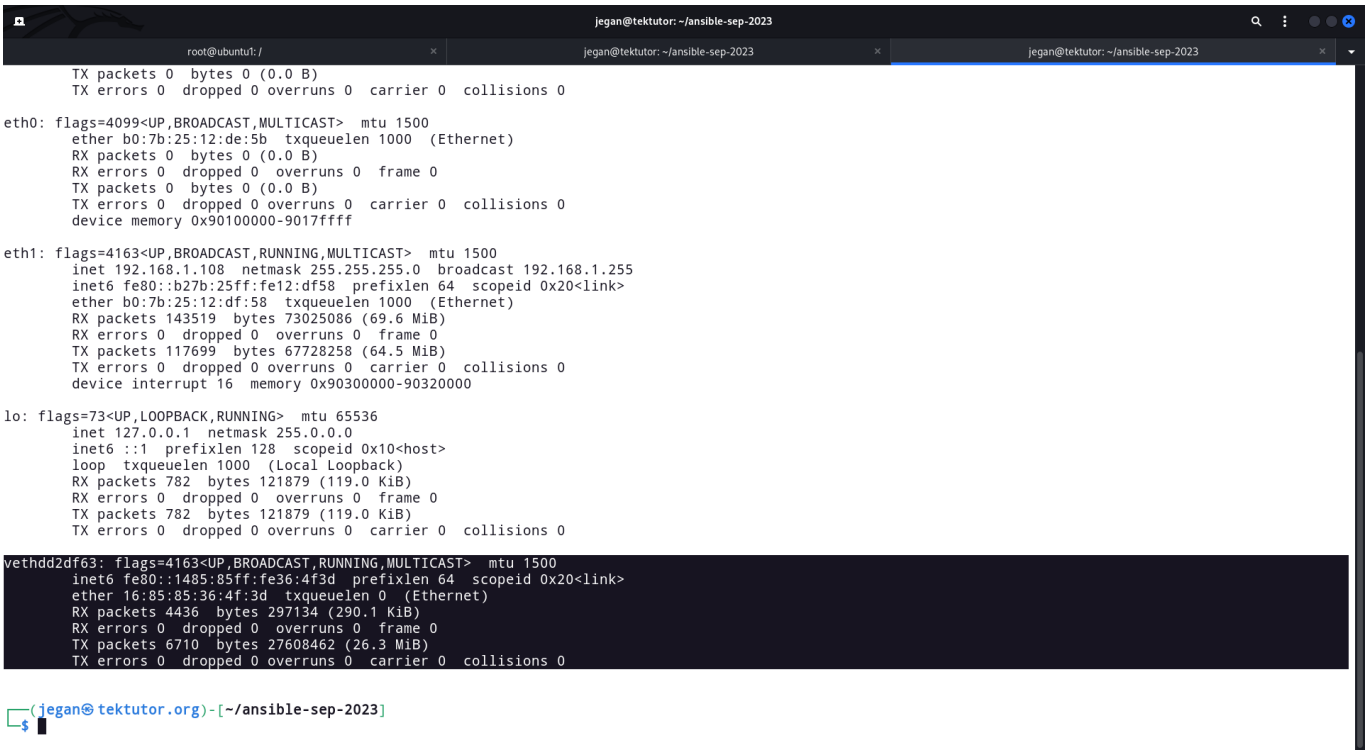
The veth suppors only uni-directional communication, hence a pair of veth devices are created for every container.

You can check the veth on local machine as shown below

```
ifconfig
```

Expected output





In order to check the veth inside the container, we need to get inside the container

```
docker exec -it ubuntu1 bash
```

Let's install network tools within the container in order to issue ifconfig command

```
apt update && apt install -y net-tools
```

You may now check ifconfig in the container shell

```
ifconfig
```

Expected output