Kubernetes 14-18 Oct 2024

To hire me as your External Openshift/Kubernetes Consultant for your organization, you can reach out to me


```
Mobile - +91 822-000-5626
Email  - jegan@tektutor.org
```


My YouTube Channel

https://www.youtube.com/@JeganathanSwaminathan/videos

My Blogs

```
https://medium.com/@jegan_50867/docker-overview-be840f727b3

https://medium.com/tektutor/container-engine-vs-container-runtime-
667a99042f3

https://medium.com/@jegan_50867/docker-commands-ba19387383b4

https://medium.com/tektutor/kubernetes-3-node-cluster-setup-50943378be41

https://medium.com/@jegan_50867/kubernetes-lightweight-developer-setup-
using-rancher-k3d-a3a94e9b5eb4

https://medium.com/tektutor/kubernetes-3-node-cluster-using-k3s-with-
docker-e325cc82fd50

https://medium.com/@jegan_50867/kubernetes-3-node-cluster-using-k3s-
d28b2c09e2f7
```

References

```
https://blog.networktocode.com/post/kubernetes-collection-ansible/

https://network-insight.net/2022/05/11/openshift-security-best-practices/

https://www.densify.com/openshift-tutorial/openshift-route/
```

Installing Code Ready Containers(OpenShift on Laptop/Desktop)

https://developers.redhat.com/products/openshift-local/overview

Using Free RedHat Developer Sandox on cloud - Openshift

https://console.redhat.com/openshift/sandbox

Installing a single node Red Hat Openshift

https://docs.openshift.com/container-platform/4.14/installing/installing_sno/install-sno-installing-sno.html

Demo - 🏃‍♀️ Installing Code Ready Containers - OpenShift Local in your Laptop/Desktop for learning purpose

Please do not attempt this exercise in our lab. This will corrupt our existing OpenShift cluster.

The instructions are for your future reference purpose. Feel free to try this in your home lab.

```
cd ~/Downloads

wget https://developers.redhat.com/content-
gateway/rest/mirror/pub/openshift-v4/clients/crc/latest/crc-linux-
amd64.tar.xz

tar xvf crc-linux-amd64.tar.xz

cd crc-linux-2.8.0-amd64/
sudo mv crc /usr/local/bin
```

Starting the CRC setup

```
crc setup
```

Expected output

```
[jegan@localhost ~]$ crc setup
INFO Using bundle path
/home/jegan/.crc/cache/crc_libvirt_4.11.1_amd64.crcbundle
INFO Checking if running as non-root
INFO Checking if running inside WSL2
INFO Checking if crc-admin-helper executable is cached
INFO Checking for obsolete admin-helper executable
INFO Checking if running on a supported CPU architecture
INFO Checking minimum RAM requirements
INFO Checking if crc executable symlink exists
INFO Checking if Virtualization is enabled
INFO Checking if KVM is enabled
INFO Checking if libvirt is installed
INFO Installing libvirt service and dependencies
INFO Using root access: Installing virtualization packages
[sudo] password for jegan:
```

```
INFO Checking if user is part of libvirt group
INFO Adding user to libvirt group
INFO Using root access: Adding user to the libvirt group
INFO Checking if active user/process is currently part of the libvirt group
INFO Checking if libvirt daemon is running
INFO Checking if a supported libvirt version is installed
INFO Checking if crc-driver-libvirt is installed
INFO Installing crc-driver-libvirt
INFO Checking if systemd-networkd is running
INFO Checking if NetworkManager is installed
INFO Checking if NetworkManager service is running
INFO Checking if /etc/NetworkManager/conf.d/crc-nm-dnsmasq.conf exists
INFO Writing Network Manager config for crc
INFO Using root access: Writing NetworkManager configuration to
/etc/NetworkManager/conf.d/crc-nm-dnsmasq.conf
INFO Using root access: Changing permissions for
/etc/NetworkManager/conf.d/crc-nm-dnsmasq.conf to 644
INFO Using root access: Executing systemctl daemon-reload command
INFO Using root access: Executing systemctl reload NetworkManager
INFO Checking if /etc/NetworkManager/dnsmasq.d/crc.conf exists
INFO Writing dnsmasq config for crc
INFO Using root access: Writing NetworkManager configuration to
/etc/NetworkManager/dnsmasq.d/crc.conf
INFO Using root access: Changing permissions for
/etc/NetworkManager/dnsmasq.d/crc.conf to 644
INFO Using root access: Executing systemctl daemon-reload command
INFO Using root access: Executing systemctl reload NetworkManager
INFO Checking if libvirt 'crc' network is available
INFO Setting up libvirt 'crc' network
INFO Checking if libvirt 'crc' network is active
INFO Starting libvirt 'crc' network
INFO Checking if CRC bundle is extracted in '$HOME/.crc'
INFO Checking if /home/jegan/.crc/cache/crc_libvirt_4.11.1_amd64.crcbundle
exists
INFO Getting bundle for the CRC executable
INFO Downloading crc_libvirt_4.11.1_amd64.crcbundle
3.23 GiB / 3.23 GiB [-----------------------------------------------------
-----------------------------------------] 100.00% 9.16 MiB p/s
INFO Uncompressing
/home/jegan/.crc/cache/crc_libvirt_4.11.1_amd64.crcbundle
crc.qcow2: 12.49 GiB / 12.49 GiB [-----------------------------------------
-------------------------------------------------------] 100.00%
oc: 118.13 MiB / 118.13 MiB [----------------------------------------------
-------------------------------------------------------] 100.00%
Your system is correctly setup for using CRC. Use 'crc start' to start the
instance
```

Starting the Local Openshift CRC single node cluster

```
crc start
```

Expected output

```
[jegan@localhost ~]$ crc start
INFO Checking if running as non-root
INFO Checking if running inside WSL2
INFO Checking if crc-admin-helper executable is cached
INFO Checking for obsolete admin-helper executable
INFO Checking if running on a supported CPU architecture
INFO Checking minimum RAM requirements
INFO Checking if crc executable symlink exists
INFO Checking if Virtualization is enabled
INFO Checking if KVM is enabled
INFO Checking if libvirt is installed
INFO Checking if user is part of libvirt group
INFO Checking if active user/process is currently part of the libvirt group
INFO Checking if libvirt daemon is running
INFO Checking if a supported libvirt version is installed
INFO Checking if crc-driver-libvirt is installed
INFO Checking if systemd-networkd is running
INFO Checking if NetworkManager is installed
INFO Checking if NetworkManager service is running
INFO Checking if /etc/NetworkManager/conf.d/crc-nm-dnsmasq.conf exists
INFO Checking if /etc/NetworkManager/dnsmasq.d/crc.conf exists
INFO Checking if libvirt 'crc' network is available
INFO Checking if libvirt 'crc' network is active
INFO Loading bundle: crc_libvirt_4.11.1_amd64...
CRC requires a pull secret to download content from Red Hat.
You can copy it from the Pull Secret section of
https://console.redhat.com/openshift/create/local.
? Please enter the pull secret
**************************************************************************
*********************************
INFO Creating CRC VM for openshift 4.11.1...
INFO Generating new SSH key pair...
INFO Generating new password for the kubeadmin user
INFO Starting CRC VM for openshift 4.11.1...
INFO CRC instance is running with IP 192.168.130.11
INFO CRC VM is running
INFO Updating authorized keys...
INFO Check internal and public DNS query...
INFO Check DNS query from host...
INFO Verifying validity of the kubelet certificates...
INFO Starting kubelet service
INFO Waiting for kube-apiserver availability... [takes around 2min]
INFO Adding user's pull secret to the cluster...
INFO Updating SSH key to machine config resource...
INFO Waiting for user's pull secret part of instance disk...
INFO Changing the password for the kubeadmin user
INFO Updating cluster ID...
INFO Updating root CA cert to admin-kubeconfig-client-ca configmap...
INFO Starting openshift instance... [waiting for the cluster to stabilize]
INFO 3 operators are progressing: image-registry, network, openshift-
controller-manager
```

```
INFO 3 operators are progressing: image-registry, network, openshift-
controller-manager
INFO 3 operators are progressing: image-registry, network, openshift-
controller-manager
INFO 3 operators are progressing: image-registry, network, openshift-
controller-manager
INFO 2 operators are progressing: image-registry, openshift-controller-
manager
INFO 3 operators are progressing: image-registry, node-tuning, openshift-
controller-manager
INFO Operator openshift-controller-manager is progressing
INFO Operator openshift-controller-manager is progressing
INFO Operator openshift-controller-manager is progressing
INFO Operator openshift-controller-manager is progressing
INFO 2 operators are progressing: authentication, openshift-controller-
manager
INFO Operator authentication is progressing
INFO Operator authentication is progressing
INFO Operator authentication is progressing
INFO Operator authentication is progressing
ERRO Cluster is not ready: cluster operators are still not stable after
10m10.227355941s
INFO Adding crc-admin and crc-developer contexts to kubeconfig...
Started the OpenShift cluster.

The server is accessible via web console at:
  https://console-openshift-console.apps-crc.testing

Log in as administrator:
  Username: kubeadmin
  Password: gwifD-9W6K9-Hn3ey-h2Wwe

Log in as user:
  Username: developer
  Password: developer

Use the 'oc' command line interface:
  $ eval $(crc oc-env)
  $ oc login -u developer https://api.crc.testing:6443
```

Testing the lab environment

```
- From the RPS cloud machine, you need to RDP to the Linux server assigned
to you
- The Linux user will be user[xy], where xy is the last two digits of your
RPS Cloud user
- We have total 3 Linux servers with the below details
  - Server 1 - 10.10.15.27 ( user01 thru user08 ) and password is rps@12345
  - Server 2 - 10.10.15.34 ( user09 thru user15 ) and password is rps@12345
  - Server 3 - 10.10.15.35 ( user16 thru user23 ) and password is rps@12345
- The Linux Server details
```

```
  - OS - Oracle Linux v9.4 64-bit
  - Hypervisor - KVM
  - Processor with 48 CPU Cores
  - 755 GB RAM
  - 17.3 TB HDD Storage
```

Check if docker installed from the linux terminal

```
docker --version
```

Check if you are able to list docker images

```
docker images
```

Check if you are able to access the kubectl - kubernetes client tool

```
kubectl version
```

Check if you are able to access the oc - openshift client tool

```
oc version
```

check if you are able to login to Red Hat Openshift cluster from terminal

```
cat ~/openshift.txt
oc login
```

Check if you are able to list the openshift nodes

```
oc get nodes
```

Day 1

Boot Loaders

```
- is a system utility that is installed in the hard disk boot sector( byte
0, sector 0 )
- MBR stands for Master Boot Record is where the boot loader application is
installed
- it is 512 bytes
- boot loader application is the first application that runs after the BIOS
POST (Power On Self Test)
- the boot loader application searches your hard disk, looking for
Operating Systems installed on it
- in case, the boot loader finds more than one OS then it gives a menu for
the user to choose which OS they want to boot into
- in otherwords, though many OS is installed in the systemm the boot loader
allows only OS to be active at any point of time
```

Hypervisor Overview

```
- Hypervisor a.k.a Virtualization
- virtualization technology allows us to run multiple OS in the same
laptop/desktop/server
- i.e many OS can be active at the same time in the same machine
- there are 2 types of Hypervisors
  1. Type 1 - Bare Metal Hypervisor ( used in Workstations and Servers )
  2. Type 2 - used in laptops/desktops/workstations
- this type of virtualization is called Heavy-weight
  - the reason is,each virtual machine requires dedicated hardware
resources
    - dedicated CPU cores
    - dedicated RAM
    - dedicated Storage ( HDD/SSD )
- the OS runs within the Virtual Machine(VM/Guest OS), hence each VM
represents one OS
- the OS that runs within the VM is a fully functional OS
```

Type 1 - Bare Metal Hypervisor

```
- Virtual Machine can be created directly on the server without any Host OS
- Examples
  - VMWare vsphere/vcenter
```

Type 2 Hypervisor

```
- Examples
  - VMWare
    - Fusion ( supports Mac OS-X )
    - Workstation ( supports Linux and Windows )
  - Oracle VirtualBox
  - Parallels ( supports Mac OS-X )
  - Microsoft Hyper-V
  - KVM - supported in all Linux distibutions
```

Server Grade Processors

- supports 128,256,512 cpu cores
- Motherboards with 8 Processor Sockets
- Processors also comes in 2 types of packaging
    - SCM (single chip module)
    - MCM (multiple chip module)

What is HyperThreading(Intel)/SMT(AMD)?

- Processors that support Hyperthreading/SMT allow each phyical core to execute 2 to 8 threads at the same time
- Hyperthreading is Intel technology
- While SMT(Simulateneous Multi Threading) is AMD's equivalent technology
- Virtualization Softwares see each Physical core as 2 Virtual Cores if they support Hyperthreading/SMT
- In some high-end server grade Processors, each Physical core are seen as 8 Virtual Cores

Multi Chip Module

- Single Integrated Chip (IC) hosting multiple Processors
- The MCM Chip can be installed in a Single Socket on your Server grade Motherboard
- Let's assume a MCM IC hosting 8 Processors each supporting 128 Cores
    - total cores supported - 8 x 128 x 2 = 2048 cores

Linux Kernel Features

- Supports two interesting features that enable container technology
    1. Namespace and
        - each container is separated from other containers as they run in a virtual sandbox environment
        - each container gets many namespaces
            - PID namespace
            - Network namespace, etc
    2. Control Group (CGroup)
        - it is through this feature, we can restrict a container hardware resource utilization
        - For instance, we can restrict a container using only 25% of CPU (if the OS has let's 4 cores, the container can only use 1 core at the max)

Containerization

```
- is an application virtualization technology
- light-weight virtualization
  - each containized application doesn't require dedicated hardware
resources
  - all containers that runs in a machine shares the Hardware resources
available in the underlying OS
- each container represents a single application
- each container get an IP address
- each container get its own file system
- container is an application process that runs in a separate namespace
- containers are not a replacement for Virtualization or OS
- containers and virtualization are complementing technology, hence they
are used in combination in real world
- each container get its own dedicated network namespace
- each container get its own dedicated network stack ( 7 OSI Layers )
- each container get its own dedicated port range ( 0 - 65535 )
- most of the containers has atleast one network card (virtual network
card)
```

What are the different Container Tools available?

```
- LXC
- Rkt ( pronounced as Rocket )
- Podman
```

Difference between Docker(Containers) and Virtualization

```
- Virtual Machine is a fully functional Operating System while a container
just hosts an application with all its dependencies
- Each Virtual Machine get its own share of Hardware resources ( CPU Cores,
RAM and Storage ), while containers share the hardware resources on the
underlying Operating System where they run
- Virtual Machines are heavy weight while Containers are light-weight
- Only a limited number of Virtual Machines can be active on laptop while
on the same laptop you could easily run 40~50 containers
```

Minimal number of Physical server to support 1000 Virtual Machines

- assume the server motherboards supports 8 Processor Sockets
- if we install MCM based Processor, i.e each IC has 4 Processors, each Processor supporting 256 CPU cores
- total number of Physical CPU cores = 8 x 4 x 256 = 8192
- total logical/virtual CPU cores = 8192 x 2 = 16384

Container Engine Overview

```
- is a high-level software that helps u managing container images and
containers
- user-friendly
- under the hood, container engines depends on Container runtimes to manage
images and containers
- Examples
  - Docker is a Container Engine that depends on containerd which in turn
depends on runC container runtime
  - Podman is a Container Engine that depends on CRI-O container runtime
```

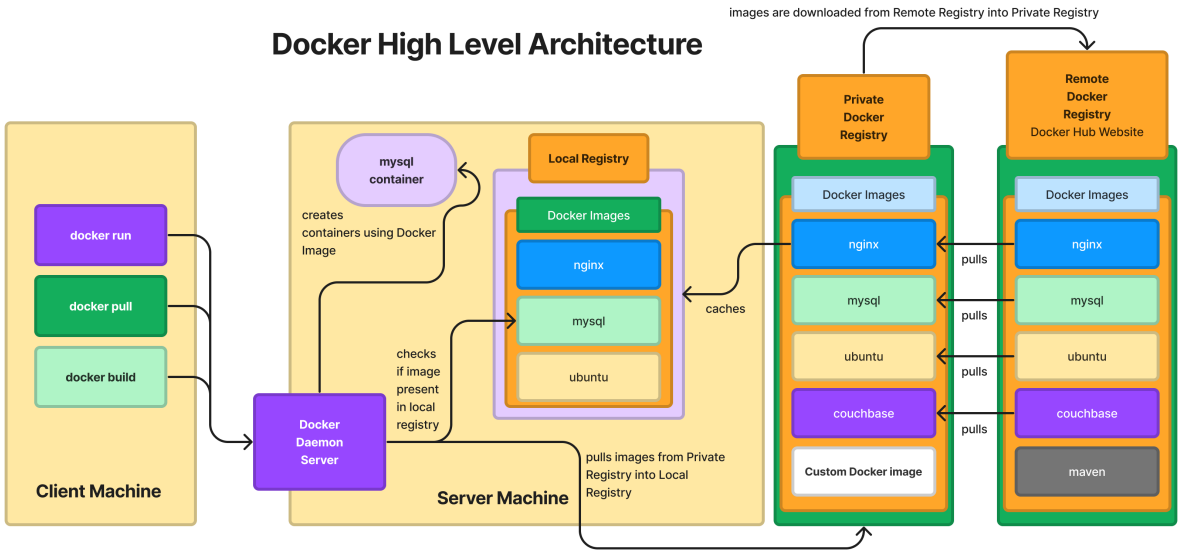
Container Runtime Overview

```
- is low-level software utility that helps us managing container images and
containers
- they are not user-friendly, hence normal end-users avoid using them
directly
- examples
  - runC container runtime
  - CRI-O container runtime
```

Docker Overview

```
- Docker is develed in go language by a company called Docker Inc
- a container engine that depends on containerd
- containerd in turn depends on runC container runtime
- a very popular container engine
- off late, due to security vulnerabilities the industry is moving away
from Docker
- some alternates to docker is containerd, podman, etc
- it comes in 2 flavours
  - Docker Community Edition - Docker CE ( open source )
  - Docker Enterprise Edition - Docker EE ( Paid version )
```

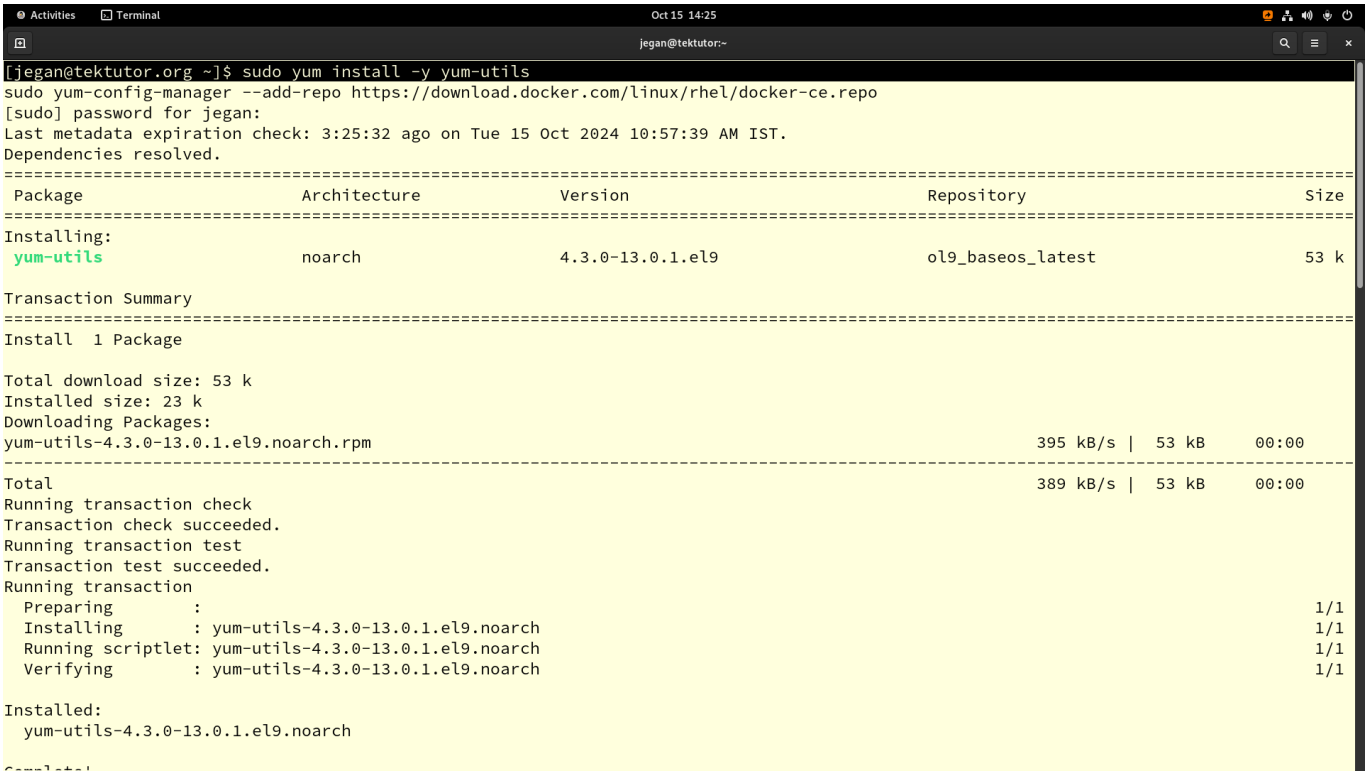
Docker High Level Architecture



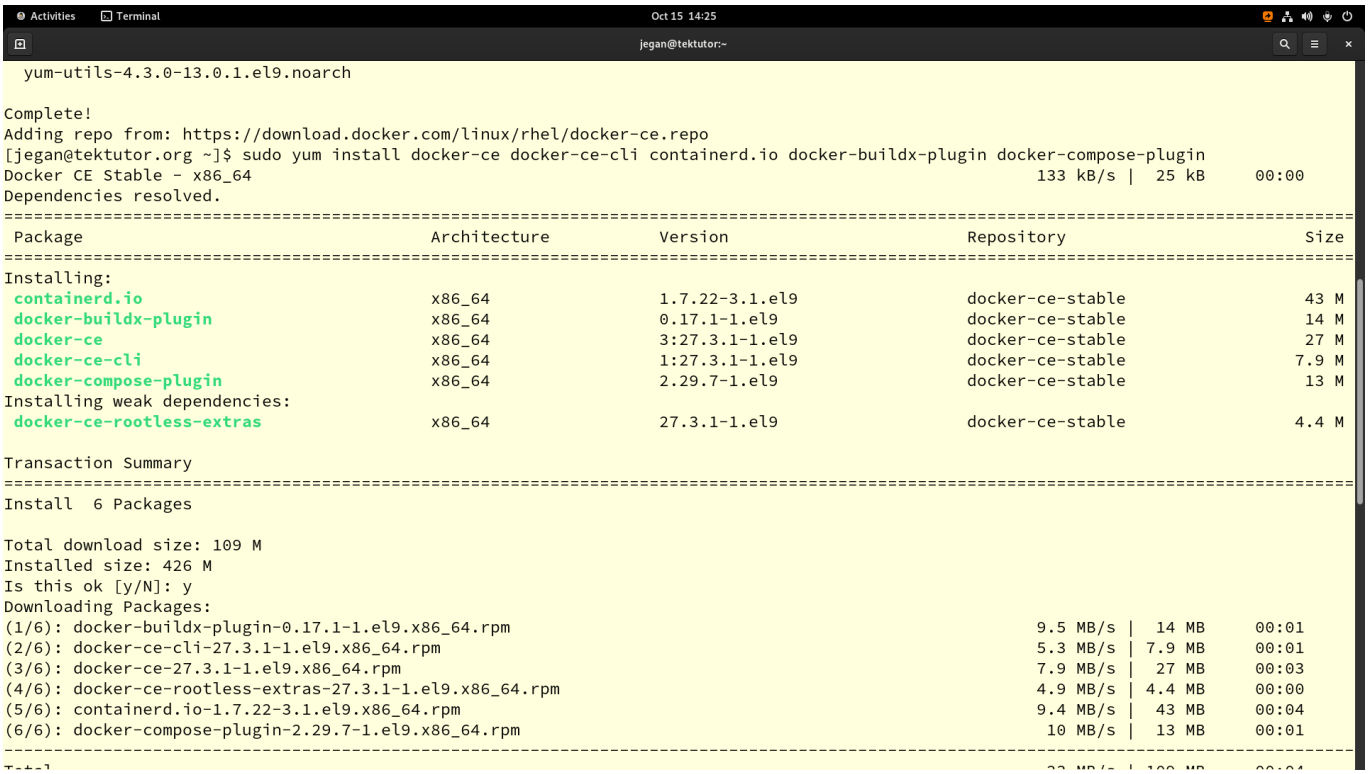
Demo - Installing Docker CE in Oracle Linux v9.4

```
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo
https://download.docker.com/linux/rhel/docker-ce.repo
sudo yum install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin
sudo systemctl enable docker
sudo systemctl start docker
sudo systemctl status docker
sudo usermod -aG docker $USER
su $USER
docker --version
docker images
```

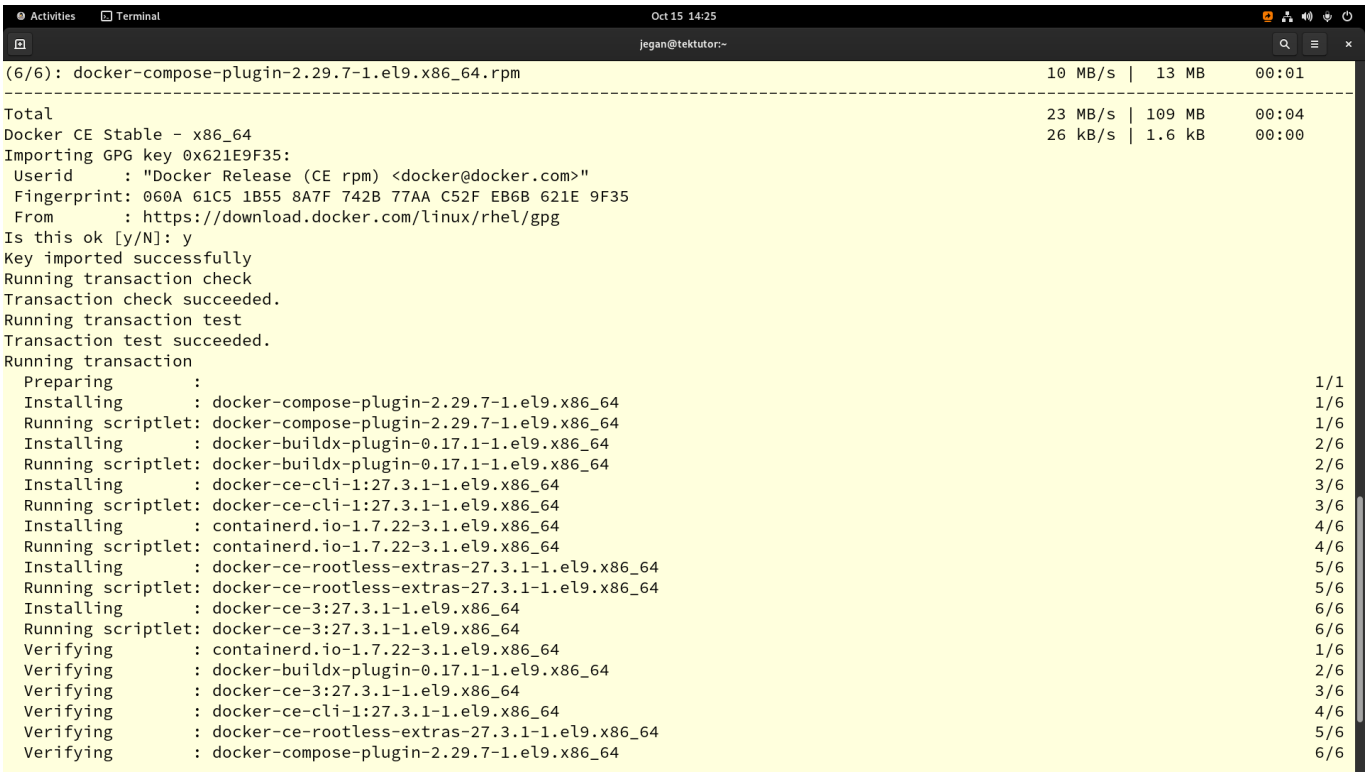
Expected output

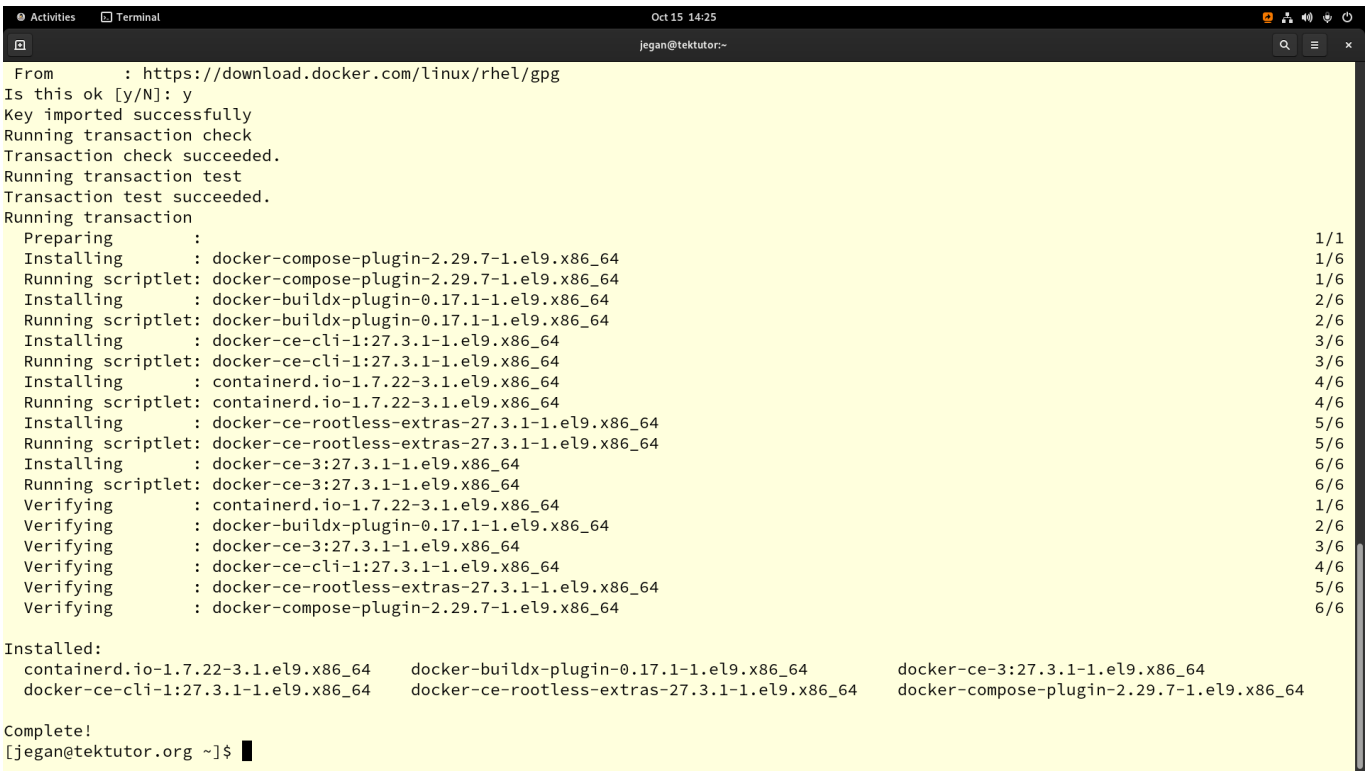


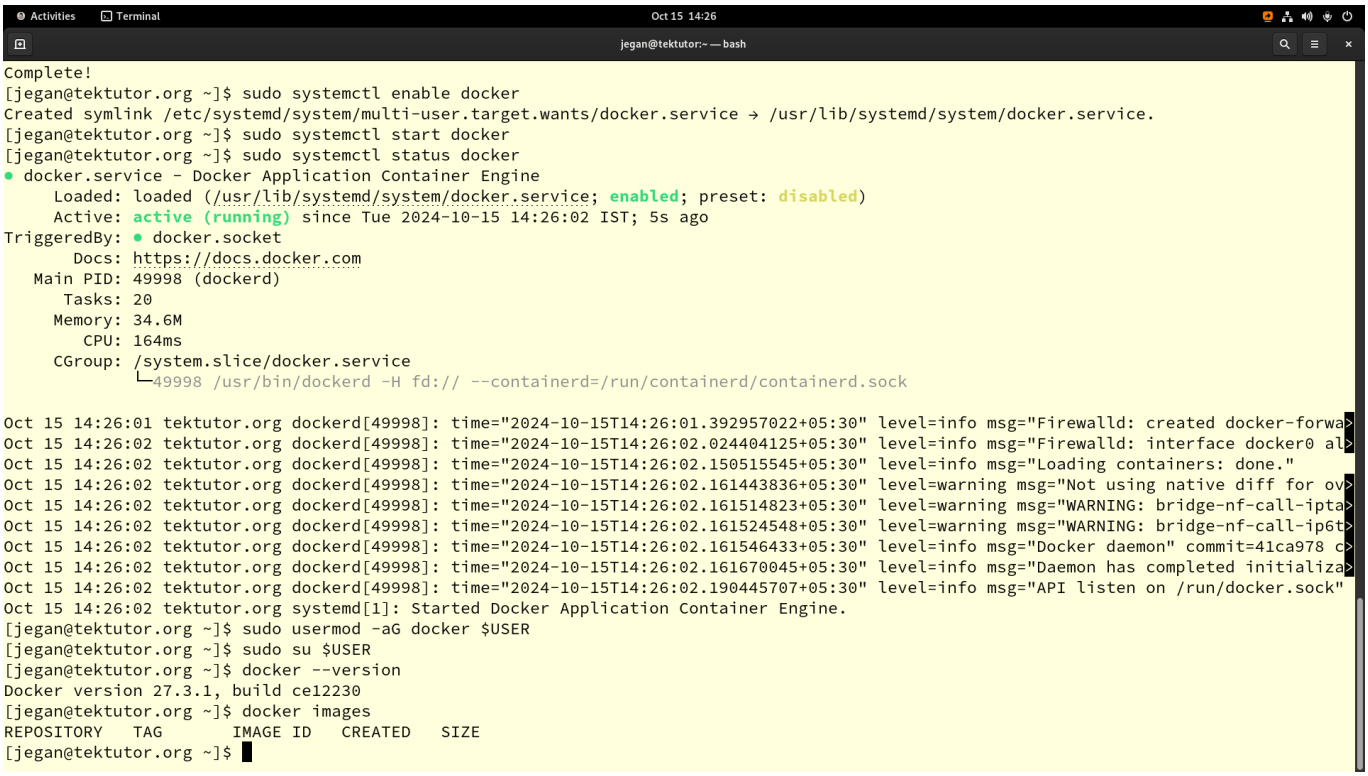
```
Activities  Terminal                    Oct 15 14:25
jegan@tektutor:~
[jegan@tektutor.org ~]$ sudo yum install -y yum-utils
sudo yum-config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
[sudo] password for jegan:
Last metadata expiration check: 3:25:32 ago on Tue 15 Oct 2024 10:57:39 AM IST.
Dependencies resolved.
================================================================================
 Package              Architecture        Version                   Repository                  Size
================================================================================
Installing:
 yum-utils            noarch              4.3.0-13.0.1.el9          ol9_baseos_latest           53 k

Transaction Summary
================================================================================
Install  1 Package

Total download size: 53 k
Installed size: 23 k
Downloading Packages:
yum-utils-4.3.0-13.0.1.el9.noarch.rpm                  395 kB/s |  53 kB     00:00
--------------------------------------------------------------------------------
Total                                                  389 kB/s |  53 kB     00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : yum-utils-4.3.0-13.0.1.el9.noarch                      1/1
  Running scriptlet: yum-utils-4.3.0-13.0.1.el9.noarch                      1/1
  Verifying        : yum-utils-4.3.0-13.0.1.el9.noarch                      1/1

Installed:
  yum-utils-4.3.0-13.0.1.el9.noarch

Complete!
```



```
Activities  Terminal                    Oct 15 14:25
jegan@tektutor:~
  yum-utils-4.3.0-13.0.1.el9.noarch

Complete!
Adding repo from: https://download.docker.com/linux/rhel/docker-ce.repo
[jegan@tektutor.org ~]$ sudo yum install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
Docker CE Stable - x86_64                              133 kB/s |  25 kB     00:00
Dependencies resolved.
================================================================================
 Package                  Architecture  Version                 Repository      Size
================================================================================
Installing:
 containerd.io            x86_64        1.7.22-3.1.el9          docker-ce-stable  43 M
 docker-buildx-plugin     x86_64        0.17.1-1.el9            docker-ce-stable  14 M
 docker-ce                x86_64        3:27.3.1-1.el9          docker-ce-stable  27 M
 docker-ce-cli            x86_64        1:27.3.1-1.el9          docker-ce-stable 7.9 M
 docker-compose-plugin    x86_64        2.29.7-1.el9            docker-ce-stable  13 M
Installing weak dependencies:
 docker-ce-rootless-extras x86_64       27.3.1-1.el9            docker-ce-stable 4.4 M

Transaction Summary
================================================================================
Install  6 Packages

Total download size: 109 M
Installed size: 426 M
Is this ok [y/N]: y
Downloading Packages:
(1/6): docker-buildx-plugin-0.17.1-1.el9.x86_64.rpm    9.5 MB/s |  14 MB     00:01
(2/6): docker-ce-cli-27.3.1-1.el9.x86_64.rpm           5.3 MB/s | 7.9 MB     00:01
(3/6): docker-ce-27.3.1-1.el9.x86_64.rpm               7.9 MB/s |  27 MB     00:03
(4/6): docker-ce-rootless-extras-27.3.1-1.el9.x86_64.rpm 4.9 MB/s | 4.4 MB     00:00
(5/6): containerd.io-1.7.22-3.1.el9.x86_64.rpm         9.4 MB/s |  43 MB     00:04
(6/6): docker-compose-plugin-2.29.7-1.el9.x86_64.rpm    10 MB/s |  13 MB     00:01
--------------------------------------------------------------------------------
```


Activities
Terminal
Oct 15 14:25
jegan@tektutor:~
(6/6): docker-compose-plugin-2.29.7-1.el9.x86_64.rpm 10 MB/s | 13 MB 00:01
Total 23 MB/s | 109 MB 00:04
Docker CE Stable - x86_64 26 kB/s | 1.6 kB 00:00
Importing GPG key 0x621E9F35:
Userid : "Docker Release (CE rpm) <docker@docker.com>"
Fingerprint: 060A 61C5 1B55 8A7F 742B 77AA C52F EB6B 621E 9F35
From : https://download.docker.com/linux/rhel/gpg
Is this ok [y/N]: y
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing : 1/1
Installing : docker-compose-plugin-2.29.7-1.el9.x86_64 1/6
Running scriptlet: docker-compose-plugin-2.29.7-1.el9.x86_64 1/6
Installing : docker-buildx-plugin-0.17.1-1.el9.x86_64 2/6
Running scriptlet: docker-buildx-plugin-0.17.1-1.el9.x86_64 2/6
Installing : docker-ce-cli-1:27.3.1-1.el9.x86_64 3/6
Running scriptlet: docker-ce-cli-1:27.3.1-1.el9.x86_64 3/6
Installing : containerd.io-1.7.22-3.1.el9.x86_64 4/6
Running scriptlet: containerd.io-1.7.22-3.1.el9.x86_64 4/6
Installing : docker-ce-rootless-extras-27.3.1-1.el9.x86_64 5/6
Running scriptlet: docker-ce-rootless-extras-27.3.1-1.el9.x86_64 5/6
Installing : docker-ce-3:27.3.1-1.el9.x86_64 6/6
Running scriptlet: docker-ce-3:27.3.1-1.el9.x86_64 6/6
Verifying : containerd.io-1.7.22-3.1.el9.x86_64 1/6
Verifying : docker-buildx-plugin-0.17.1-1.el9.x86_64 2/6
Verifying : docker-ce-3:27.3.1-1.el9.x86_64 3/6
Verifying : docker-ce-cli-1:27.3.1-1.el9.x86_64 4/6
Verifying : docker-ce-rootless-extras-27.3.1-1.el9.x86_64 5/6
Verifying : docker-compose-plugin-2.29.7-1.el9.x86_64 6/6

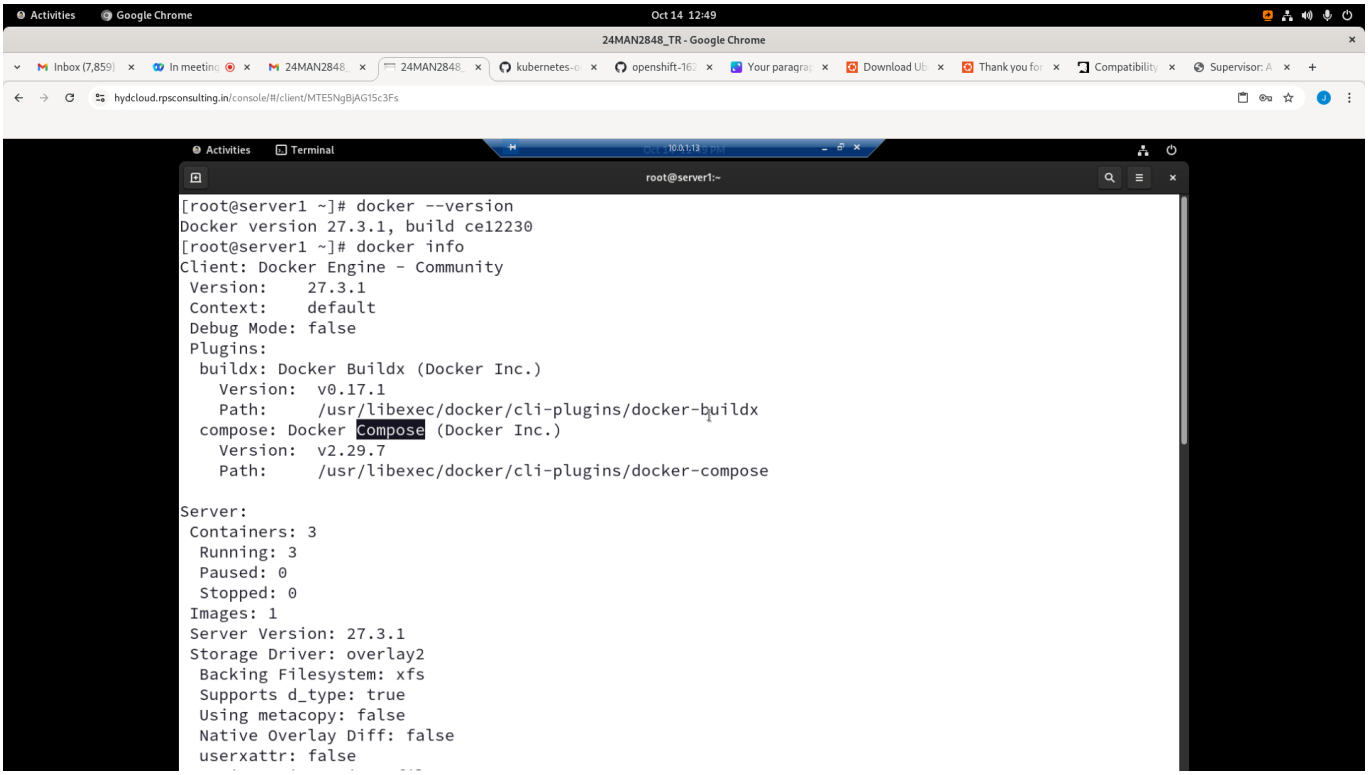

Activities
Terminal
Oct 15 14:25
jegan@tektutor:~
From : https://download.docker.com/linux/rhel/gpg
Is this ok [y/N]: y
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing : 1/1
Installing : docker-compose-plugin-2.29.7-1.el9.x86_64 1/6
Running scriptlet: docker-compose-plugin-2.29.7-1.el9.x86_64 1/6
Installing : docker-buildx-plugin-0.17.1-1.el9.x86_64 2/6
Running scriptlet: docker-buildx-plugin-0.17.1-1.el9.x86_64 2/6
Installing : docker-ce-cli-1:27.3.1-1.el9.x86_64 3/6
Running scriptlet: docker-ce-cli-1:27.3.1-1.el9.x86_64 3/6
Installing : containerd.io-1.7.22-3.1.el9.x86_64 4/6
Running scriptlet: containerd.io-1.7.22-3.1.el9.x86_64 4/6
Installing : docker-ce-rootless-extras-27.3.1-1.el9.x86_64 5/6
Running scriptlet: docker-ce-rootless-extras-27.3.1-1.el9.x86_64 5/6
Installing : docker-ce-3:27.3.1-1.el9.x86_64 6/6
Running scriptlet: docker-ce-3:27.3.1-1.el9.x86_64 6/6
Verifying : containerd.io-1.7.22-3.1.el9.x86_64 1/6
Verifying : docker-buildx-plugin-0.17.1-1.el9.x86_64 2/6
Verifying : docker-ce-3:27.3.1-1.el9.x86_64 3/6
Verifying : docker-ce-cli-1:27.3.1-1.el9.x86_64 4/6
Verifying : docker-ce-rootless-extras-27.3.1-1.el9.x86_64 5/6
Verifying : docker-compose-plugin-2.29.7-1.el9.x86_64 6/6
Installed:
containerd.io-1.7.22-3.1.el9.x86_64 docker-buildx-plugin-0.17.1-1.el9.x86_64 docker-ce-3:27.3.1-1.el9.x86_64
docker-ce-cli-1:27.3.1-1.el9.x86_64 docker-ce-rootless-extras-27.3.1-1.el9.x86_64 docker-compose-plugin-2.29.7-1.el9.x86_64
Complete!
[jegan@tektutor.org ~]$



Lab - Checking docker version and information

```
docker --version
docker info
```

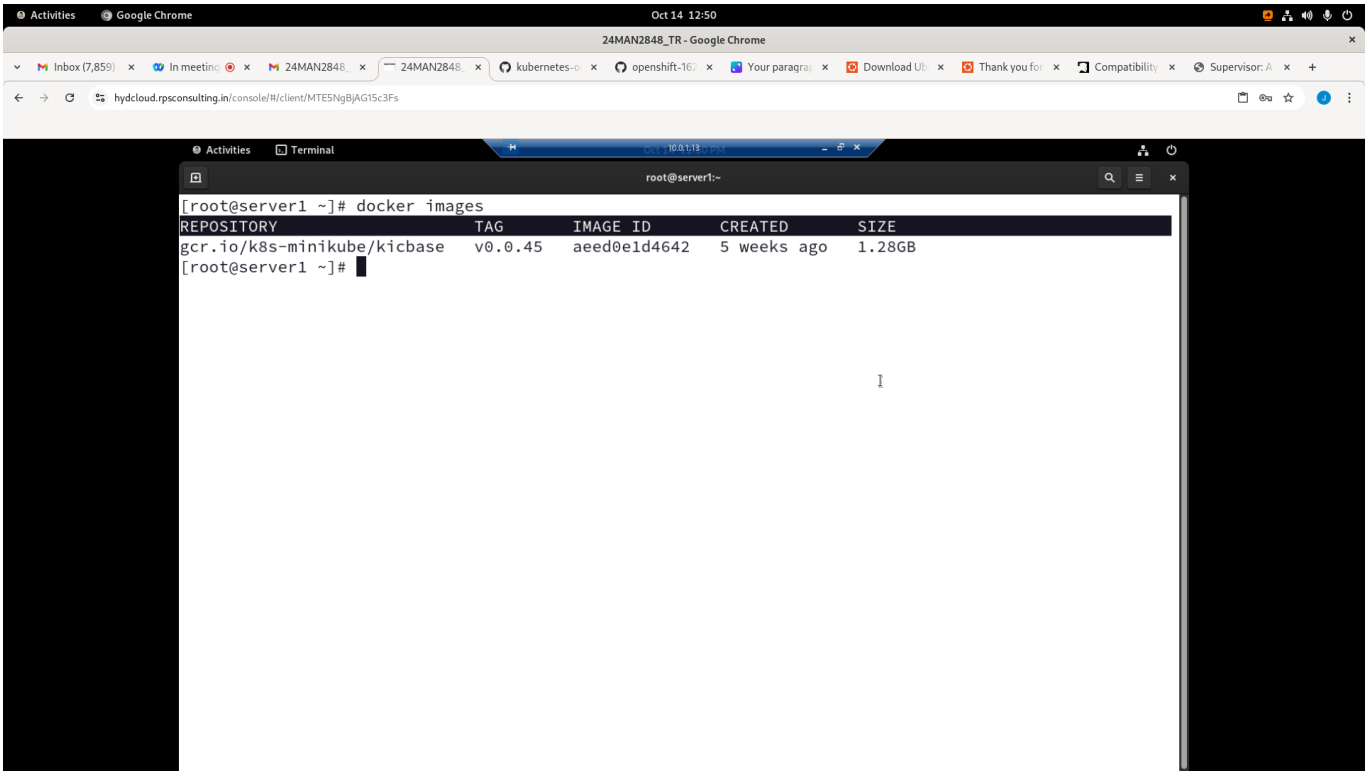
Expected output



Lab - Listing docker images in your local docker registry

```
docker images
```

Expected output



Lab - Create a container and run it in background

Create and run the container in the background(as a daemon)

```
docker run -dit --name ubuntu-jegan --hostname ubuntu-jegan ubuntu:24.04
/bin/bash
```

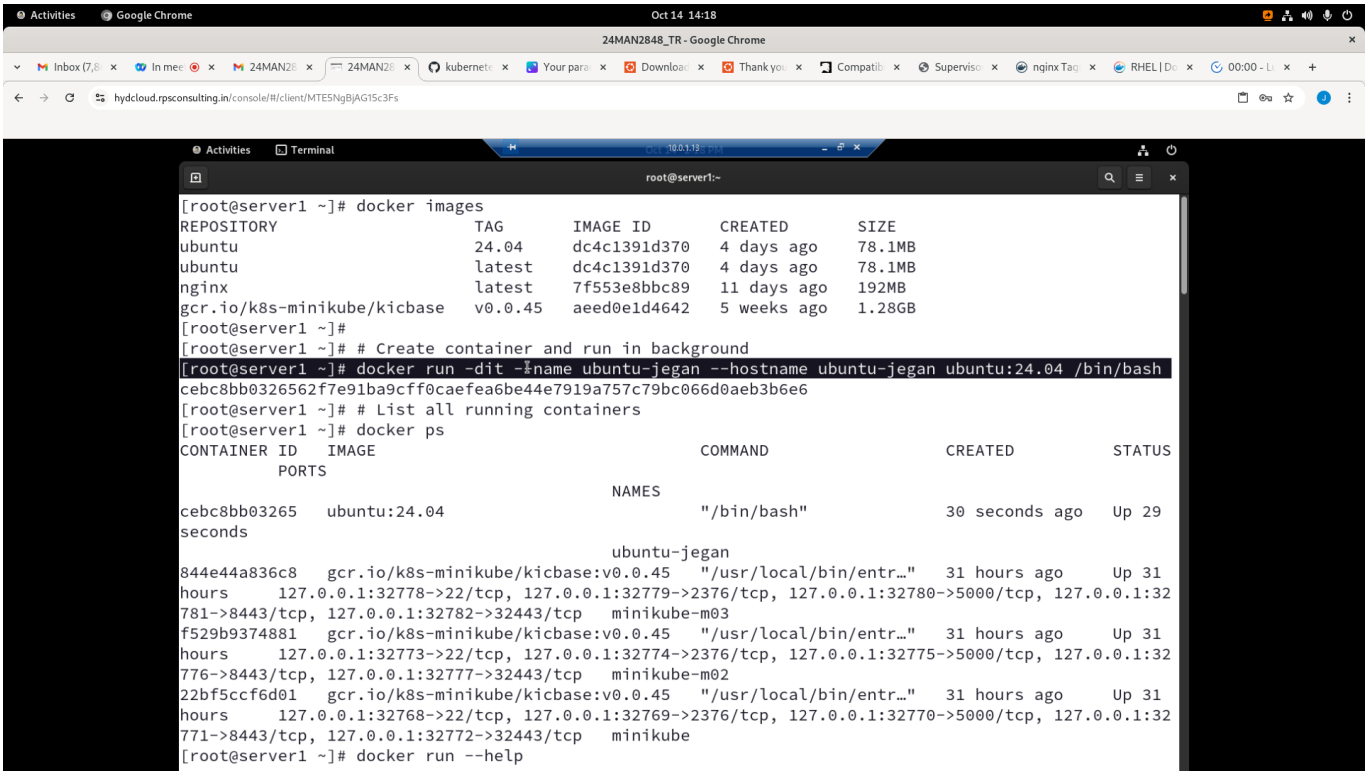
In the above command

```
dit - stands for detached interactive terminal
name - unique name of the container, replace 'jegan' with your name
hostname - though name and hostname can be different, as a best practice
make sure they are same to avoid confusion
ubuntu:24.04 - docker image name
/bin/bash - application that you wish to run inside the container
```

List all the running contianers

```
docker ps
```

Expected output



Lab - Listing all containers including the exited ones

Stopping a running container

```
docker stop ubuntu-jegan
```

List all containers

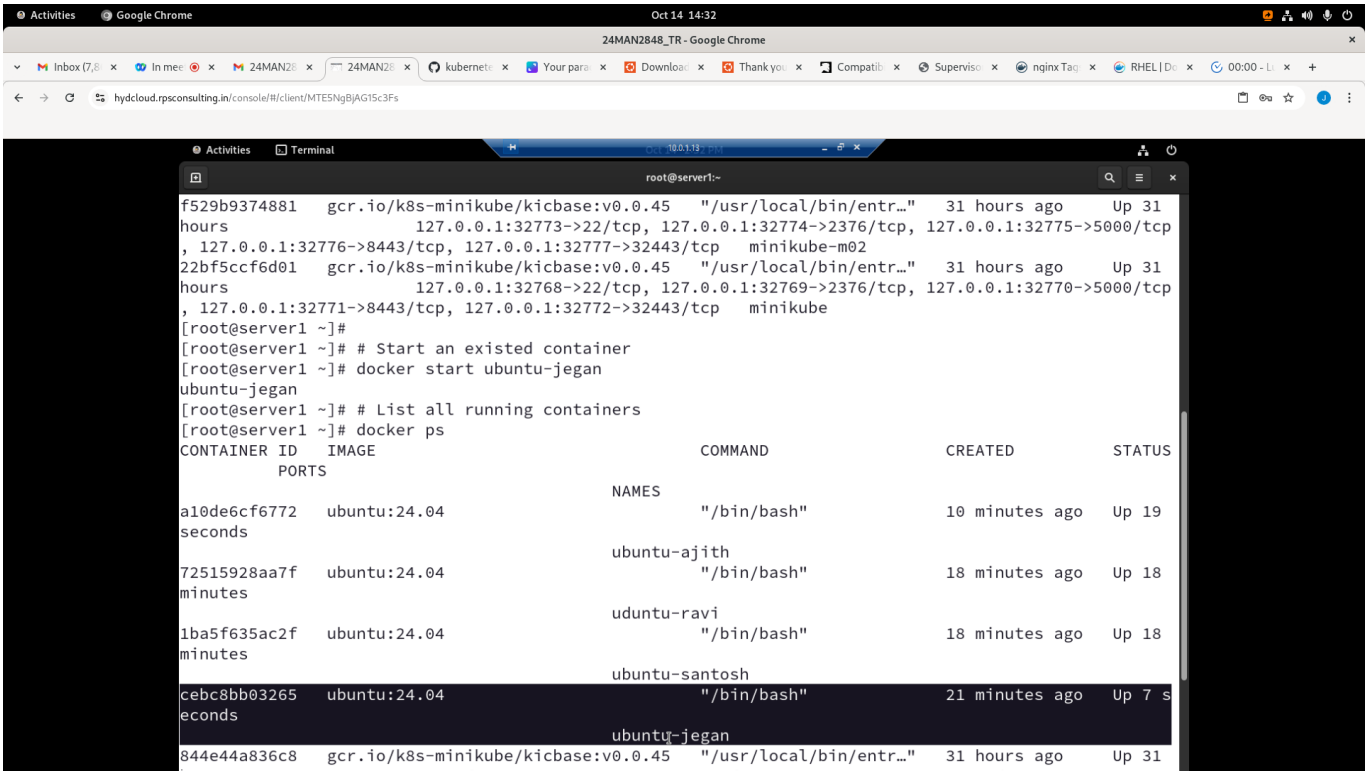
```
docker ps -a
```

Expected output

Lab - Starting an exited container

```
docker ps -a
docker start ubuntu-jegan
docker ps
```

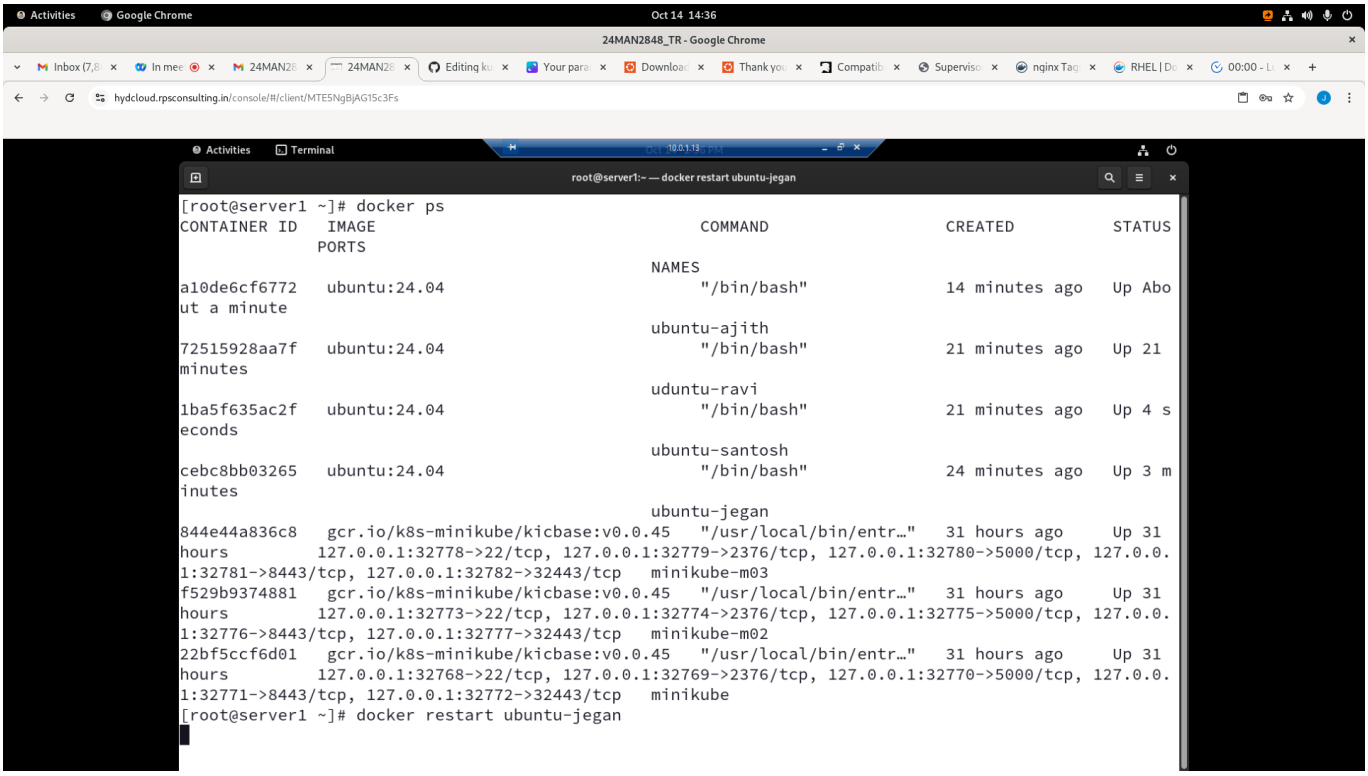
Expected output

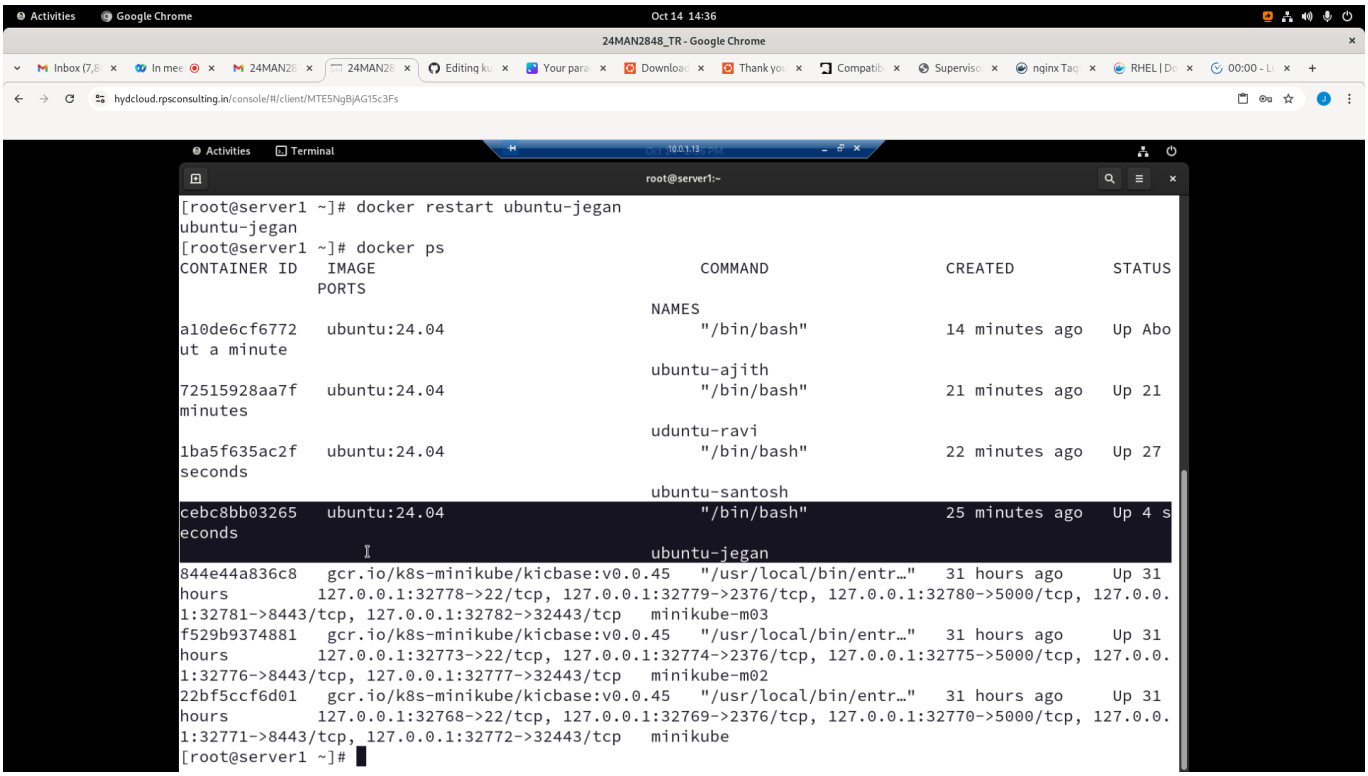


Lab - Restart a running container

```
docker ps
docker restart ubuntu-jegan
docker ps
```

Expected output

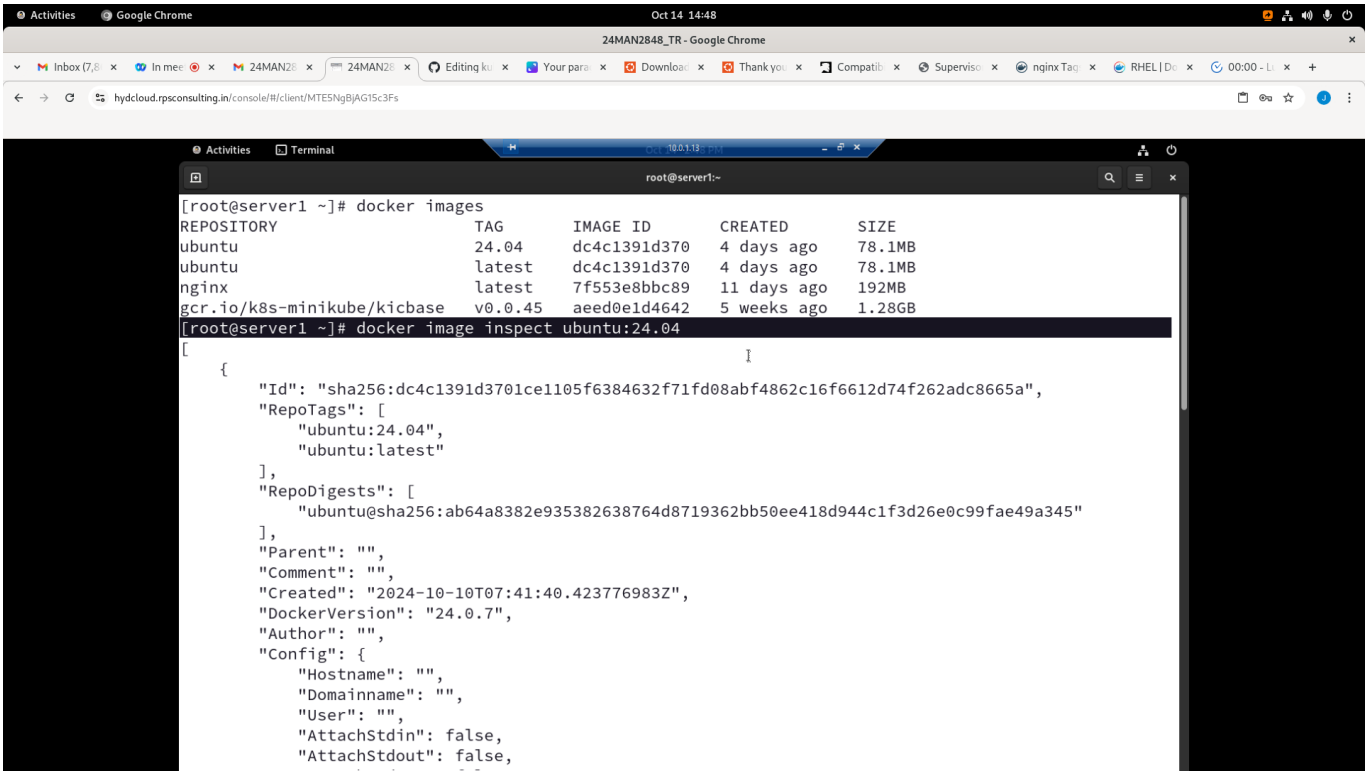


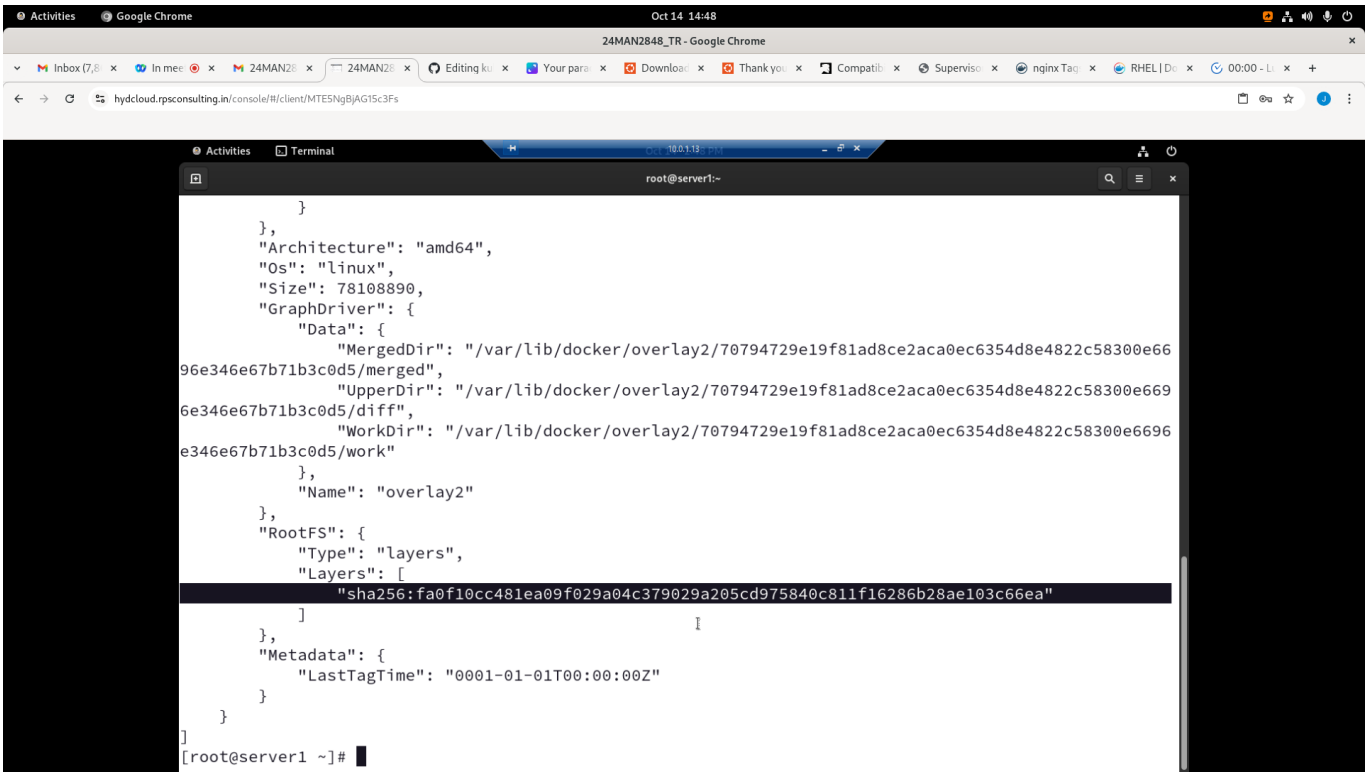


Lab - Finding more details about a docker image

```
docker images
docker image ubuntu:24.04
```

Expected output





Lab - Creating a custom Docker Image

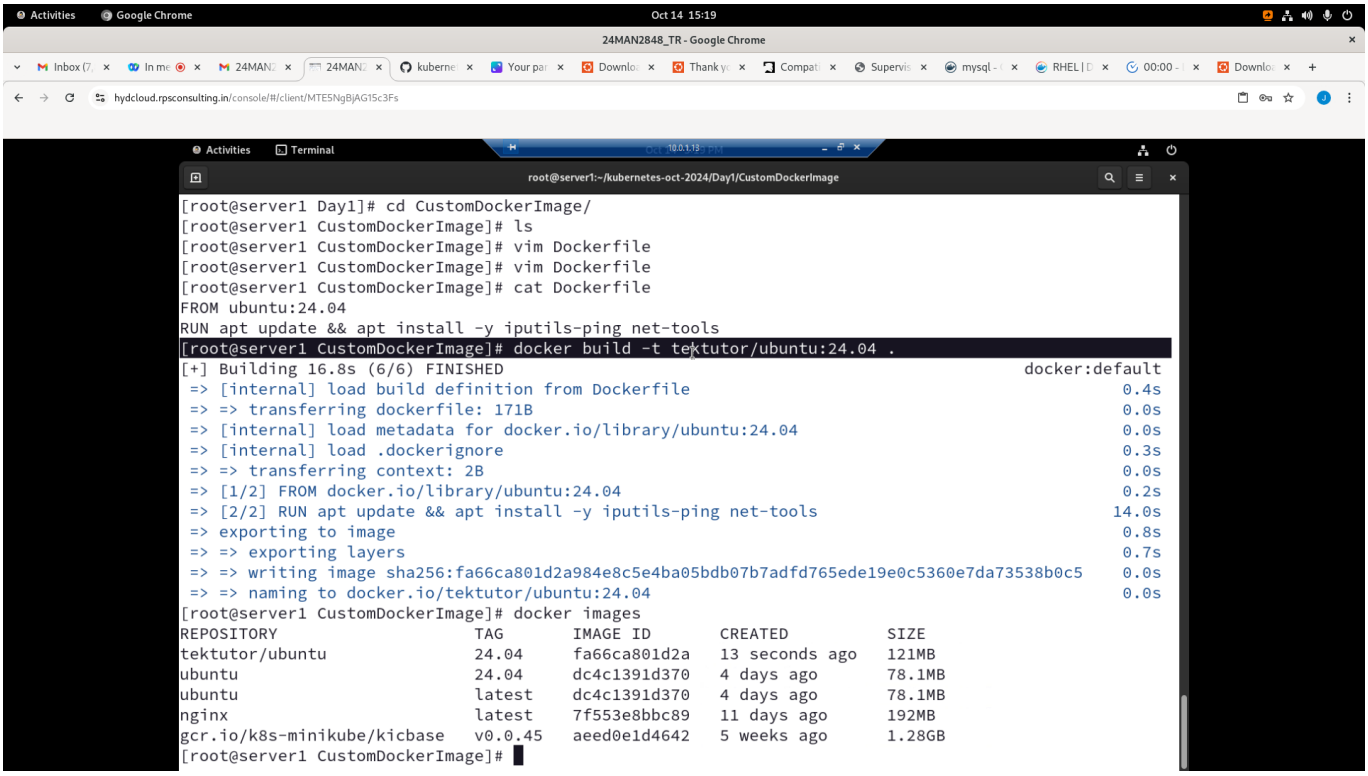
Create a file named Dockerfile with the below content

```
FROM ubuntu:24.04
RUN apt update && apt install -y iputils-ping net-tools
```

Build a custom docker image

```
docker build -t tektutor/ubuntu:24.04 .
docker images
```

Expected output



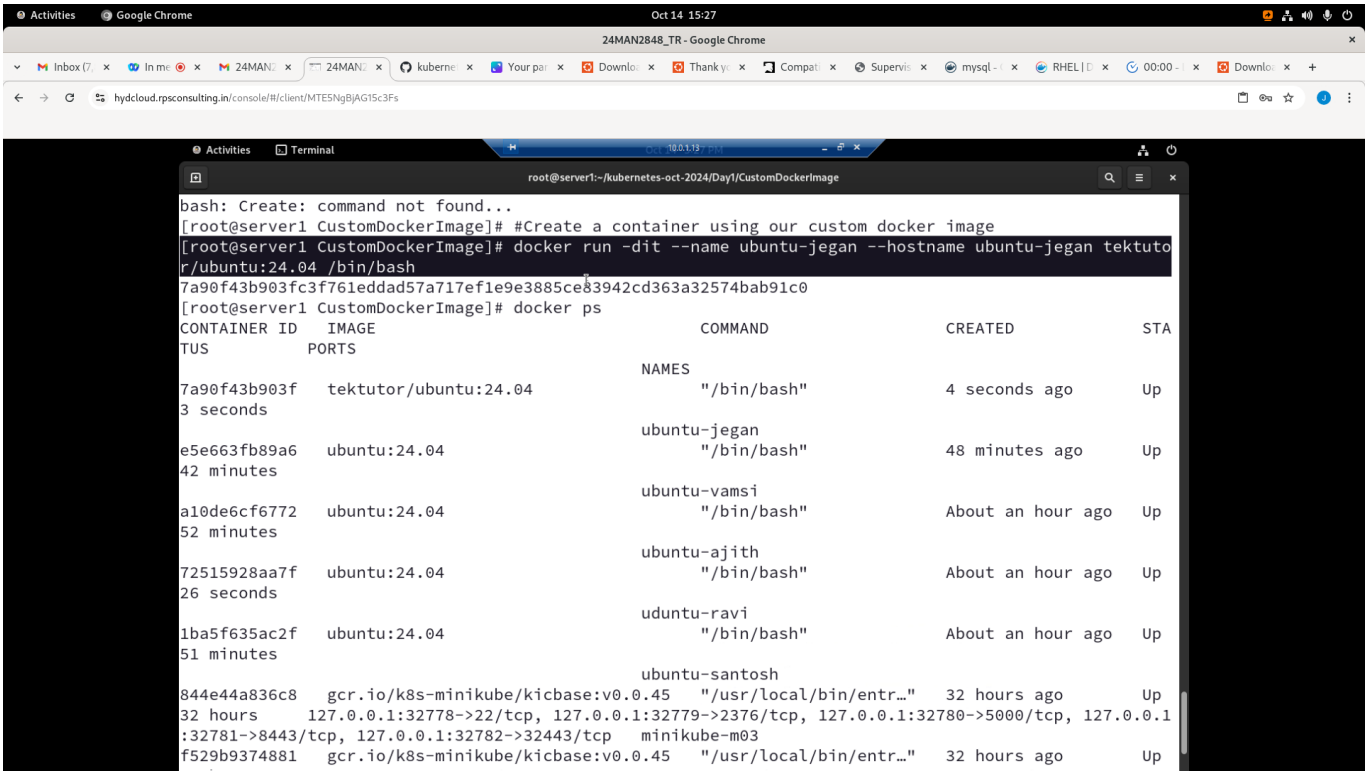
Delete the existing container

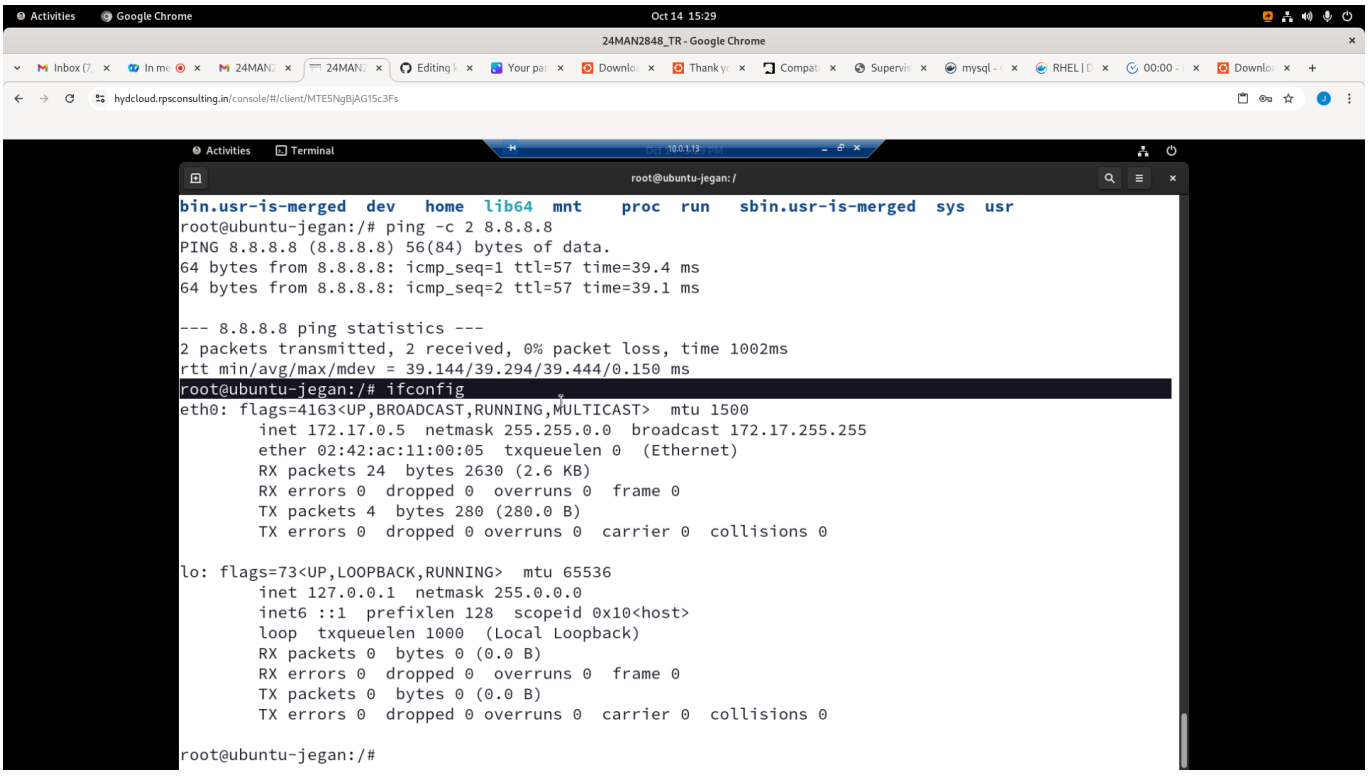
```
docker rm -f ubuntu-jegan
docker ps -a
```

Create a container using our custom docker image

```
docker run -dit --name ubuntu-jegan --hostname ubuntu-jegan
tektutor/ubuntu:24.04 /bin/bash
docker ps
docker exec -it ubuntu-jegan bash
ping -c 2 8.8.8.8
ifconfig
exit
```

Expected output





Lab - another example of custom docker image

Create a Dockerfile with below content

```
FROM ubuntu:24.04
RUN apt update && apt install -y iputils-ping net-tools
RUN apt update && apt install -y default-jdk maven
```

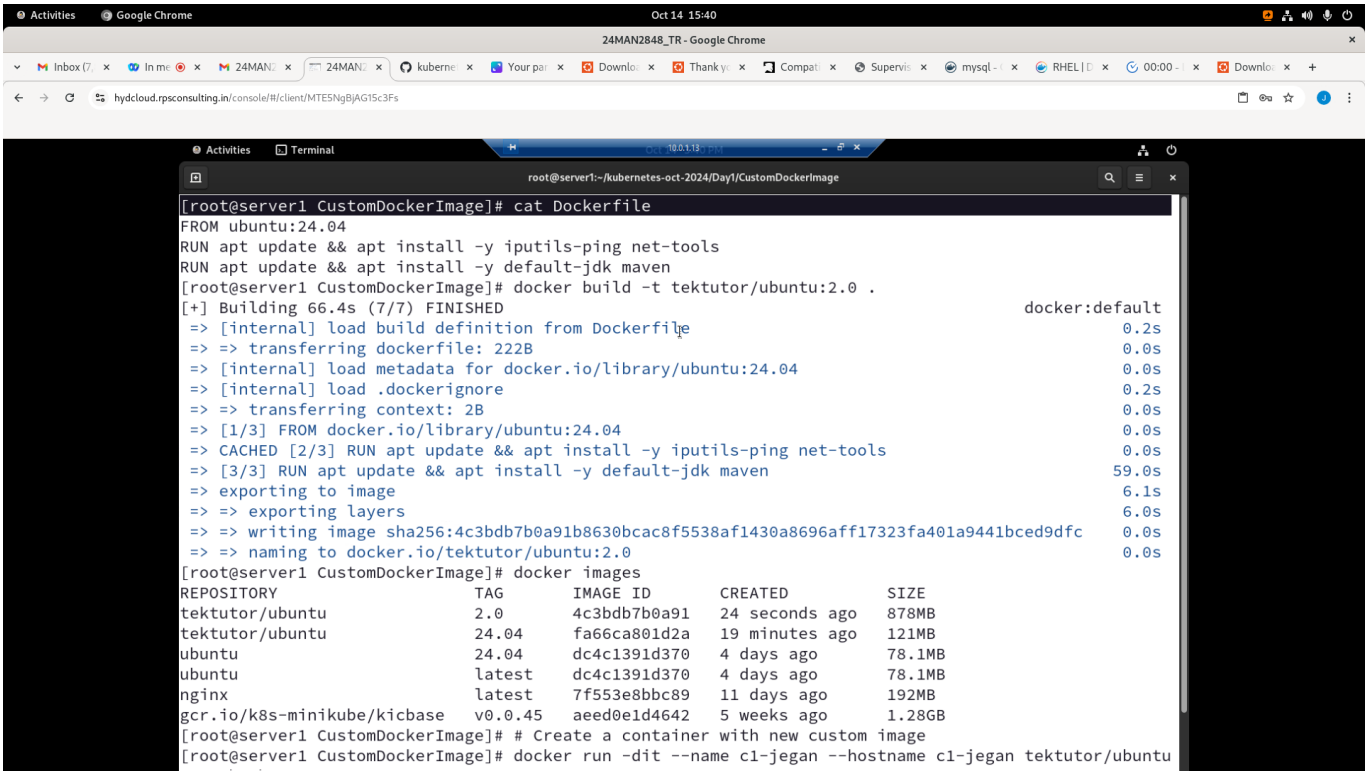
Build the custom image

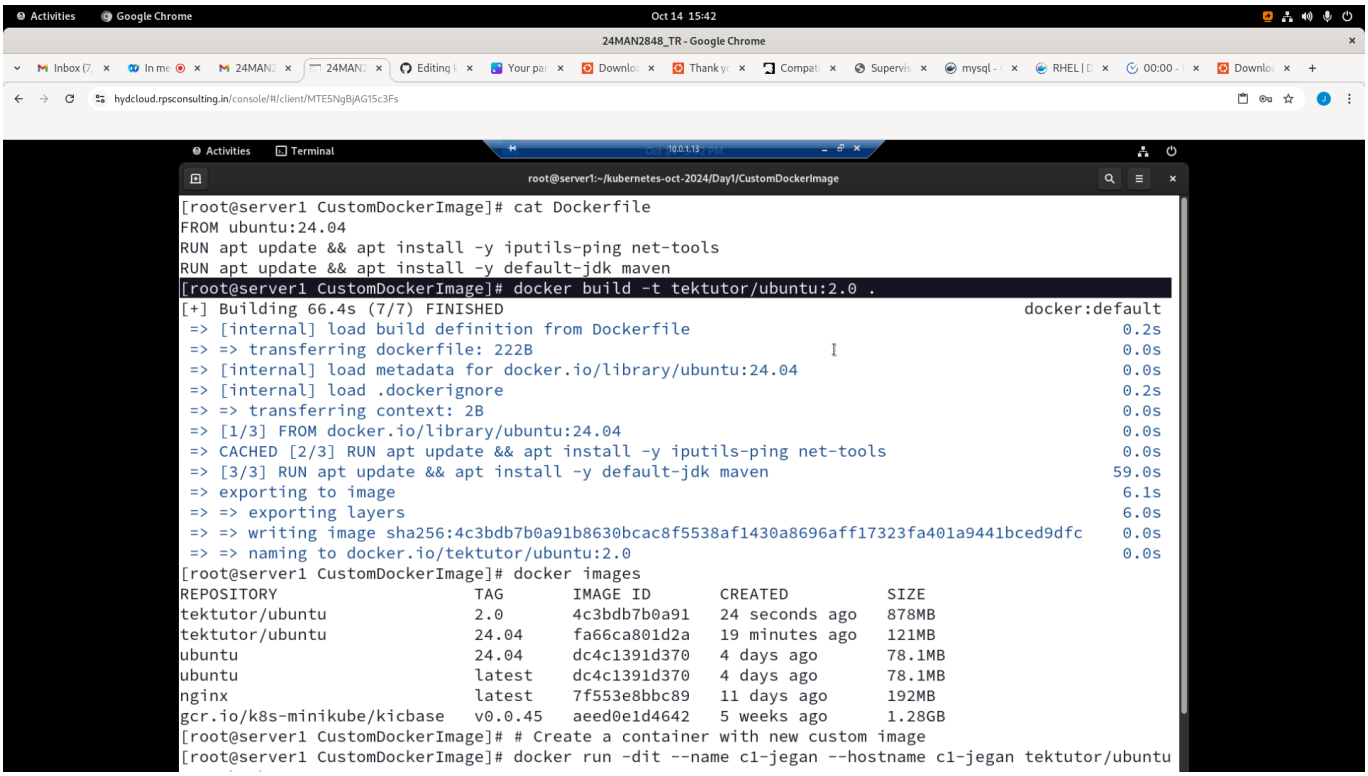
```
docker build -t tektutor/ubuntu:2.0 .
docker images
```

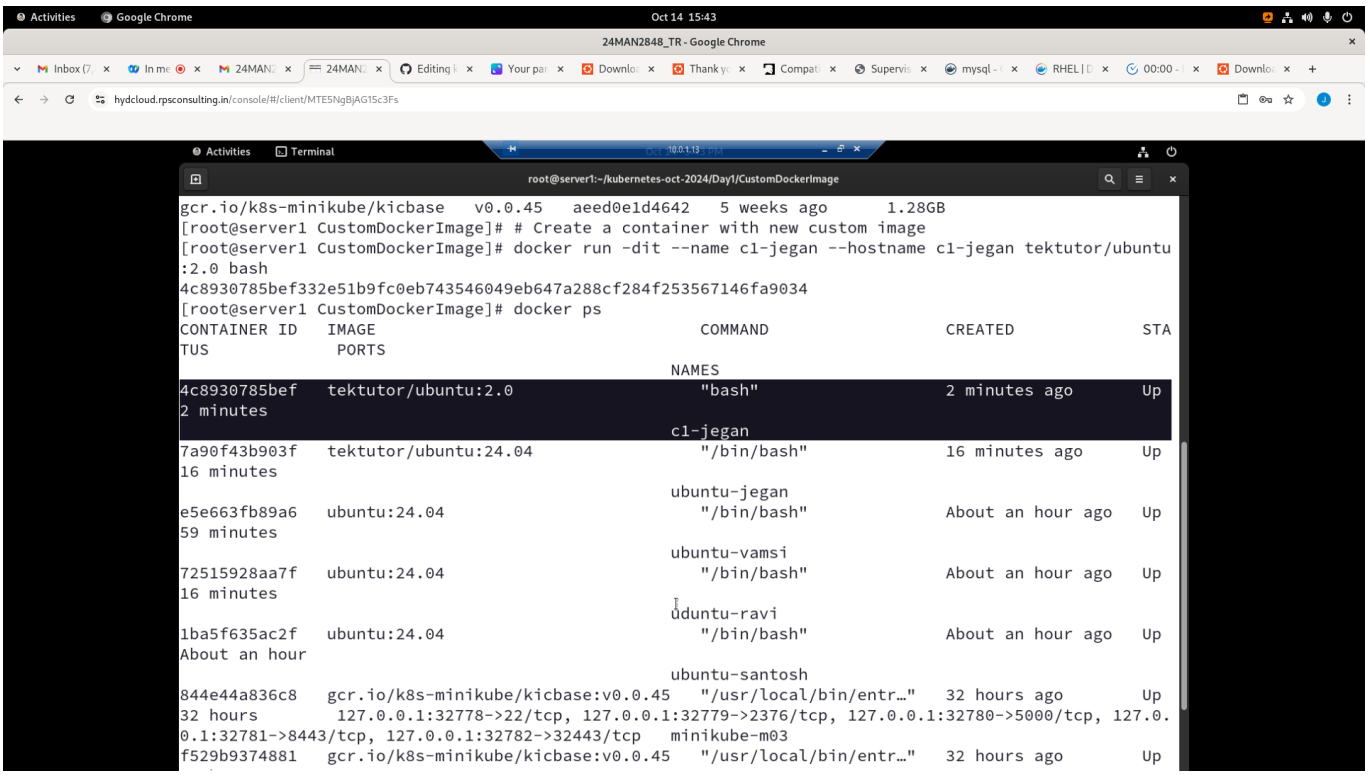
Create a container using custom image

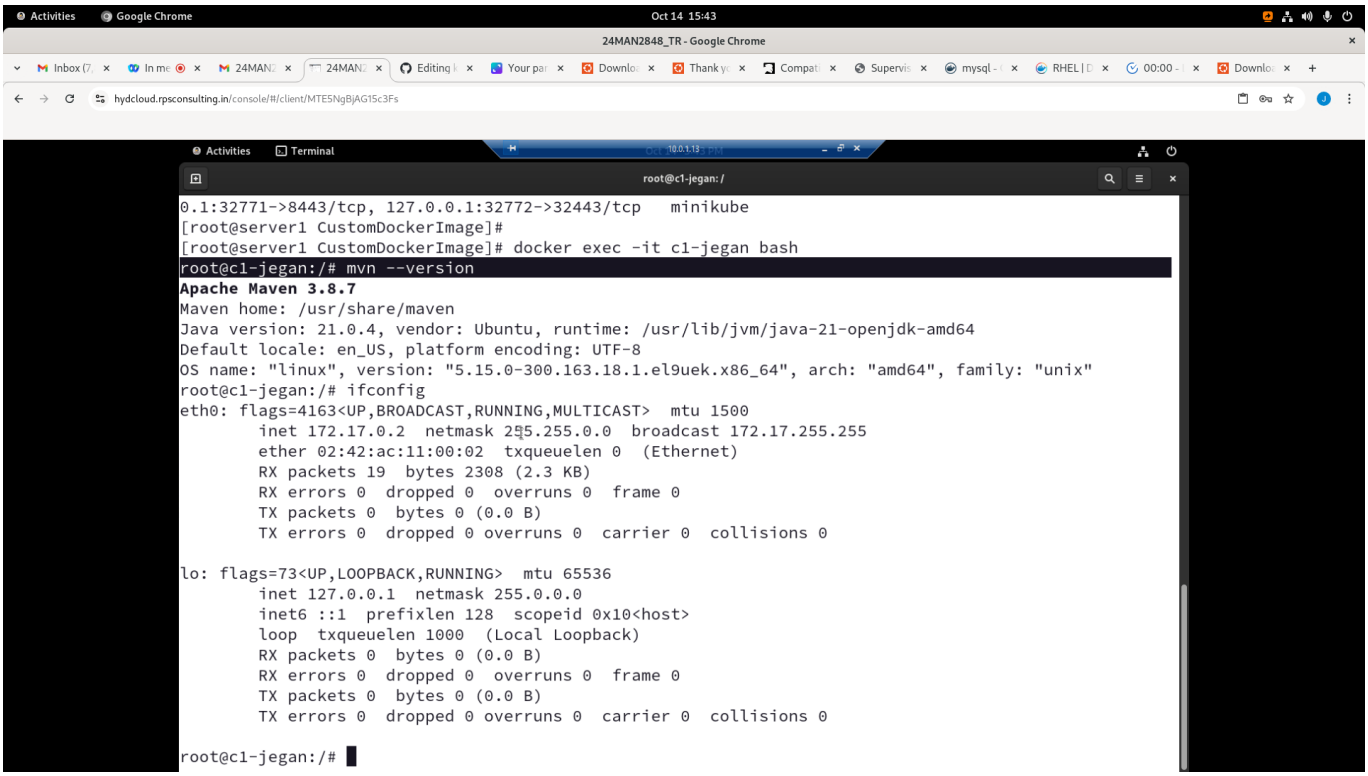
```
docker run -dit --name c1-jegan --hostname c1-jegan tektutor/ubuntu:2.0
bash
docker ps
docker exec -it c1-jegan bash
mvn --version
exit
```

Expected output




Activities
Google Chrome
Oct 14 15:42
24MAN2848_TR - Google Chrome
hydcloud.rpsconsulting.in/console/#/client/MTE5NgBjAG15c3Fs
Activities
Terminal
root@server1:~/kubernetes-oct-2024/Day1/CustomDockerImage
[root@server1 CustomDockerImage]# cat Dockerfile
FROM ubuntu:24.04
RUN apt update && apt install -y iputils-ping net-tools
RUN apt update && apt install -y default-jdk maven
[root@server1 CustomDockerImage]# docker build -t tektutor/ubuntu:2.0 .
[+] Building 66.4s (7/7) FINISHED docker:default
=> [internal] load build definition from Dockerfile 0.2s
=> => transferring dockerfile: 222B 0.0s
=> [internal] load metadata for docker.io/library/ubuntu:24.04 0.0s
=> [internal] load .dockerignore 0.2s
=> => transferring context: 2B 0.0s
=> [1/3] FROM docker.io/library/ubuntu:24.04 0.0s
=> CACHED [2/3] RUN apt update && apt install -y iputils-ping net-tools 0.0s
=> [3/3] RUN apt update && apt install -y default-jdk maven 59.0s
=> exporting to image 6.1s
=> => exporting layers 6.0s
=> => writing image sha256:4c3bdb7b0a91b8630bcac8f5538af1430a8696aff17323fa401a9441bced9dfc 0.0s
=> => naming to docker.io/tektutor/ubuntu:2.0 0.0s
[root@server1 CustomDockerImage]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
tektutor/ubuntu 2.0 4c3bdb7b0a91 24 seconds ago 878MB
tektutor/ubuntu 24.04 fa66ca801d2a 19 minutes ago 121MB
ubuntu 24.04 dc4c1391d370 4 days ago 78.1MB
ubuntu latest dc4c1391d370 4 days ago 78.1MB
nginx latest 7f553e8bbc89 11 days ago 192MB
gcr.io/k8s-minikube/kicbase v0.0.45 aeed0e1d4642 5 weeks ago 1.28GB
[root@server1 CustomDockerImage]# # Create a container with new custom image
[root@server1 CustomDockerImage]# docker run -dit --name c1-jegan --hostname c1-jegan tektutor/ubuntu

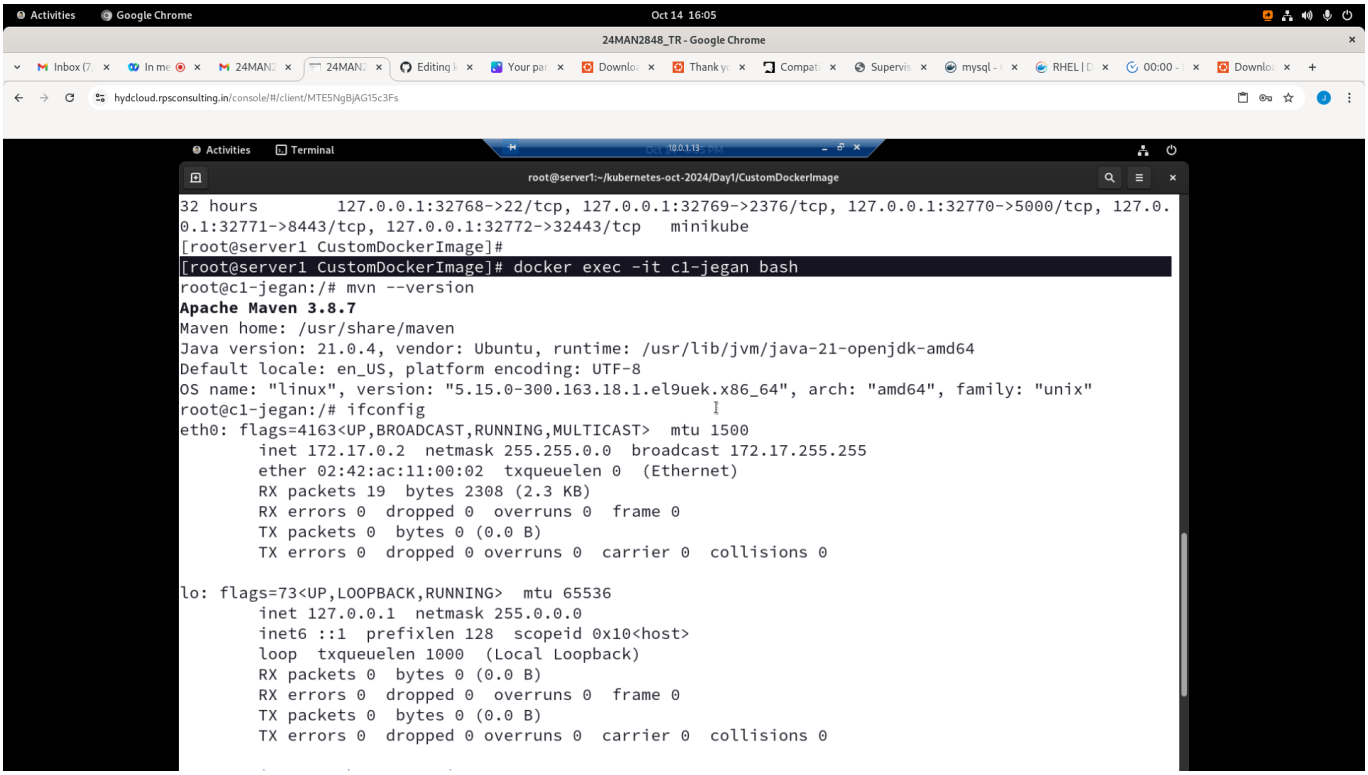

Activities
Google Chrome
Oct 14 15:43
24MAN2848_TR - Google Chrome
hydcloud.rpsconsulting.in/console/#/client/MTE5NgBjAG15c3Fs
Activities
Terminal
root@server1:~/kubernetes-oct-2024/Day1/CustomDockerImage
gcr.io/k8s-minikube/kicbase v0.0.45 aeed0e1d4642 5 weeks ago 1.28GB
[root@server1 CustomDockerImage]# # Create a container with new custom image
[root@server1 CustomDockerImage]# docker run -dit --name c1-jegan --hostname c1-jegan tektutor/ubuntu
:2.0 bash
4c8930785bef332e51b9fc0eb743546049eb647a288cf284f253567146fa9034
[root@server1 CustomDockerImage]# docker ps
CONTAINER ID IMAGE COMMAND CREATED STA
TUS PORTS
NAMES
4c8930785bef tektutor/ubuntu:2.0 "bash" 2 minutes ago Up
2 minutes
c1-jegan
7a90f43b903f tektutor/ubuntu:24.04 "/bin/bash" 16 minutes ago Up
16 minutes
ubuntu-jegan
e5e663fb89a6 ubuntu:24.04 "/bin/bash" About an hour ago Up
59 minutes
ubuntu-vamsi
72515928aa7f ubuntu:24.04 "/bin/bash" About an hour ago Up
16 minutes
uduntu-ravi
1ba5f635ac2f ubuntu:24.04 "/bin/bash" About an hour ago Up
About an hour
ubuntu-santosh
844e44a836c8 gcr.io/k8s-minikube/kicbase:v0.0.45 "/usr/local/bin/entr…" 32 hours ago Up
32 hours 127.0.0.1:32778->22/tcp, 127.0.0.1:32779->2376/tcp, 127.0.0.1:32780->5000/tcp, 127.0.
0.1:32781->8443/tcp, 127.0.0.1:32782->32443/tcp minikube-m03
f529b9374881 gcr.io/k8s-minikube/kicbase:v0.0.45 "/usr/local/bin/entr…" 32 hours ago Up

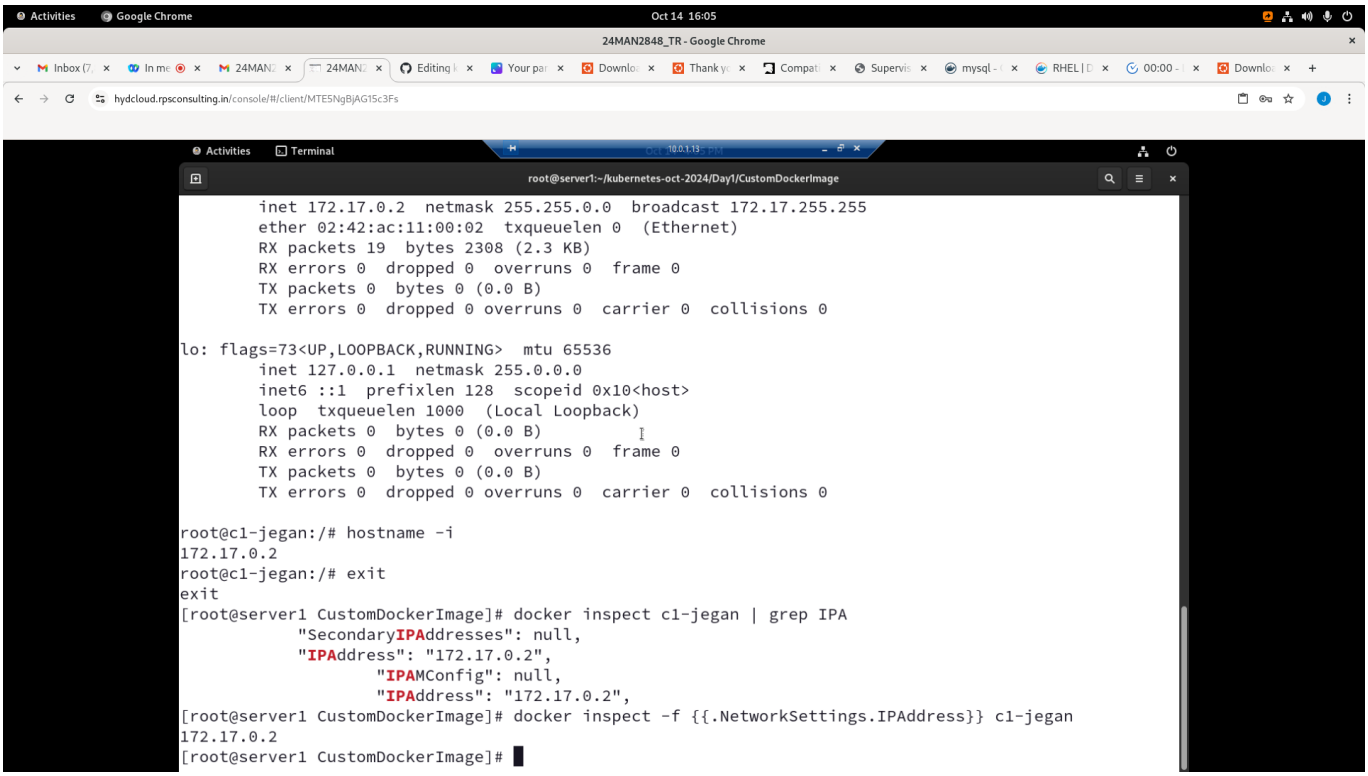


Lab - Find IP address of a container

```
docker ps
docker inspect c1-jegan | grep IPA
docker inspect -f {{.NetworkSettings.IPAddress}} c1-jegan
```

Expected output

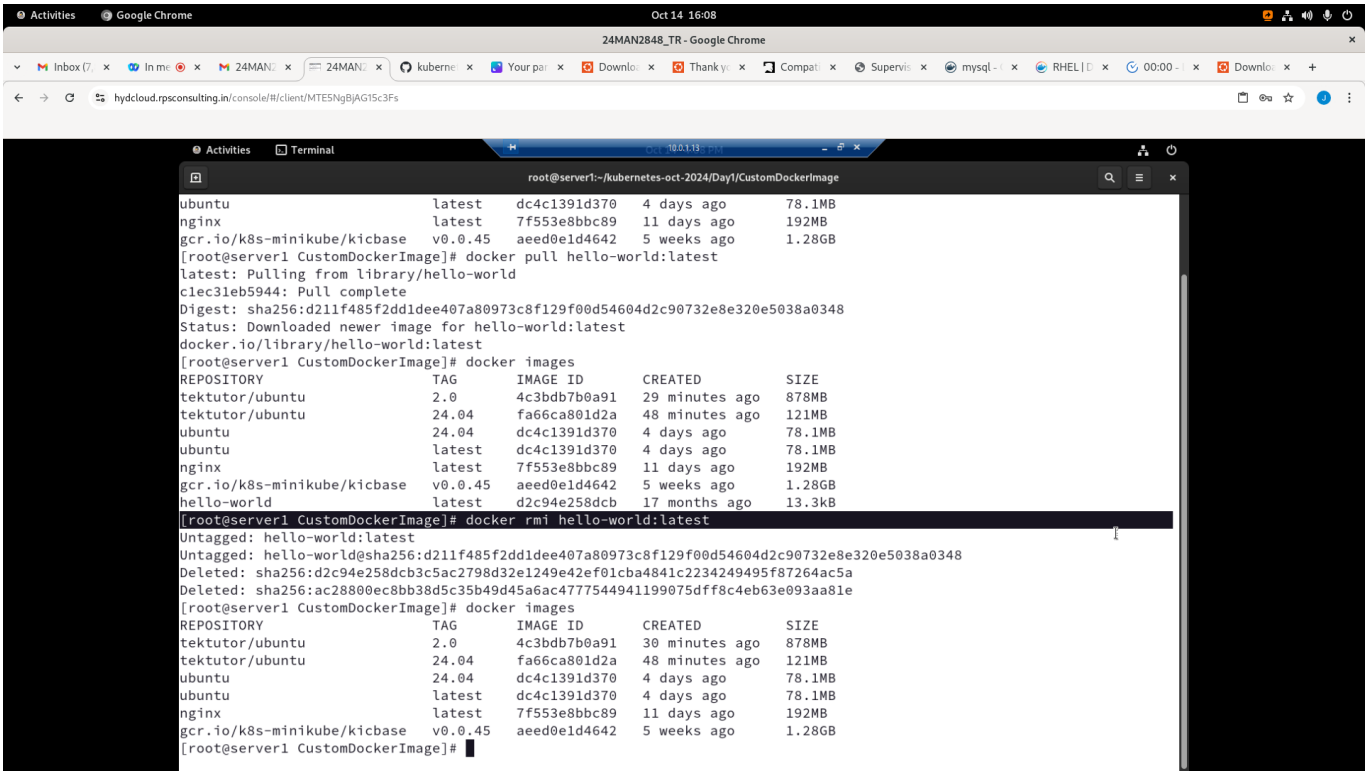




Lab - Deleteing a docker image from local registry

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

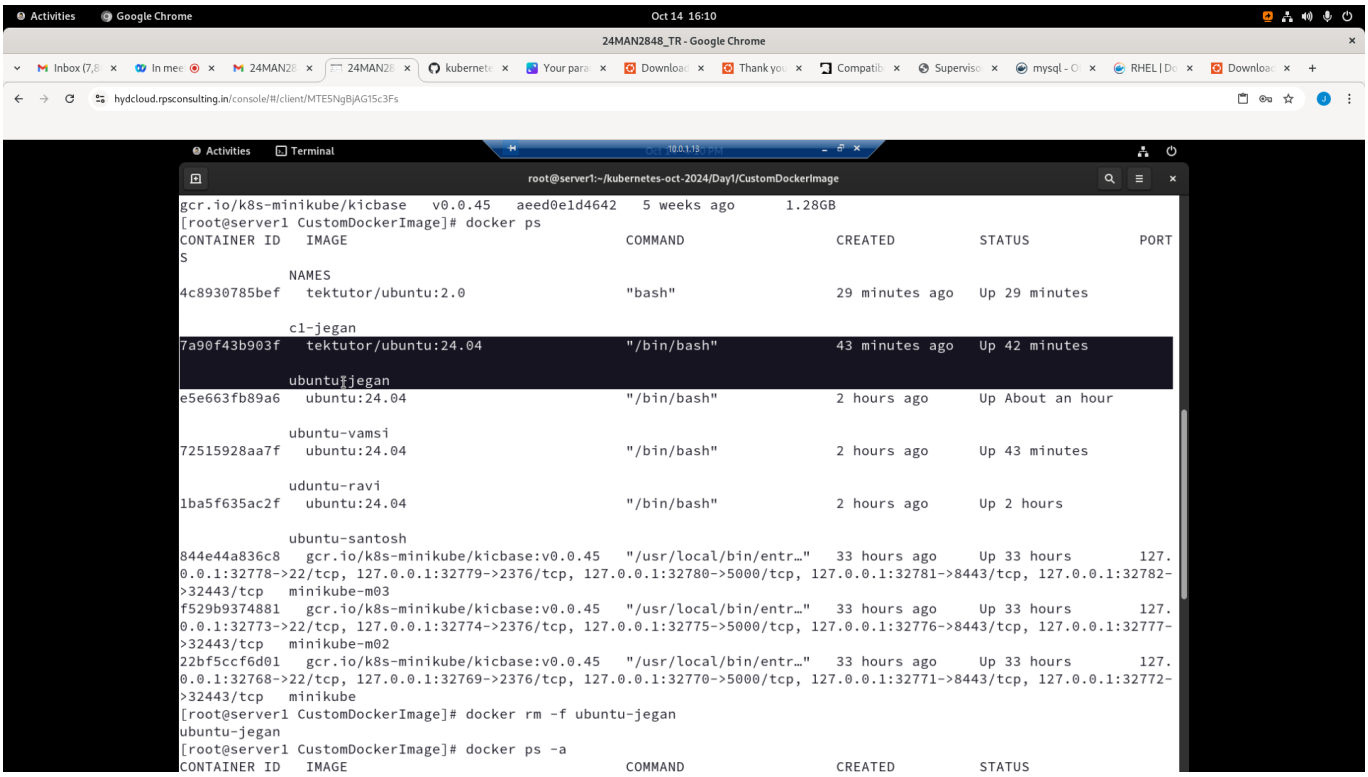
Expected output

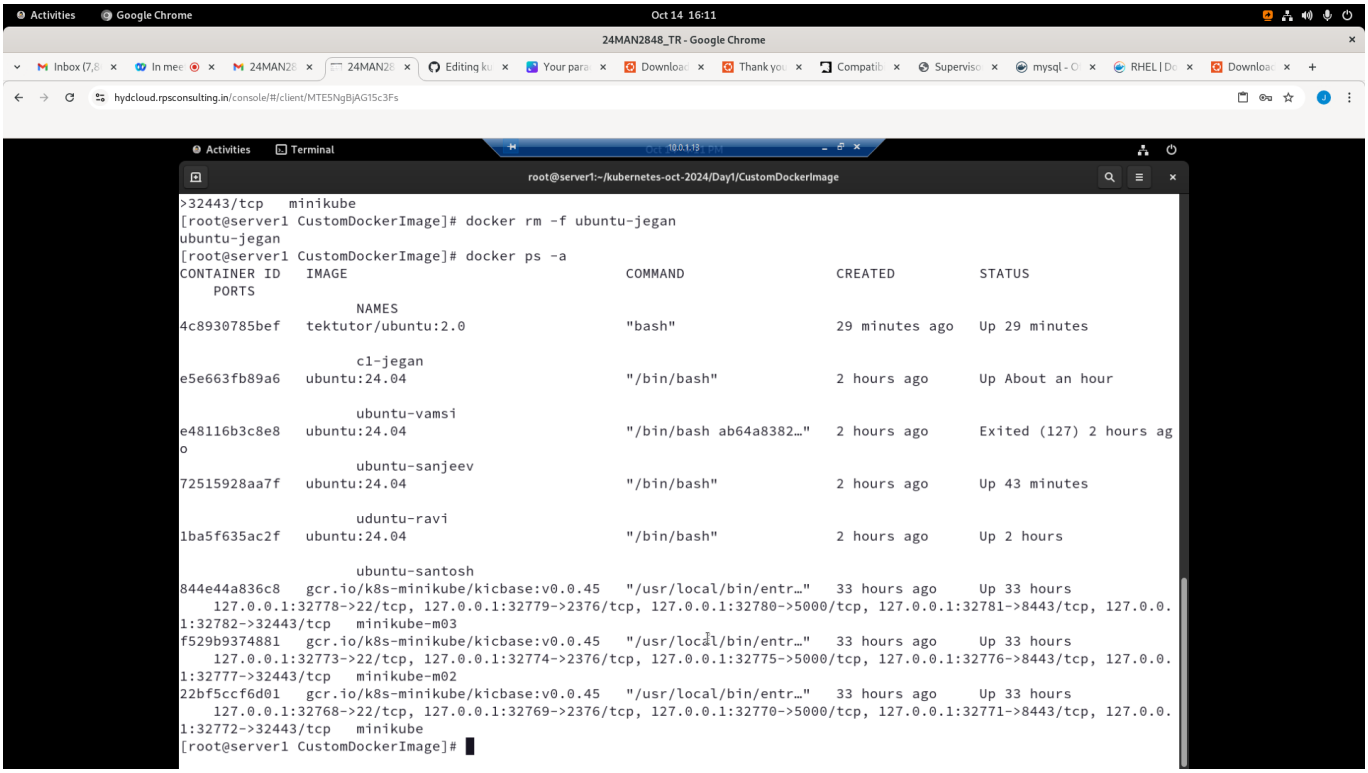


Lab - Deleting a running container

```
docker ps
docker rm -f ubuntu-jegan
docker ps -a
```

Expected output





Lab - Port forward to expose a containerized application for external access

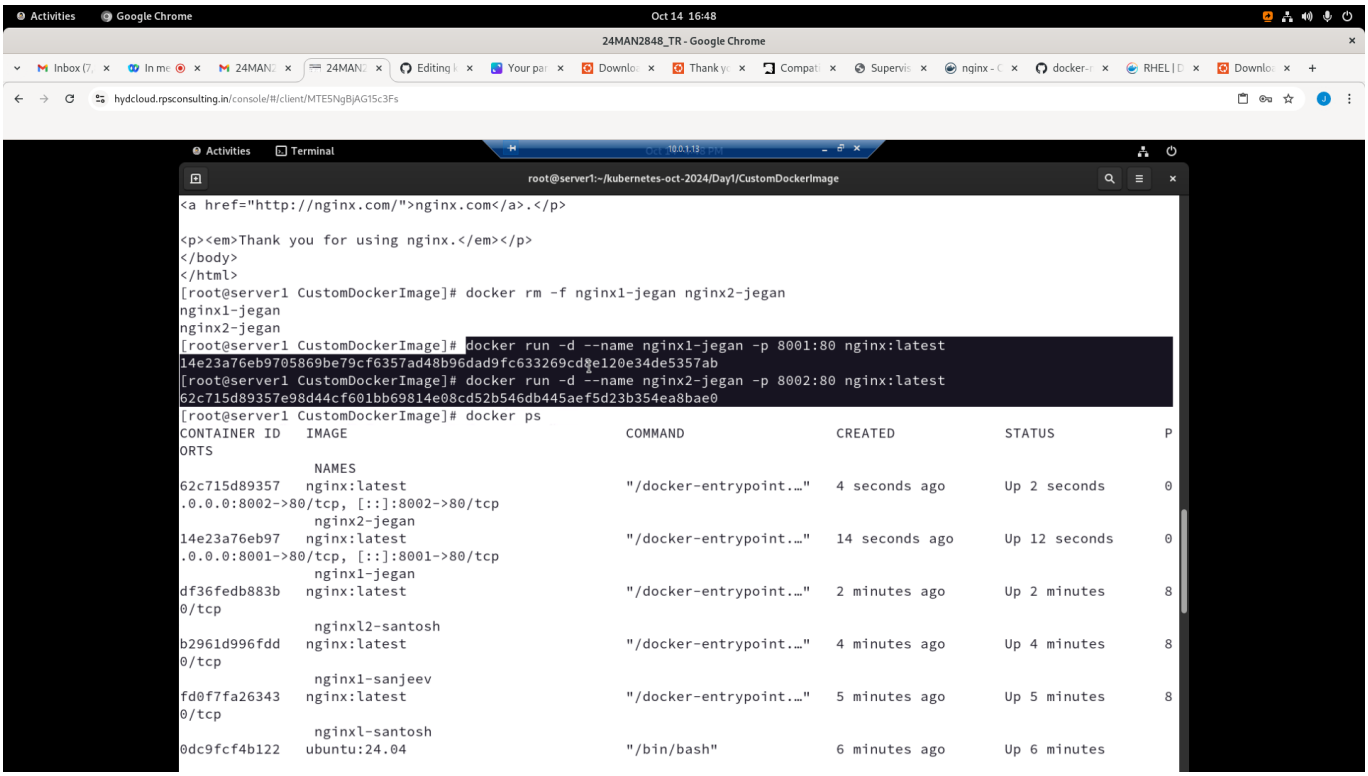
```
docker run -d --name nginx1-jegan -p 8001:80 nginx:latest
docker run -d --name nginx2-jegan -p 8002:80 nginx:latest
docker ps
```

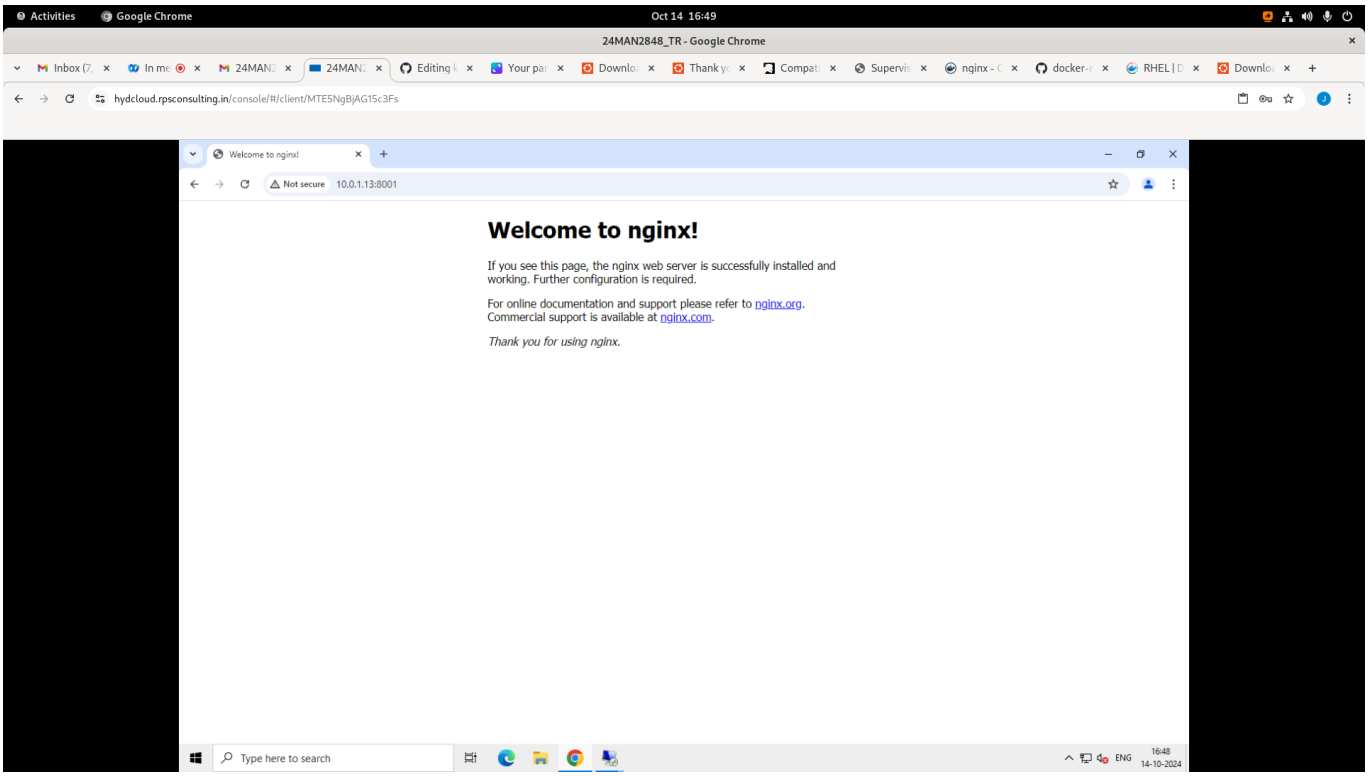
Accessing the nginx web page from external machine

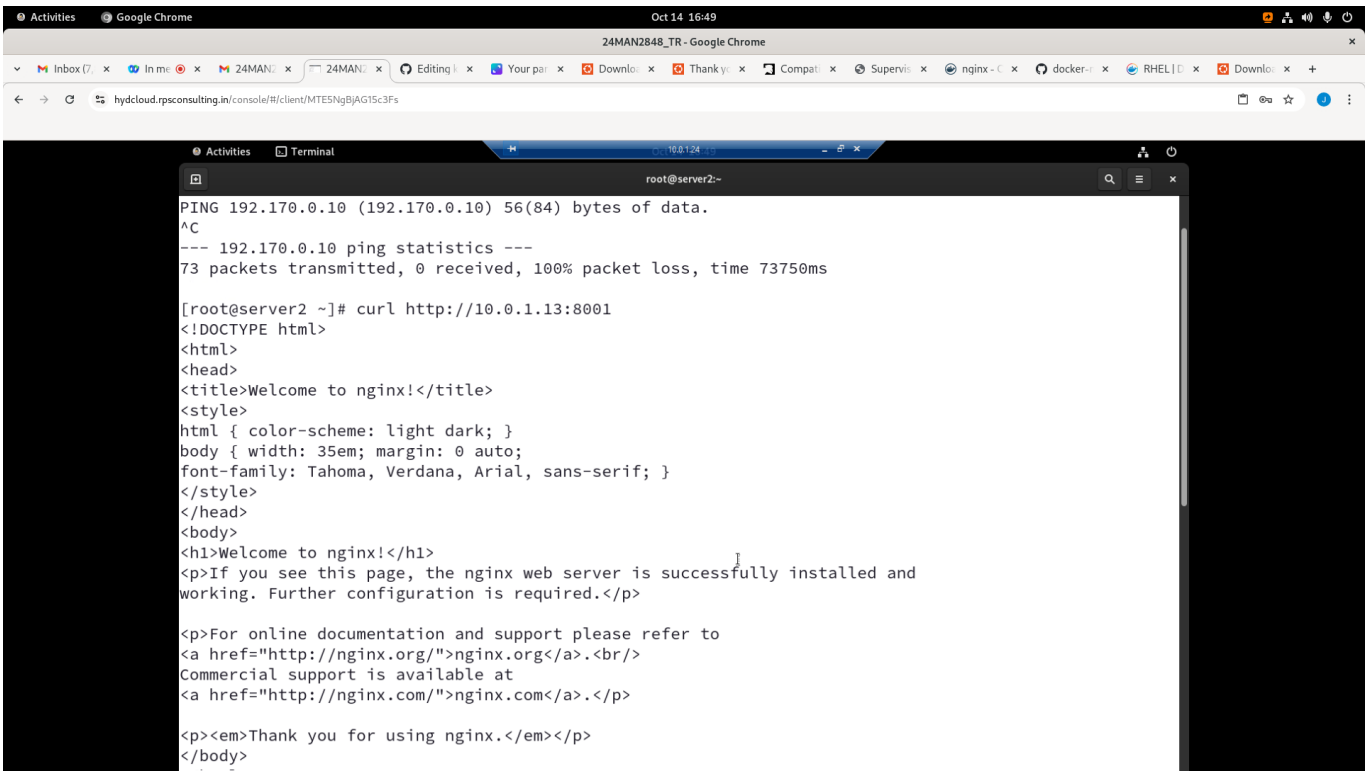
```
http://10.0.1.13:8001
http://10.0.1.13:8002
```

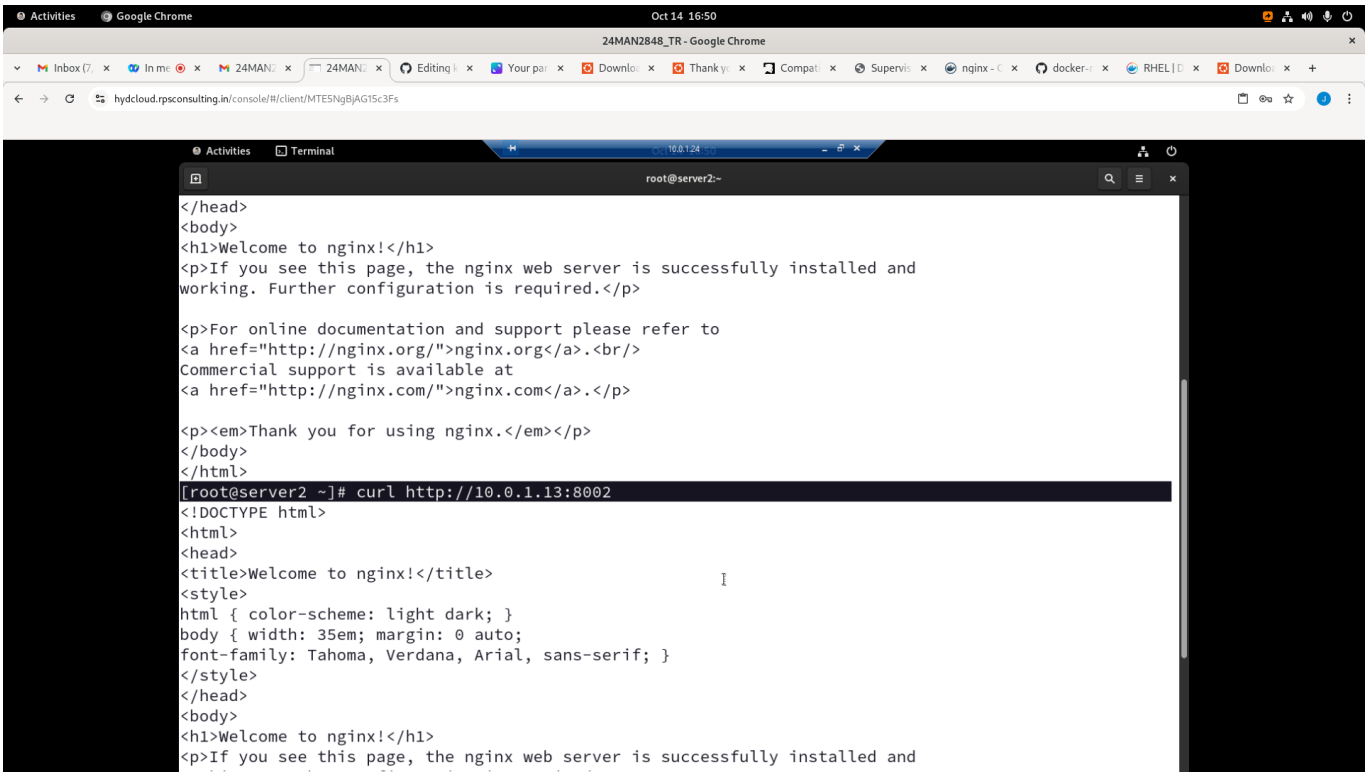
In the above url, 10.0.1.13 is the IP address of server1 linux machine, any request received at port 8001 will be forwarded to nginx1-jegan container at port 80

Expected output




Activities
Google Chrome
Oct 14 16:49
24MAN2848_TR - Google Chrome
hydcloud.rpsconsulting.in/console/#/client/MTE5NgBjAG15c3Fs
Welcome to nginx!
Not secure 10.0.1.13:8001
Welcome to nginx!
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.
Thank you for using nginx.
Type here to search
ENG
16:48
14-10-2024

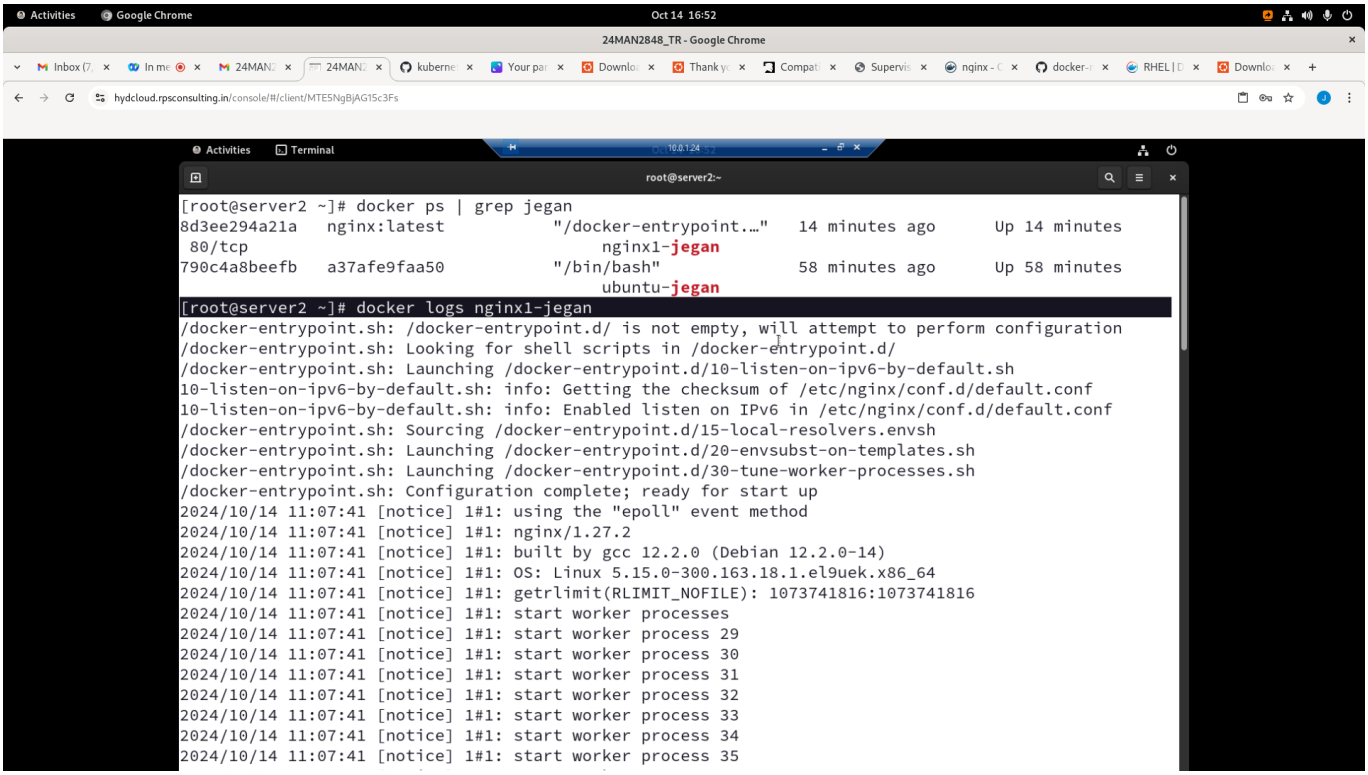

Activities
Google Chrome
Oct 14 16:49
24MAN2848_TR - Google Chrome
hydcloud.rpsconsulting.in/console/#/client/MTE5NgBjAG15c3Fs
Activities
Terminal
root@server2:~
PING 192.170.0.10 (192.170.0.10) 56(84) bytes of data.
^C
--- 192.170.0.10 ping statistics ---
73 packets transmitted, 0 received, 100% packet loss, time 73750ms
[root@server2 ~]# curl http://10.0.1.13:8001
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.
For online documentation and support please refer to
nginx.org.
Commercial support is available at
nginx.com.
Thank you for using nginx.
</body>



Lab - Checking containerized application logs

```
docker ps | grep jegan
docker logs nginx1-jegan
```

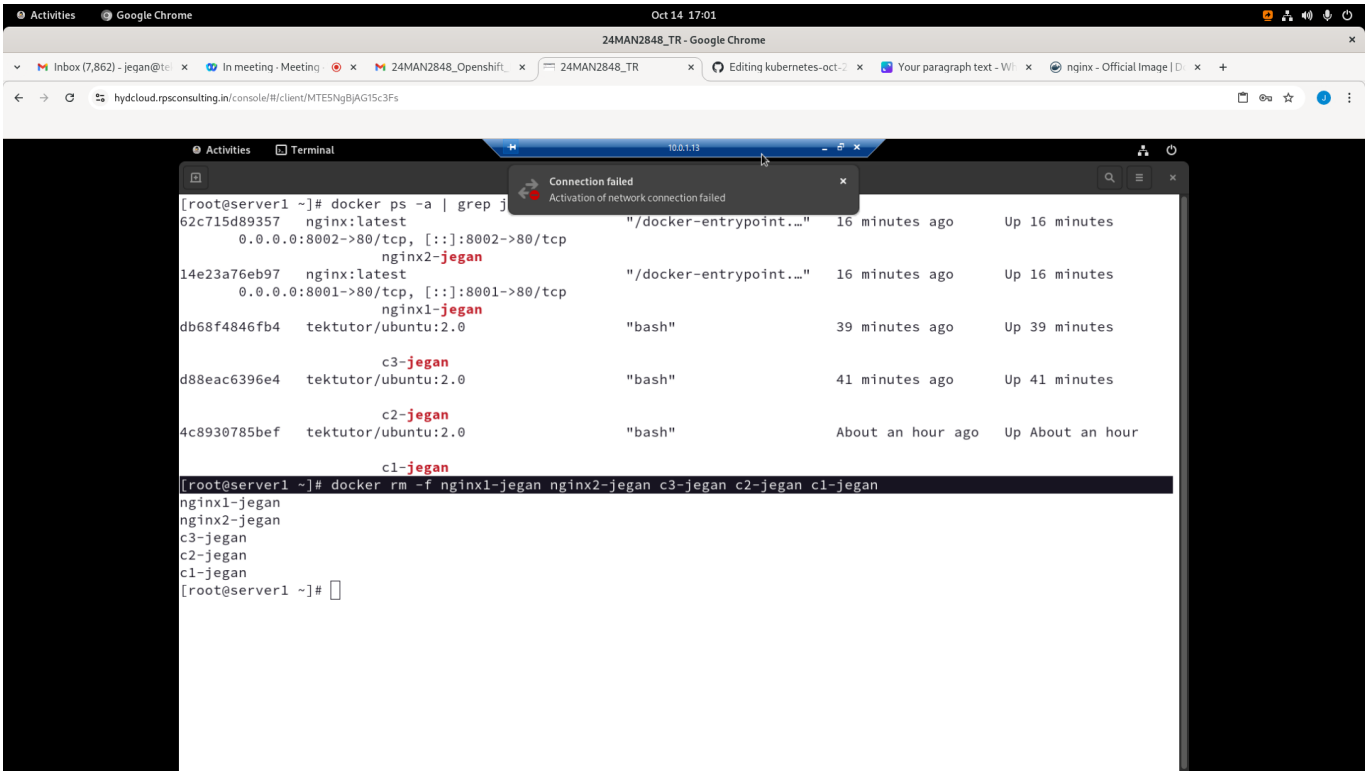
Expected output

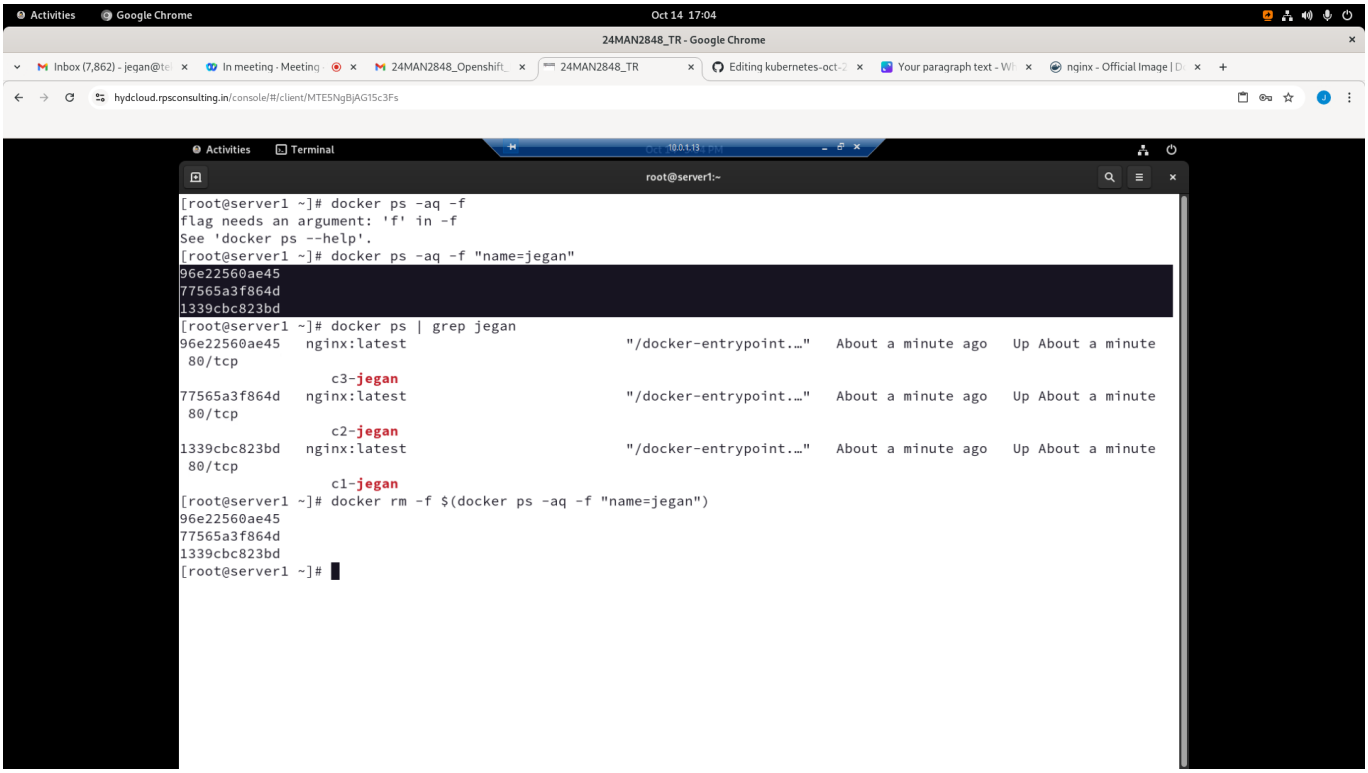


Lab - Deleting multiple containers with single docker command

```
docker ps | grep jegan
docker rm -f nginx1-jegan nginx2-jegan c3-jegan c2-jegan c1-jegan
docker rm -f $(docker ps -aq -f "name=jegan")
```

Expected output



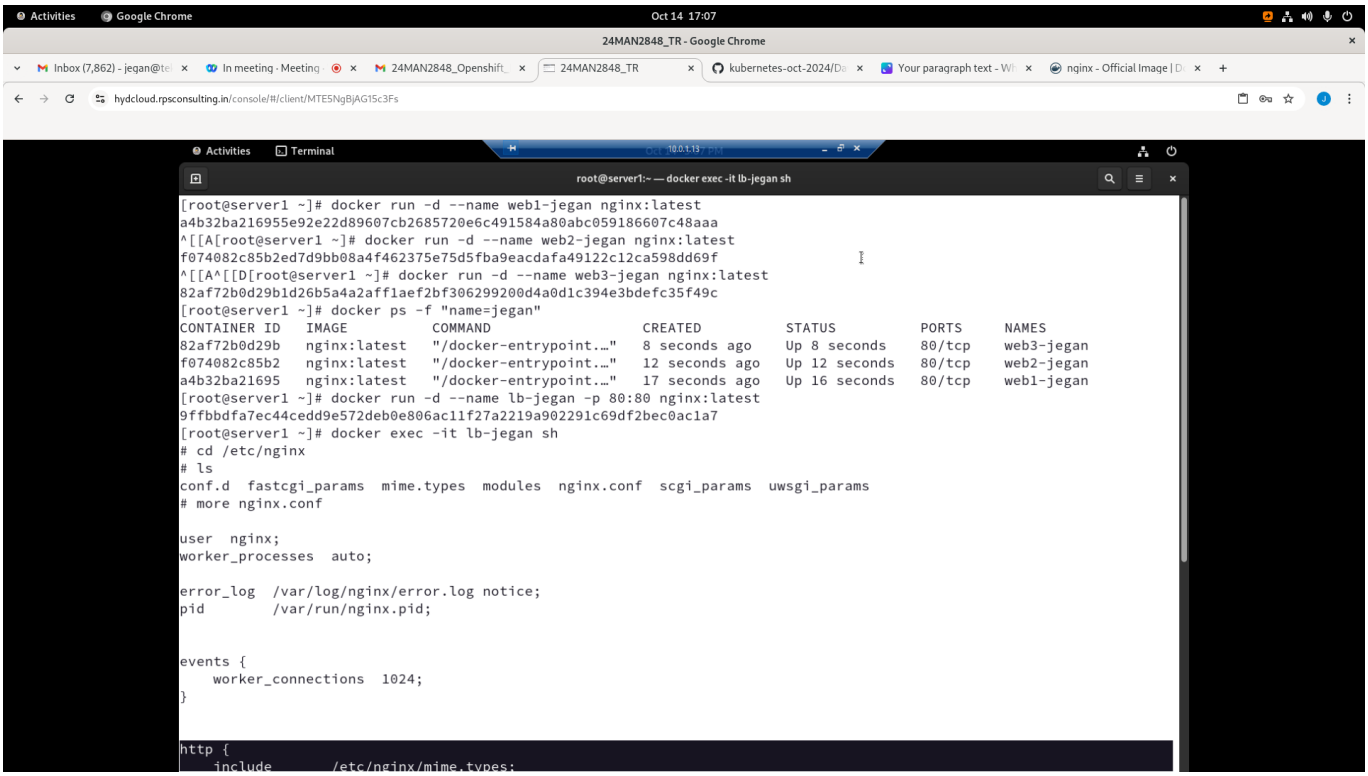


Lab - Setup a load balancer using nginx

Let's create 3 nginx web servers without port forwarding

```
docker run -d --name web1-jegan nginx:latest
docker run -d --name web2-jegan nginx:latest
docker run -d --name web3-jegan nginx:latest
docker ps -f "name=jegan"
```

Expected output



Let's create a load balancer container using nginx with port forwarding to make accessible from external machines

```
docker run -d --name lb-jegan -p 80:80 nginx:latest
docker ps -f "name=jegan"
```

Let's copy the nginx.conf file from lb-jegan container to local system

```
docker cp lb-jegan:/etc/nginx/nginx.conf .
```

Let's modify the http block as shown below nginx.conf on our local machine as shown below

```
http {
    upstream backend {
        server 172.17.0.2:80;
        server 172.17.0.7:80;
        server 172.17.0.9:80;
    }
```

```
    server {
        location / {
            proxy_pass http://backend;
        }
    }
}
```

We need to copy the modify nginx.conf file back in to the lb container

```
docker cp nginx.conf lb-jegan:/etc/nginx/nginx.conf
```

We need to restart the lb container to apply config changes

```
docker restart lb-jegan
```

We need to check if the nginx container continues to run after config changes

```
docker ps
```

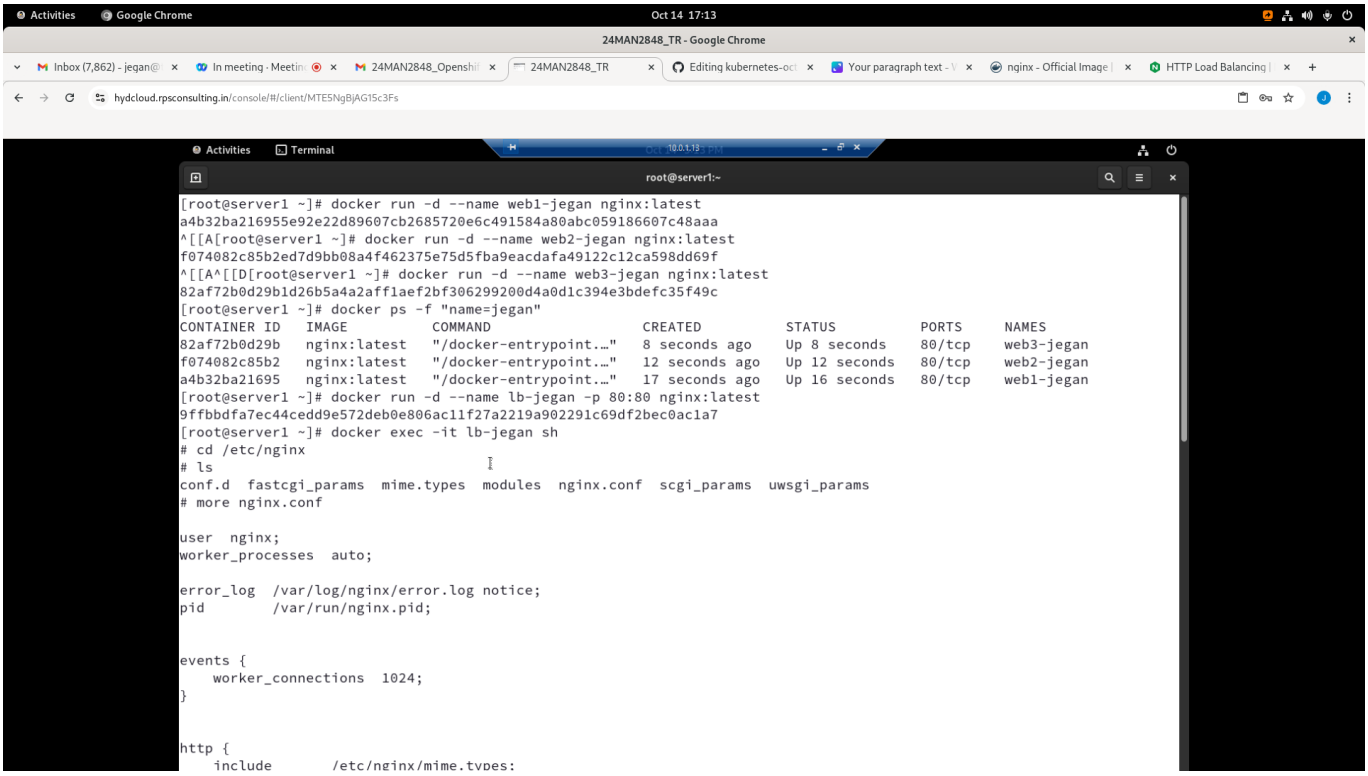
Let's customize the web1-jegan, web2-jegan and web3-jegan container pages

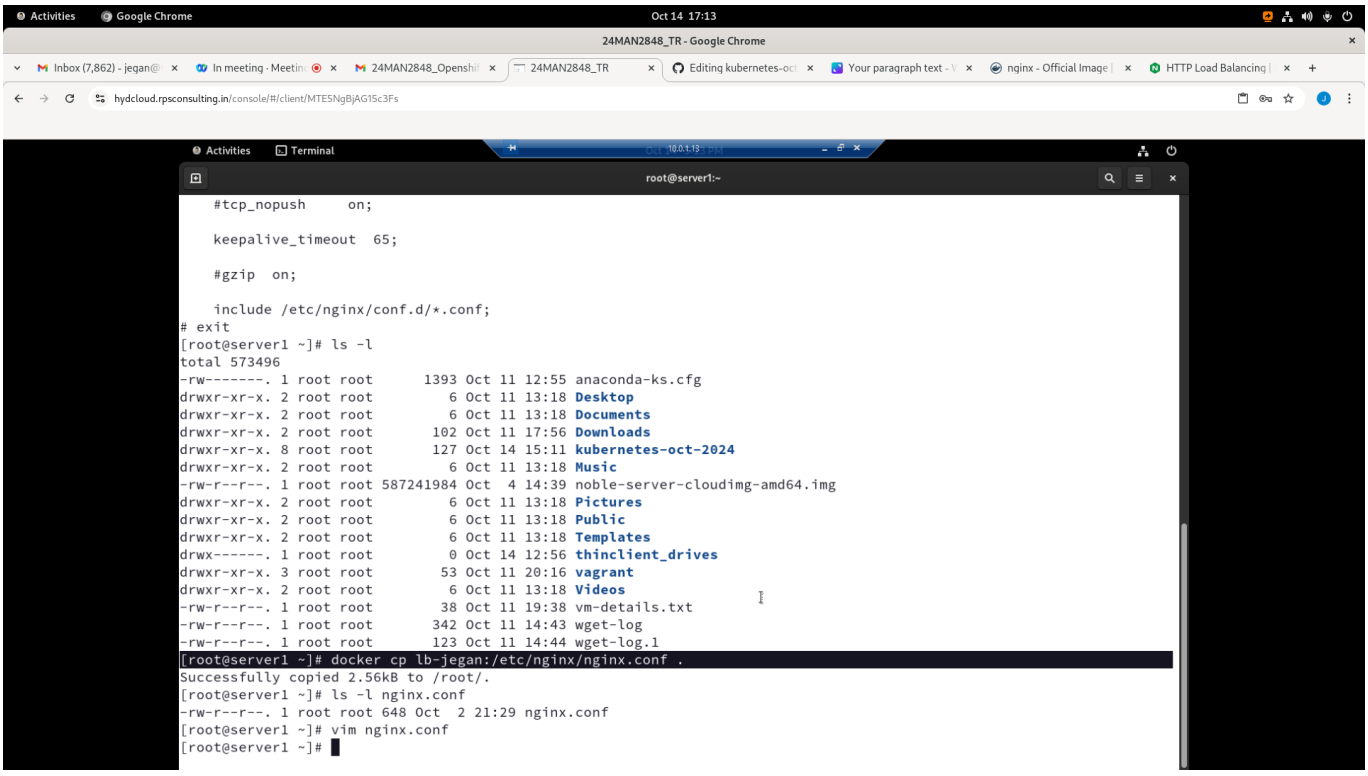
```
echo "Server 1" > index.html
docker cp index.html web1-jegan/usr/share/nginx/html/index.html

echo "Server 2" > index.html
docker cp index.html web2-jegan/usr/share/nginx/html/index.html

echo "Server 3" > index.html
docker cp index.html web3-jegan/usr/share/nginx/html/index.html
```

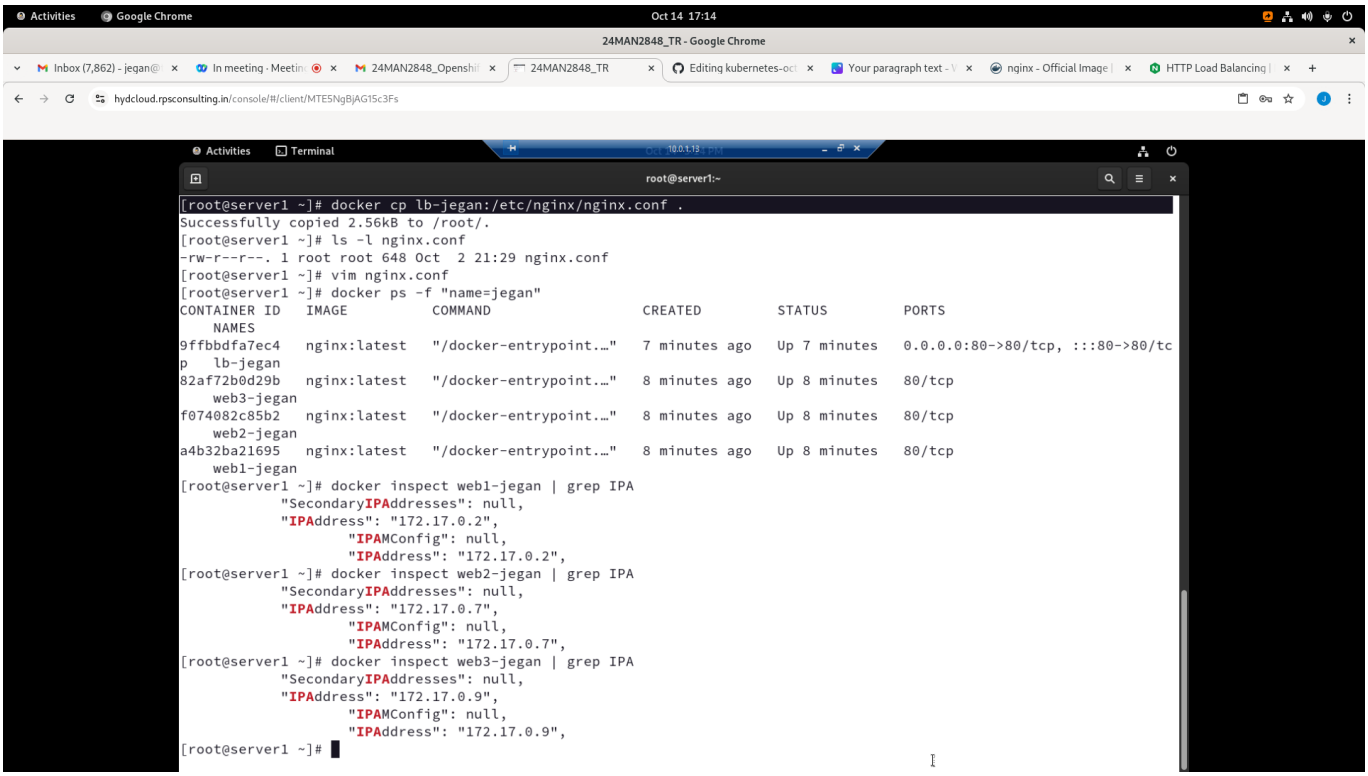
Let's check if load balancer is routing the calls in round-robin fashion from the web browser

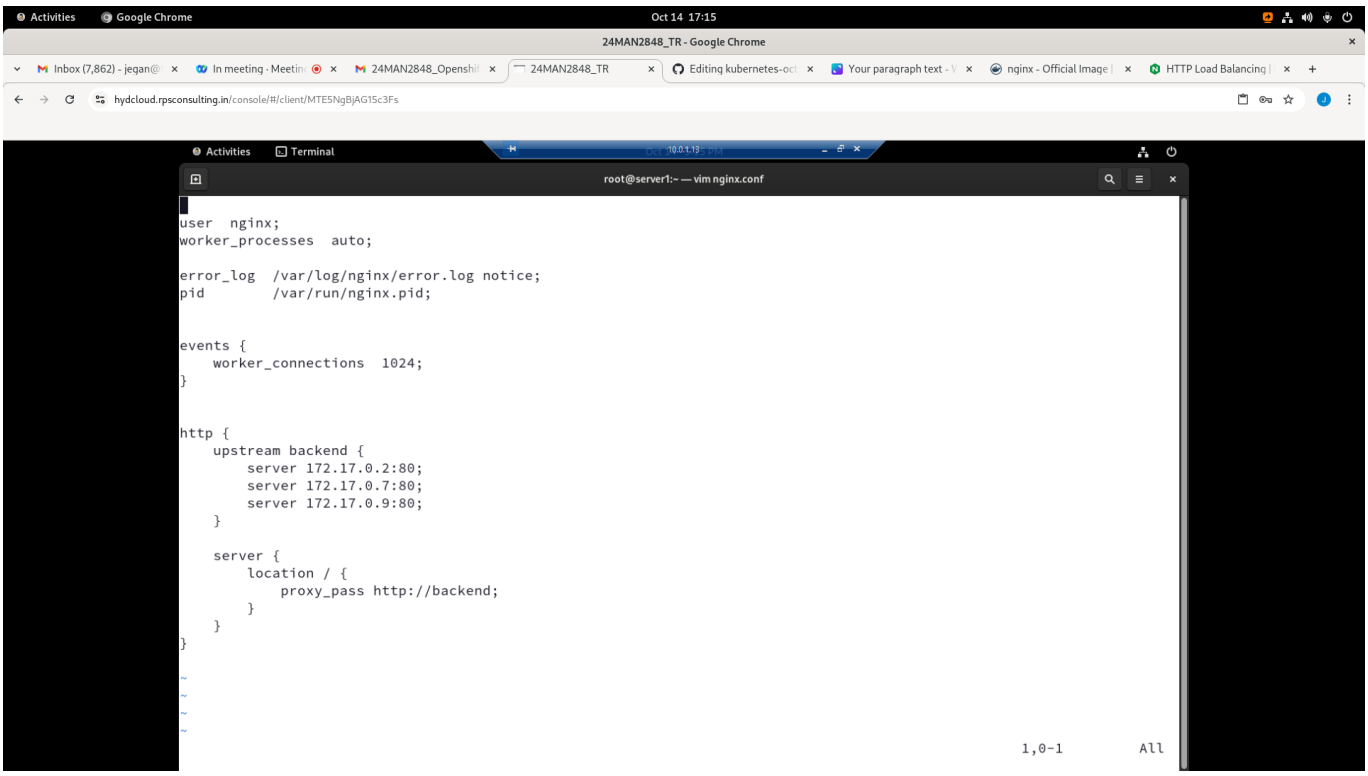
```
http://localhost:80
http://10.0.1.13
```

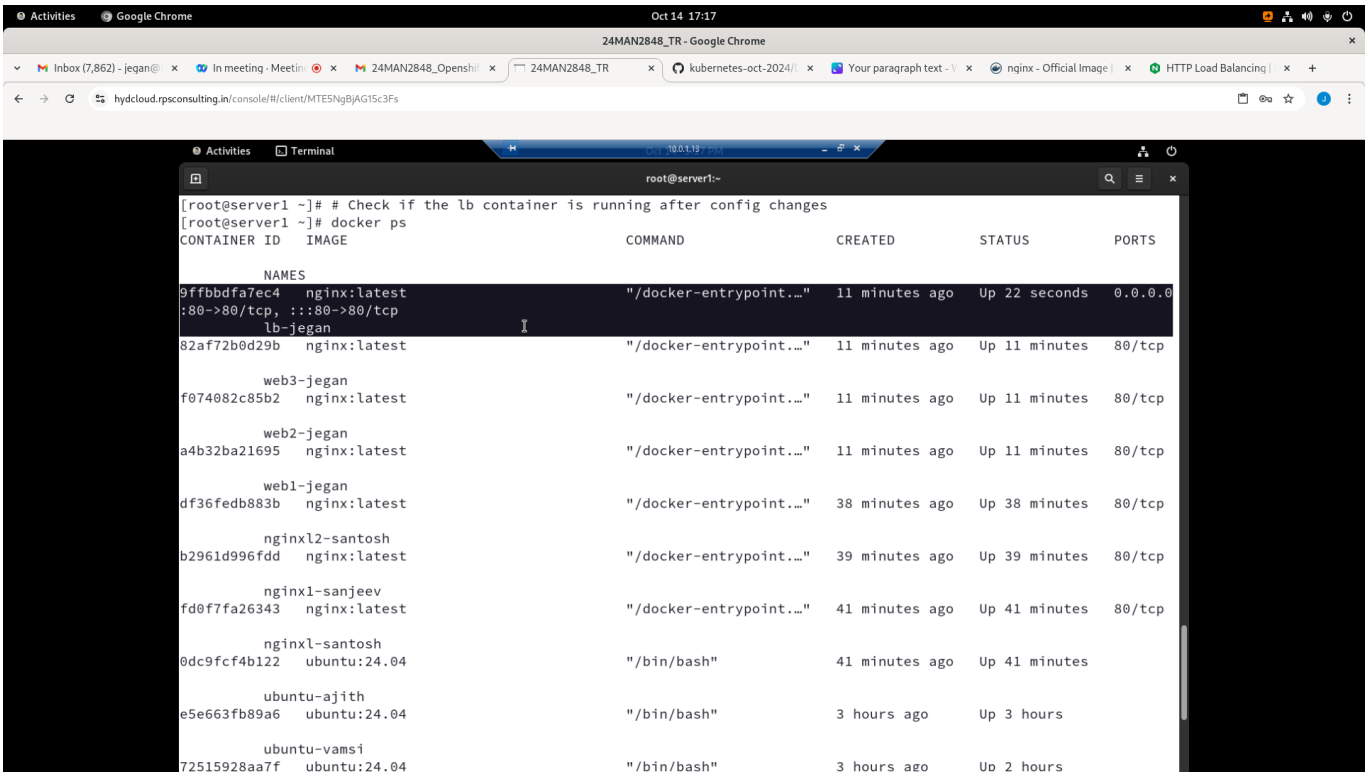
Expected output

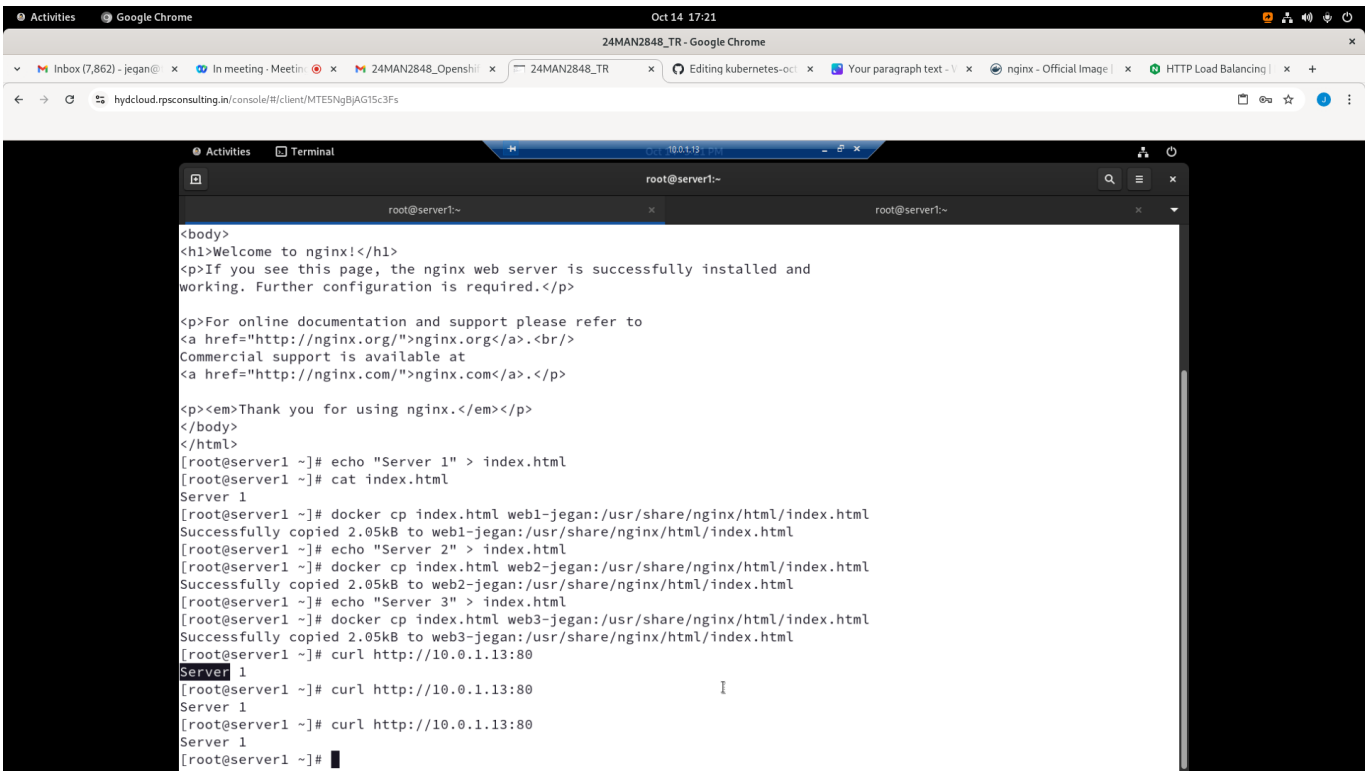





Activities
Google Chrome
Oct 14 17:14
24MAN2848_TR - Google Chrome
hydcloud.rpsconsulting.in/console/#/client/MTE5NgBjAG15c3Fs
Activities
Terminal
root@server1:~
[root@server1 ~]# docker cp lb-jegan:/etc/nginx/nginx.conf .
Successfully copied 2.56kB to /root/.
[root@server1 ~]# ls -l nginx.conf
-rw-r--r--. 1 root root 648 Oct 2 21:29 nginx.conf
[root@server1 ~]# vim nginx.conf
[root@server1 ~]# docker ps -f "name=jegan"
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
9ffbbdfa7ec4 nginx:latest "/docker-entrypoint.…" 7 minutes ago Up 7 minutes 0.0.0.0:80->80/tcp, :::80->80/tc
p lb-jegan
82af72b0d29b nginx:latest "/docker-entrypoint.…" 8 minutes ago Up 8 minutes 80/tcp
web3-jegan
f074082c85b2 nginx:latest "/docker-entrypoint.…" 8 minutes ago Up 8 minutes 80/tcp
web2-jegan
a4b32ba21695 nginx:latest "/docker-entrypoint.…" 8 minutes ago Up 8 minutes 80/tcp
web1-jegan
[root@server1 ~]# docker inspect web1-jegan | grep IPA
"SecondaryIPAddresses": null,
"IPAddress": "172.17.0.2",
"IPAMConfig": null,
"IPAddress": "172.17.0.2",
[root@server1 ~]# docker inspect web2-jegan | grep IPA
"SecondaryIPAddresses": null,
"IPAddress": "172.17.0.7",
"IPAMConfig": null,
"IPAddress": "172.17.0.7",
[root@server1 ~]# docker inspect web3-jegan | grep IPA
"SecondaryIPAddresses": null,
"IPAddress": "172.17.0.9",
"IPAMConfig": null,
"IPAddress": "172.17.0.9",
[root@server1 ~]#


Activities
Google Chrome
Oct 14 17:15
24MAN2848_TR - Google Chrome
hydcloud.rpsconsulting.in/console/#/client/MTE5NgBjAG15c3Fs
Activities
Terminal
root@server1:~ — vim nginx.conf
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log notice;
pid /var/run/nginx.pid;
events {
worker_connections 1024;
}
http {
upstream backend {
server 172.17.0.2:80;
server 172.17.0.7:80;
server 172.17.0.9:80;
}
server {
location / {
proxy_pass http://backend;
}
}
}
1,0-1 All

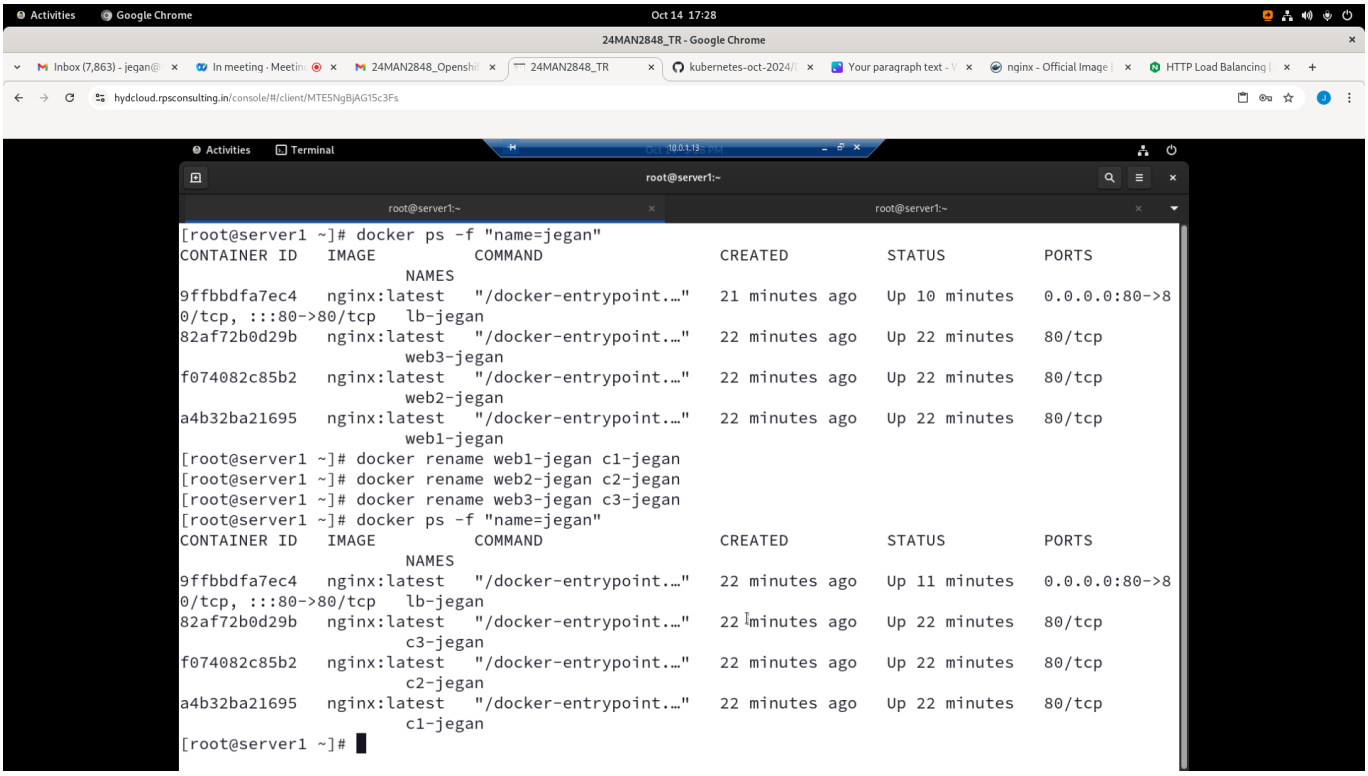




Lab - Renaming containers

```
docker ps -f "name=jegan"
docker rename web1-jegan c1-jegan
docker rename web2-jegan c2-jegan
docker rename web3-jegan c3-jegan
docker ps -f "name=jegan"
```

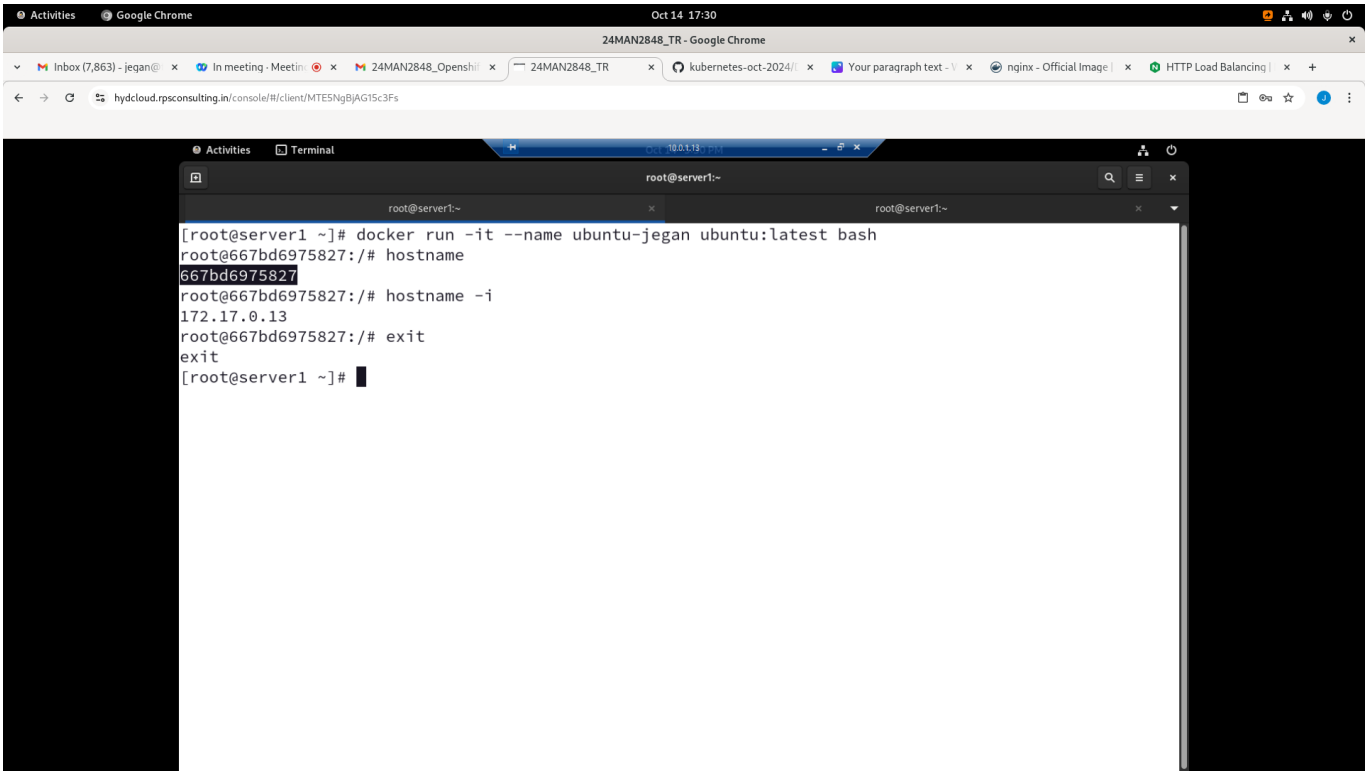
Expected output



Lab - Creating a container in foreground(interactive) mode

```
docker run -it --name ubuntu-jegan ubuntu:24.04 bash
hostname
hostname -i
exit
```

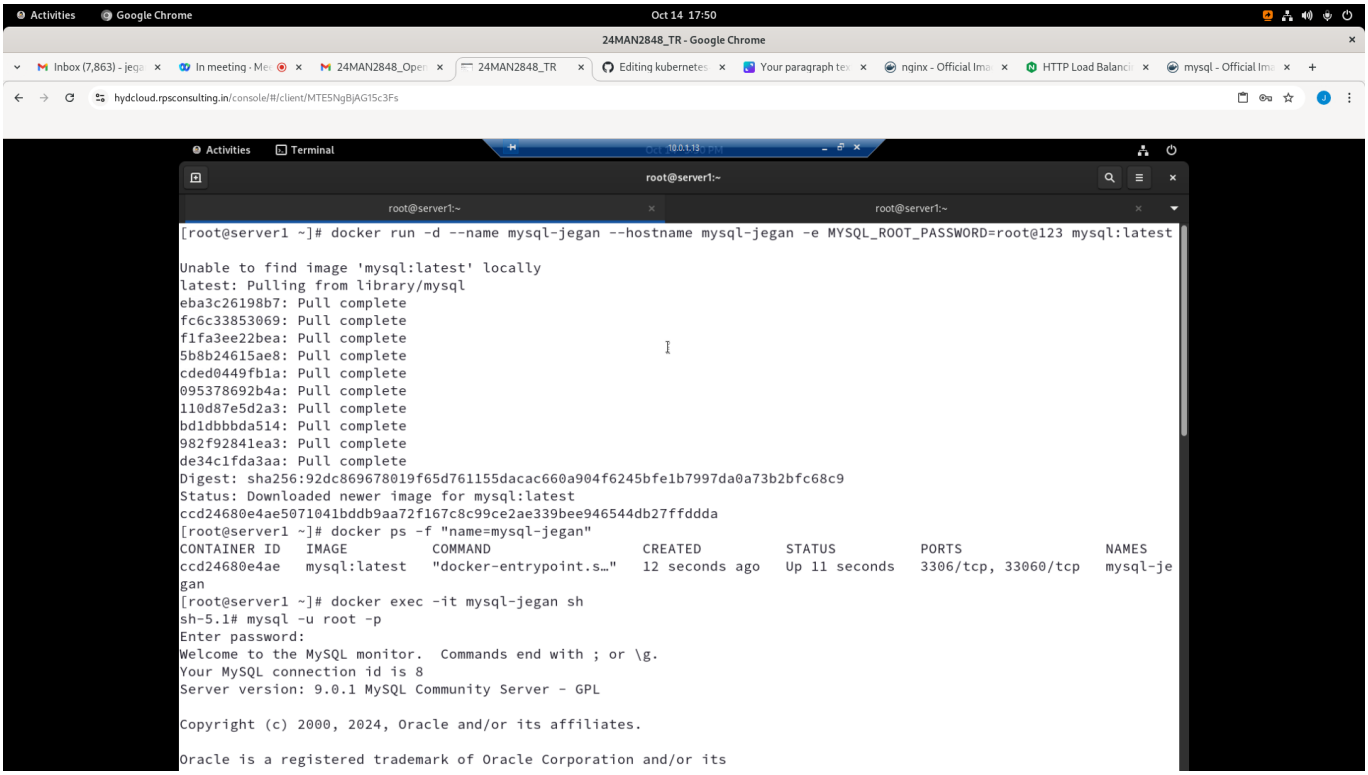
Expected output

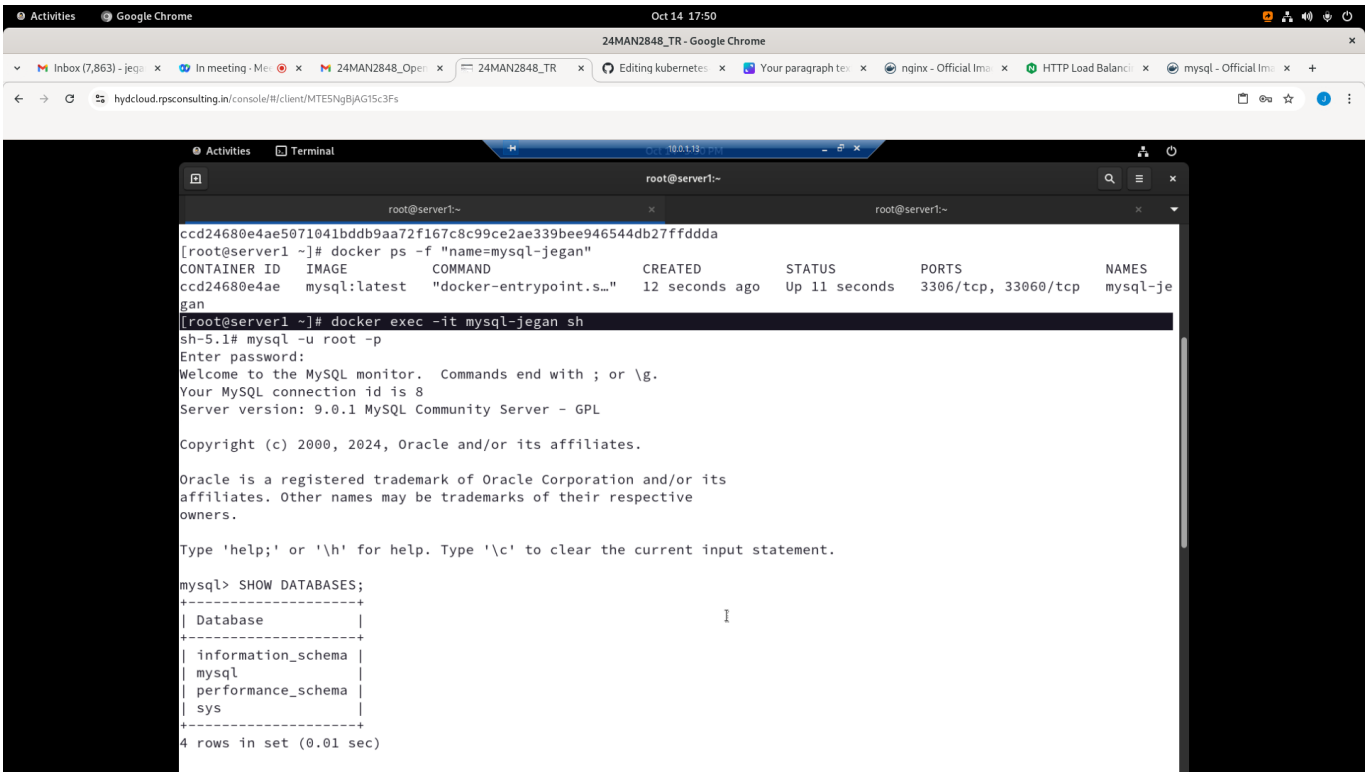


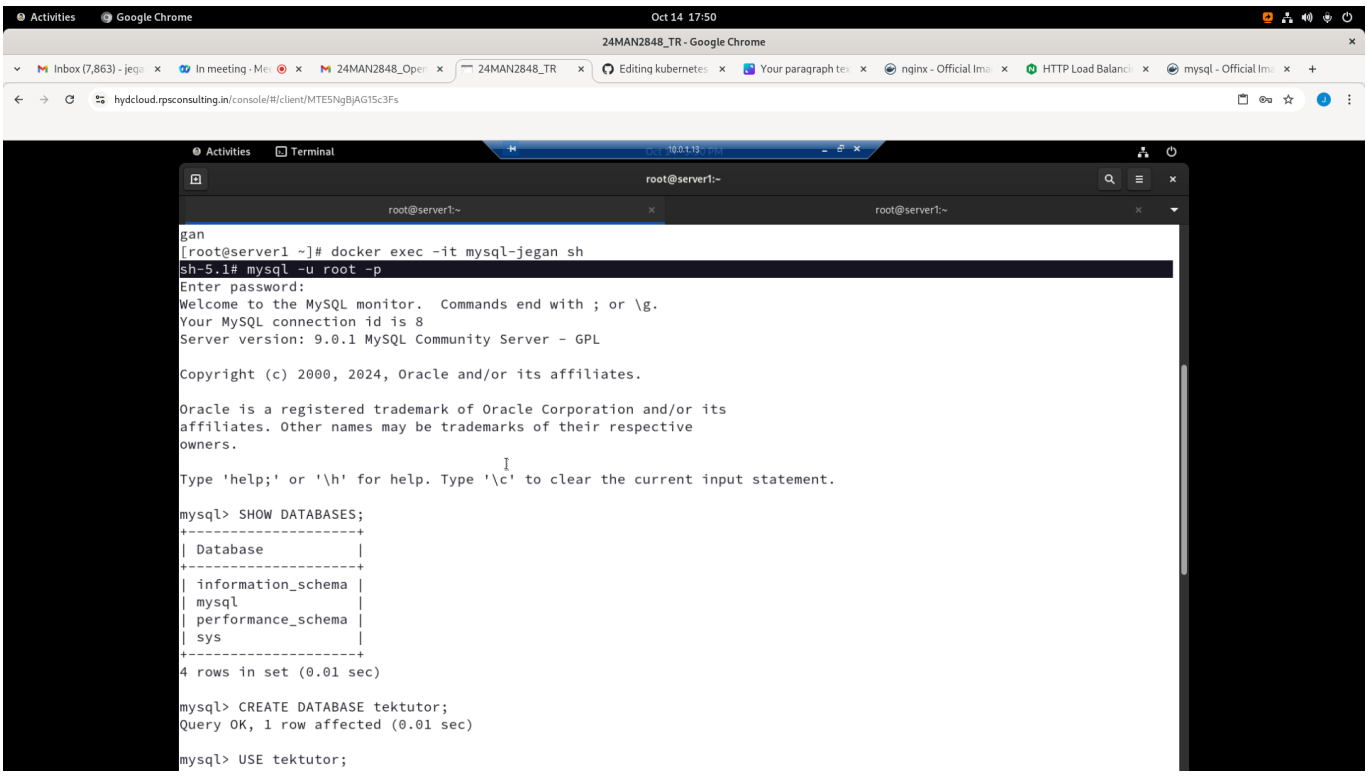
Lab - Creating mysql server container

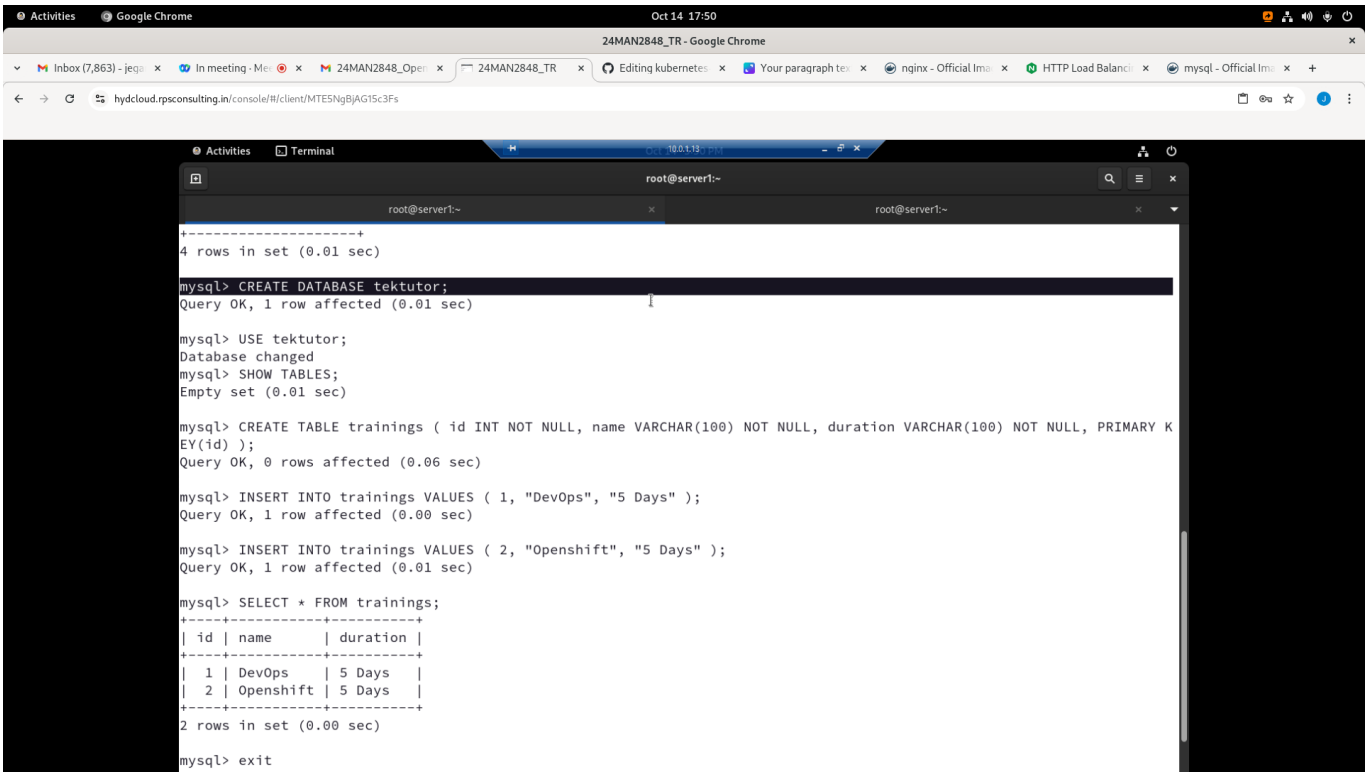
```
docker run -d --name mysql-jegan --hostname mysql-jegan -e
MYSQL_ROOT_PASSWORD=root@123 mysql:latest
docker ps -f "name=mysql-jegan"
docker exec -it mysql-jegan sh
mysql -u root -p
SHOW DATABASES;
CREATE DATABASE tektutor;
USE tektutor;
CREATE TABLE trainings ( id INT NOT NULL, name VARCHAR(100) NOT NULL,
duration VARCHAR(50) NOT NULL, PRIMARY KEY(id) );
INSERT INTO trainings VALUES ( 1, "DevOps", "5 Days" );
INSERT INTO trainings VALUES ( 2, "Openshift", "5 Days" );
SELECT * FROM trainings;
exit
exit
```

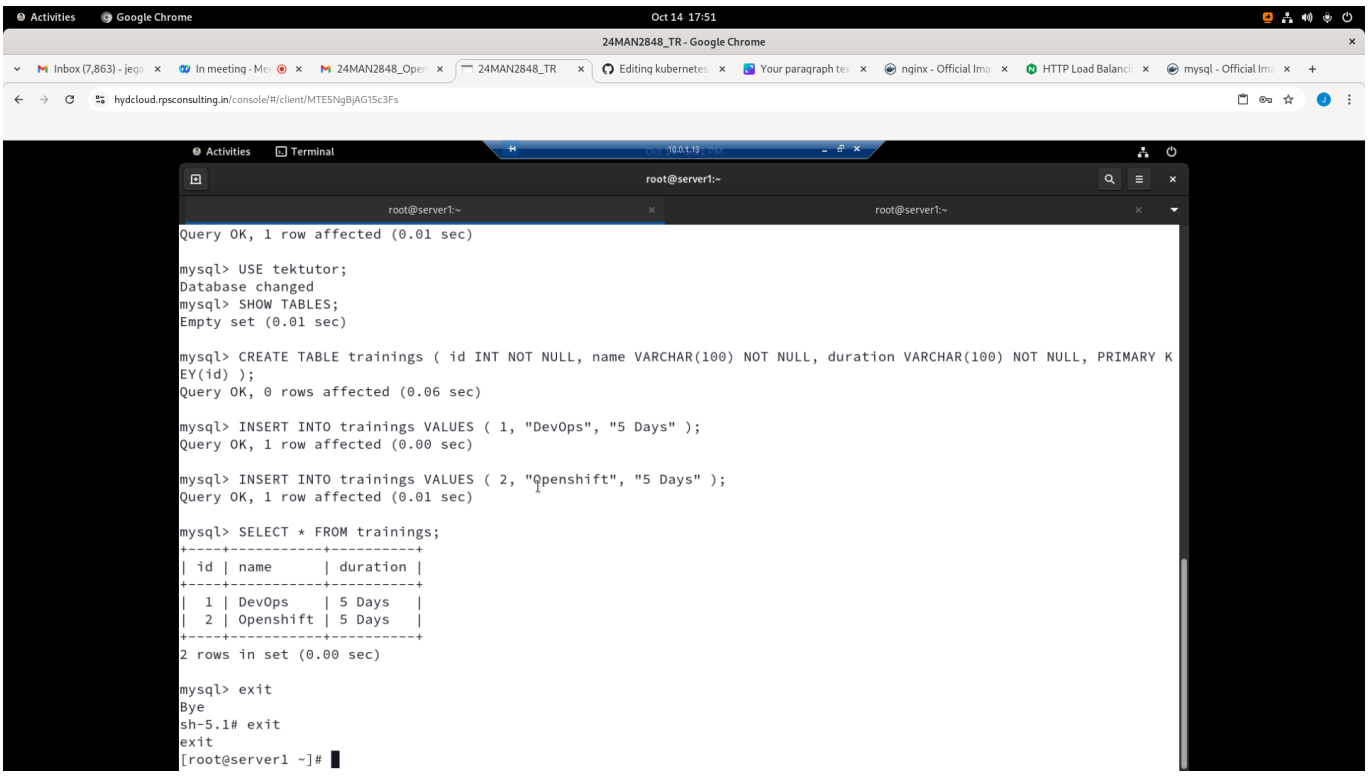
Expected output




Activities
Google Chrome
Oct 14 17:50
24MAN2848_TR - Google Chrome
hydcloud.rpsconsulting.in/console/#/client/MTE5NgBjAG15c3Fs
Activities
Terminal
root@server1:~
root@server1:~
root@server1:~
ccd24680e4ae5071041bddb9aa72f167c8c99ce2ae339bee946544db27ffddda
[root@server1 ~]# docker ps -f "name=mysql-jegan"
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
ccd24680e4ae mysql:latest "docker-entrypoint.s…" 12 seconds ago Up 11 seconds 3306/tcp, 33060/tcp mysql-je
gan
[root@server1 ~]# docker exec -it mysql-jegan sh
sh-5.1# mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 9.0.1 MySQL Community Server - GPL
Copyright (c) 2000, 2024, Oracle and/or its affiliates.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> SHOW DATABASES;
+--------------------+
| Database |
+--------------------+
| information_schema |
| mysql |
| performance_schema |
| sys |
+--------------------+
4 rows in set (0.01 sec)


Activities
Google Chrome
Oct 14 17:50
24MAN2848_TR - Google Chrome
hydcloud.rpsconsulting.in/console/#/client/MTE5NgBjAG15c3Fs
Activities
Terminal
root@server1:~
root@server1:~
root@server1:~
gan
[root@server1 ~]# docker exec -it mysql-jegan sh
sh-5.1# mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 9.0.1 MySQL Community Server - GPL
Copyright (c) 2000, 2024, Oracle and/or its affiliates.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> SHOW DATABASES;
+--------------------+
| Database |
+--------------------+
| information_schema |
| mysql |
| performance_schema |
| sys |
+--------------------+
4 rows in set (0.01 sec)
mysql> CREATE DATABASE tektutor;
Query OK, 1 row affected (0.01 sec)
mysql> USE tektutor;





Let's delete the mysql container

```
docker rm -f mysql-jegan
docker ps -a
```

Let's create a folder in our local linux server

```
mkdir -p /tmp/mysql-jegan
```

Let's mount the above path inside the container using volume mounting to force the container store the data in an external storage

```
docker run -d --name mysql-jegan --hostname mysql-jegan -e
MYSQL_ROOT_PASSWORD=root@123 -v /tmp/mysql-jegan:/var/lib/mysql
mysql:latest
docker ps
docker exec -it mysql-jegan sh
mysql -u root -p
SHOW DATABASES;
CREATE DATABASE tektutor;
USE tektutor;
CREATE TABLE trainings ( id INT NOT NULL, name VARCHAR(100) NOT NULL,
duration VARCHAR(50) NOT NULL, PRIMARY KEY(id) );
INSERT INTO trainings VALUES ( 1, "DevOps", "5 Days" );
INSERT INTO trainings VALUES ( 2, "Openshift", "5 Days" );
SELECT * FROM trainings;
exit
exit
```

Let's delete the mysql-jegan container

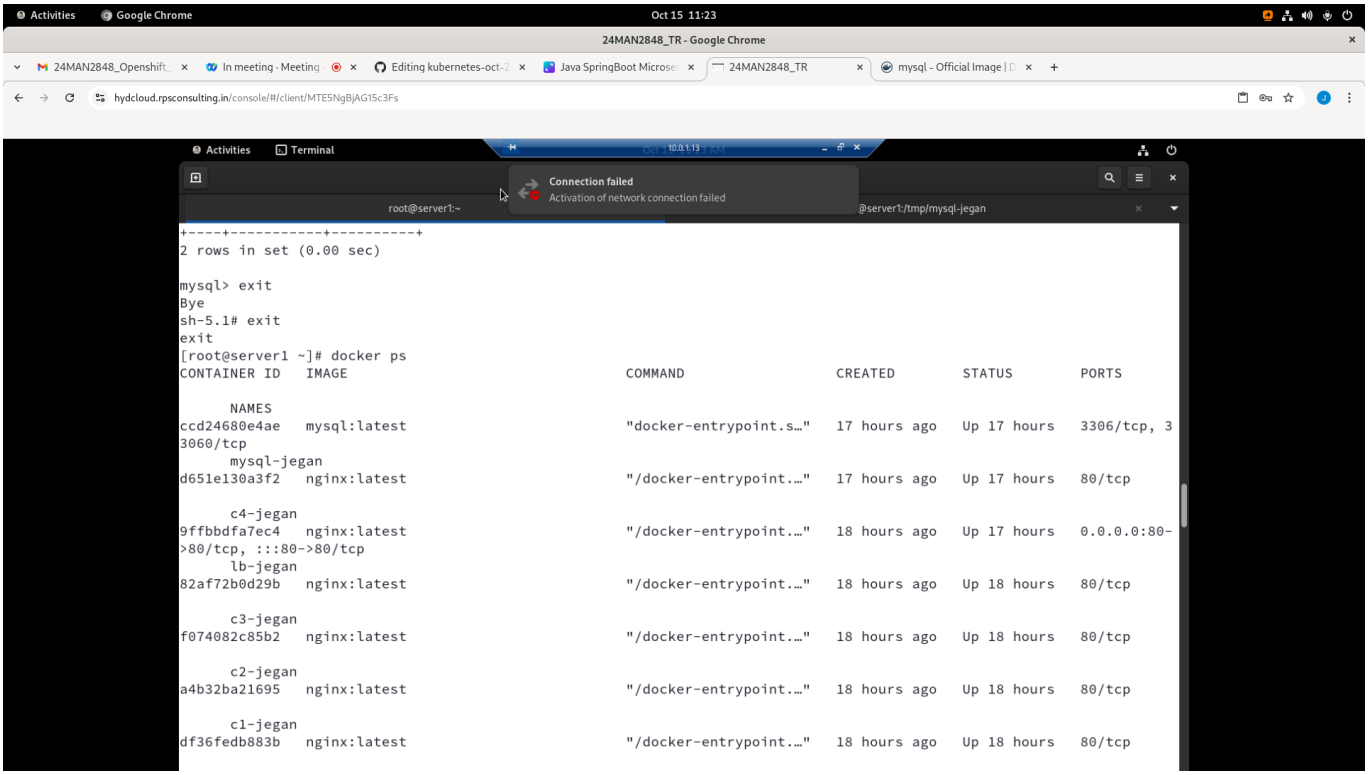
```
docker rm -f mysql-jegan
```

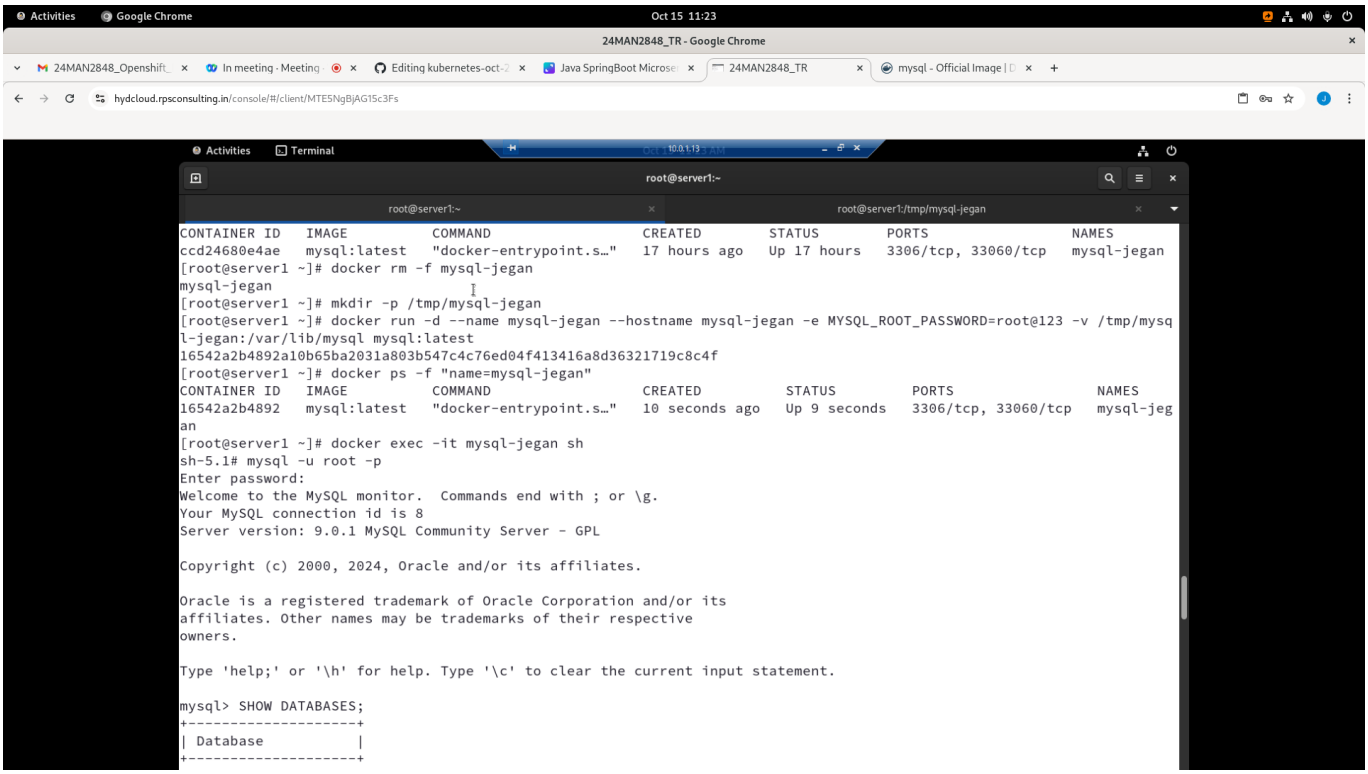
Let's recreate a new container using the same local disk path

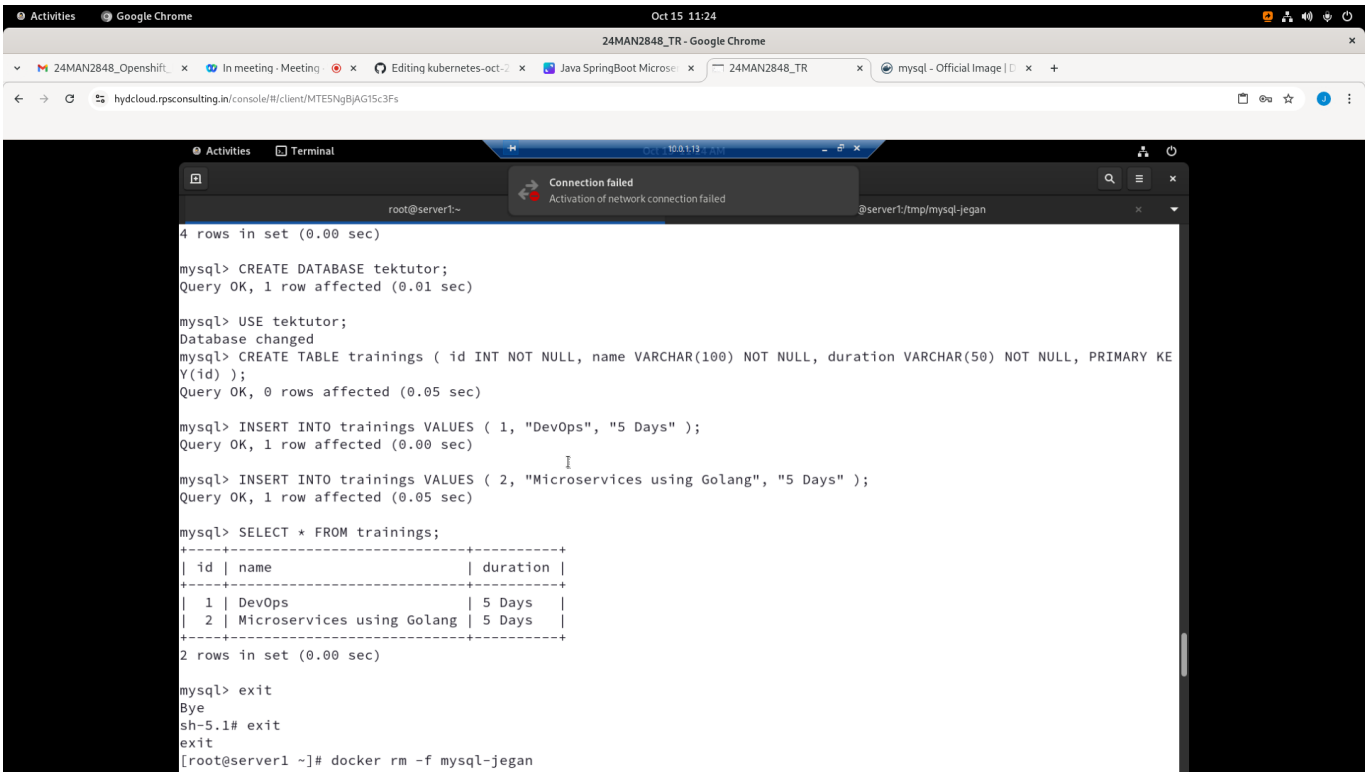
```
docker run -d --name mysql-jegan --hostname mysql-jegan -e
MYSQL_ROOT_PASSWORD=root@123 -v /tmp/mysql-jegan:/var/lib/mysql
mysql:latest
docker ps
docker exec -it mysql-jegan sh
mysql -u root -p
SHOW DATABASES;
USE tektutor;
SHOW TABLES
SELECT * FROM trainings;
exit
exit
```

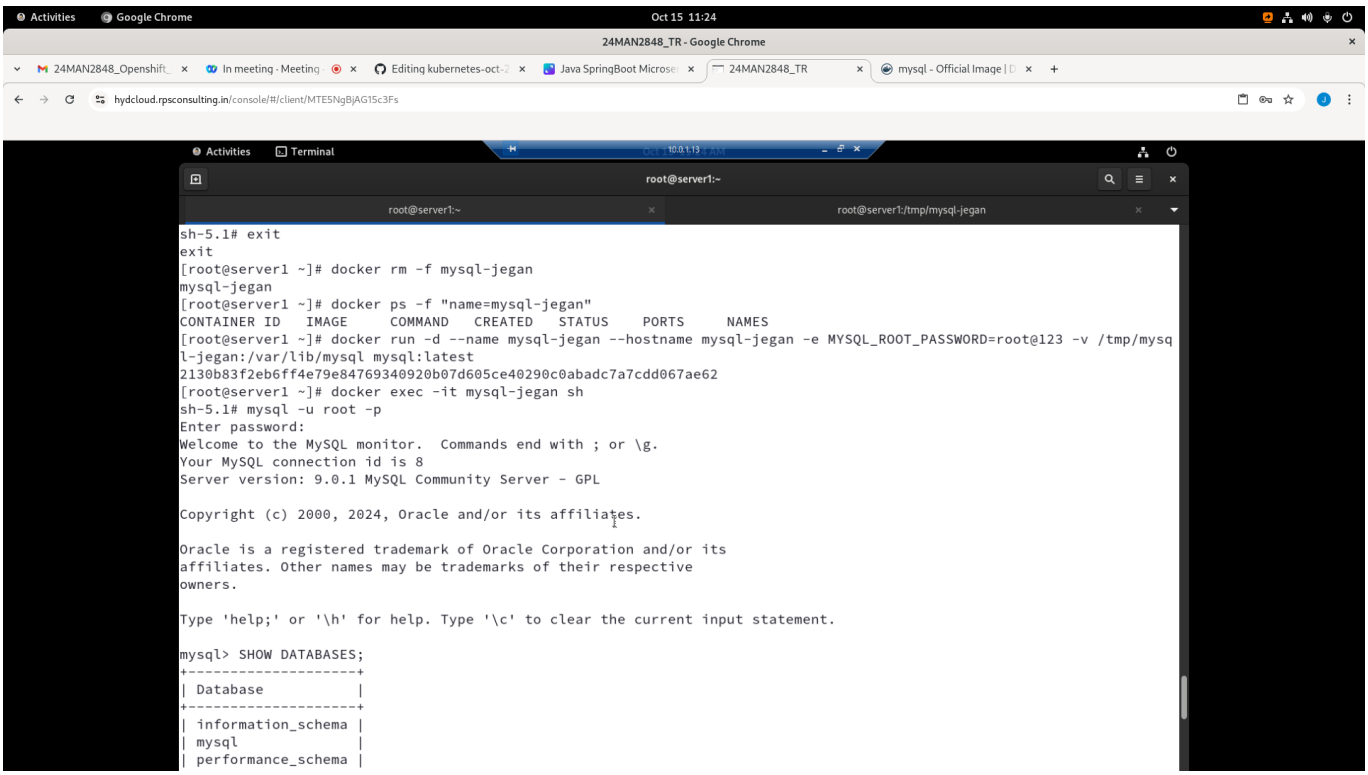
As you noticed, the data is intact even though we delete the original container that created those records.

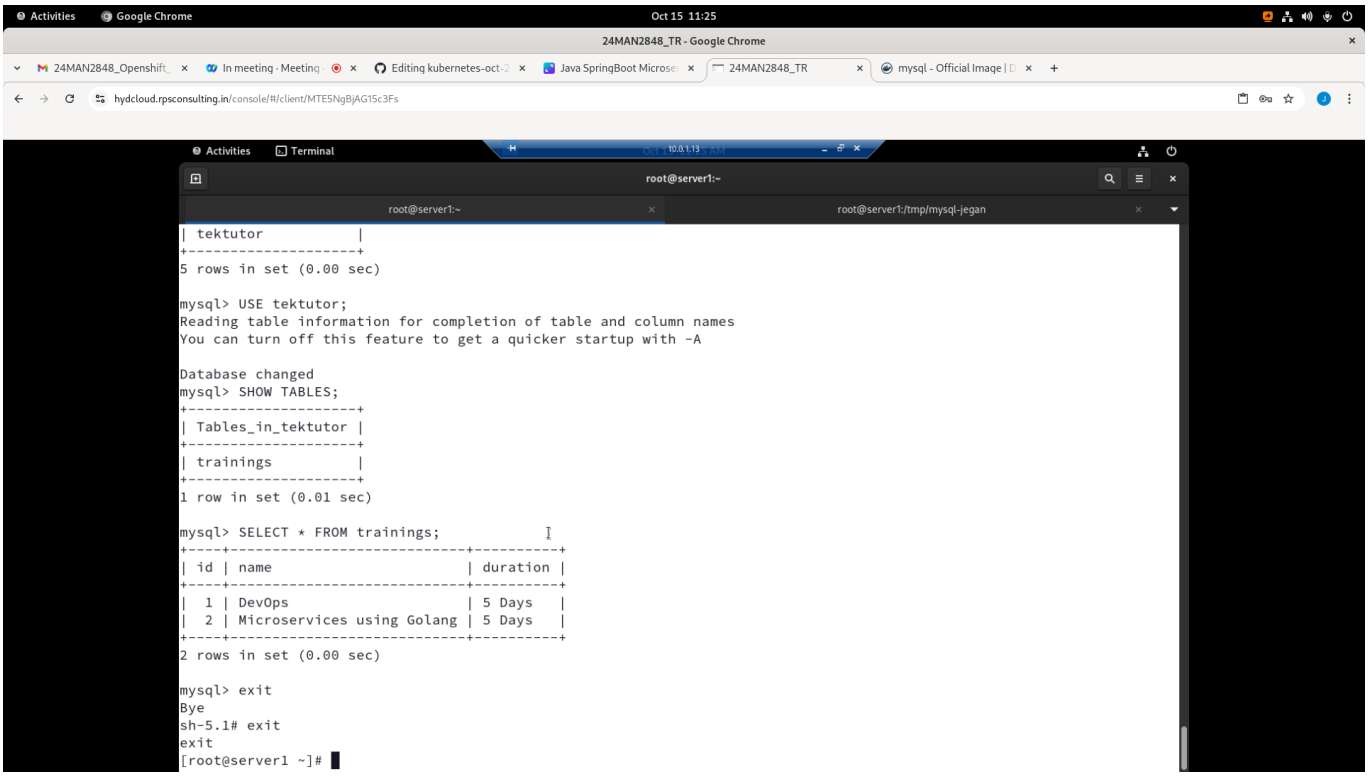
Expected output






Activities
Google Chrome
Oct 15 11:24
24MAN2848_TR - Google Chrome
hydcloud.rpsconsulting.in/console/#/client/MTE5NgBjAG15c3Fs
Activities
Terminal
Connection failed
Activation of network connection failed
root@server1:~
root@server1:/tmp/mysql-jegan
4 rows in set (0.00 sec)
mysql> CREATE DATABASE tektutor;
Query OK, 1 row affected (0.01 sec)
mysql> USE tektutor;
Database changed
mysql> CREATE TABLE trainings (id INT NOT NULL, name VARCHAR(100) NOT NULL, duration VARCHAR(50) NOT NULL, PRIMARY KE
Y(id));
Query OK, 0 rows affected (0.05 sec)
mysql> INSERT INTO trainings VALUES (1, "DevOps", "5 Days");
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO trainings VALUES (2, "Microservices using Golang", "5 Days");
Query OK, 1 row affected (0.05 sec)
mysql> SELECT * FROM trainings;
+----+-----------------------------+----------+
| id | name | duration |
+----+-----------------------------+----------+
| 1 | DevOps | 5 Days |
| 2 | Microservices using Golang | 5 Days |
+----+-----------------------------+----------+
2 rows in set (0.00 sec)
mysql> exit
Bye
sh-5.1# exit
exit
[root@server1 ~]# docker rm -f mysql-jegan


Activities
Google Chrome
Oct 15 11:24
24MAN2848_TR - Google Chrome
hydcloud.rpsconsulting.in/console/#/client/MTE5NgBjAG15c3Fs
Activities
Terminal
root@server1:~
root@server1:~
root@server1:/tmp/mysql-jegan
sh-5.1# exit
exit
[root@server1 ~]# docker rm -f mysql-jegan
mysql-jegan
[root@server1 ~]# docker ps -f "name=mysql-jegan"
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[root@server1 ~]# docker run -d --name mysql-jegan --hostname mysql-jegan -e MYSQL_ROOT_PASSWORD=root@123 -v /tmp/mysq
l-jegan:/var/lib/mysql mysql:latest
2130b83f2eb6ff4e79e84769340920b07d605ce40290c0abadc7a7cdd067ae62
[root@server1 ~]# docker exec -it mysql-jegan sh
sh-5.1# mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 9.0.1 MySQL Community Server - GPL
Copyright (c) 2000, 2024, Oracle and/or its affiliates.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> SHOW DATABASES;
+--------------------+
| Database |
+--------------------+
| information_schema |
| mysql |
| performance_schema |


Activities
Google Chrome
Oct 15 11:25
24MAN2848_TR - Google Chrome
hydcloud.rpsconsulting.in/console/#/client/MTE5NgBjAG1Sc3Fs
Activities
Terminal
root@server1:~
root@server1:~
root@server1:/tmp/mysql-jegan
| tektutor |
+--------------------+
5 rows in set (0.00 sec)
mysql> USE tektutor;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> SHOW TABLES;
+--------------------+
| Tables_in_tektutor |
+--------------------+
| trainings |
+--------------------+
1 row in set (0.01 sec)
mysql> SELECT * FROM trainings;
+----+-----------------------------+----------+
| id | name | duration |
+----+-----------------------------+----------+
| 1 | DevOps | 5 Days |
| 2 | Microservices using Golang | 5 Days |
+----+-----------------------------+----------+
2 rows in set (0.00 sec)
mysql> exit
Bye
sh-5.1# exit
exit
[root@server1 ~]#

Day 2

You might be interested in these blogs

```
https://medium.com/@jegan_50867/kubernetes-lightweight-developer-setup-
using-rancher-k3d-a3a94e9b5eb4

https://medium.com/@jegan_50867/kubernetes-3-node-cluster-using-k3s-
d28b2c09e2f7

https://medium.com/@jegan_50867/kubernetes-3-node-cluster-using-k3s-with-
docker-e325cc82fd50

https://kubernetes.io/docs/concepts/storage/persistent-volumes/

https://www.tecmint.com/install-nfs-server-on-ubuntu/
```

Recommnended Projects to learn Kubernetes internals by doing the setup by hand

```
https://github.com/kelseyhightower/kubernetes-the-hard-way
```

What are the options to install Kubernetes?

1. Minikube - Good for learning purpose but not production grade
2. MicroK8s - Lightweight setup, good for learning and production
3. K3S - Lightweight setup, good for learning and production grade
4. Kubernetes setup using kubeadm - https://medium.com/tektutor/kubernetes-3-node-cluster-setup-50943378be41
5. Kubespray - Ansible playbook - Production Grade

Info - Container Orchestration Platform Overview

```
- Container Orchestration Platforms supports the below features
  - deploying containerized application workloads
  - scale up/down based on user-traffic to our applications
  - provides an environment to make our applications highly available(HA)
  - load-balancing
  - monitoring tools
  - rolling update
    - upgrading your live application from one version to the other without
any downtime
  - rollback
    - rolling back to previous to older versions when the new version is
```

```
found to be unstable
  - exposing application for internal or external use via services
  - supports service discovery - i.e services can be accessed using their
name
- examples
  - Docker SWARM
  - Google Kubernetes
  - Opensource Openshift (OKD)
  - Red Hat Openshift
  - EKS - AWS Managed Kubernetes Service
  - AKS - Azure Kubernetes Service
  - ROSA - AWS Managed Openshift Service
  - ARO - Azure Managed Openshift Service
```

Info - Docker SWARM

```
- Docker SWARM is a Container Orchestration Platform developed by Docker
Inc as a open source project
- it is Docker's native Container Orchestration Platform
- it only support Docker containerized application workloads
- it is very light-weight, easy to setup, easy to learn and easy to
maintain
- it is not production grade
- it is very good for learning or dev/qa environment or for R&D purpose
```

Info - Kubernetes Overview

```
- Container Orchestration Platform developed by Google
- Google used this product internally for several years before they made it
opensource
- it is time-tested, robus Container Orchestration Platform thats works for
simple or very complex applications
- ideal for dev/qa/prod
- supports many different container runtimes and engines
- it is an opensource project that can be used free of cost for personal
and commercial use
- supports only command-line, setup can only be done in Linux
- we don't get support from Google as it is opensource project
- however, we can get support from Google if we use GKE from Google cloud
- however, we can get support from Amazon if we use EKS from AWS cloud
- however, we can get support from Microsoft if we use AKS from Azure cloud
```

Info - OKD

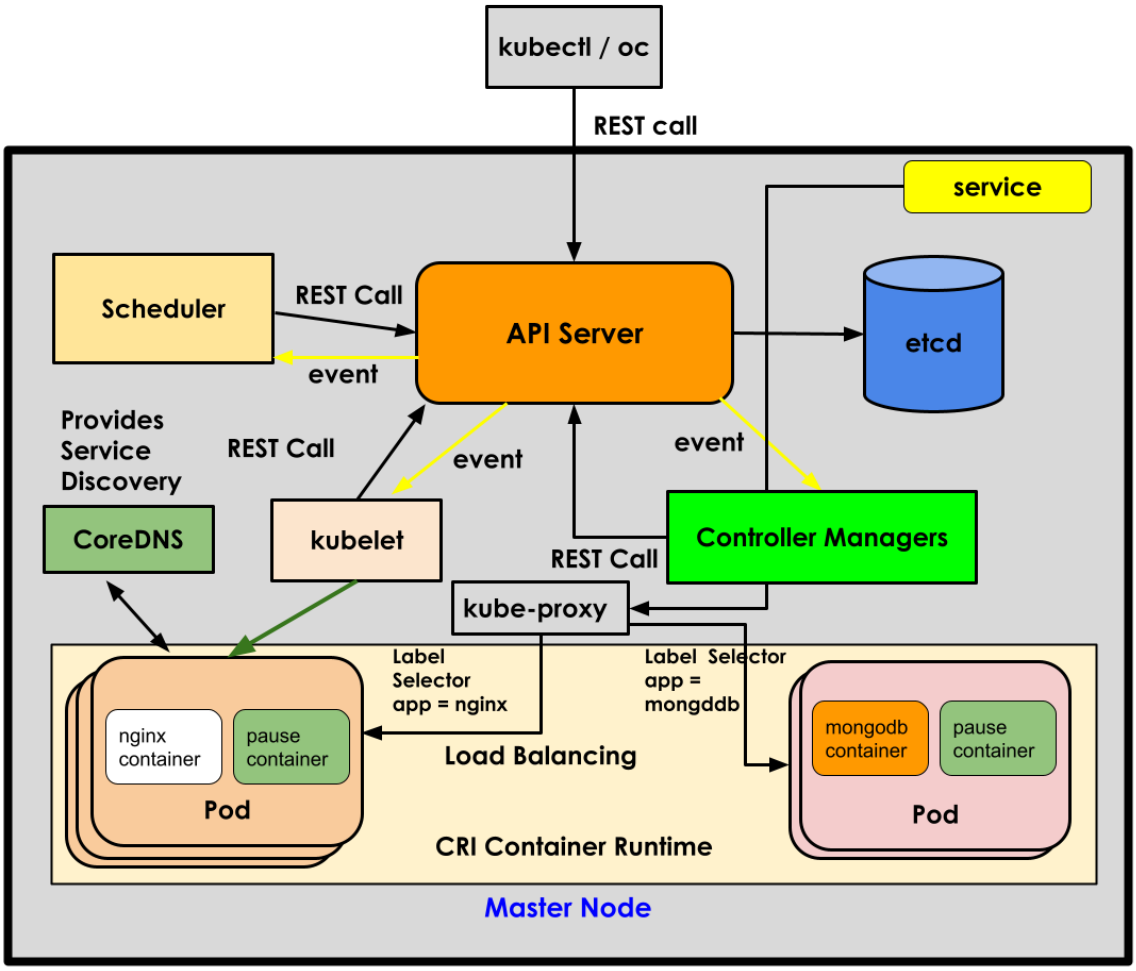
```
- opensource OpenShift Container Orchestration Platform
- supports CRI-O container runtime and Podman Container Engine only
```

```
- supports CLI and webconsole
- supports user management
- developed on top of Kubernetes with many addition features
- it is a superset of Google Kubernetes
- you won't get support from any specific company as it is open source, you
are on your own, you only get community support
```

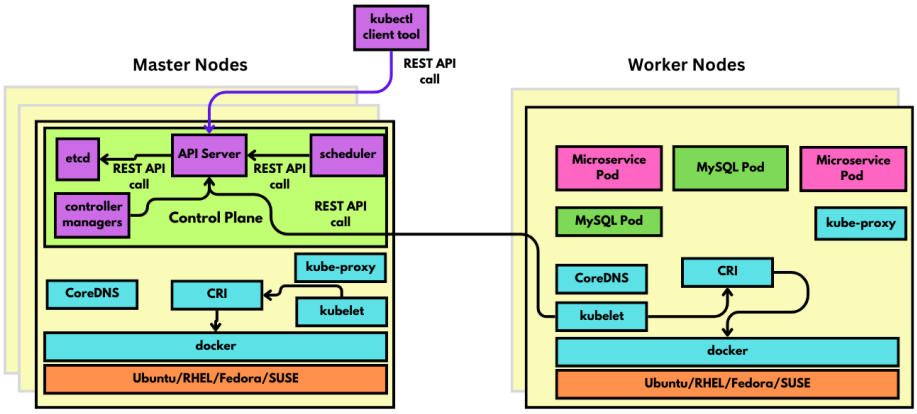
Info - Red Hat Openshift

```
- it is commercial Container Orchestration Platform developed by Red Hat
- developed on top of opensource OKD
- supports CLI and Webconsole
- we will get support from Red Hat ( an IBM company )
```

Info - Kubernetes High Level Architecture



Kubernetes High Level Architecture



Kubernetes Cluster

```
- a group of Linux servers that works together as a Kubernetes Cluster
- the Linux servers can be an onpremise server, virtual machine that runs
```

```
locally or on a public cloud
- these Linux servers are called Kubernetes Nodes
- Kubernetes Nodes are of two types
  - Master Node and
  - Worker Node
- Master Node
  - control plane Pods runs only in master nodes
  - a Master node can also be configured to run user application pods just
like a worker node
  - master can behave as Master and Worker Node if they are configured
- Worker Node
  - this is where typically user application Pods are deployed
```

Info - Kubernetes Control Plane

```
- Control Plane Components only run in master nodes
- below are the Control Plane Components
  1. API Server
  2. etcd database
  3. scheduler
  4. Controller Managers
```

Info - API Server

```
- is the heart of Kubernetes - Container Orchestration Platform
- it supports REST API for all Kubernetes features
- API Server is the only components that has write access to etcd database
- API Server saves and manages the cluster and application status in the
etcd database
- every API Server has its own dedicated etcd database
- Whenever new records are added, edited, deleted in the etcd database, API
Server will send broadcasting events about the change
```

Info - etcd

```
- it is key/value database
- it is an independent opensource database that is developed and maintained
independent of Kubernetes/Openshift
- distributed database that can be used even outside the scope of
Kubernetes/Openshift
```

Info - scheduler

```
- this component is responsible to schedule containerized application loads
onto healthy nodes within the Kubernetes cluster
- when we deploy our application into Kubernetes(k8s), it creates Pods,
within Pods containers are created, our application runs inside one of
those containers
- the scheduler decides on which Pod goes into which node ( master-1,
master-2, master-3, worker-1, worker-2, worker-3 )
- however, it won't schedule the Pods on its own, it is send the scheduling
recommendations to API Server via REST calls
```

Info - Controller Managers

```
- a group of controllers
- is a collection of many
- is a Pod that runs in every master node
- controller is an application that runs in a never ending while loop
- each controller manages one type of Openshift resource
- For example
  - Deployment Controller manages Deployment
  - ReplicaSet Controller manages ReplicaSet
  - Job Controller manages Jobs
  - CronJob Controller manages CronJobs
  - StatefulSet Controller manages StatefulSet
  - DaemonSet Controller manager DaemonSet
  - Endpoint Controller manages Pod endpoints used in services
```

Info - Deployment Controller

```
- When pods are requested by the user, it will try to spread the pods on
multiples nodes but there is no assurance
- it is possible all the pods from a specific deployment may be scheduled
to the same nodes as well
- Deployment controller doesn't put any constraint on scheduling, hence it
is upto the scheduler to decide which pod goes to which node
- what is the guarantee offered
  - at point of time, the desired number of pods will be always running
  - but they can run on any nodes
```

Info - DaemonSet Controller

```
- unlike the Deployment Controller, daemonset controller ensures one Pod
per node are deployed
- the number of Pods deployed will be equivalent to the number of nodes in
your openshift cluster
- hence, the pods created as part of daemonset are distributed always one
```

```
Pod per node
- What is the practical usecase for this?
  - prometheus pod to collect performance metrics need to run in every node
  - kube-proxy pod in Kubernetes/Openshift runs in every node
- what is the guarantee offerred
  - if 5 nodes are there in cluster, 5 pods would be created
  - each Pod would be scheduled to different nodes in the cluster
  - the total number of pods and nodes would match
```

Info - StatefulSet Controller

```
- used to deploy stateful applications
- they tend to use external storage in general
- generally when we have deploy database applications as a cluster that
synchronizes data
- creating a cluster of database varies for every database, hence cluster
creation is our responsibility
- statefulset provides the required sections/provisions to create a
cluster, but it won't create a cluster of databases out of the box
- scaling up/down the number of db pods and make them work as cluster is
very complex, hence we also need to do some configurations to ensure they
are running as a cluster
```

Info - Job Controller

```
- is used to do any one time activity
- the one time activity will run in a container which of part of a Pod
- example
  - taking one time backup of etcd database
```

Info - CronJob Controller

```
- is used to do any repeating tasks
- it can be scheduled to run on a particular day/week/month and particular
time
- example
  - taking backup, every friday midnight
```

Pod Overview

```
- literal english - a group of whales is called a Pod
- Logical grouping of containers
- containers is where our application will be running
- it is a JSON file that is stored in etcd database
```

```
- docker logo is whale, that's how the Pod terminology was coined
- is a group of related containers
- a Kubernetes resource which is defined as JSON document
- Pod definition is stored in etcd and maintained by API Server
```

Info - Labels

```
- key/values pairs
- labels are used to map child objects
- every components has one to many labels
- the deployment will identify its respective replicaset using labels as a
selector
- the replicaset will identify its respective pods using labels as a
selector
- For example
  - Each Deployment has one or more ReplicaSets
  - Each ReplicaSet has one or more Pod
  - Each Pod has one or more Containers
```

ReplicaSet Overview

```
- is a Kubernetes resource, that resides in etcd database
- it is a JSON file that is stored in etcd database
- it captures the below details
  - container image that must be used while deploying the application
containers
  - desired number of Pod instances that must be running
  - actual number of Pod instances that are running
- ReplicaSet controller takes the ReplicaSet definition as an input and it
ensures the desired and actual Pod counts are matching always
```

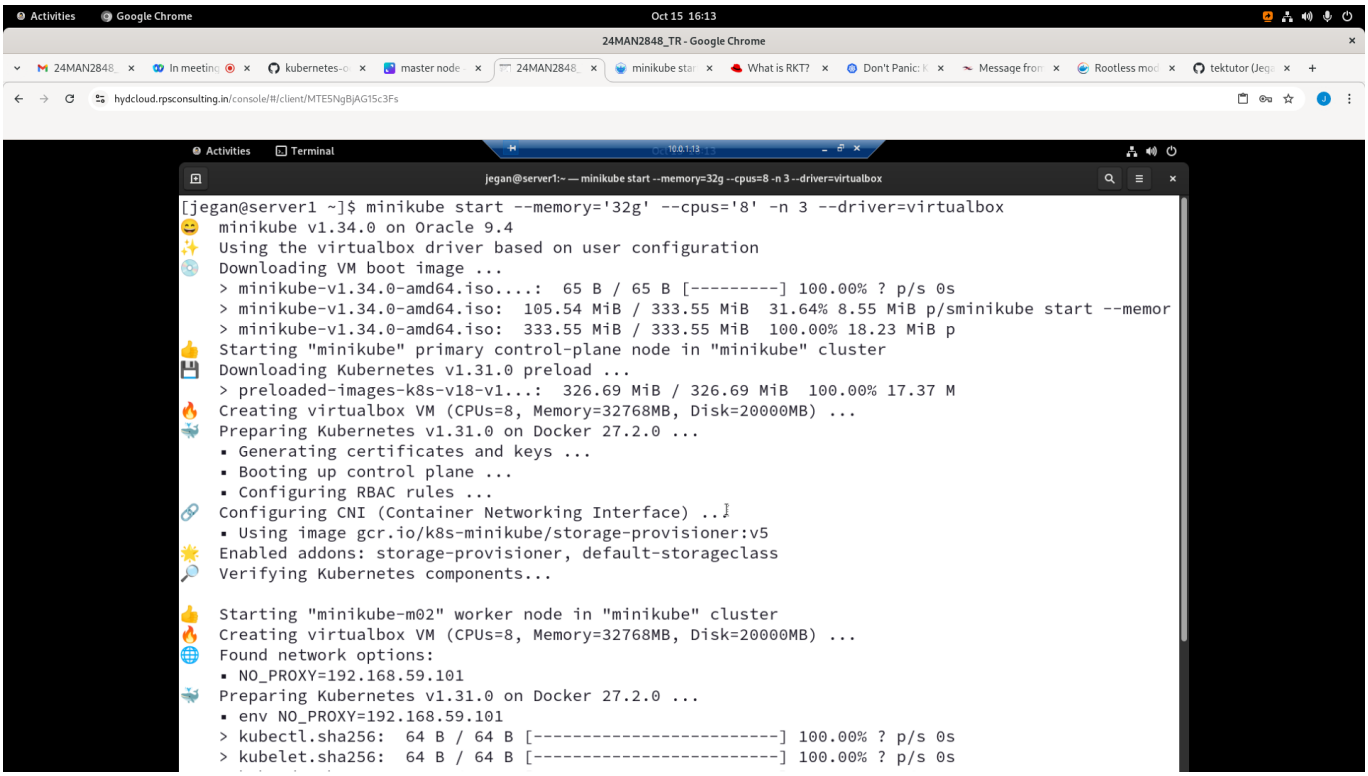
Deployment Overview

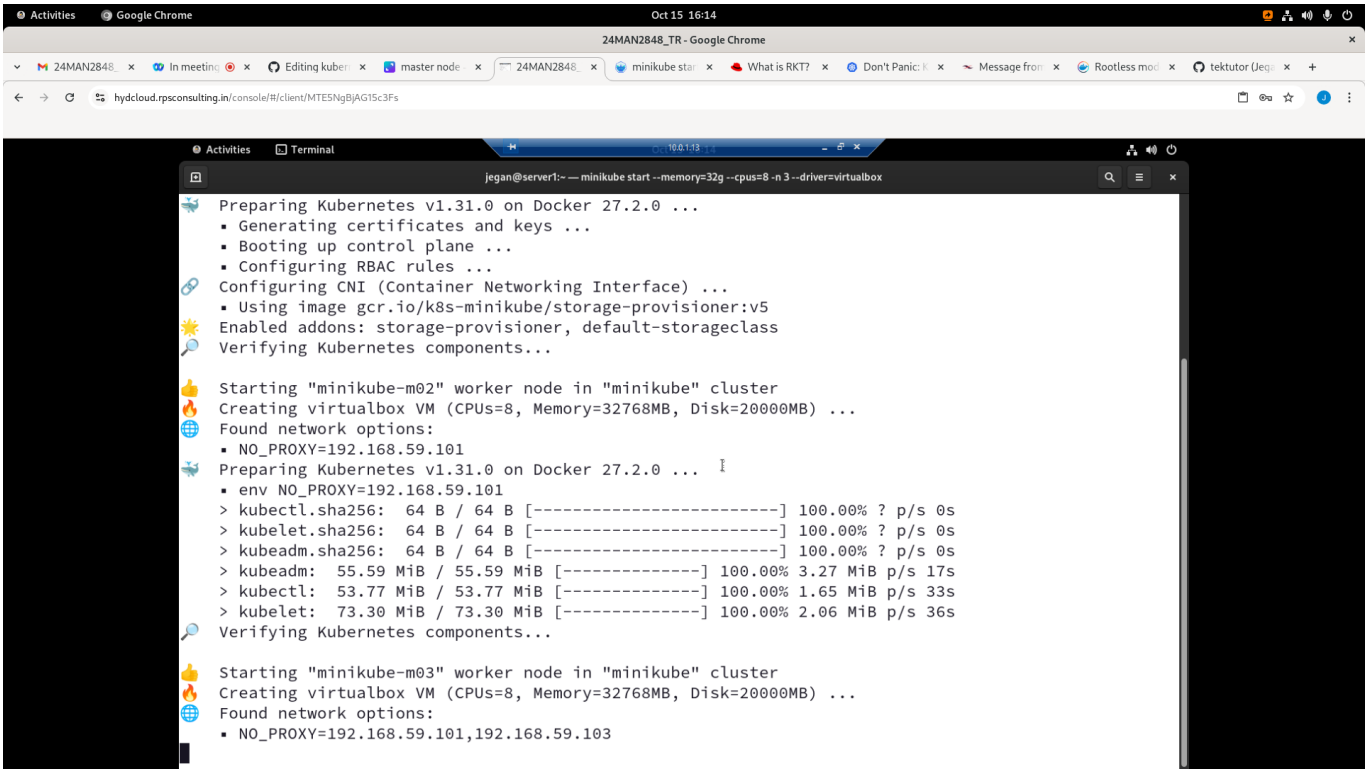
```
- is a Kubernetes resource, that resides in etcd database
- it is a JSON file that is stored in etcd database
- it captures the below details
  - name of the deployment
  - container image that must be used while deploying the respective Pods
  - number of Pod instances that must be running
- Deployment Controller takes the Deployment definition as an input and it
ensures the desired and actual Pod counts are matching
  - Deployment Controller creates the ReplicaSet with desired number of Pod
instance count, while the ReplicaSet controller is the one that manages the
Pods for the respective ReplicaSet
- is used to deploy stateless application
```

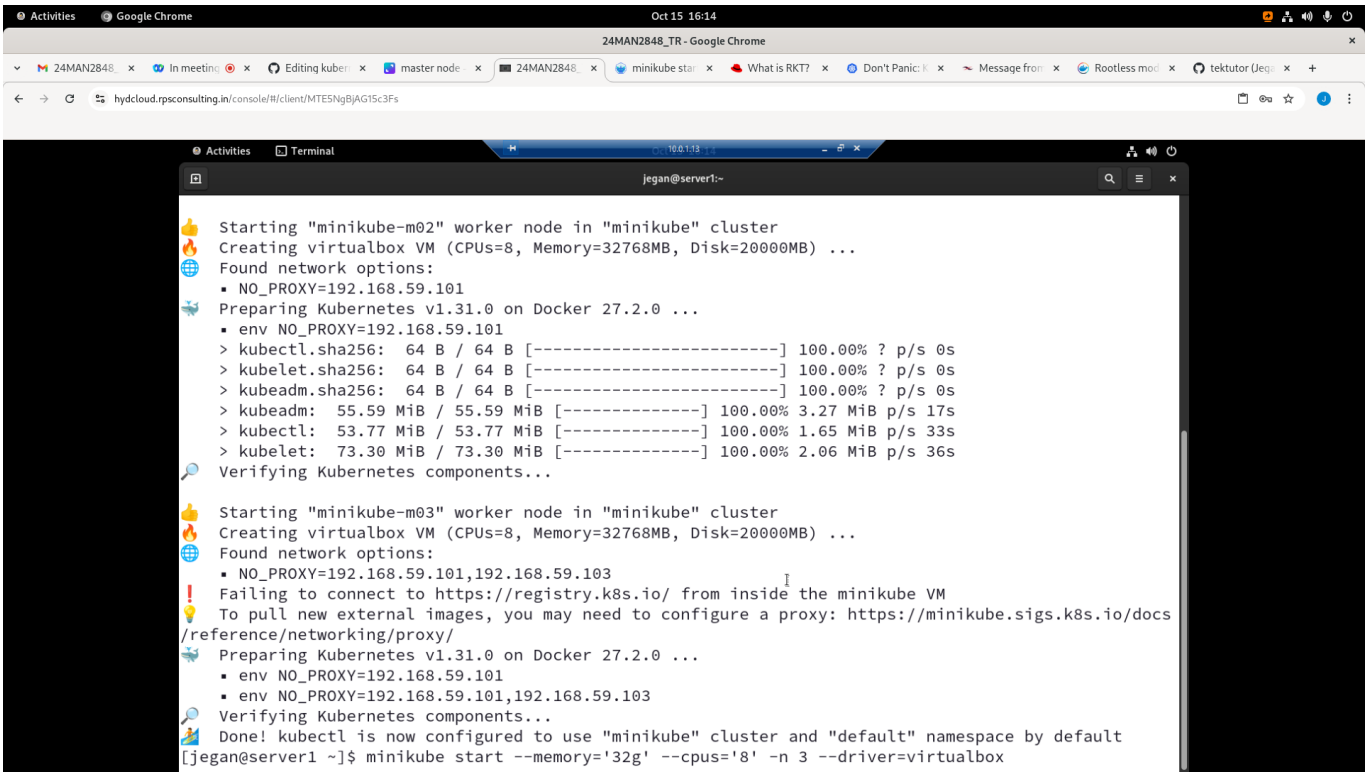
Setup 3 node Kubernetes cluster using Minikube

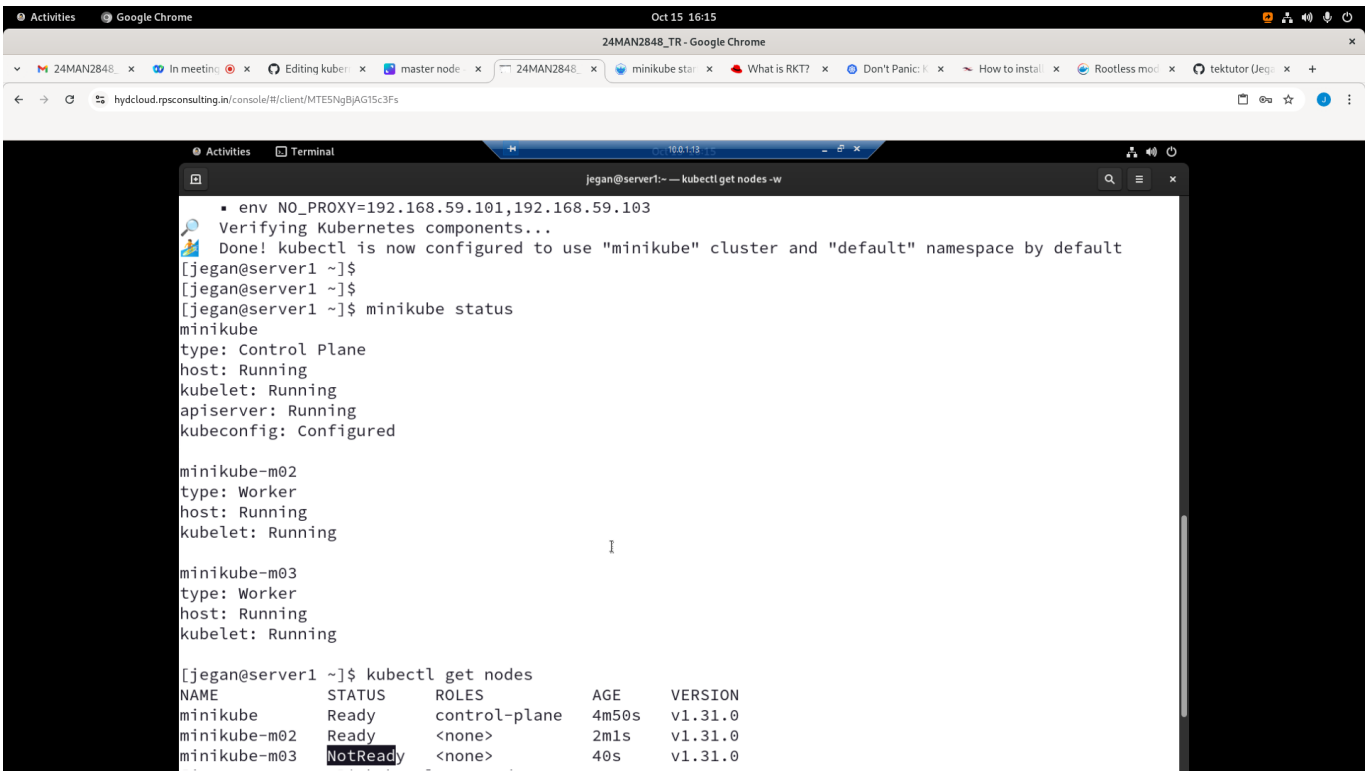
```
docker --version
docker images
minikube delete
minikube start --memory='32g' --cpus='8' -n 3 --driver=virtualbox
minikube status
kubectl get nodes
```

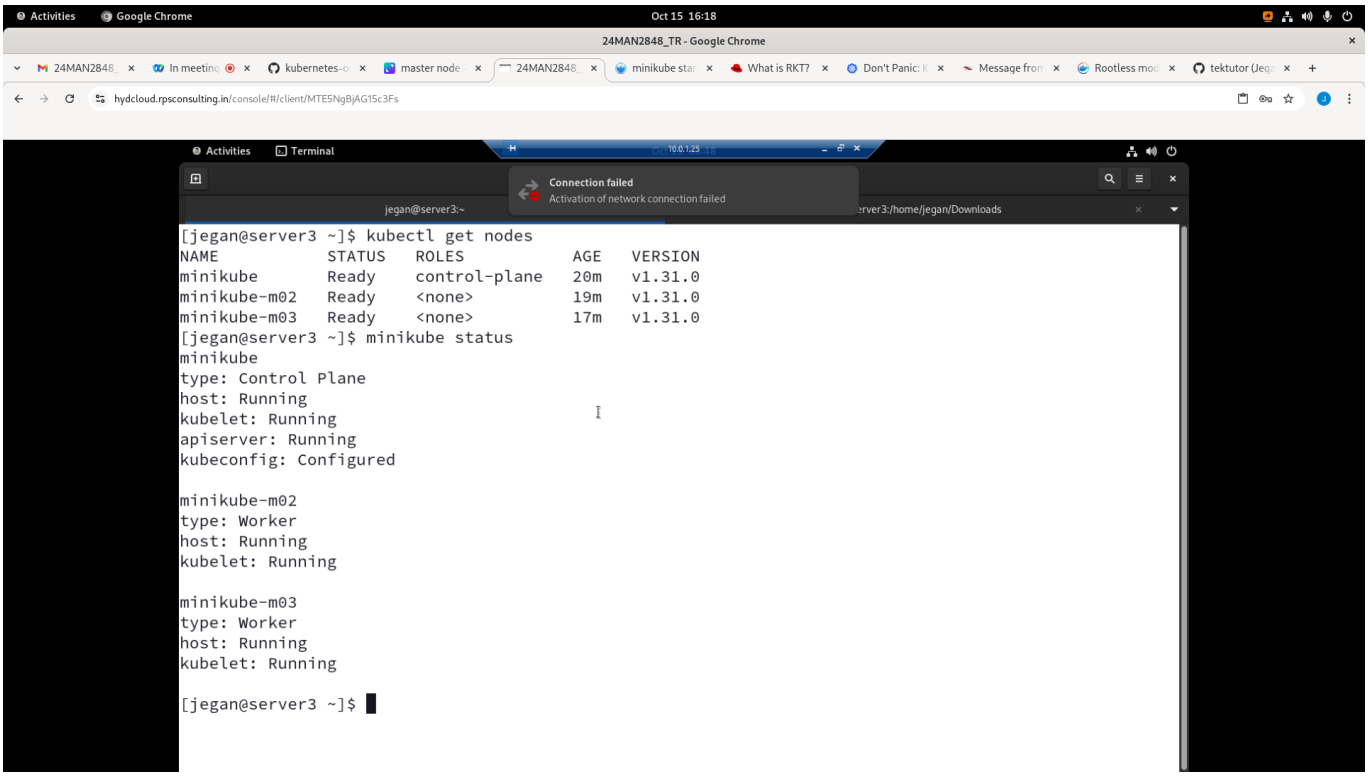
Expected output






Activities
Google Chrome
Oct 15 16:14
24MAN2848_TR - Google Chrome
hydcloud.rpsconsulting.in/console/#/client/MTE5NgBjAG15c3Fs
Activities
Terminal
jegan@server1:~
Starting "minikube-m02" worker node in "minikube" cluster
Creating virtualbox VM (CPUs=8, Memory=32768MB, Disk=20000MB) ...
Found network options:
▪ NO_PROXY=192.168.59.101
Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
▪ env NO_PROXY=192.168.59.101
> kubectl.sha256: 64 B / 64 B [--------------------------] 100.00% ? p/s 0s
> kubelet.sha256: 64 B / 64 B [--------------------------] 100.00% ? p/s 0s
> kubeadm.sha256: 64 B / 64 B [--------------------------] 100.00% ? p/s 0s
> kubeadm: 55.59 MiB / 55.59 MiB [--------------] 100.00% 3.27 MiB p/s 17s
> kubectl: 53.77 MiB / 53.77 MiB [--------------] 100.00% 1.65 MiB p/s 33s
> kubelet: 73.30 MiB / 73.30 MiB [--------------] 100.00% 2.06 MiB p/s 36s
Verifying Kubernetes components...
Starting "minikube-m03" worker node in "minikube" cluster
Creating virtualbox VM (CPUs=8, Memory=32768MB, Disk=20000MB) ...
Found network options:
▪ NO_PROXY=192.168.59.101,192.168.59.103
Failing to connect to https://registry.k8s.io/ from inside the minikube VM
To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
▪ env NO_PROXY=192.168.59.101
▪ env NO_PROXY=192.168.59.101,192.168.59.103
Verifying Kubernetes components...
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
[jegan@server1 ~]$ minikube start --memory='32g' --cpus='8' -n 3 --driver=virtualbox


Activities
Google Chrome
Oct 15 16:15
24MAN2848_TR - Google Chrome
hydcloud.rpsconsulting.in/console/#/client/MTE5NgBjAG15c3Fs
Activities
Terminal
jegan@server1:~ — kubectl get nodes -w
▪ env NO_PROXY=192.168.59.101,192.168.59.103
Verifying Kubernetes components...
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
[jegan@server1 ~]$
[jegan@server1 ~]$
[jegan@server1 ~]$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
minikube-m02
type: Worker
host: Running
kubelet: Running
minikube-m03
type: Worker
host: Running
kubelet: Running
[jegan@server1 ~]$ kubectl get nodes
NAME STATUS ROLES AGE VERSION
minikube Ready control-plane 4m50s v1.31.0
minikube-m02 Ready <none> 2m1s v1.31.0
minikube-m03 NotReady <none> 40s v1.31.0



Just for your reference - do not try this in rps lab environment

Create an Ubuntu 24.04 virtual machine for master-1 node and install docker

```
sudo apt-get update
sudo apt install apt-transport-https curl -y
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
dearmor -o /etc/apt/keyrings/docker.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null
sudo apt-get update

# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-
```

```
plugin docker-compose-plugin
```

Install kubernetes tools

```
sudo apt-get update
# apt-transport-https may be a dummy package; if so, you can skip that
package
sudo apt-get install -y apt-transport-https ca-certificates curl gpg

# If the directory `/etc/apt/keyrings` does not exist, it should be created
before the curl command, read the note below.
# sudo mkdir -p -m 755 /etc/apt/keyrings
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo
gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo tee
/etc/apt/sources.list.d/kubernetes.list

sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl

sudo systemctl enable --now kubelet
```

Configure kubelet service, sudo vim /etc/default/kubelet

```
KUBELET_EXTRA_ARGS="--cgroup-driver=cgroupfs"
```

Append below docker configuration in /etc/docker/daemon.json

```
{
      "exec-opts": ["native.cgroupdriver=systemd"],
      "log-driver": "json-file",
      "log-opts": {
         "max-size": "100m"
       },
      "storage-driver": "overlay2"
}
```

Disable virtual memory and comment all the line in /etc/fstab

```

```

```
sudo swapoff -a
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

Configure kubeadm, sudo vim /etc/systemd/system/kubelet.service.d/10-kubeadm.conf and add the below line

```
Environment="KUBELET_EXTRA_ARGS=--fail-swap-on=false"
```

Restart docker

```
sudo systemctl daemon-reload
sudo systemctl enable kubelet
sudo systemctl enable docker
sudo systemctl start docker
sudo systemctl status docker
```

Enable kernel modules

```
sudo modprobe overlay
sudo modprobe br_netfilter
sudo sysctl -w net.ipv4.ip_forward=1
sudo sysctl --system
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Install Flannedl network addon

```
kubectl apply -f https://github.com/flannel-
io/flannel/releases/latest/download/kube-flannel.yml
```

Check the installation

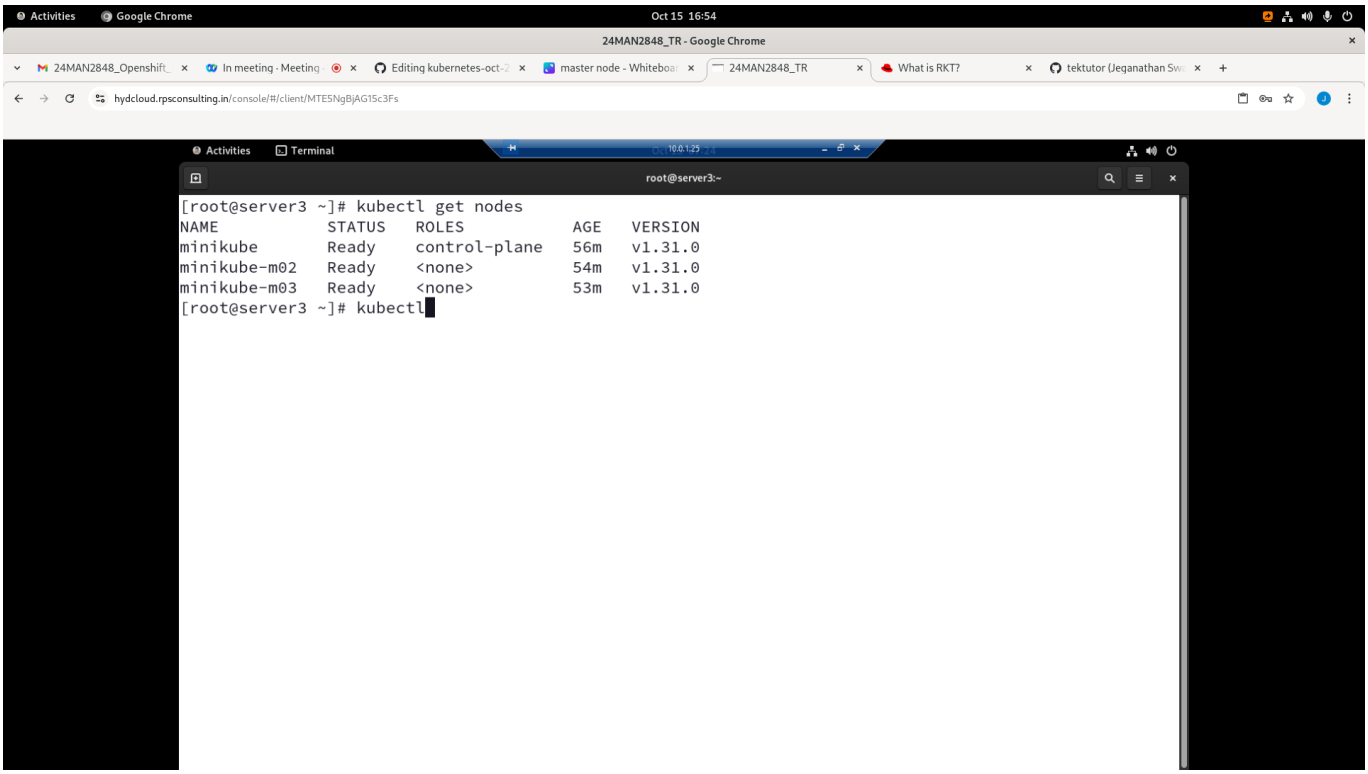
```
kubectl get nodes -o wide
kubectl get pods --all-namespaces
```

Use the join token to join worker nodes

Lab - Listing Kubernetes nodes in the cluster

```
kubectl get nodes
```

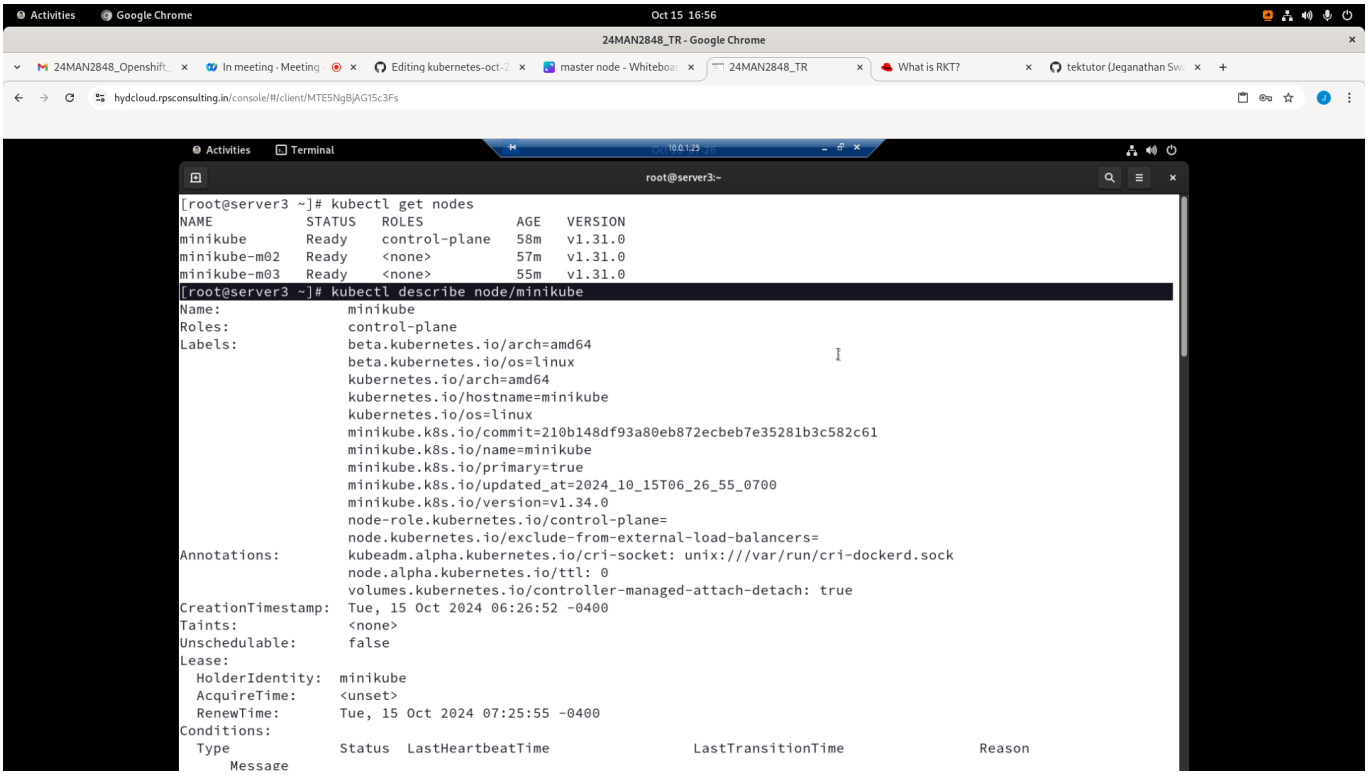
Expected output

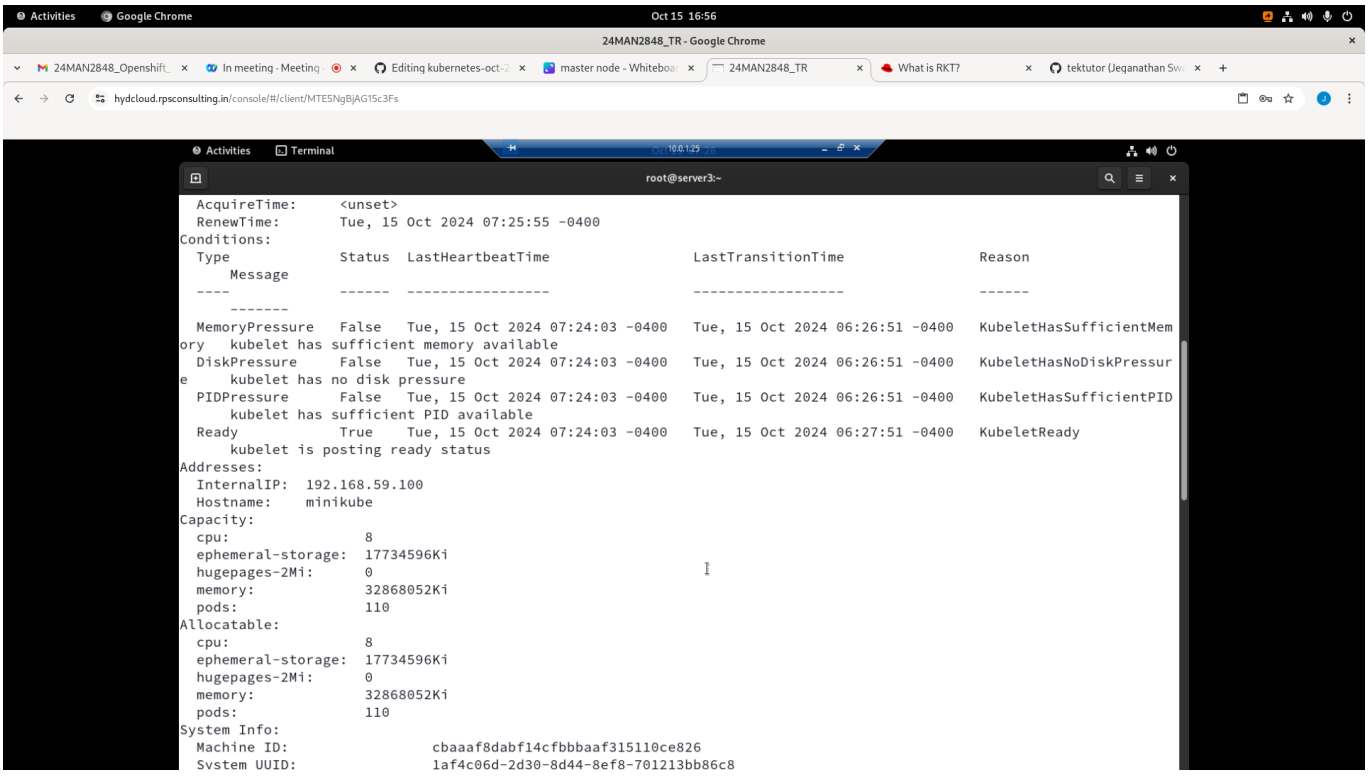


Lab - Finding meta data about master node

```
kubectl get nodes
kubectl describe node/minikube
```

Expected output





Lab - Finding meta data about master node

```
kubectl get nodes
kubectl describe node/minikube-m02
```

Expected output

Lab - Deploying our first application into Kubernetes cluster

Let's create a namespace before deploying nginx

```
kubectl create namespace jegan
```

```
kubectl create deployment nginx --image=nginx:latest --replicas=3 -n jegan
```

Listing the deployments

```
kubectl get deployments -n jegan
kubectl get deployment -n jegan
kubectl get deploy -n jegan
```

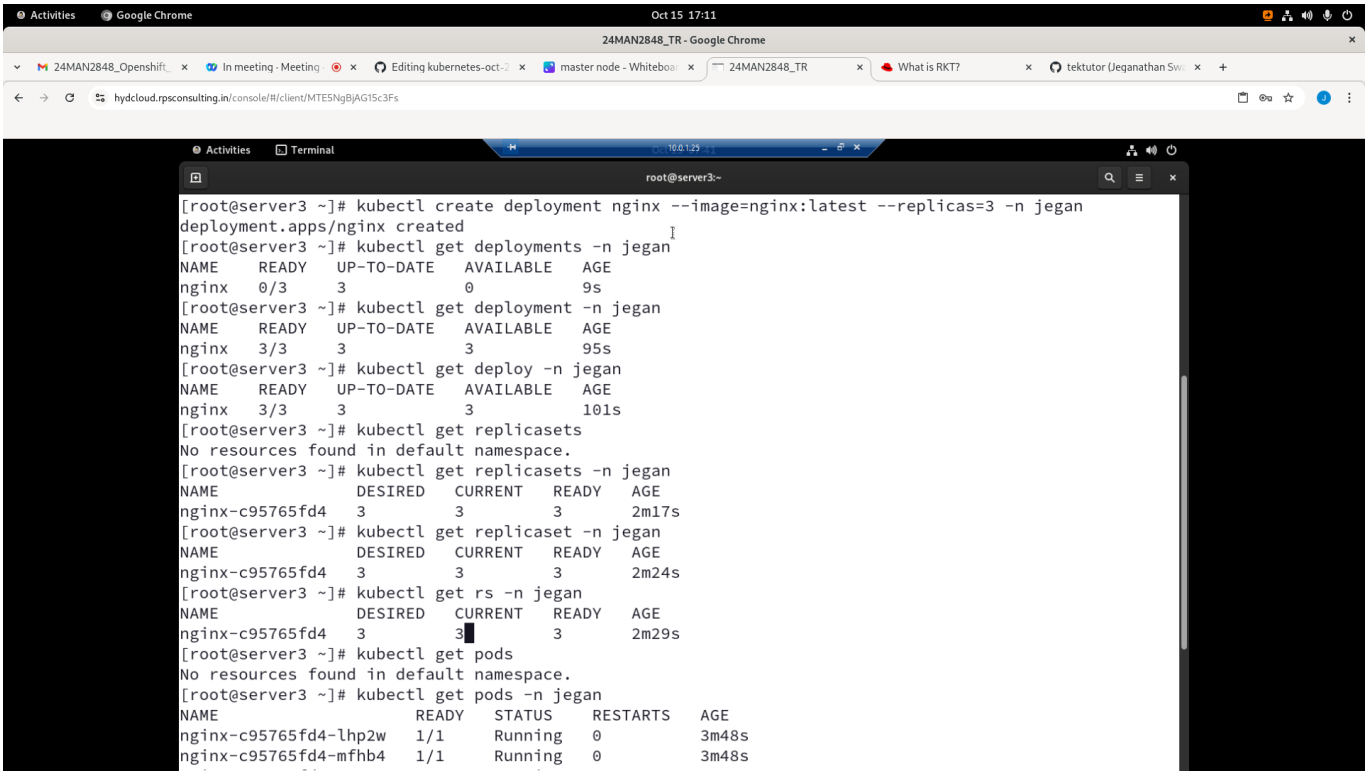
Listing the replicasets

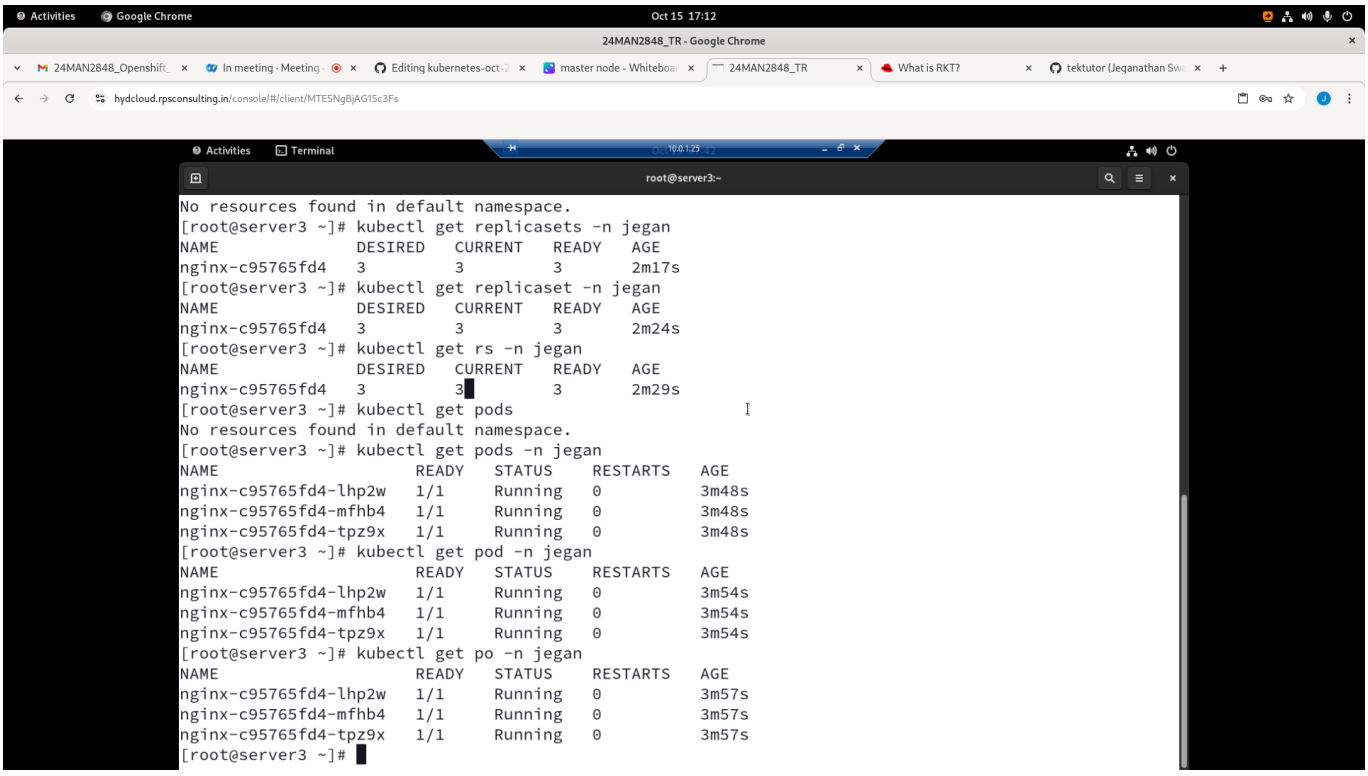
```
kubectl get replicasets -n jegan
kubectl get replicaset -n jegan
kubectl get rs -n jegan
```

Listing the pods

```
kubectl get pods -n jegan
kubectl get pod -n jegan
kubectl get po -n jegan
```

Expected output

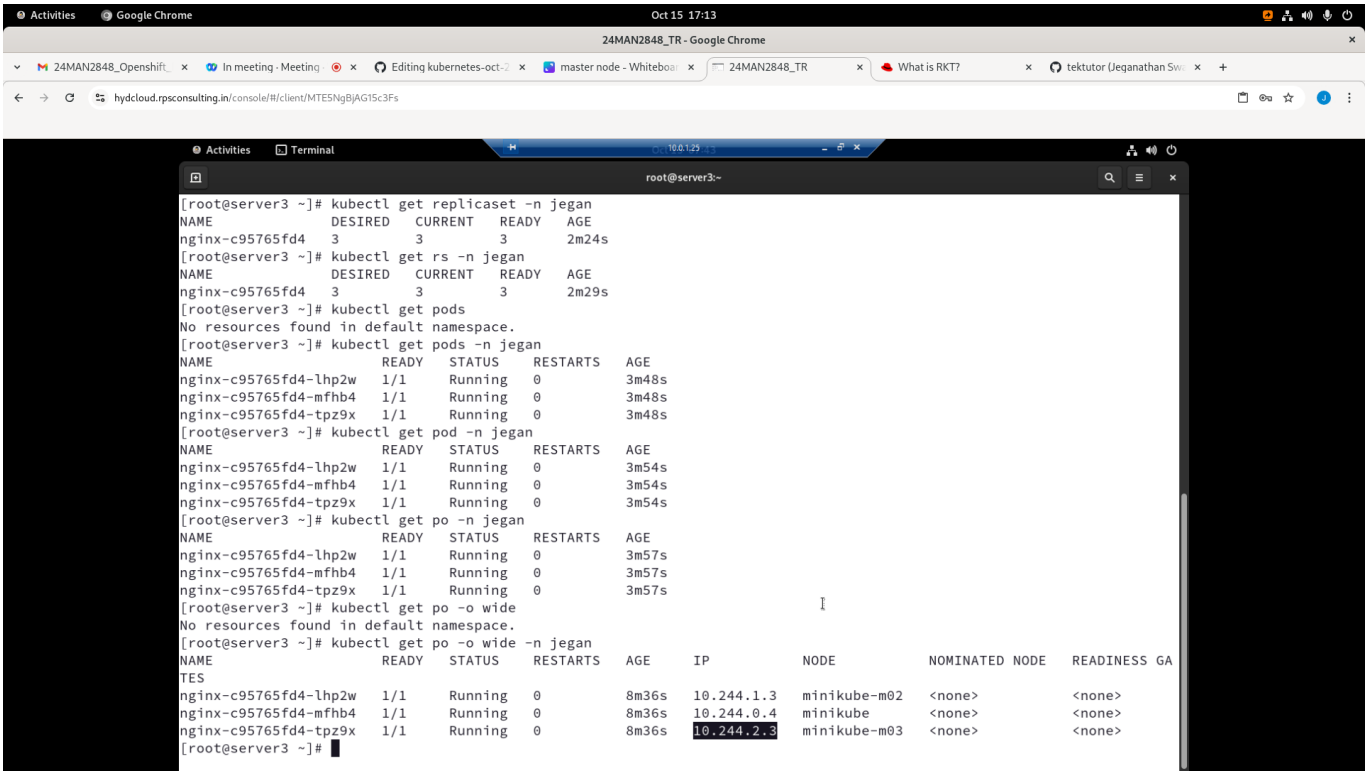




Finding on which node each is running along with their Pod IP Address

```
kubectl get po -o wide
```

Expected output

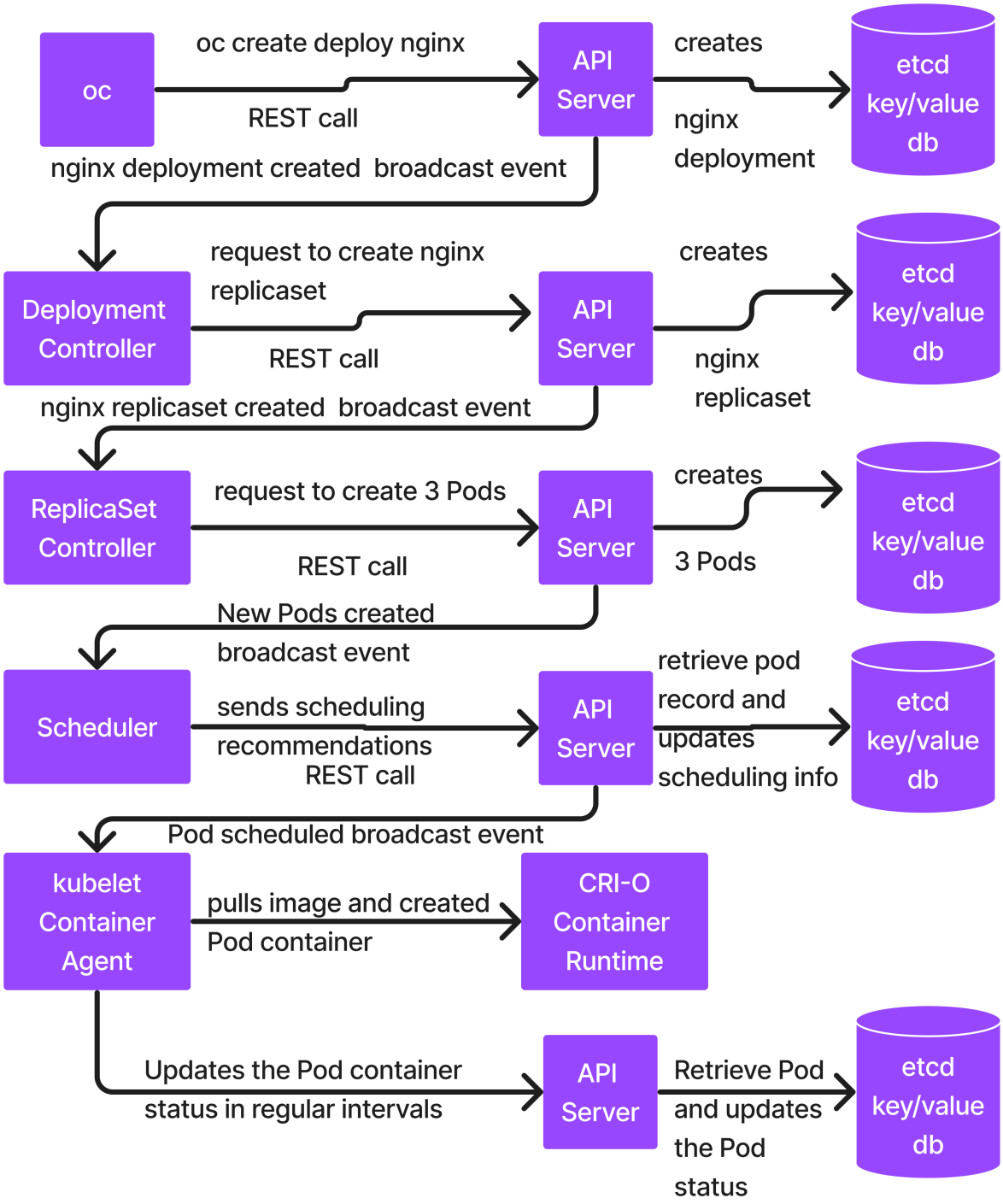


Info - What happens internally in Kubernetes cluster when we deploy an application

```
kubectl create deployment nginx --image=nginx:latest --replicas=3 -n jegan
```

```
- kubectl client tools will make a REST call to API server, requesting to
create a deployment with name nginx with 3 Pod instance using container
image nginx:latest
- API Server receives the request from kubectl and it creates a deployment
record in the etcd database
- API Server broadcasts an event to indicate a new deployment by name nginx
is created
- Deployment Controller receives the event, it then makes a REST call to
API server, requesting to create a replicaset for nginx deployments
- API Server receives the request from Deployment Controller, it then
creates a replicaset as requested by Deployment controller
- API Server broadcasts an event to indicate a new replicaset is created
for nginx deployment
- Replicaset Controller receives the event, it then makes a REST call to
API Server, requesting to create 3 Pod entries in etcd
- API Server receives the request from ReplicaSet Controller, it then
create 3 Pods records(JSON) in the etcd database
- API Server broadcasts an event for every new Pod created
- Scheduler receives the event, it then identifies a healthy node where
that Pod can be scheduled.  Scheduler sends the scheduling recommendataions
for each Pod to API Server via REST call
- API Server receives the scheduling recommendations from Scheduler, it
```

```
retrieves the respective Pod entries from etcd and then it updates the
scheduling details
- API Server broadcasts an event for every Pod with scheduling details
- kubelet container agent that runs in the scheduled node receives the
event, it then interacts with container runtime to download the image
- kubelet creates a container with the image downloaded
- kubelet starts the container
- kubelet reports the status of each container to API Server via REST call
in regular intervals ( hearbeat notifications )
- API receives the status from kubelet container agents running on each
node and it updates the Pod status in the etcd database
```



Info - Kubernetes Service

```
- a way to expose an application that runs in Kubernetes for
internal/external access
- represents a group of load balanced pods from a single deployment
- services supports 2 types of scope
  1. Internal and
  2. External
- Internal Service
```

```
  - ClusterIP Service - this is accessible only within the Kubernetes
cluster where the application Pods are running
- External Service
  - NodePort Service - this is accessible to external users
  - LoadBalancer Service - generally used in public cloud environment like
AWS, GCP, Azure, etc.,
```

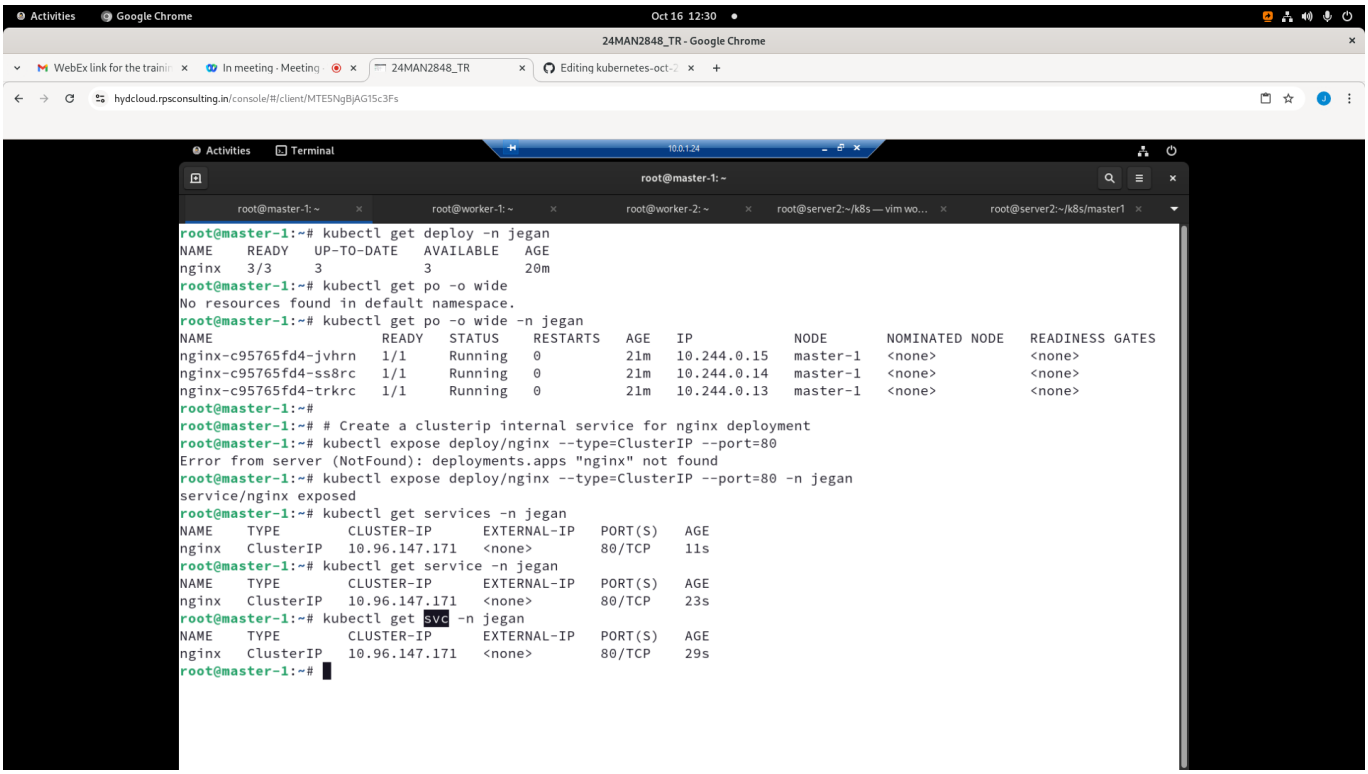
Lab - Creating an internal service from nginx deployment

```
kubectl create namespace jegan
kubectl create deployment nginx --image=nginx:latest --replicas=3
kubectl get deploy,rs,po
```

Let's create an internal service

```
kubectl expose deploy/nginx --type=ClusterIP --port=80
kubectl get services
kubectl get service
kubectl get svc
kubectl describe svc/nginx
```

Expected output



Lab - Set context to your namespace

```
kubectl config set-context --current --namespace=jegan
```

Lab - Accessing ClusterIP Internal service

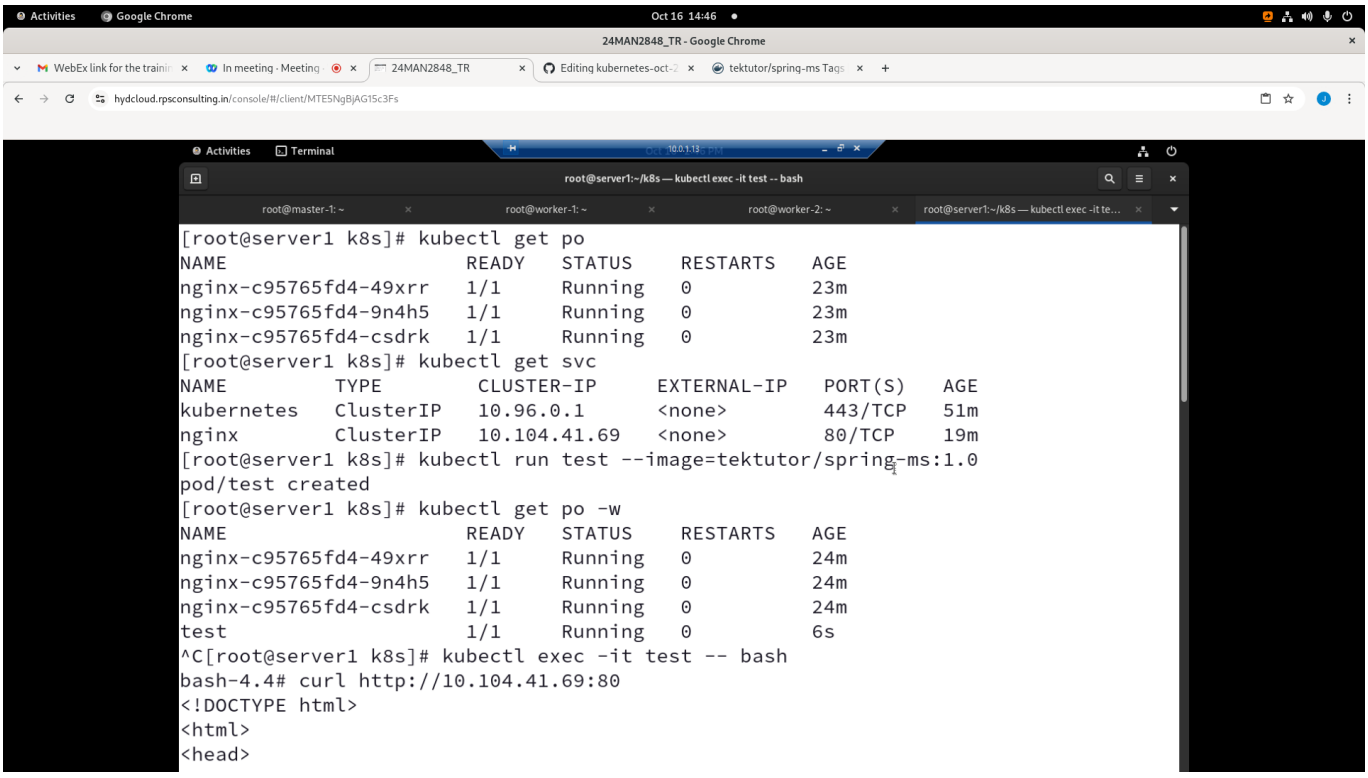
Let's create a test pod

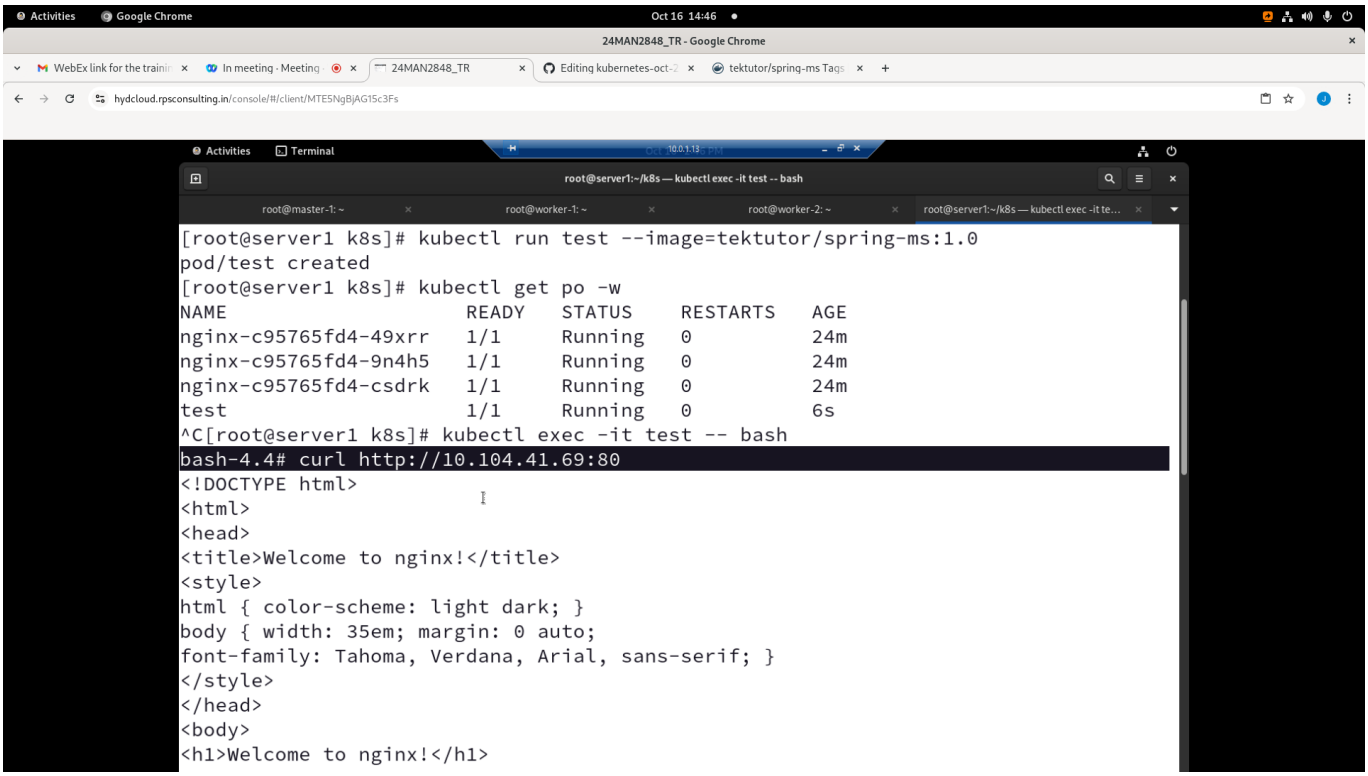
```
kubectl config set-context --current --namespace=jegan
kubectl run test --image=tektutor/spring-ms:1.0
kubectl get po
```

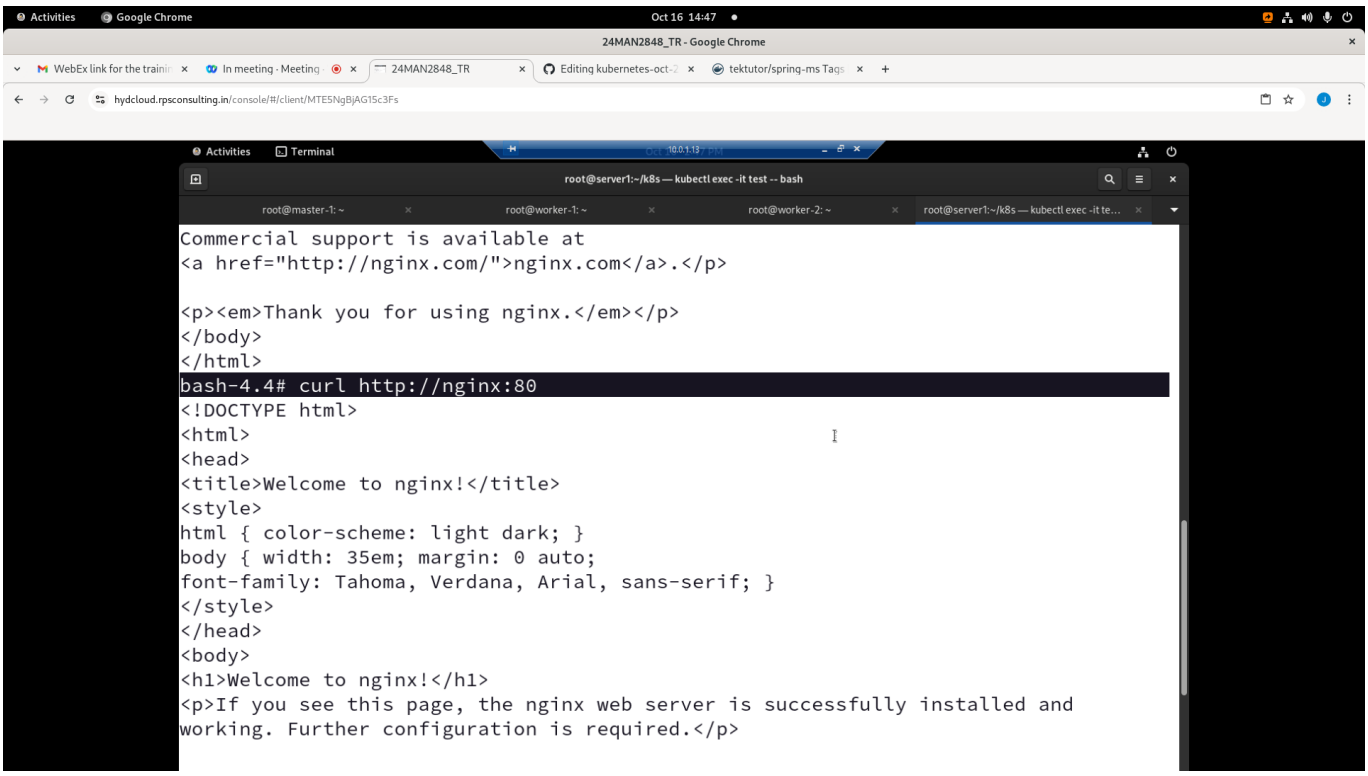
Let's get inside the test pod shell

```
kubectl exec -it test -- bash
curl http://<service-ip>:<service-port>
curl http://<service-name>:<service-port>

curl http://nginx:80
```

Expected output







Lab - Creating a nodeport external service

First of all, we need to delete the clusterip internal service

```
kubectl config set-context --current --namespace=jegan
kubectl delete svc/nginx
kubectl get svc
```

Let's list the deployment

```
kubectl get deploy
```

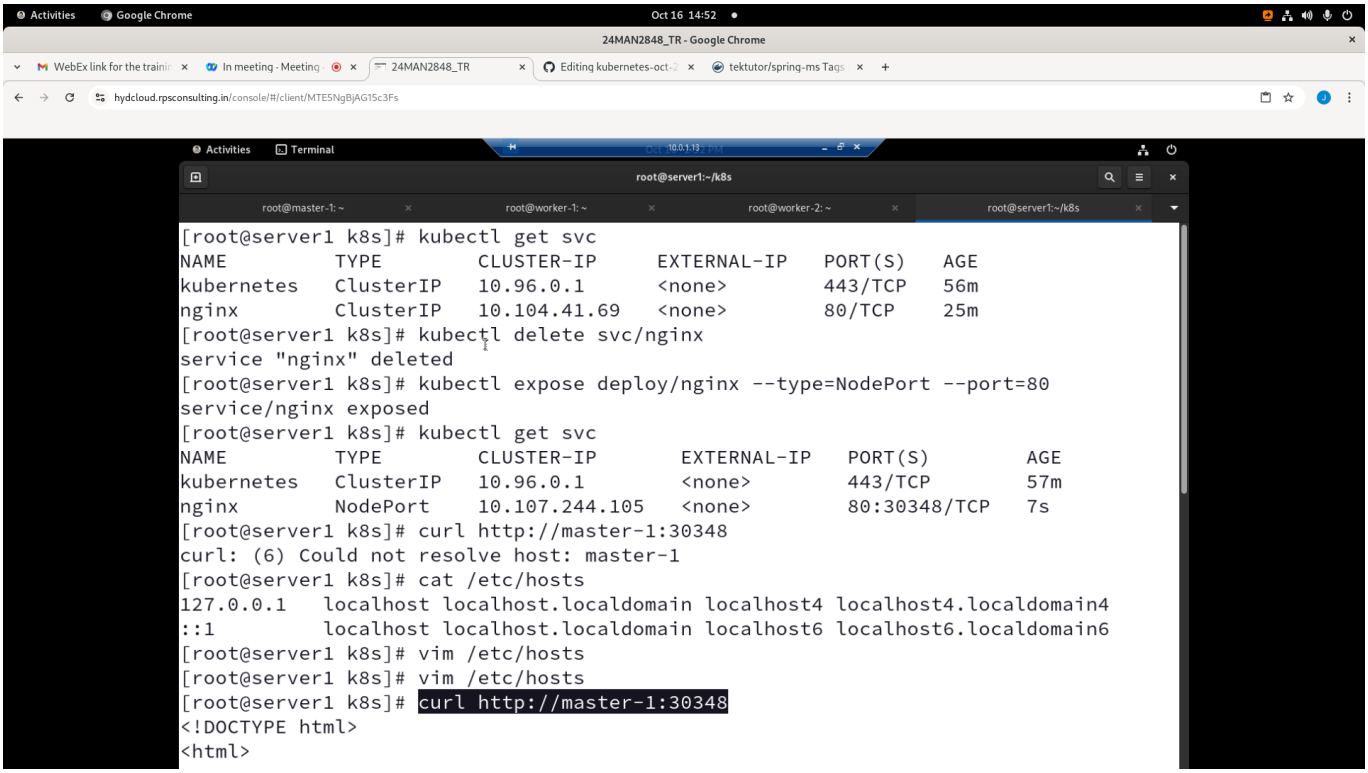
Let's create the nodeport external service

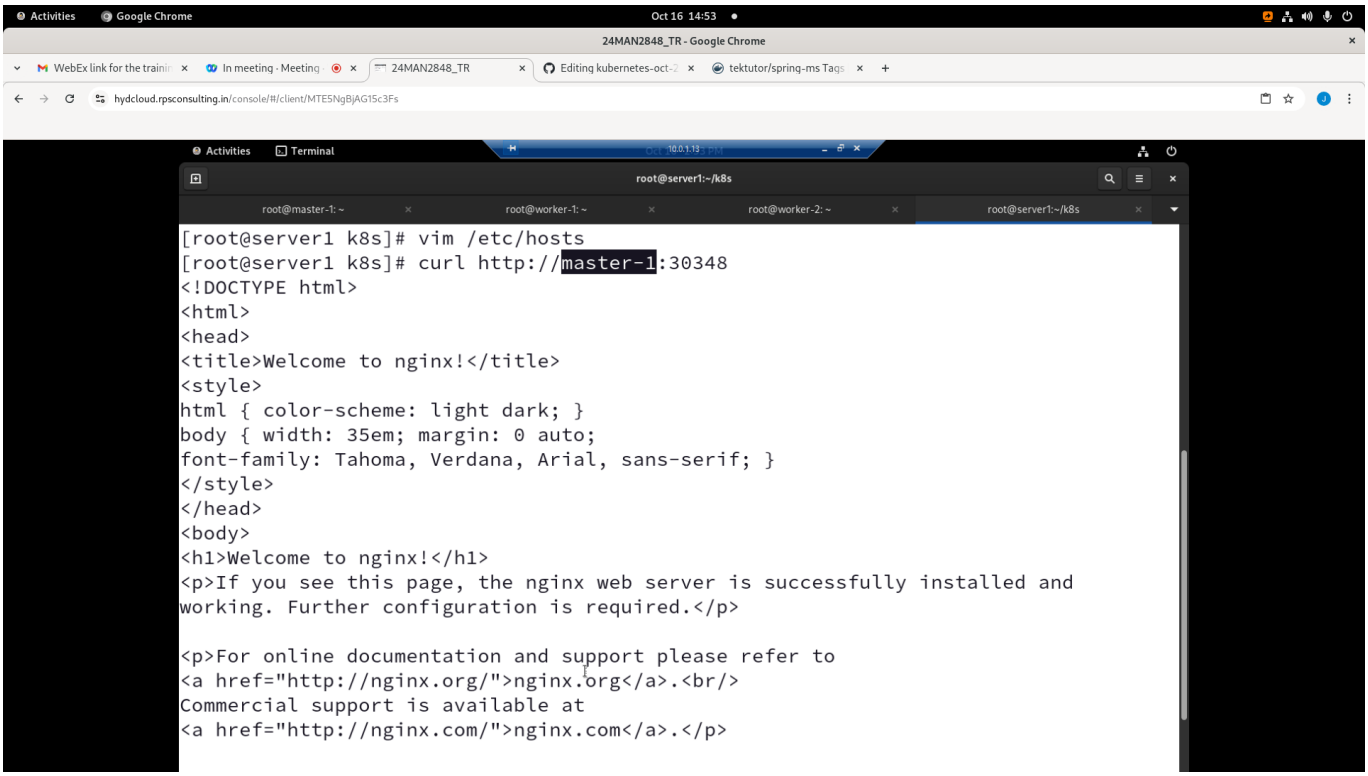
```
kubectl expose deploy/nginx --type=NodePort --port=80
kubectl get svc
kubectl describe svc/nginx
```

Accessing the nodeport service, let's find the node ip address

```
kubectl get nodes -o wide
curl http://master-1:<node-port>
curl http://master-1:30348
```

Expected output





Lab - Port forward (Meant for developer test)

In the below command, you may replace 8080 with any other port that is available on your system

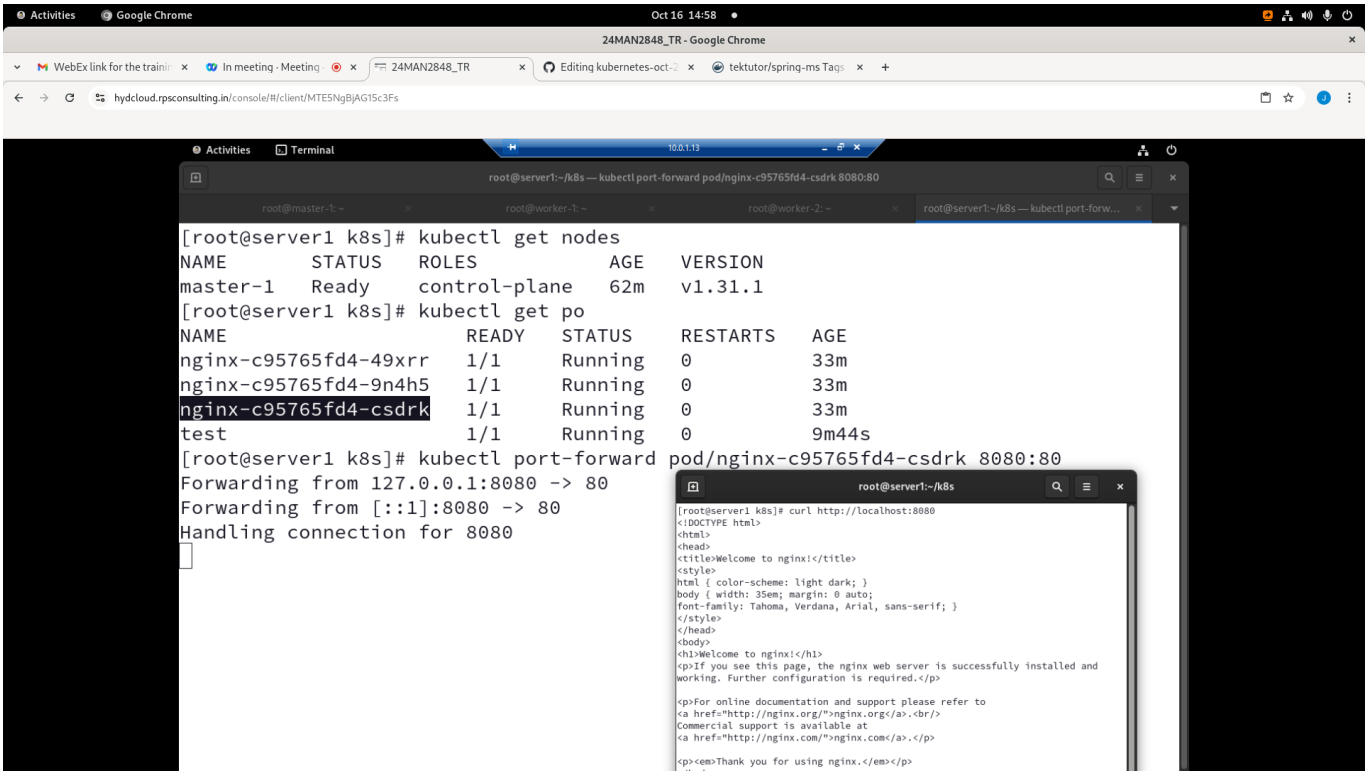
```
kubectl get po
kubectl port-forward pod/nginx-c95765fd4-csdrk 8080:80
```

Open a new terminal window

```
curl http://localhost:8080
```

Once you are done you can stop the port-forward by pressing Ctrl+C

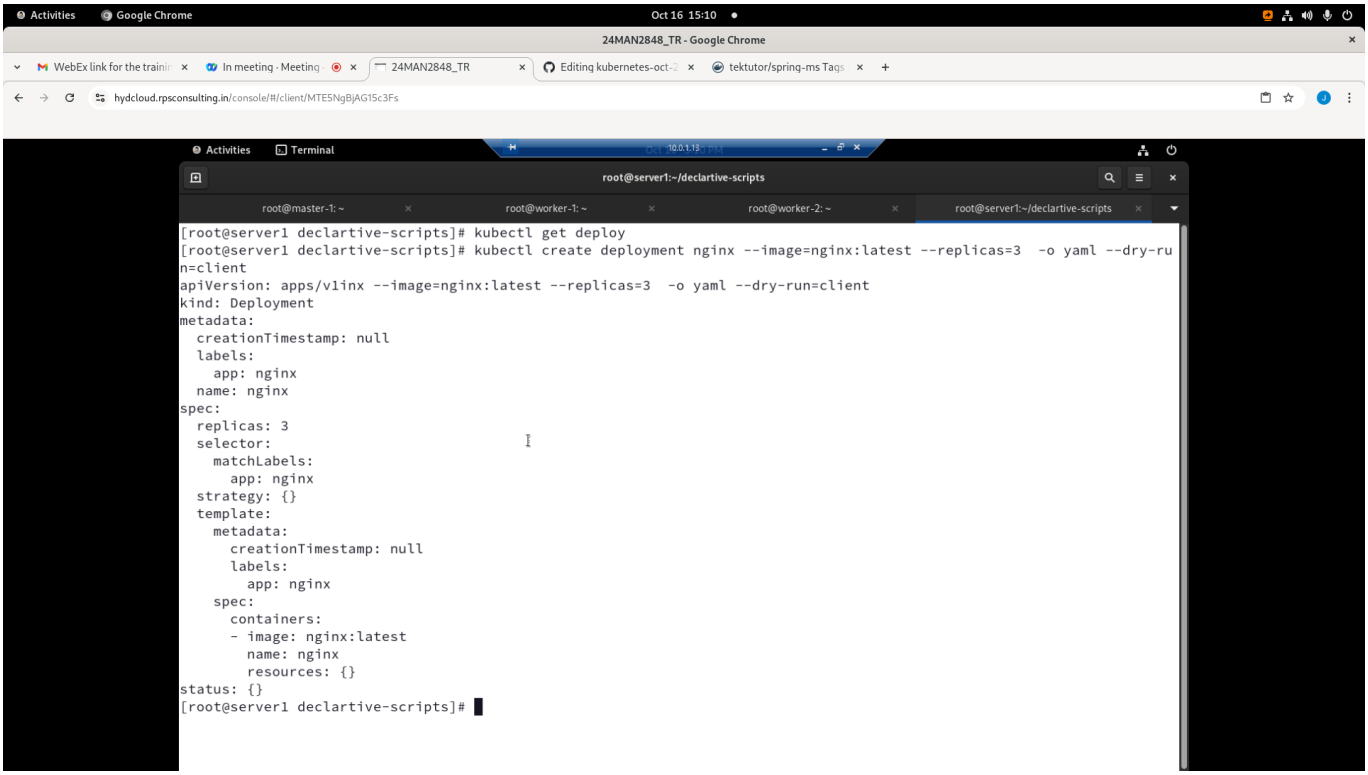
Expected output

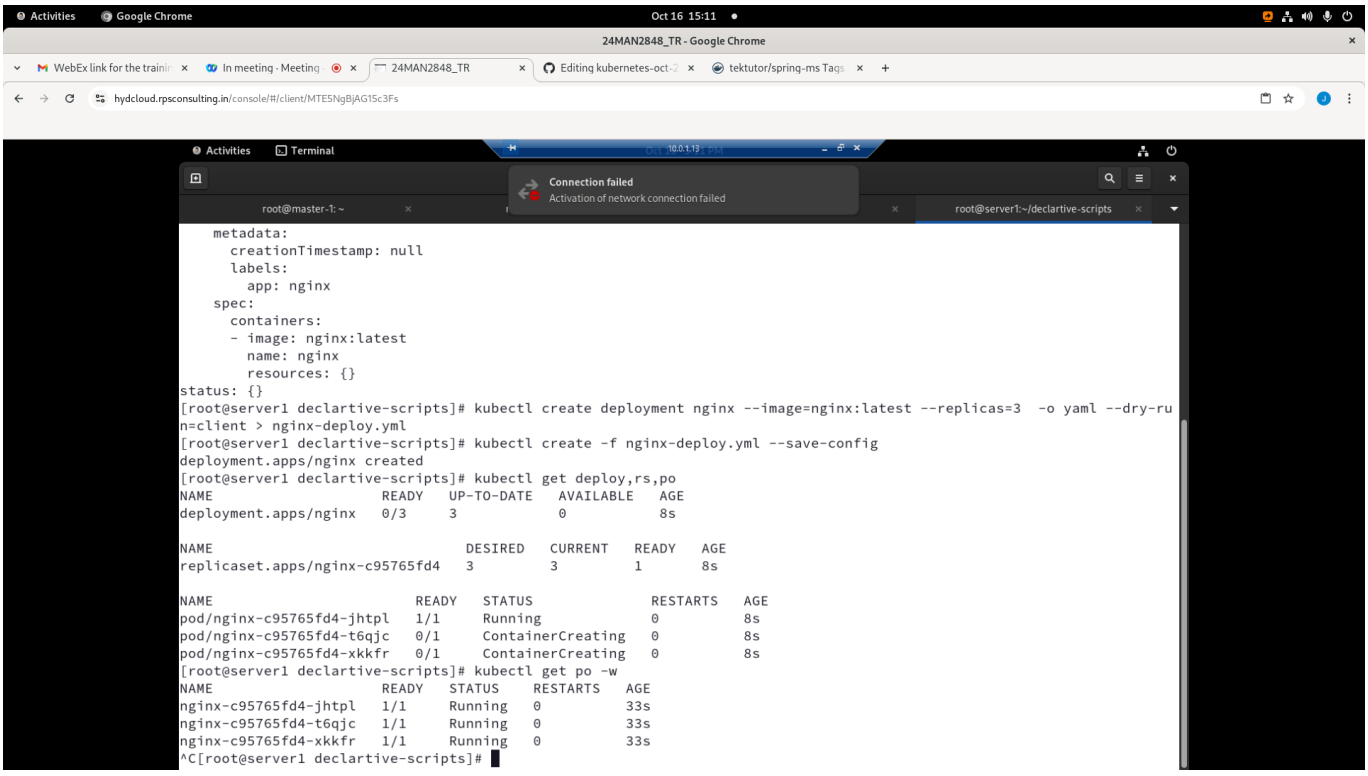


Lab - Autogenerating declarative manifest script for nginx deployment

```
kubectl create deployment nginx --image=nginx:latest --replicas=3 -o json --dry-run=client
kubectl create deployment nginx --image=nginx:latest --replicas=3 -o yaml --dry-run=client

kubectl create deployment nginx --image=nginx:latest --replicas=3 -o yaml --dry-run=client > nginx-deploy.yml

kubectl config set-context --current --namespace=jegan
kubectl create -f nginx-deploy.yml --save-config
kubectl get deploy,rs,po
```

Expected output



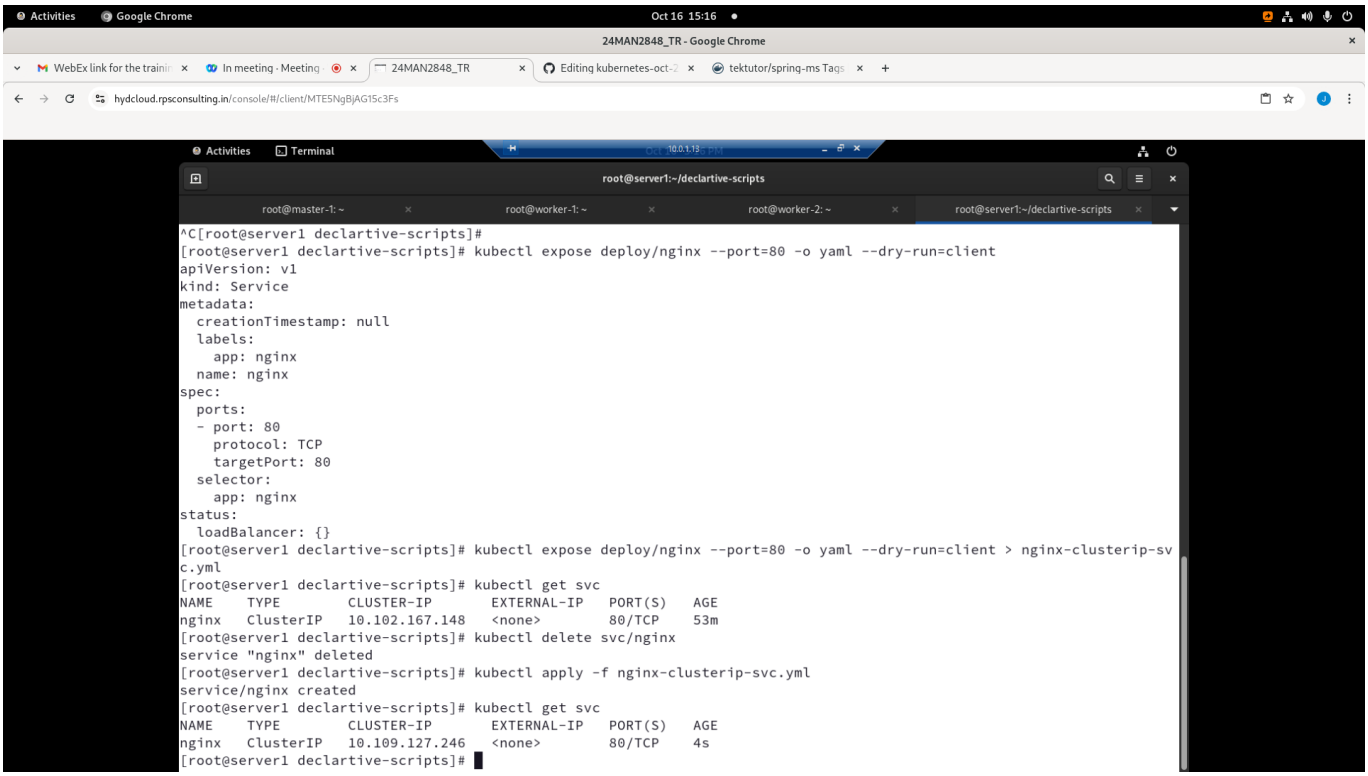


Day 3

Lab - Declaratively creating a clusterip internal service

```
kubectl expose deploy/nginx --type=ClusterIP --port=80 -o yaml --dry-
run=client
kubectl expose deploy/nginx --type=ClusterIP --port=80 -o yaml --dry-
run=client > nginx-clusterip-svc.yml
kubectl apply -f nginx-clusterip-svc.yml

kubectl get svc
kubectl describe svc/nginx
```

Expected output



Lab - Declaratively creating nodeport external service

Let's delete the clusterip service in declarative style

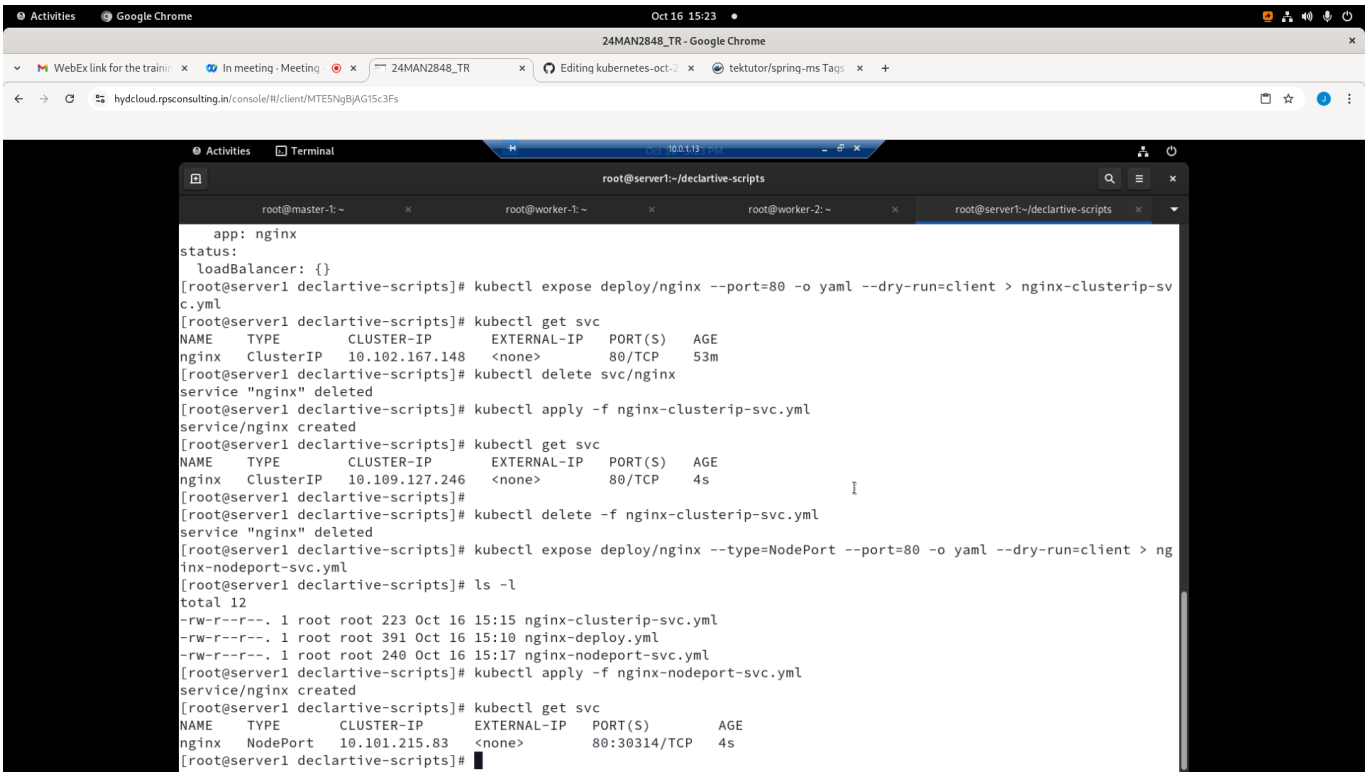
```
kubectl delete -f nginx-clusterip-svc.yml
```

Let's create the nodeport external service in declartive style

```
kubectl expose deploy/nginx --type=NodePort --port=80 -o yaml --dry-
run=client
```

```
kubectl expose deploy/nginx --type=NodePort --port=80 -o yaml --dry-
run=client > nginx-nodeport-svc.yml
kubectl apply -f nginx-nodeport-svc.yml

kubectl get svc
kubectl describe svc/nginx
```

Expected output



Lab - Creating a Pod in declarative style

Create a file with below content and save the file as pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: frontend
spec:
  containers:
  - name: my-container
    image: tektutor/spring-ms:1.0
```

Let's create the pod

```
kubectl create -f pod.yml --save-config
kubectl get po
```

Let's modify the pod.yml as shown below

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: web
    tier: frontend
spec:
  containers:
  - name: my-container
    image: tektutor/spring-ms:1.0
```

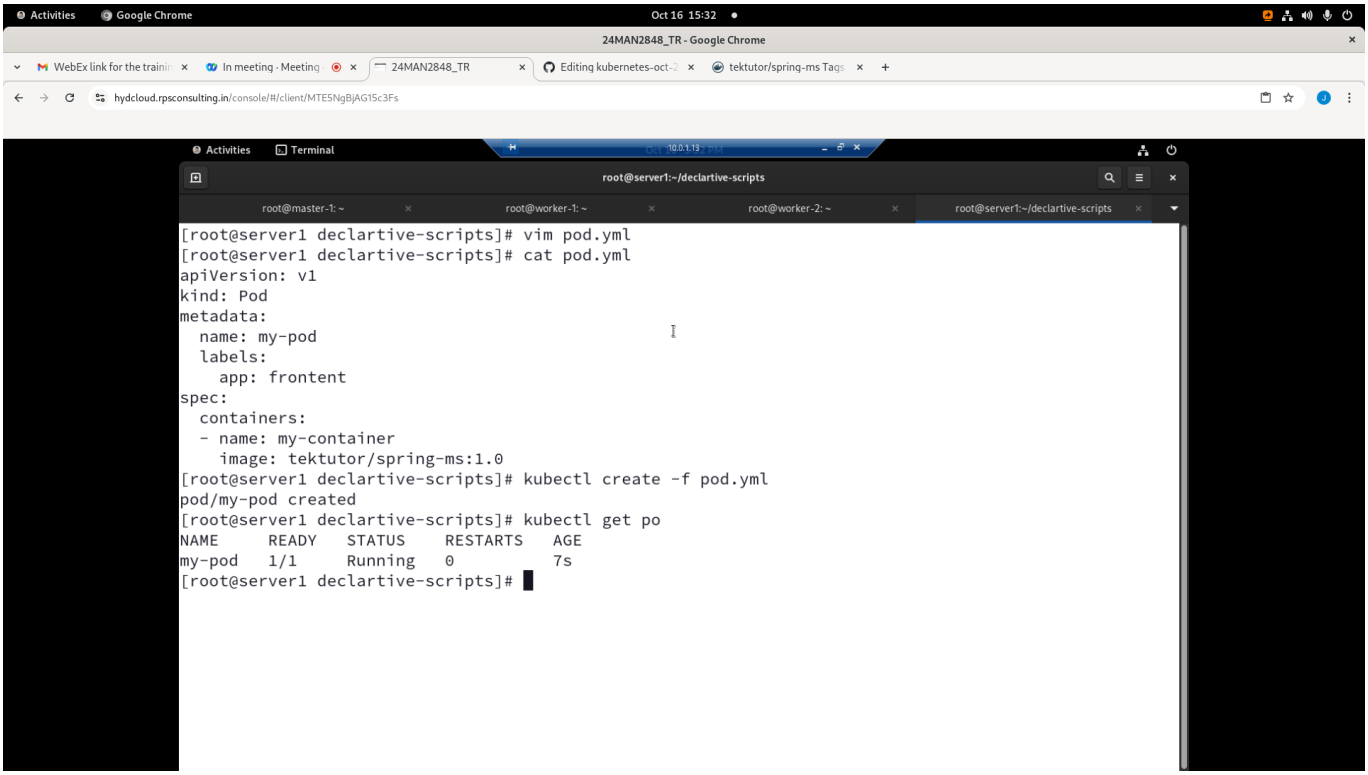
Let's apply the delta changes on the already existing kubernetes resource

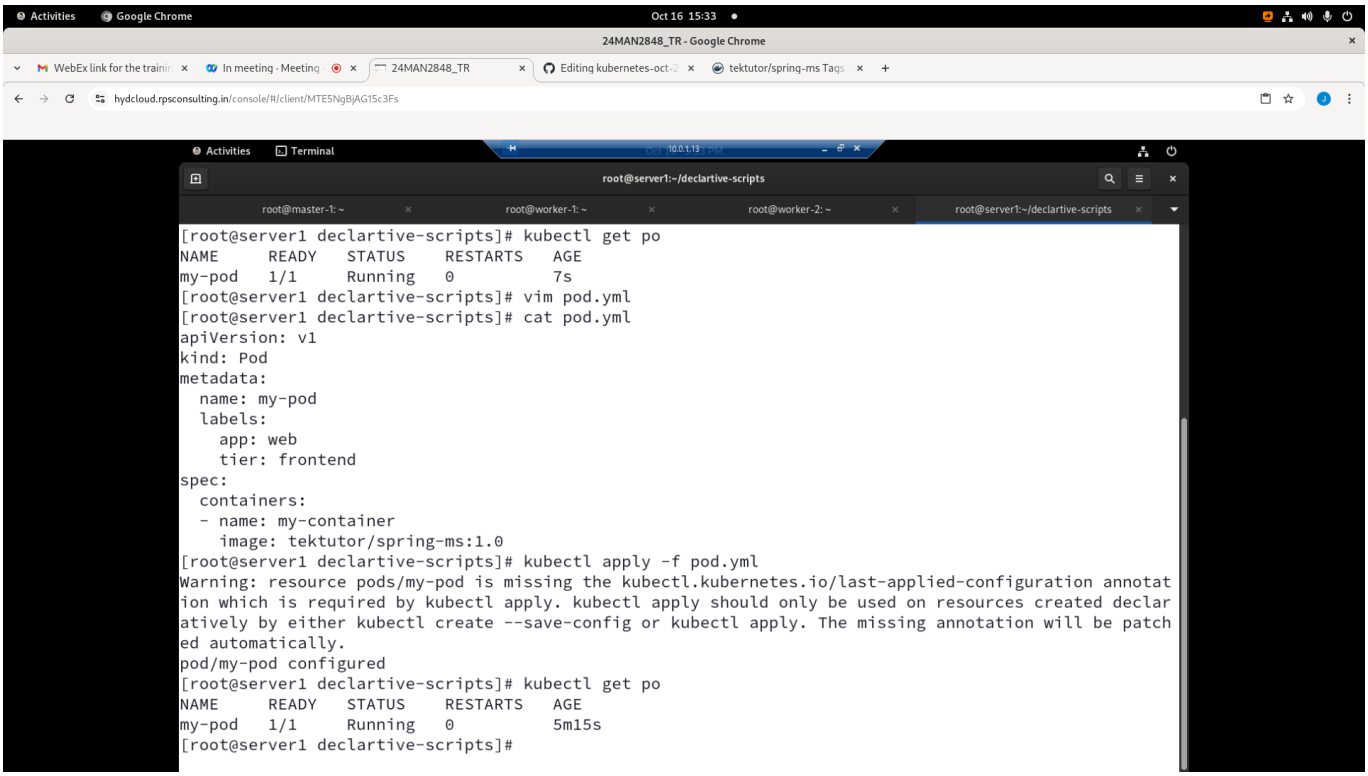
```
kubectl apply -f pod.yml
kubectl get po
```

Once you are done with this exercise, you may delete it as shown below

```
kubectl delete -f pod.yml
```

Expected output





Lab - Deploying your custom application into Kubernetes cluster

We need to clone the source code first

```
cd ~
git clone https://github.com/tektutor/spring-ms.git
rm *.yml *.yaml
```

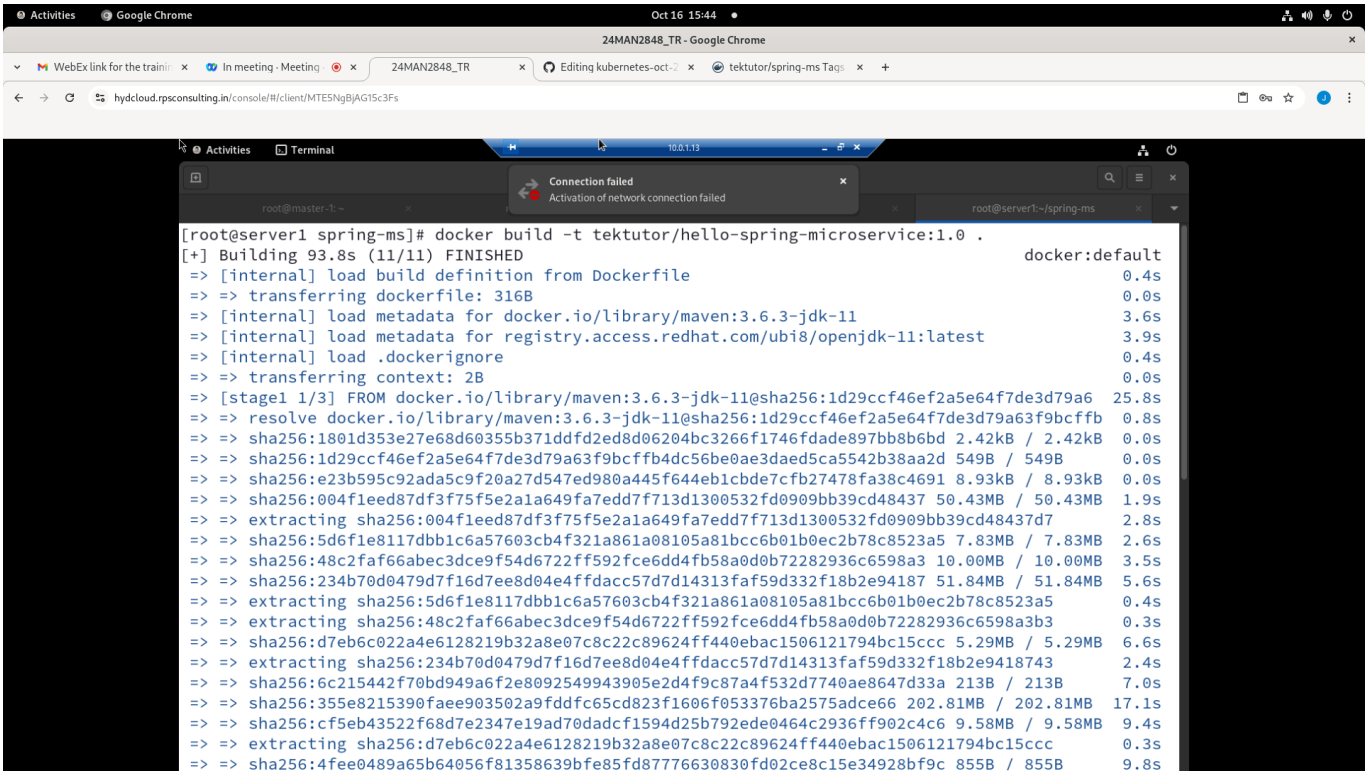
Let's build the custom docker image

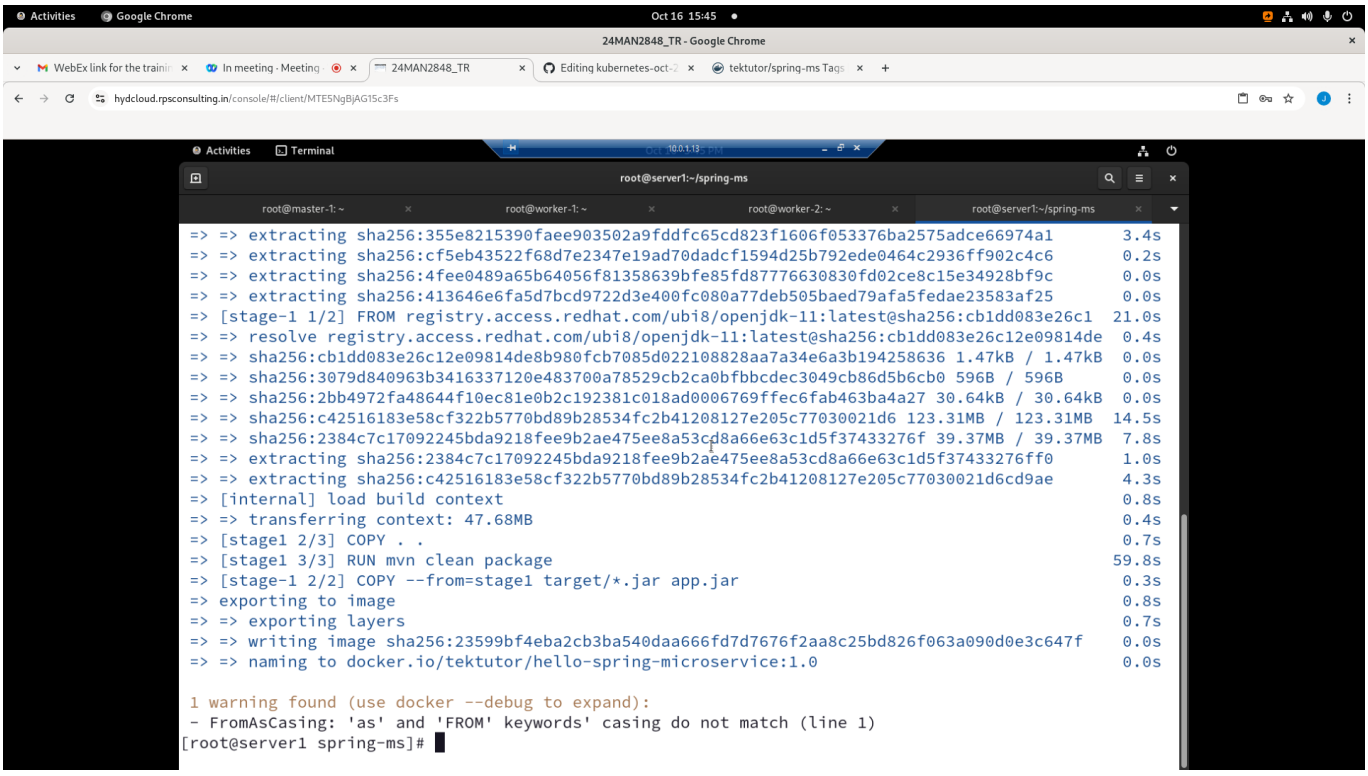
```
docker build -t tektutor/hello-spring-microservice:1.0 .
docker images
docker login
docker push tektutor/hello-spring-microservice:1.0
```

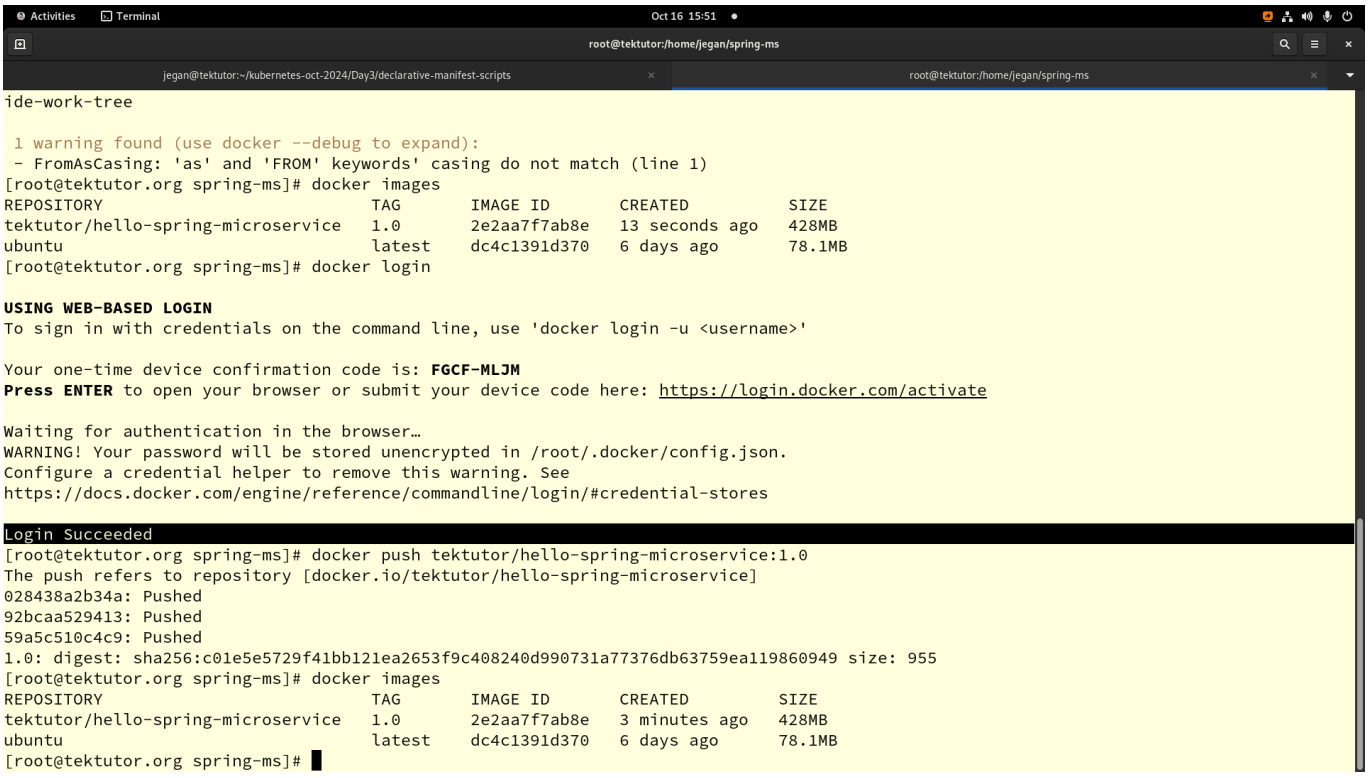
Let's deploy our custom application into K8s cluster

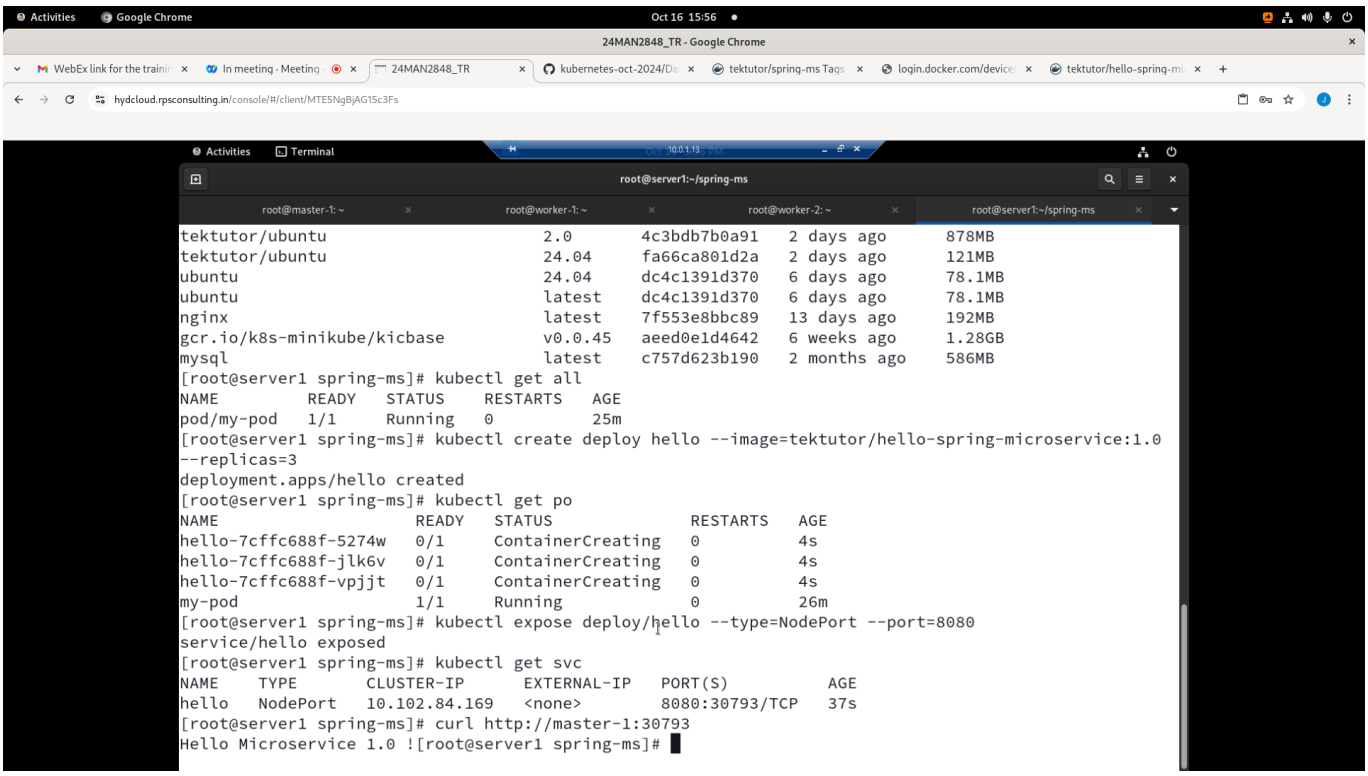
```
kubectl create deployment hello --image=tektutor/hello-spring-
microservice:1.0 ---replicas=2
kubectl get deploy,rs,po
```

Expected output









Lab - Creating a loadbalancer service in declarative style

Let's create an external loadbalancer service for hello deployment

```
kubectl expose deploy/hello --type=LoadBalancer --port=8080 -o yaml --dry-
run=client
kubectl expose deploy/hello --type=LoadBalancer --port=8080 -o yaml --dry-
run=client > hello-lb-svc.yml
kubectl apply -f hello-lb-svc.yml
kubectl get svc
```

But in order for the loadbalancer service to acquire an external IP, the kubernetes administrator has to install an operator called MetallB. The Metallb operator configures our local k8s cluster to work like it works in AWS/Azure or any public cloud.

Each time someone creates a LoadBalancer service, the metallb controller will be watching, when it detects some one creating a new loadbalancer service, it configures the metallb load balancer to route the traffic to our pods just like how it works in AWS/Azure.

Let's install the metallb operator

```
kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.14.8/config/manifests/
metallb-native.yaml
```

Lab - Let's add a new type of resource to K8s cluster

Create a file named training-crd.yml with the below content

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: trainings.tektutor.org
spec:
  group: tektutor.org
  scope: Namespaced
  names:
    kind: Training
    listKind: TrainingList
    plural: trainings
    singular: training
    shortNames:
    - train
  version:
  - name: v1
    served: true
    storage: true
    schema:
      openAPIV3Schema:
        type: object
        properties:
          training:
            type: string
          duration:
            type: string
          from:
            type: string
          to:
            type: string
```

Let's create a training resource

```
apiVersion: tektutor.org/v1
kind: Training
metadata:
  name: devops-training
spec:
  training: "Advanced DevOps"
  duration: "5 Days"
  from: "4th Nov 2023"
  to: "8th Nov 2023"
```

Info - Kubernetes Operator Overview

```
- Kubernetes Operator helps us extend the Kubernetes API or used to add new
functionality to Kubernetes
- Operator is a combination of one or more Controllers and Custom Resources
```

Demo - Configure the metallb operator

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: default
  namespace: metallb-system
spec:
  ipAddressPools:
  - tektutor-metallb-addresspool
```

Lab - Scale up nginx deployment

```
kubectl create namespace jegan
kubectl create deployment nginx --image=nginx:latest --replicas=3
kubectl get po
kubectl scale deploy/nginx --replicas=5
kubectl get po
```

Expected output

Lab - Scale down nginx deployment

```
kubectl get po
kubectl scale deploy/nginx --replicas=3
```

```
kubectl get po
```

Expected output

Lab - Rolling update

Let's create a namespace and deploy nginx v1.8

```
kubectl create namespace jegan
kubectl config set-context --current --namespace=jegan
kubeclt create deployment nginx --image=nginx:1.18 --replicas=3
kubectl get po -o yaml | grep image
kubectl get rs
kubectl get deploy
```

Let's perform rolling update (upgrade nginx image from 1.18 to 1.19)

```
kubectl set image deploy/nginx nginx=nginx:1.19
```

Let's observe if 2 replicasets are created under nginx deployment

```
kubectl get rs
```

Let's check the pod

```
kubectl get po
```

Let's check the status of the rolling update

```
kubectl rollout status deploy/nginx
```

Let's check the image used by nginx pods after rolling update completed successfully

```
kubectl get po -o yaml | grep image
```

Rolling back to previous version of image

```

```

```
kubectl rollout undo deploy/nginx
kubectl rollout status deploy/nginx
kubectl get po -o yaml | grep image
```

Info - Ingress

```
- a routing rule
- for an Ingress to work, basically 3 things are required
  1. Ingress rule ( we will be writing this a yaml file )
  2. Ingress Controller (Nginx Controller or HAProxy Ingress Controller)
  3. Either Nginx Load Balancer or HAProxy Load Balancer
- For instance,
  - We have a bank website with login, balance enquiry, fundtransfer,
cheque request, logout, etc.,
  - assume we have develped each of the above features as a microservice,
hence each one will be a separate deployment
  - Using ingress we will get a public url, so base path we can route the
traffic to different microservices
  - E.g
    - www.somebank.com - home page
    - www.somebank.com/login - this should be forwarded to the login
microservice K8s service
    - www.somebank.com/fundtransfer - this should be forwarded to the
fundtransfer microservice K8s service
```

Info - ReplicationController vs ReplicaSet

```
- In older version of Kubernetes, the only way we could deploy stateless
application is via ReplicationController
- The ReplicationController supports both Rolling Update and Scale up/down
- One Controller does two things, which violates Single Responsibility
Principle ( SOLID - SRP Principle )
- In latest version of kubernetes, they refactored(broken down)
ReplicationController functionality into Deployment and ReplicaSet
- The Deployment supports rolling update to stateless applications, while
the ReplicaSet supports scale up/down
- ReplicationController is still supported for backward compatility and
legacy application
- We should strictly avoid using ReplicationController for deploying new
application
```

Info - Persistent Volume (PV)

```
- is a external storage that can be used by the applications runing within
Pod
```

```
- this can be provisioned by Administrator either manually or dynamically
- created on the cluster scope, which any pod running in any project
namespace can claim and use PV
- In case the PV is manually provisioned, the administrator will have
create Persistent volume
  - with a specific size capacity
  - with specific access modes
    - ReadWriteOnce
    - ReadWriteMany
    - etc
  - StorageClass(optional)
  - Labels ( optional )
```

Info - Persistent Volume Claims (PVC)

```
- is the way your application can request for external storage
- PV will have to define
  - the size of the storage required
  - storageclass(optional)
  - access mode
  - labels (optional)
```

Info - What is Ingress?

```
- routing/forwarding rules
- Ingress helps in forwarding the calls to multiple different services
pointing to different deployments
- Ingress is not a service
- We can declaratively create ingress rules, which are retreived by Ingress
Controller, which then configures the load balancer with the forwarding
rules we listing in the ingress
- For Ingress to work, we need the below
  - Ingress ( rules )
  - Ingress Controller
  - Load Balancer
```

Info - What is Ingress Controller?

```
- Ingress Controller is Controller like Deployment Controller, ReplicaSet
Controller
- Ingress Controller keeps an eye on every new Ingress created in any
project namespace
- Ingress Controller monitors any change done to existing Ingress resources
under any project namespace
- Ingress Controller also will monitor when Ingress is deleted in any
project namespace
```

```
- Ingress Controller picks the rules we mentioned in the Ingress resource
and configures the load balancer accordingly
- There are two popular ingress controllers
  - Nginx Ingress Controller
  - HAProxy Ingress Controller
- In our lab setup, we are using HAProxy Load Balancer, hence we need to
use HAProxy Ingress Controller
```

Day4

Docker Network Model

```
- Each container gets an unique private IP on the system it runs
- Containers can be connected to Docker Network
- Docker creates a default bridge network called docker0 with subnet
172.17.0.0/16 ( 65535 IP addresses )
- Containers get their IP address from the subnet range assigned to the
Docker network
- Containers running on same network can communicate with each other
directly
- a single container can be connected to multiple networks, which means
they may get multiple IP addresses
```

Subnet

```
- 172.17.0.0/16
- IPV4 IP address
- 172 - 1 byte
- 17 - 1 byte
- 0 - 1 byte
- 0 - 1 byte
- IPv4 - 4 bytes ( 32 bits )
- CIDR - Classless Inter Domain Routing
- First IP in 172.17.0.0/16 - 172.17.0.0
- 172.17.0.1
- 172.17.0.255
- 172.17.1.0
- 172.17.1.255
- 256 x 256 = 65535 IP addresses are there in 172.17.0.0/16 Network
```

Kubernetes Network Model

```
- Kubernetes doesn't implement Network, it only publishes its Network
requirements as Kubernetes Network specification, while third-party vendors
implements the Kubernetes Network specification, which is referred as
Kubernetes Network Model
- Each Pod should get an unique cluster wide IP address within the
Kubernetes cluster
- Pod Network
  - all pods should be able to communicate with each other whether they run
in same node or different nodes
  - kubelet should be able to communicate with the pods that runs on the
node where kubelet is running
```

```
- Service Network
  - Pods are temporary as they get removed in the process of scale up/down,
rolling update, etc.
  - application developers should not use Pod IP to access them
  - every service should get an unique cluster wide name and IP address
  - service IP once assigned to a service will not change until the service
exists
  - i.e service IP is stable
  - service represents a group of Load Balanced Pods behind them
- Ingress
  - makes services accessible outside the Kubernetes cluster
- Network Policy
  - helps controlling communication between Pods
  - helps controlling Pod incoming/outgoing traffic
```

Kubernetes Network CNI Addons

```
- the Kubernetes Network Model(Specification) is implemented by third-party
CNI plugins/addons
- Some of the Kubernetes CNI Network addons are
  - Calico
  - Weave Net
  - Flannel
  - Antrea
  - Canal
  - Cilium
  - Gateway API
  - Multus
  - Romana
  - Spiderpool
  - Nuage
  - NST-T
  - Knitter
  - OVN-Kubernetes
  - Nodus
  - Contrail
  - Contiv
  - ACI
- Popular addons most commonly used by many companies
  - Flannel
  - Calico
  - WeaveNet
```

Flannel

You may find this article interesting

```
https://mvallim.github.io/kubernetes-under-the-hood/documentation/kube-
flannel.html
```

Let's understand flannel briefly

```
- one of the oldest and most mature CNI plugins available
- is a simple, lightweight layer 3 fabric for Kubernetes
- uses Overlay Network
- developed by CoreOS
- operates on Layer 3 of the OSI model and uses the VX-LAN as its default
backend to move network packets between nodes
- provides access to basic networking features and requires limited amount
of administration to set up and maintain
- supports a variety of backends like VX-LAN, host-gateway, AWS VPC,
AliVPC, IPIP, and IPSec etc.,
- overlay network is a network that is layered on top of another network
- overlay network can be used to handle pod-to-pod traffic between nodes
- Overlay networks work by encapsulating network packets
- when a pod initiates a connection to an IP address outside of the
cluster, the node hosting the pod will use SNAT (Source Network Address
Translation) to map the source address of the packet from the pod IP to the
node IP
- is a great entry level choice for Kubernetes cluster networking
- drawbacks
  - doesn't support Network Policy
  - as each packets are encapsulated by sender and de-encapsulated by the
receiver it impacts overvall network performance negatively
- when flannel CNI is installed in Kubernetes, one Flannel Pod gets created
on each Kubernetes node
- flannel either uses Kubernetes API to store network configurations in
etcd or gets direct write access to etcd database just like API Server
- when kubernetes master node is bootstrapped(installed), we need to assign
a Pod network subnet
- eg: kubeadm init --pod-network-cidr=10.244.0.0/16
- in the above example 10.244.0.0/16 ( 65535 IP addresses are allocated for
Pods created in the K8s cluster )
- the above subnet is sub-divided into small subnets(IP ranges) and
allocated to every node by Flannel
  - For example
    - Master 1 - 10.244.1.0/24 ( Upto 256 IP addresses, as Nodes by default
can support only 110 Pods this is more than enough )
    - Master 2 - 10.244.2.0/24
    - Master 3 - 10.244.3.0/24
    - Worker 1 - 10.244.4.0/24
    - Worker 2 - 10.244.5.0/24
    - Worker 3 - 10.244.6.0/24
- the subnet for every node is
```

Calico Overview

```
- Calico
  - implemented by company called Tigera
  - comes in 2 flavours
    - opensource and
    - enterprise
  - most popular and commonly used in Kubernetes/Openshift CNI
  - provides both Network and Network Policy
  - operates on Layer 3 of the OSI model and uses the BGP(Border Gateway
Protocol) protocol to move network packets between nodes
  - BGP is one of the fundamental building blocks of the internet, with
exceptional scaling characteristics
  - Using BGP, Calico directs packets natively, without needing to wrap
them in additional layers of encapsulation
```

Weave Net Overview

```
- is a flexible networking solution for Kubernetes/Openshift clusters
- developed by a company called WeaveWorks
- Weave comes in 2 flavours
  - opensource and paid
- weave routes packets using fast datapath method
- weave routes packets uses a slower network method called sleeve packet
forward when fast datapath fails
- is easy to install and configure
- creates a mesh overlay network to connect all the nodes in the cluster
- Weave is a good choice for organizations that need a flexible and
scalable networking solution for their Kubernetes/Openshift clusters
```

Day5

Lab connection details

```
From the RPS cloud windows machine you need to open Remote Desktop
connection and type

Server 1 ( 10.0.1.13 )
- user01 to user08
- user17 to user20

Server 3 ( 10.0.1.25 )
- user09 to user16
- user21 to user24
```

Red Hat Openshift Overview

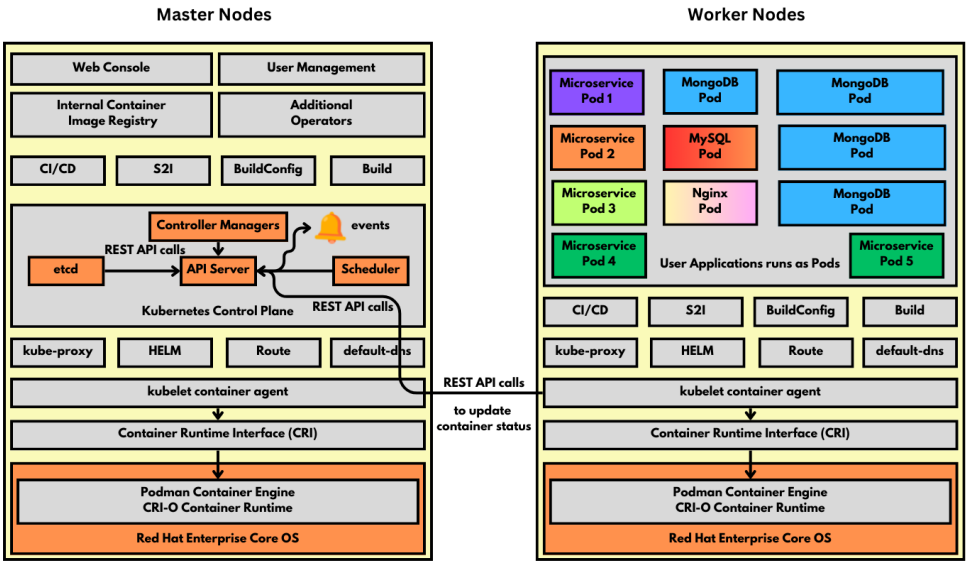
```
- is Red Hat's Kubernetes distribution
- developed on top of opensource Google Kubernetes with many additional
features
- it is a superset of Kubernetes, hence all features of Kubernetes are also
supported in Openshift
- Red Hat Openshift supports only RHCOS in master nodes and either
RHCOS/RHEL in worker nodes
- supports only CRI-O Container runtime and Podman Container engine
- enterprise product that requires commercial license
- supports many additional features
  - Web console
  - Internal Openshift Image Registry
  - Source to Image (S2I)
    - deploying application from source code
    - deploying application using Dockerfile
  - supports CI/CD
  - supports routes to expose application for external access
  - supports user management
- AWS supports managed Red Hat Openshift cluster called ROSA
  - Load Balancer creates an external Load Balancer supported by AWS

- Azure supports managed Red Hat Openshift cluster called ARO
  - Load Balancer creates an external Load Balancer supported by Azure

- As Red Hat Openshift makes use of Red Hat Enterprise Core OS, it is
secure already
  - Ports below 1024 are not allowed as it is reserved for internal use
  - not all applications can be deployed with root access
  - it can be made more secure by using network policy like we do in
Kubernetes
```

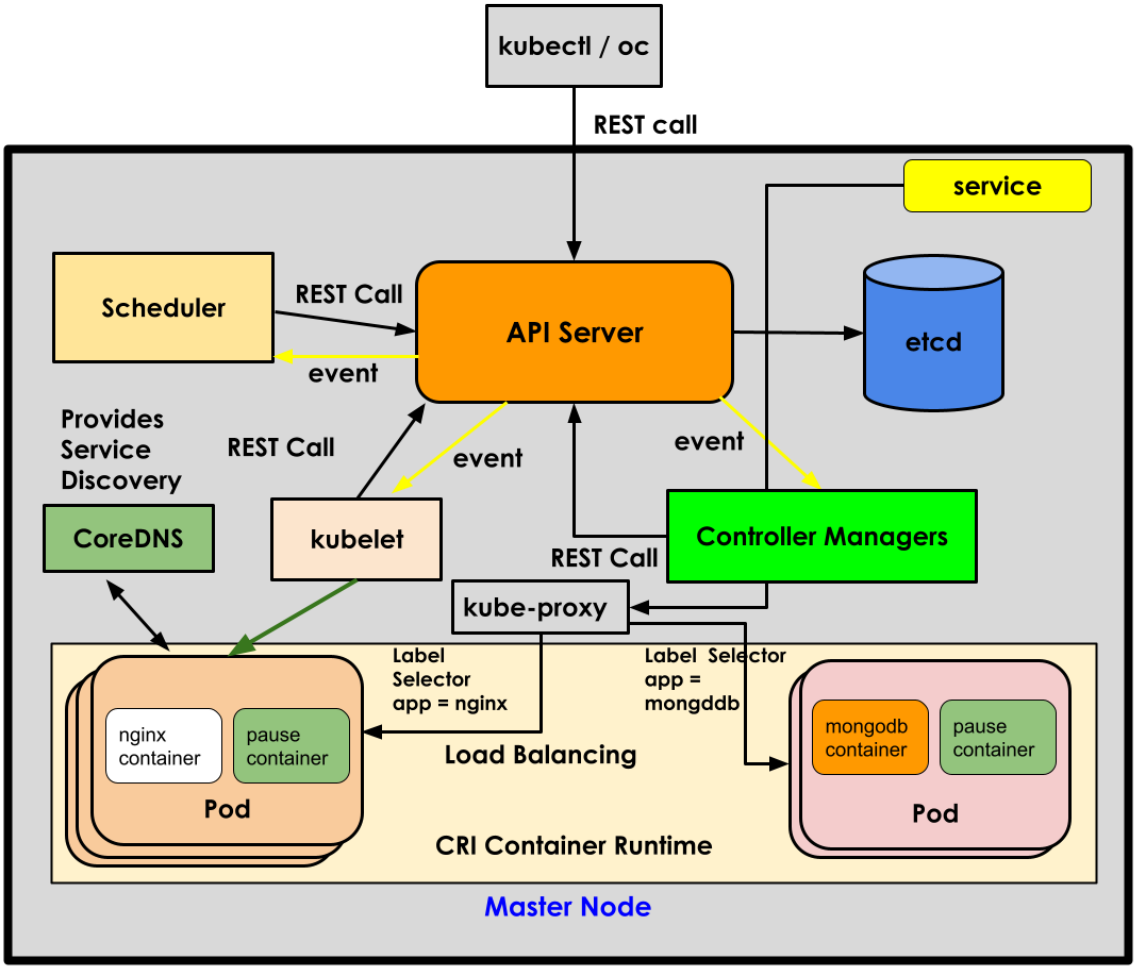
```
  - it will enforce best practices are followed which are not taken so
seriously
```

Info - Openshift onPrem vs Cloud

```
- onPrem Openshift
  - installation of openshift we need to take care
  - Red Hat Openshift license are taken care by us
  - backup is our responsibility
  - we need to decide the master/worker node hardware configuration as per
our application workload and user traffic
  - add new nodes into cluster is done manually using Openstack, VMWare
vSphere, etc.,
  - Metallb operator or similar operators must be used to support
LoadBalancer service in a on-prem openshift setup like our lab setup

- AWS ROSA
  - installation of openshift is taken care by AWS
  - Red Hat Openshift license is taken care by AWS
  - Hardware configuration of master nodes are decided and managed by AWS
  - backup of etcd database is taken care by AWS
  - LoadBalancer service when created it will automatically create AWS
ALB/ELB as it is tighly integrated with AWS
```

Red Hat Openshift High-Level Architecture





Kube config file

- the oc/kubectl client tools requires a config file that has connection details to the API Server(load balancer)

- the config file is generally kept in user home directory, .kube folder and the default name of kubeconfig is config
- optionally we could also use the --kubeconfig flag with the oc command to point to a config file
- it is also possible to use a KUBECONFIG environment variable to point to the config file
- Just to give an idea, it is possible that your Kubernetes/OpenShift is running in AWS/Azure but you could install oc/kubectl client tool on your laptop with a config file and still run all the oc/kubectl commands from your laptop without going to aws/azure

To print the content of kubeconfig file

```
cat ~/.kube/config
```

About Red Hat Enterpise Core OS (RHCOS)

- an optimized operating system created especially for the use of Container Orchestration Platforms
- each version of RHCOS comes with a specific version of Podman Container Engine and CRI-O Container Runtime
- RHCOS enforces many best practices and security features
- it allows writing to only folders the application will has read/write access
- if an application attempts to modify a read-only folder RHCOS will not allow those applications to continue running
- RHCOS also reserves many Ports for the internal use of Openshift
- User applications will not have write access to certain reserved folders, user applications are allowed to perform things as non-admin users only, only certain special applications will have admin/root access

Points to remember

- Red Hat Openshift uses RedHat Enterprise Linux Core OS
- RHCOS has many restrictions or insists best practises
- RHEL Core OS reserves ports under 1024 for its internal use
- Many folders within the OS is made as ready only
- Any application Pod attempts to perform write operation on those restricted folders will not be allowed to run
- For detailed documentation, please refer official documentation here https://docs.openshift.com/container-platform/4.8/architecture/architecture-rhcos.html

Info - Pod Lifecycle

- Pending - Container image gets downloads or there are no Persistent Volume to bound and claim them
- Running - The Pod is scheduled to a node and all containers in the Pod are up and running
- Succeeded - All containers in the Pod have terminated succesfully and not be restarted
- Failed - All containers in the Pod have terminated but one or more containers terminated with non-zero status or was terminated by Openshift

- Unknown - For some reason, the state of the Pod could not be obtained may be there is some problem in communicating to the node where the Pod is running

Info - Container Lifecycle

- Waiting - pulling the container image
- Running - container is running without issues
- Terminated - container in the Terminated state began execution and then either ran to completion or failed for some reason

Info - NodePort vs Route

```
- NodePort is an external service
- It is a K8s features, which is also supported in openshift
- Kubernetes/Openshift reserve ports in range 30000-32767 for the purpose
of NodePorts
- For each, NodePort service we create one of the ports from the above
range will be alloted for the service
- the chose nodeport is opened in all the nodes for the nodeport service
- if we create 100 nodeport services, we end up opening 100 firewall ports
on all the nodes, which is a security concern
- also nodeport service is not end-user friendly or developer friendly as
they are accessed via node hostname/ip address, ideally the end-user should
not have worry about about how many nodes are part of openshift
- routes is based on Kubernetes ingress, which provides an easy to access
public url which is user-friendly as opposed to nodeport service
- hence, in openshift for internal service, we can create clusterip service
- for external access, we just need to expose the clusterip service as a
route
- we don't have use node-port service in openshift
```

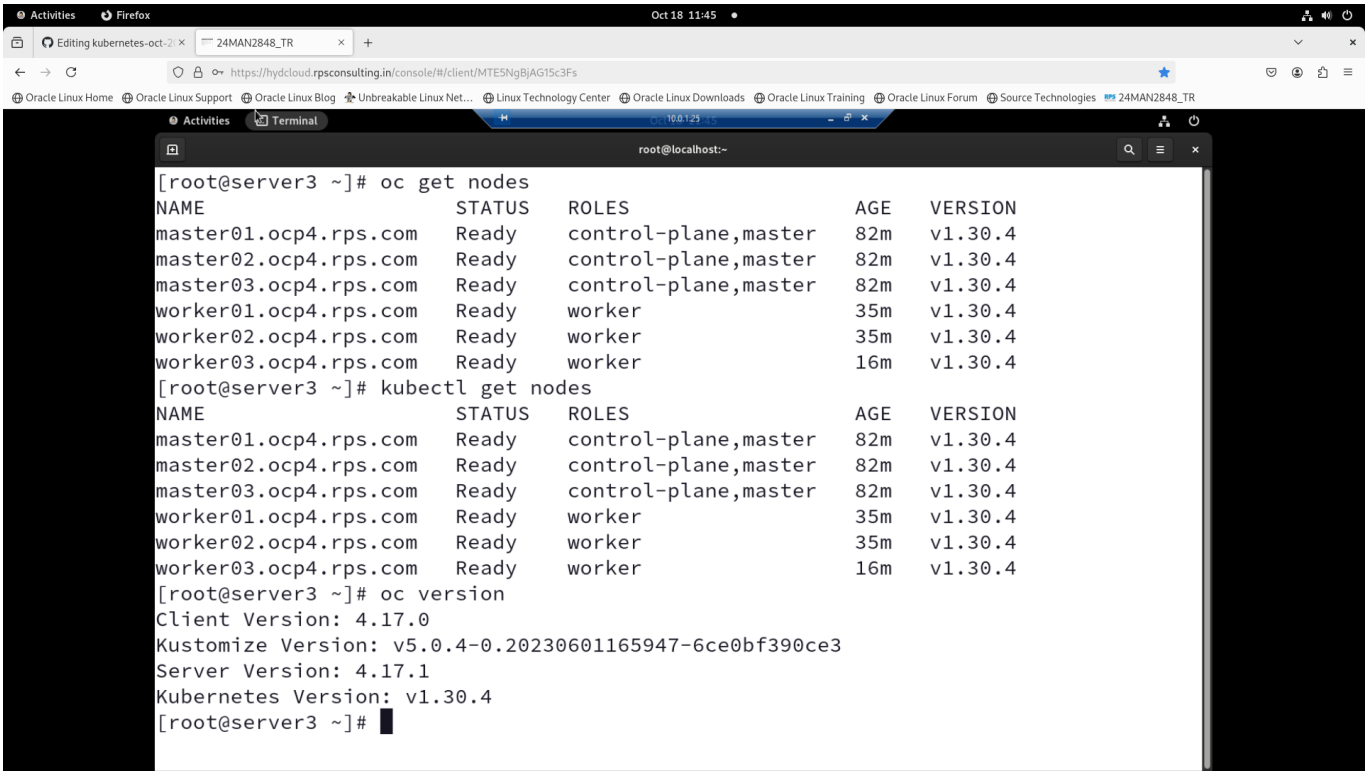
Info - Deployment vs DeploymentConfigs

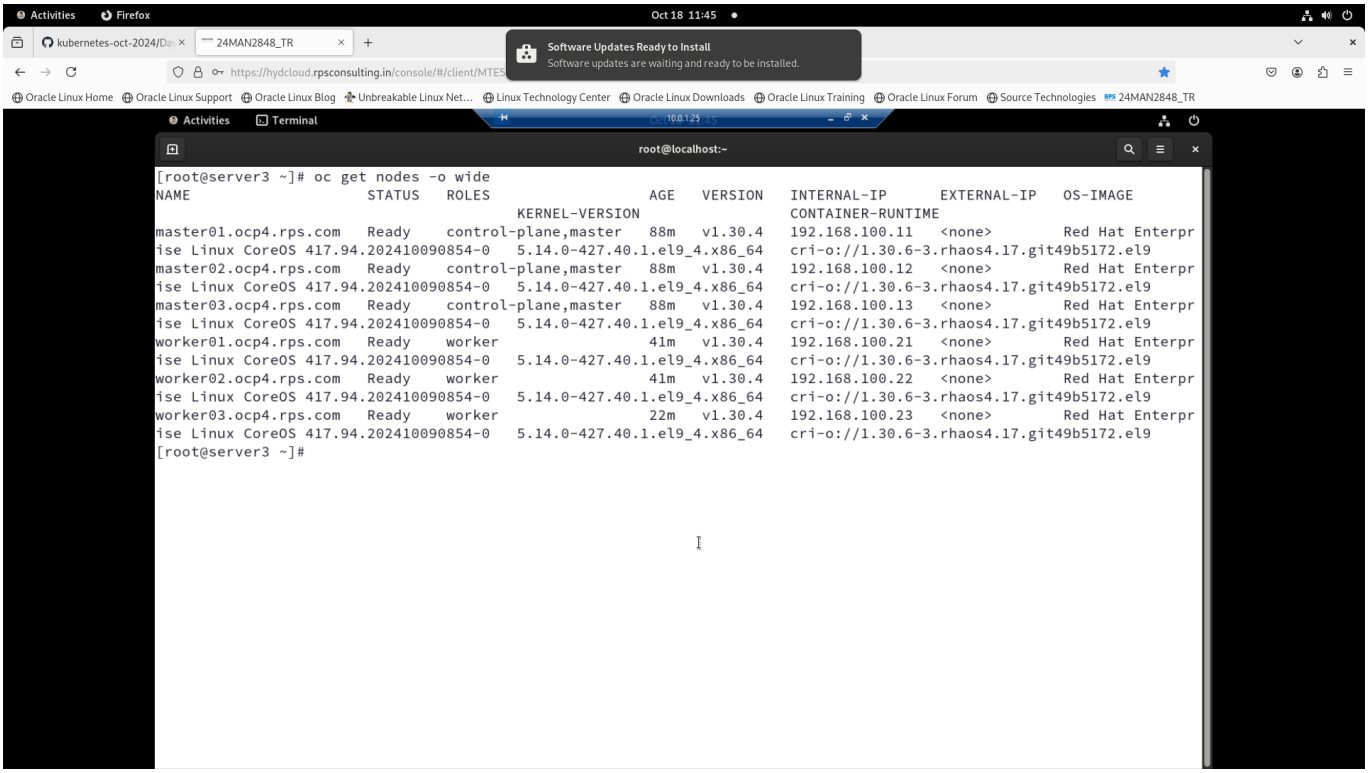
```
- In older version of Kubernetes, we had to use ReplicationController to
deploy applications into Kubernetes/Openshift
- The Red Hat openshift team, at that time added DeploymentConfig to allow
deploying application in the declarative style as the ReplicationController
doesn't support deploying application in the declarative style
- Meanwhile, the Google Kubernetes team & community added Deployment and
ReplicaSet resource as an alternate for ReplicationController
- Hence, in Openshift the Red Hat team deprecated the use of
DeploymentConfig as Deployment and DeploymentConfig pretty does the same
- Kubernetes, deprecated the use of ReplicationController
- Hence, whenever we deploy new appplication we need choose Deployment over
the DeploymentConfig as DeploymentConfig internally uses
ReplicationController
```

Lab - List the openshift nodes

```
oc get nodes
kubectl get nodes
oc get nodes -o wide
kubectl get nodes -o wide
oc version
kubectl version
```

Expected output

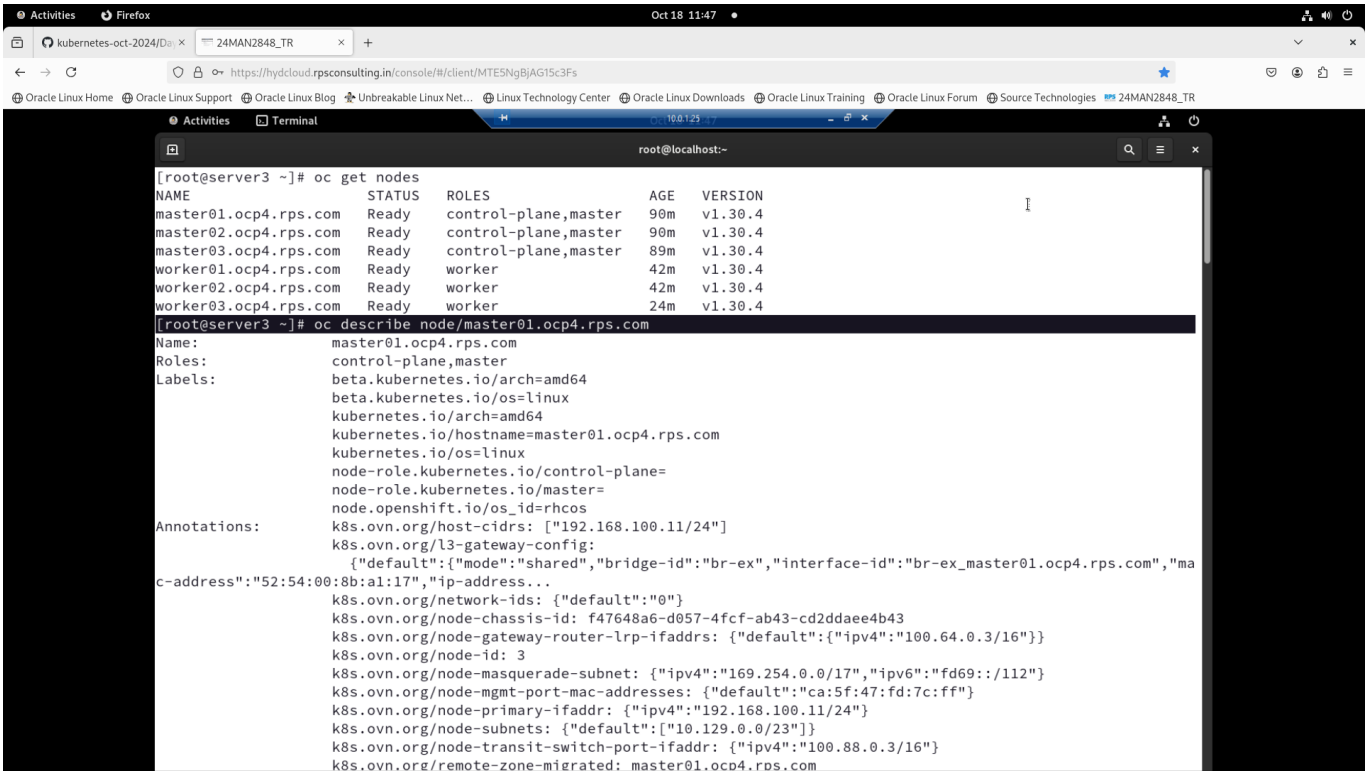


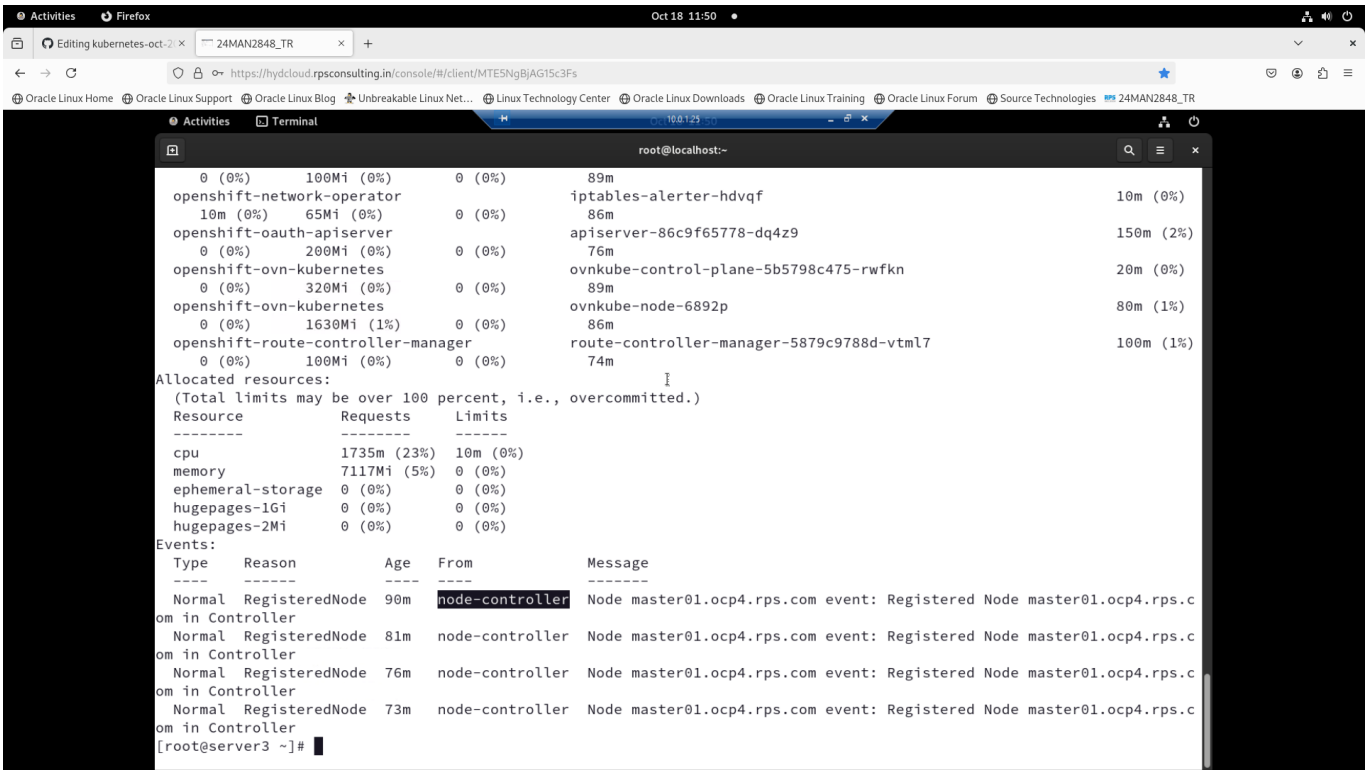


Lab - Finding more details about an openshift node

```
oc describe node/master01.ocp4.rps.com
```

Expected output



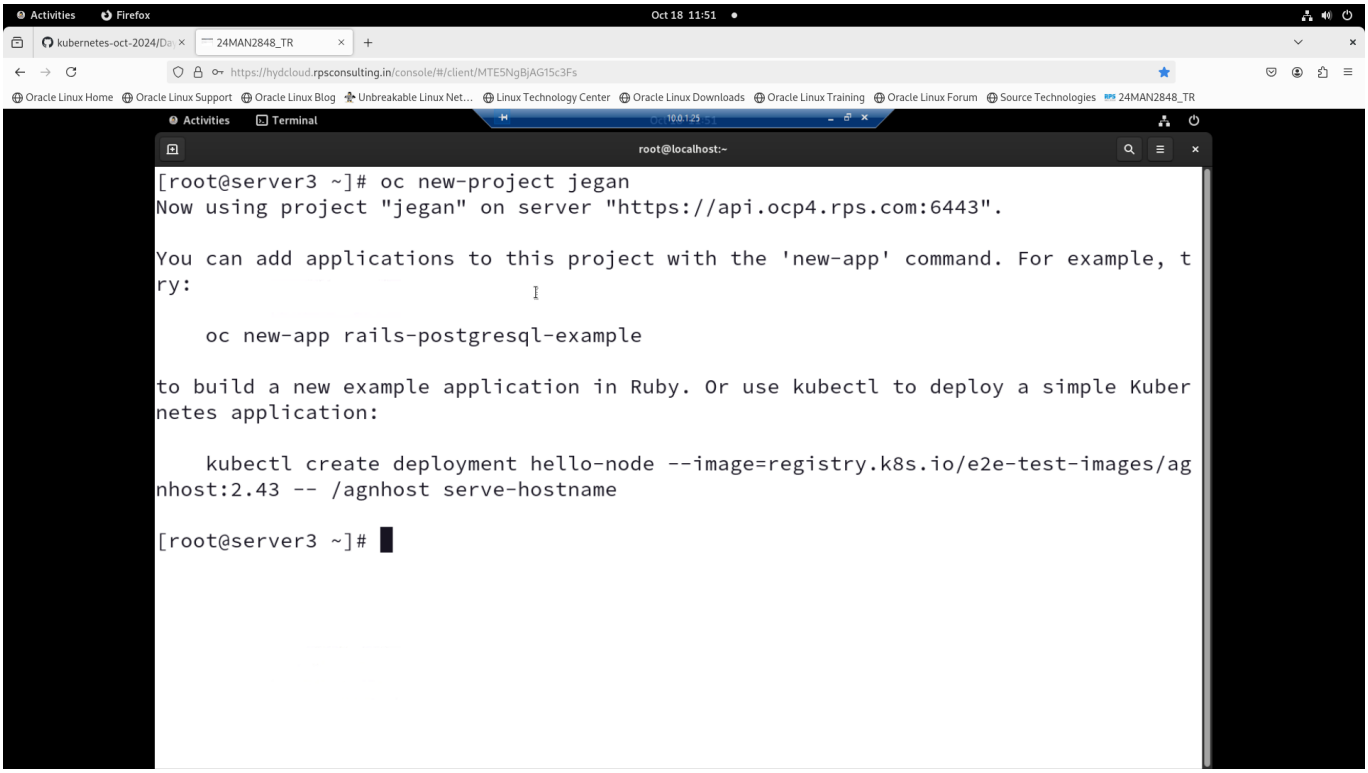


Lab - Create a new project

The below command will create a new project and switches to the project

```
oc new-project jegan
```

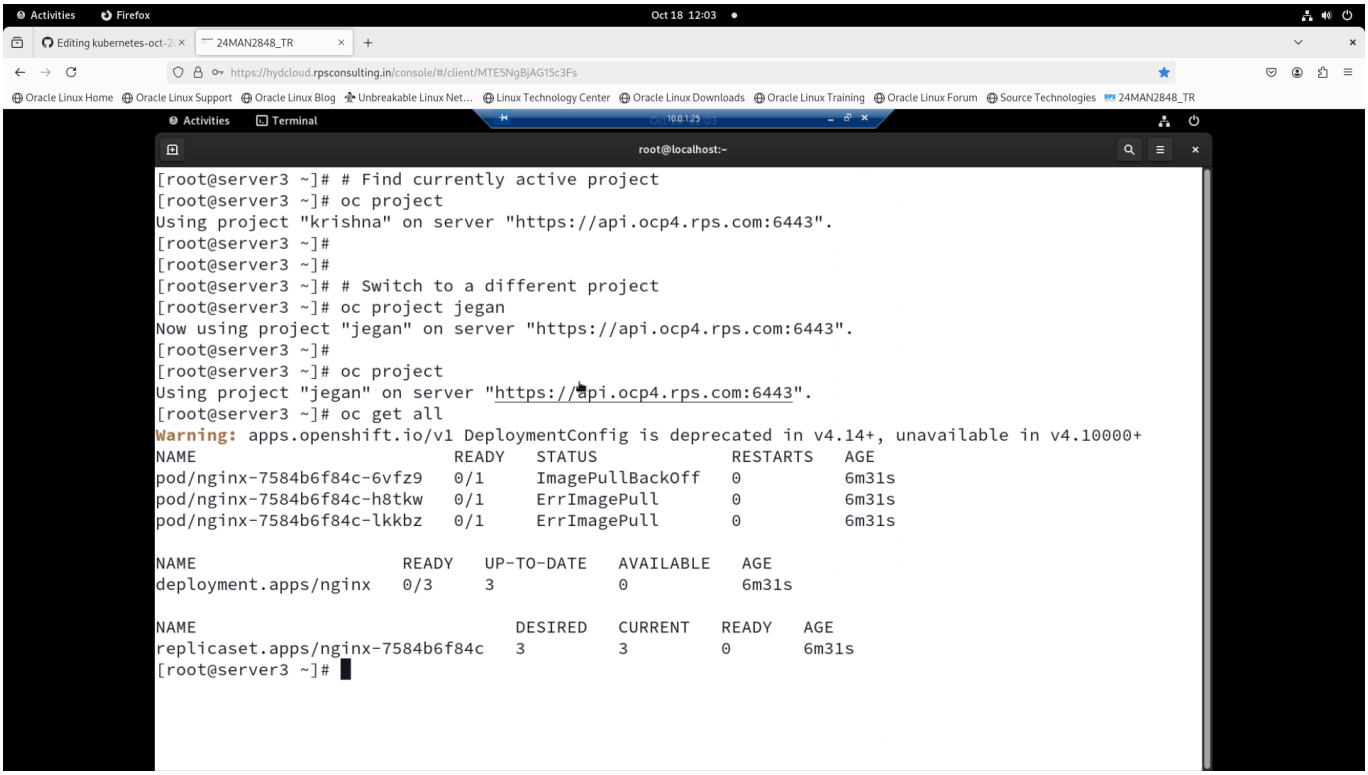
Expected output



Lab - Let's create a nginx deployment

```
oc project jegan
oc create deploy nginx --image=nginx:latest --replicas=3
oc get deploy,rs,po
```

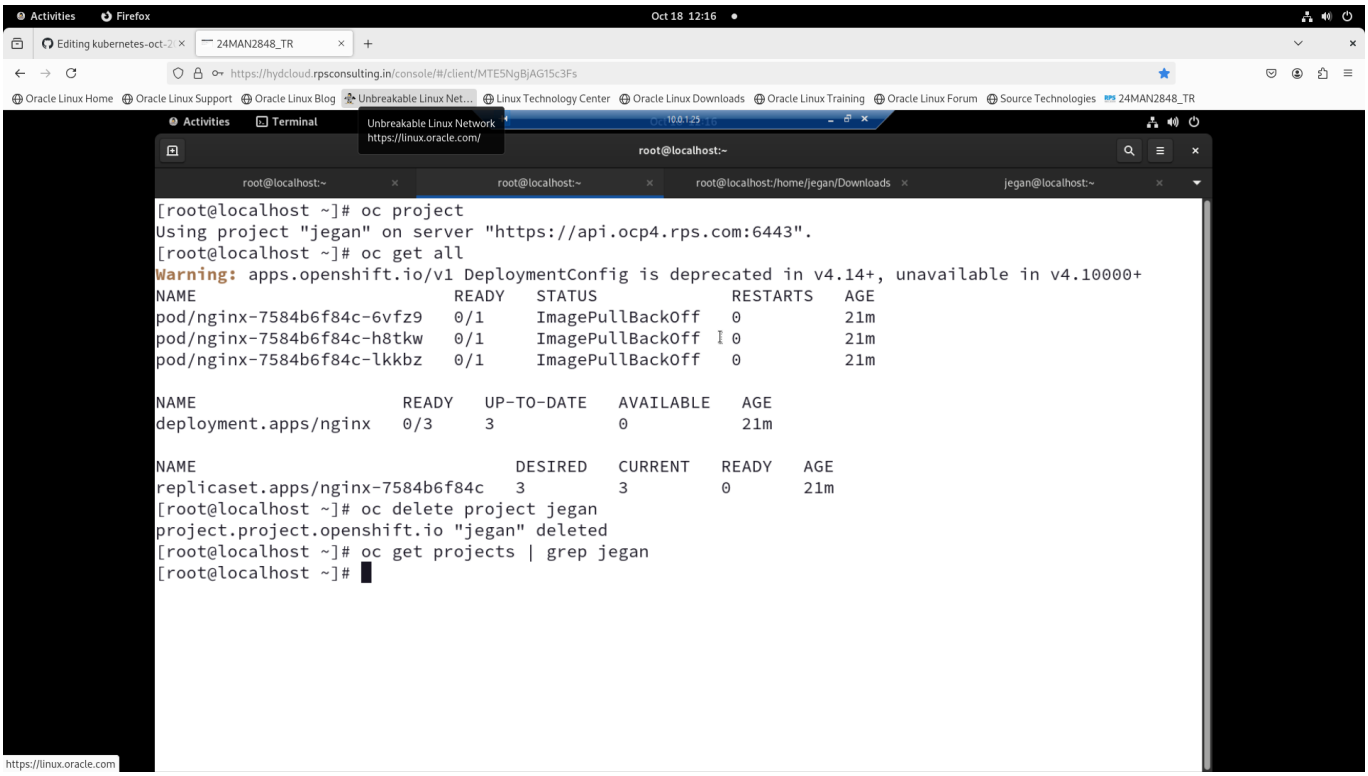
Expected output



Lab - Deleting a project along with all resources in it

```
oc project
oc get all
oc delete project jegan
oc get projects | grep jegan
```

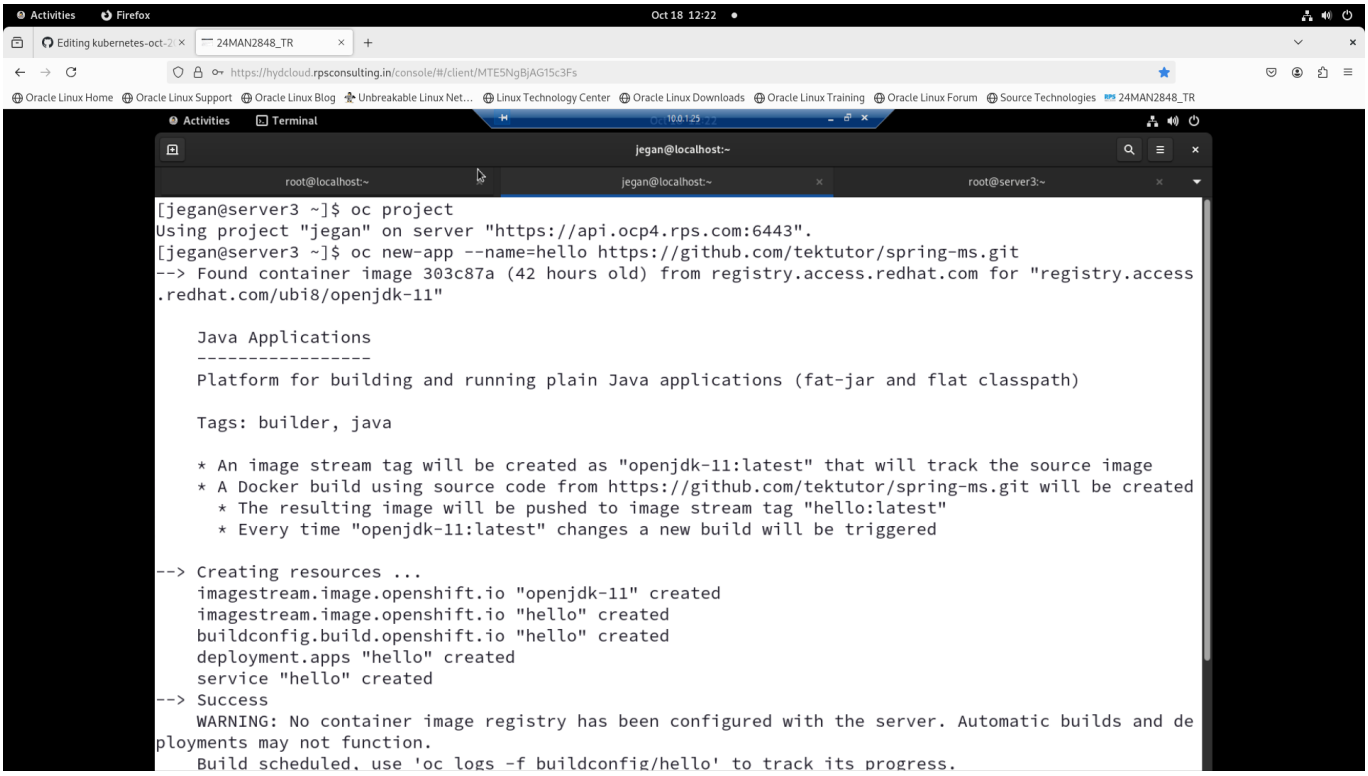
Expected output

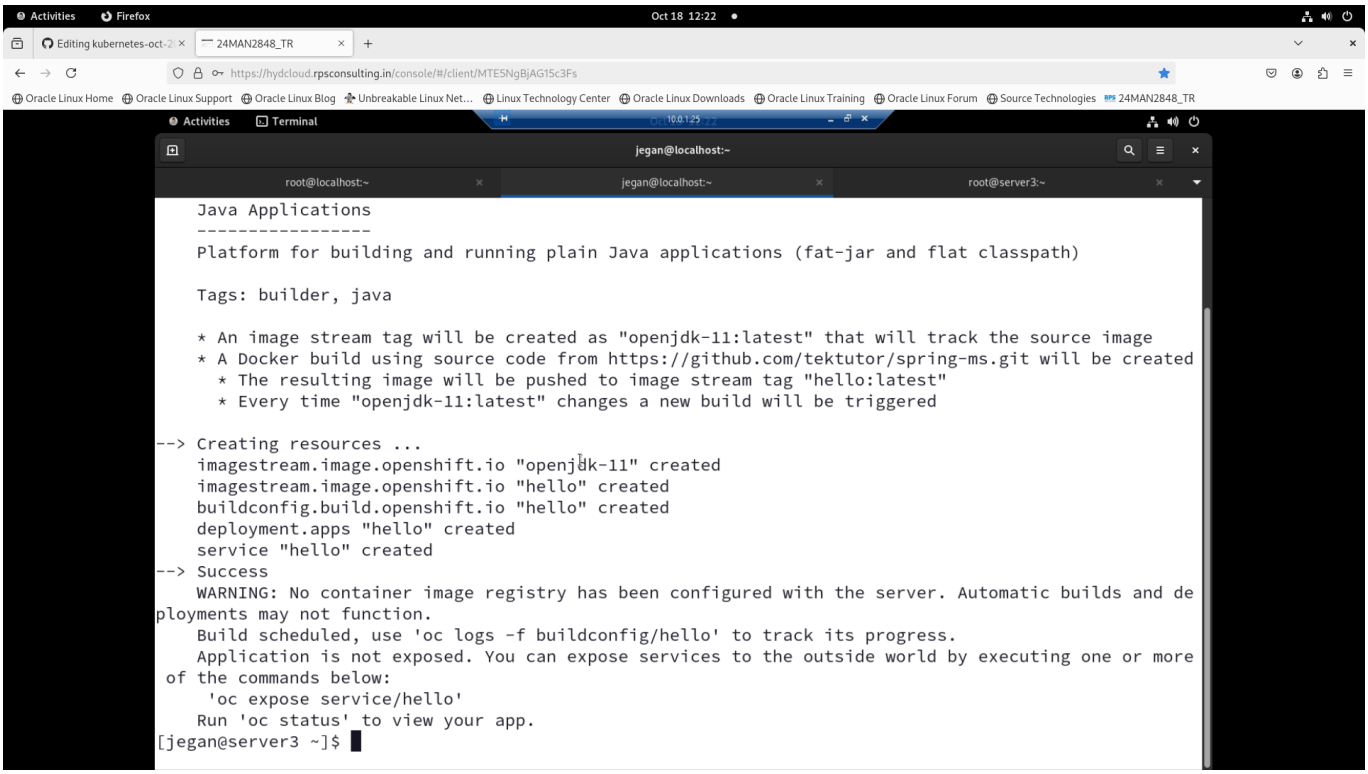


Lab - Deploying an application in openshift S2I(source to Image)

```
oc project jegan
oc new-app --name=hello https://github.com/tektutor/spring-ms.git
```

Expected output

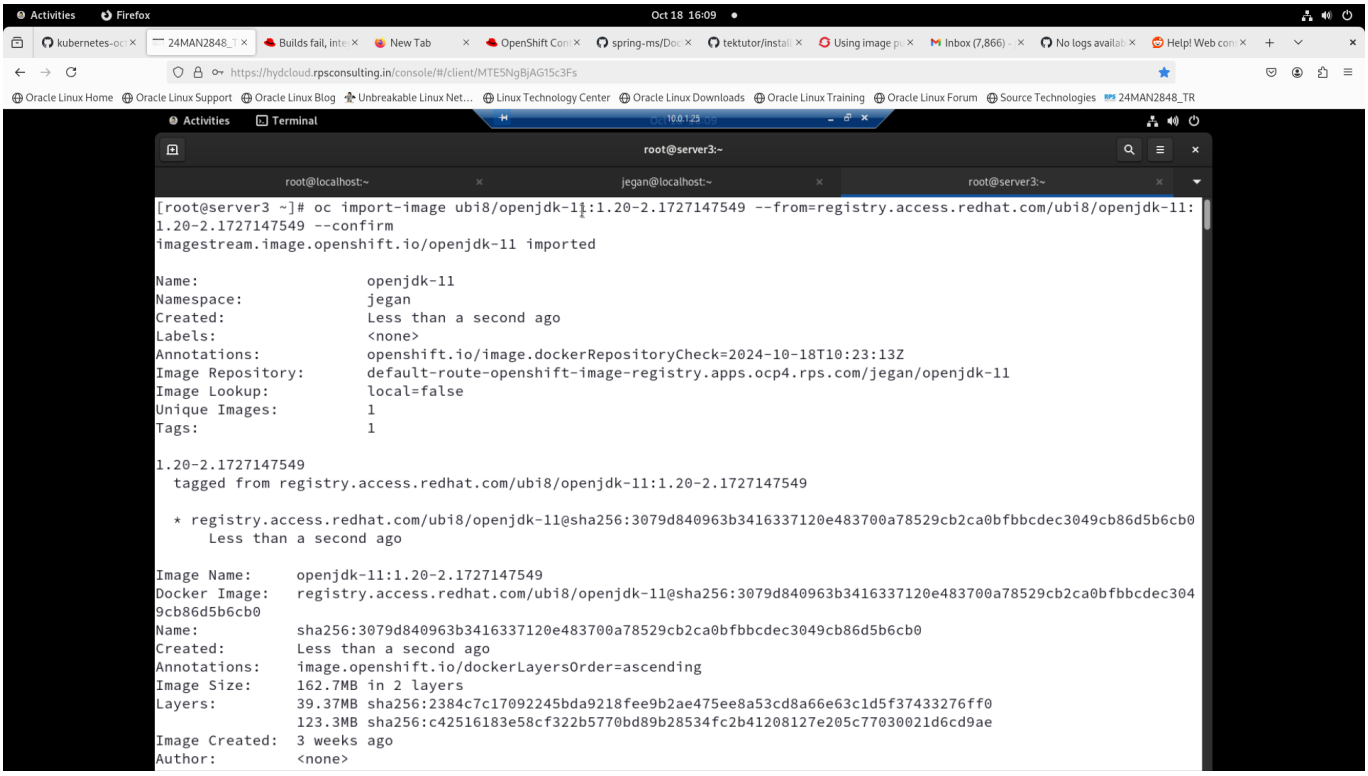


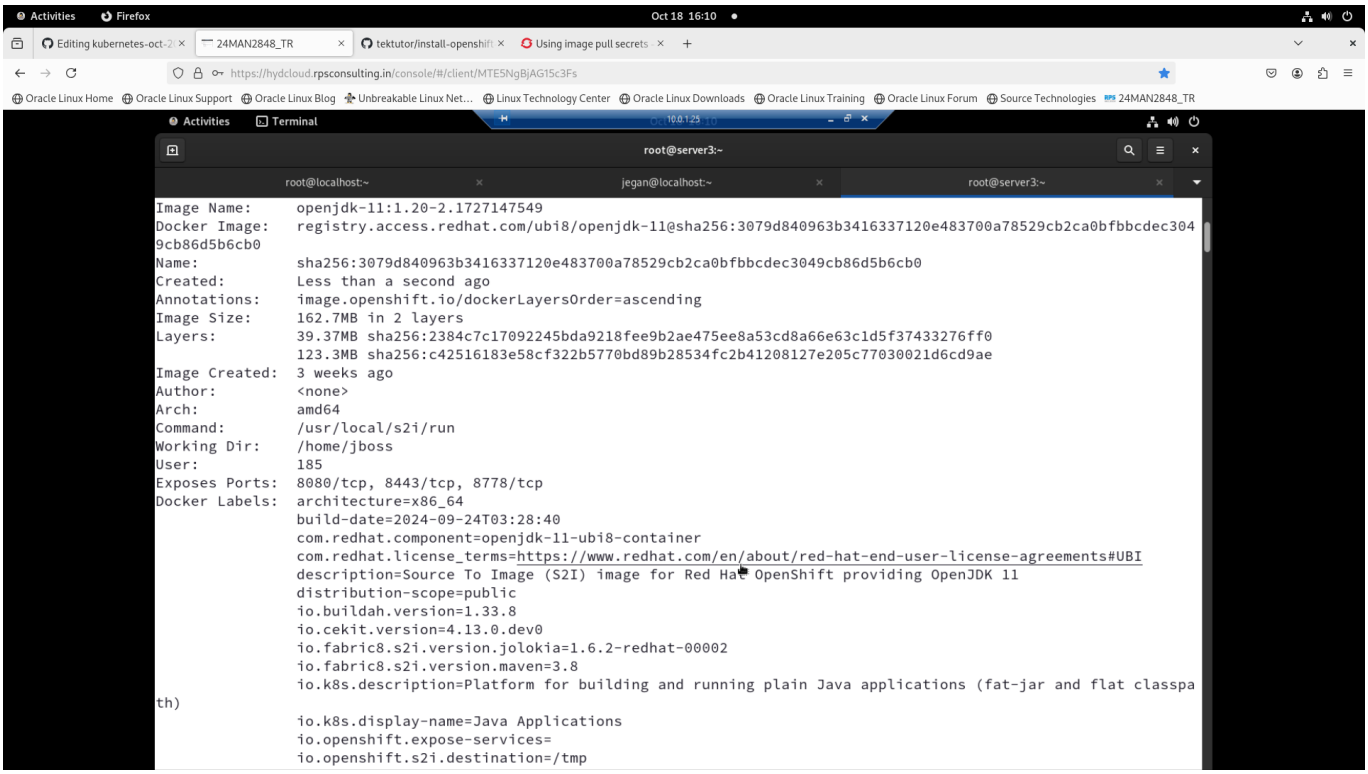


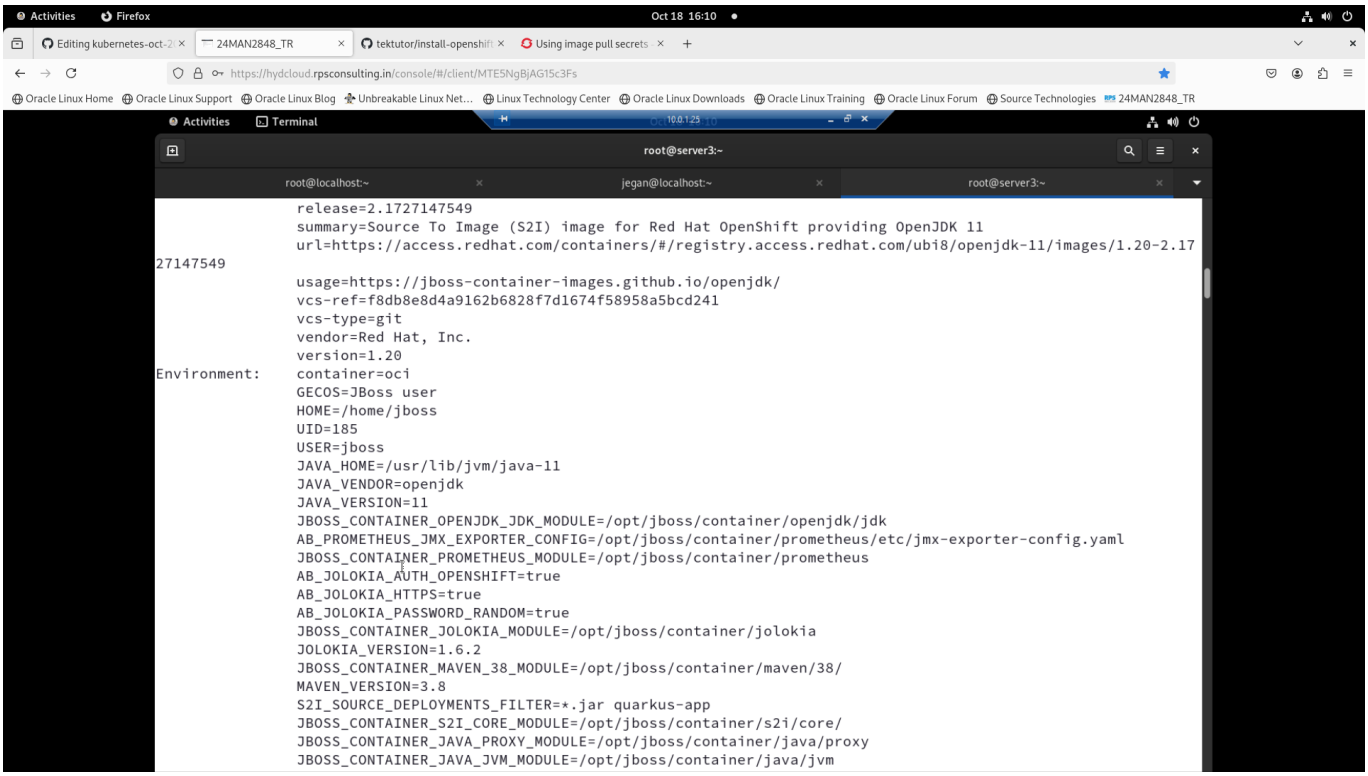
Lab - Importing image into Openshift Internal registry

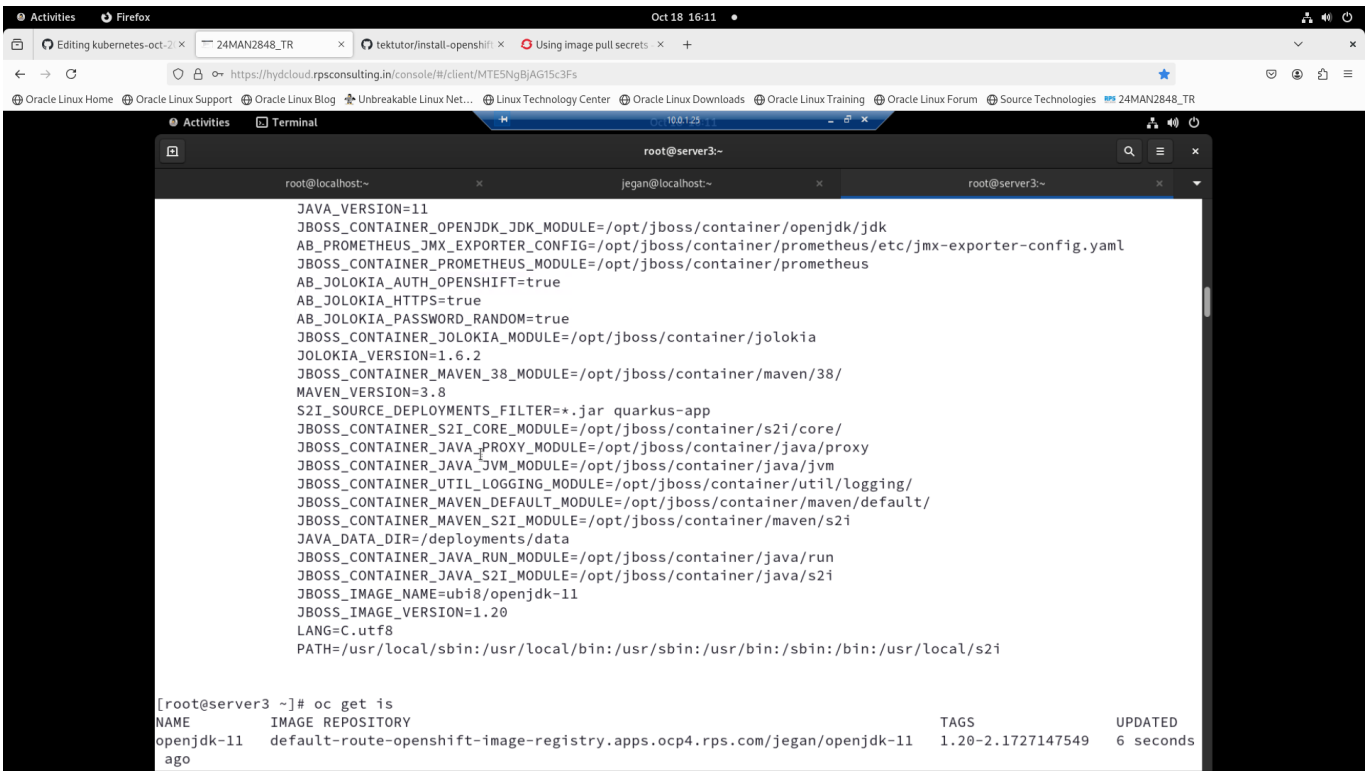
```
oc import-image ubi8/openjdk-11:1.20-2.1727147549 --
from=registry.access.redhat.com/ubi8/openjdk-11:1.20-2.1727147549 --confirm
```

Expected output





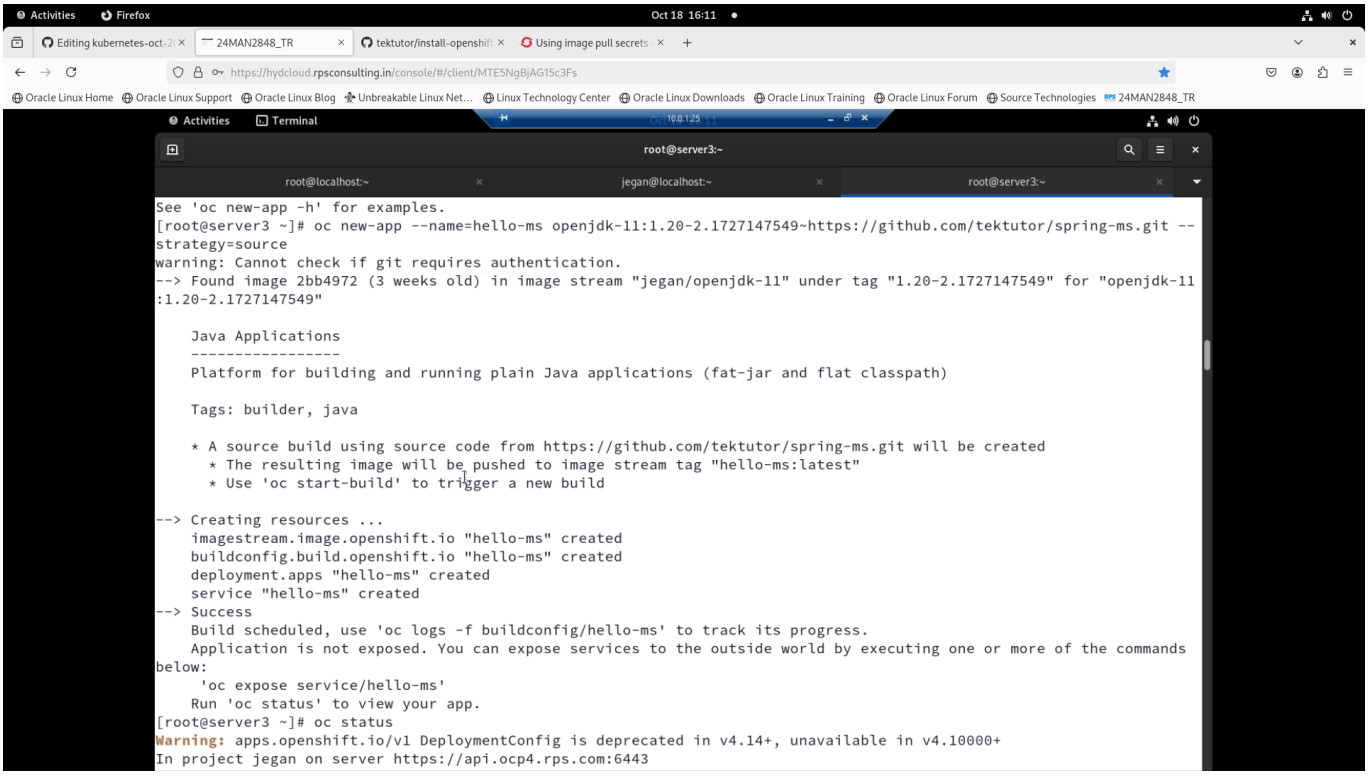


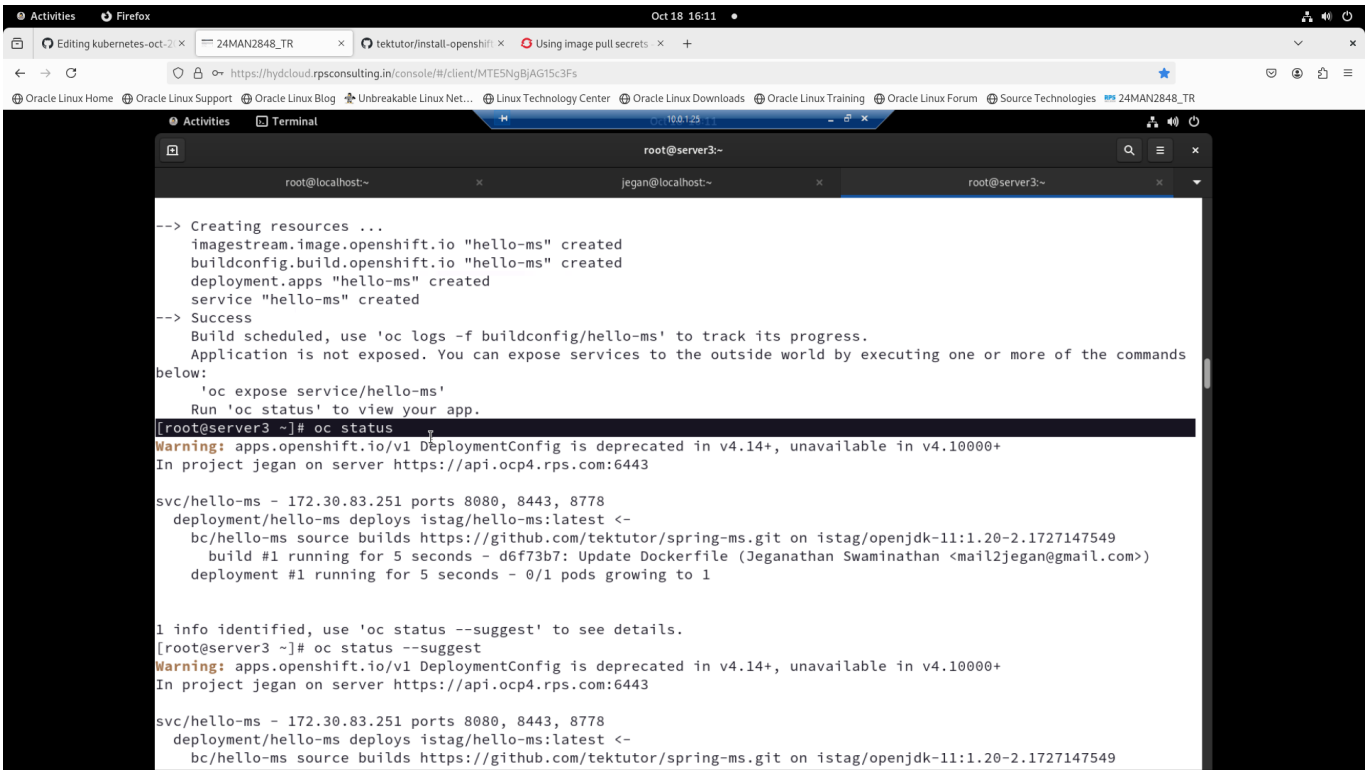


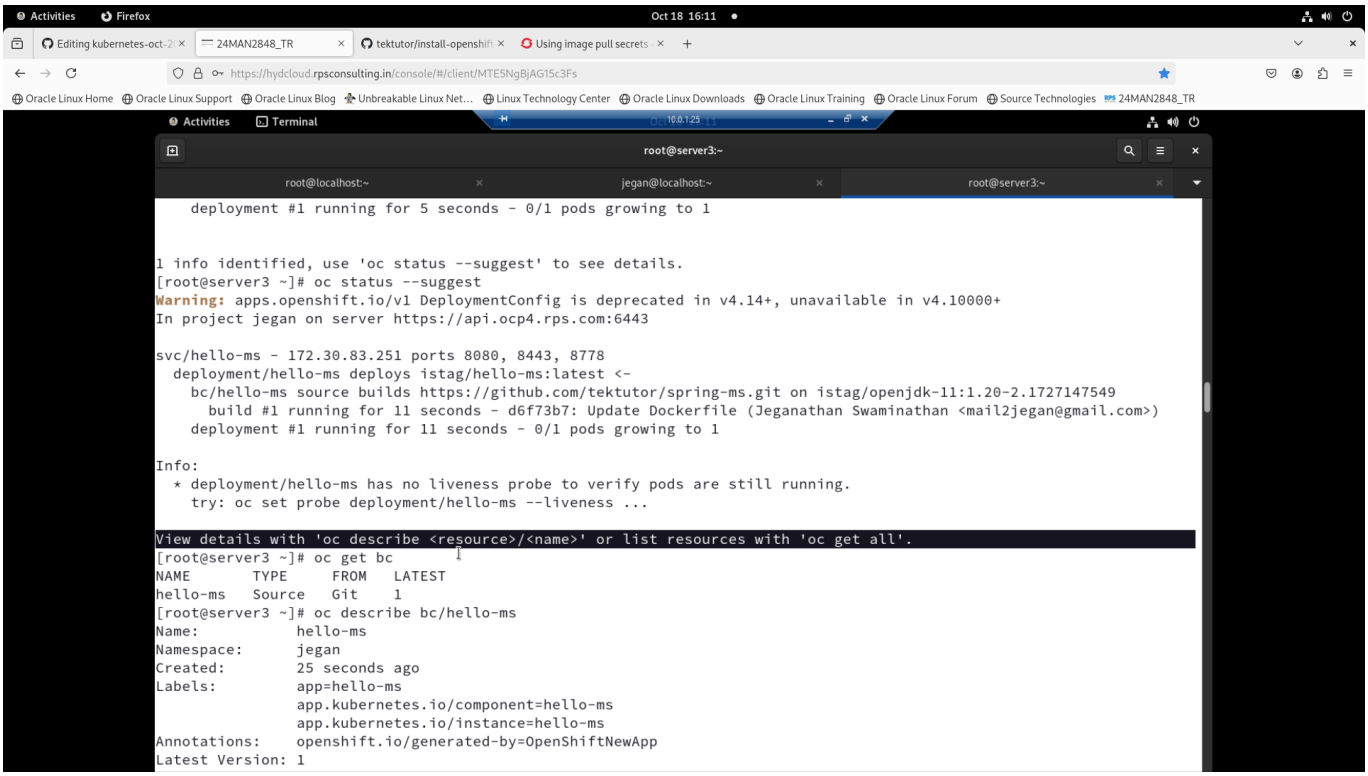
Lab - Deploy your custom application using S2I source strategy

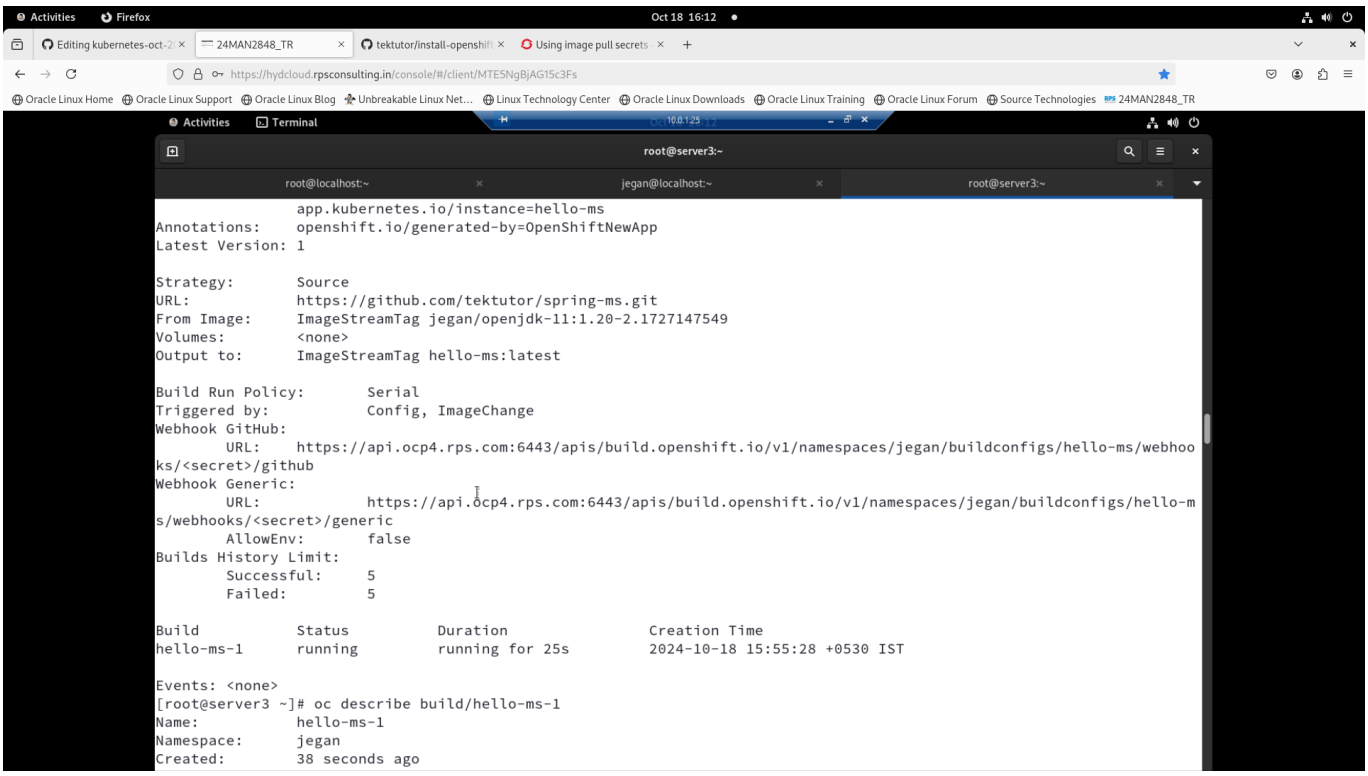
```
oc new-project jegan
oc new-app --name=hello openjdk-11:1.20-
2.1727147549~https://github.com/tektutor/spring-ms.git --strategy=source
```

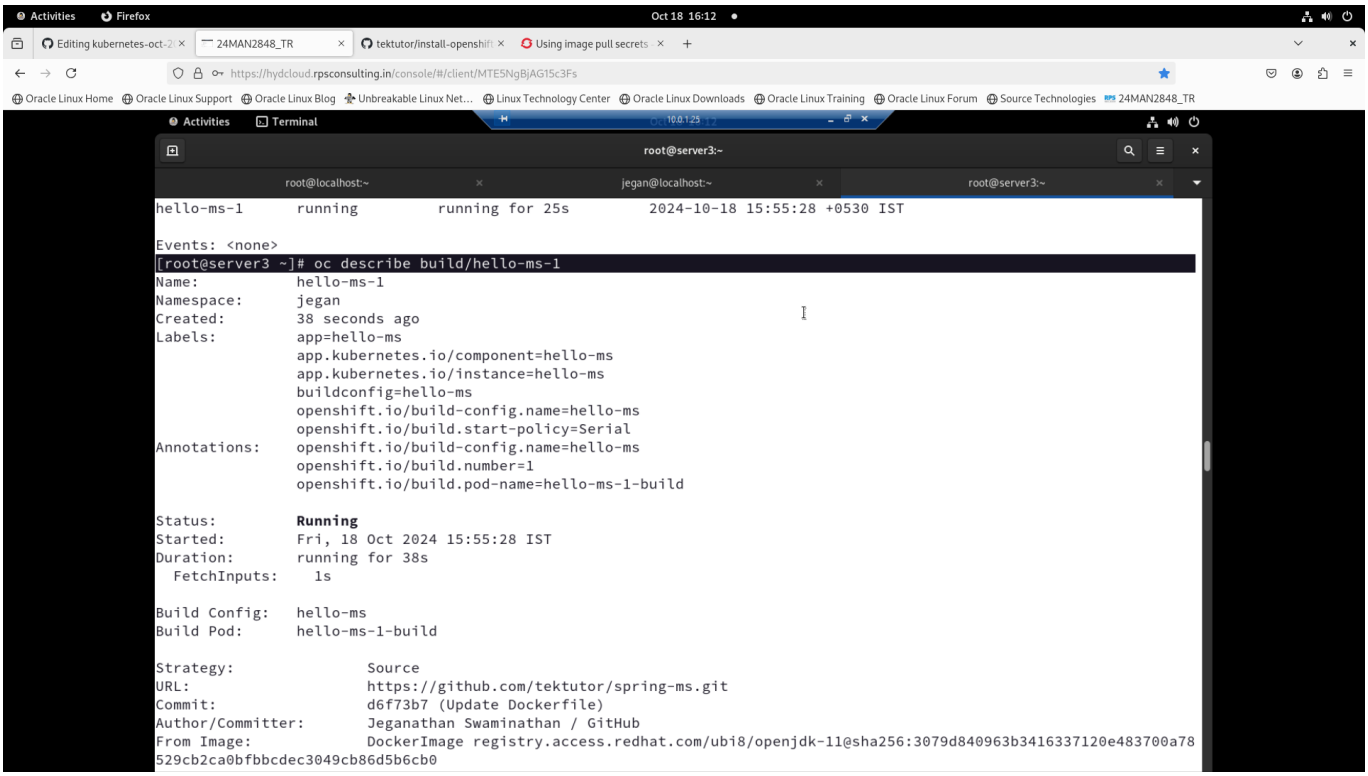
Expected output

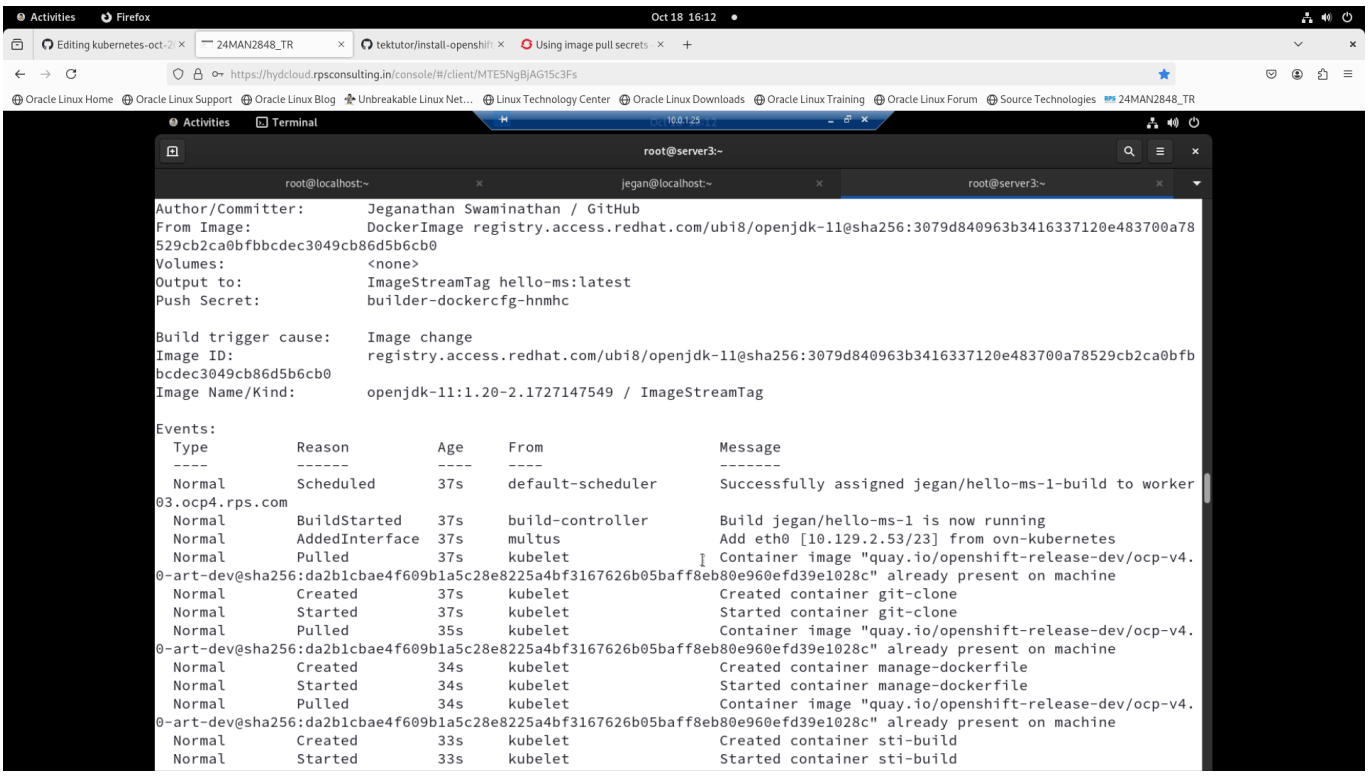


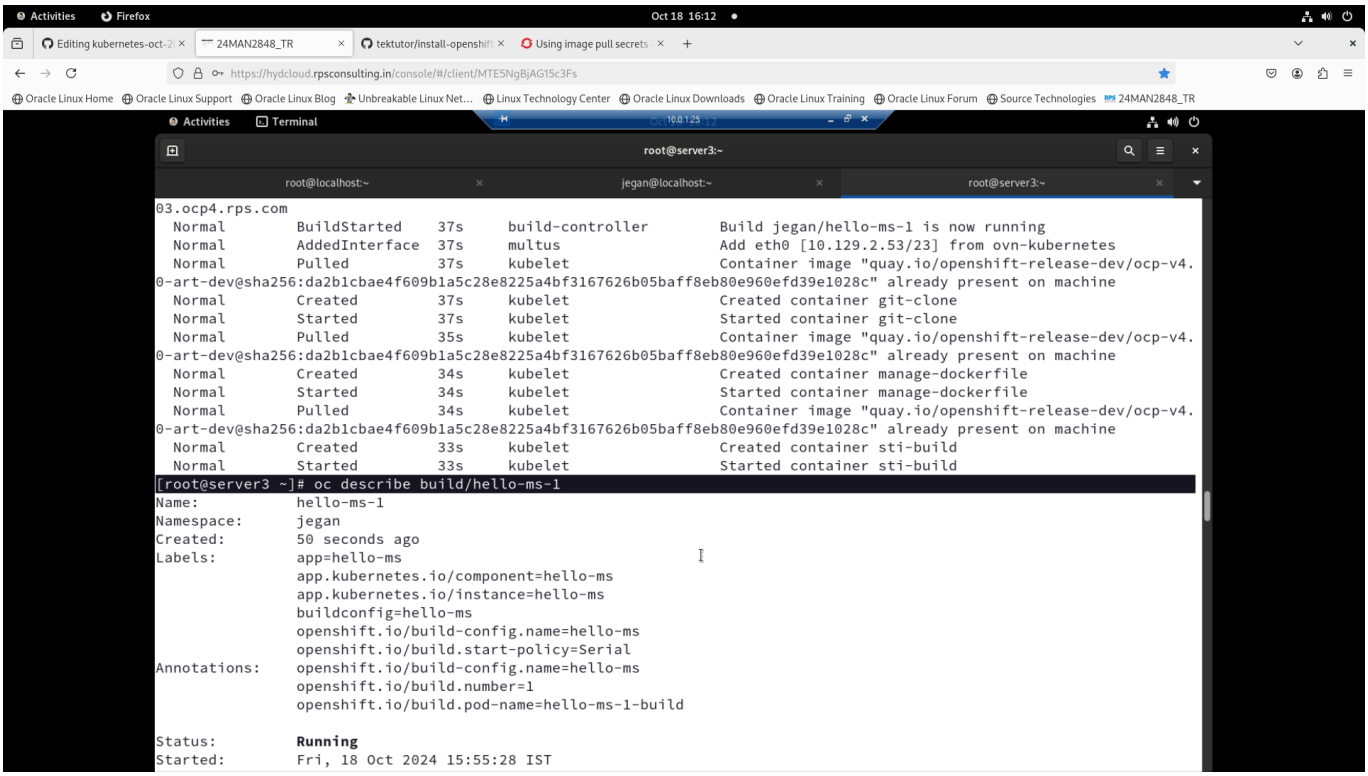



Activities
Firefox
Oct 18 16:11
Editing kubernetes-oct-2
24MAN2848_TR
tektutor/install-openshift
Using image pull secrets
https://hydcloud.rpsconsulting.in/console/#/client/MTE5NgBjAG15c3Fs
Oracle Linux Home
Oracle Linux Support
Oracle Linux Blog
Unbreakable Linux Net...
Linux Technology Center
Oracle Linux Downloads
Oracle Linux Training
Oracle Linux Forum
Source Technologies
24MAN2848_TR
Activities
Terminal
root@server3:~
root@localhost:~
jegan@localhost:~
root@server3:~
deployment #1 running for 5 seconds - 0/1 pods growing to 1
1 info identified, use 'oc status --suggest' to see details.
[root@server3 ~]# oc status --suggest
Warning: apps.openshift.io/v1 DeploymentConfig is deprecated in v4.14+, unavailable in v4.10000+
In project jegan on server https://api.ocp4.rps.com:6443
svc/hello-ms - 172.30.83.251 ports 8080, 8443, 8778
deployment/hello-ms deploys istag/hello-ms:latest <-
bc/hello-ms source builds https://github.com/tektutor/spring-ms.git on istag/openjdk-11:1.20-2.1727147549
build #1 running for 11 seconds - d6f73b7: Update Dockerfile (Jeganathan Swaminathan <mail2jegan@gmail.com>)
deployment #1 running for 11 seconds - 0/1 pods growing to 1
Info:
* deployment/hello-ms has no liveness probe to verify pods are still running.
try: oc set probe deployment/hello-ms --liveness ...
View details with 'oc describe <resource>/<name>' or list resources with 'oc get all'.
[root@server3 ~]# oc get bc
NAME TYPE FROM LATEST
hello-ms Source Git 1
[root@server3 ~]# oc describe bc/hello-ms
Name: hello-ms
Namespace: jegan
Created: 25 seconds ago
Labels: app=hello-ms
app.kubernetes.io/component=hello-ms
app.kubernetes.io/instance=hello-ms
Annotations: openshift.io/generated-by=OpenShiftNewApp
Latest Version: 1


Activities
Firefox
Oct 18 16:12
Editing kubernetes-oct-2
24MAN2848_TR
tektutor/install-openshift
Using image pull secrets
https://hydcloud.rpsconsulting.in/console/#/client/MTE5NgBjAG15c3Fs
Oracle Linux Home
Oracle Linux Support
Oracle Linux Blog
Unbreakable Linux Net...
Linux Technology Center
Oracle Linux Downloads
Oracle Linux Training
Oracle Linux Forum
Source Technologies
24MAN2848_TR
Activities
Terminal
root@server3:~
root@localhost:~
jegan@localhost:~
root@server3:~
app.kubernetes.io/instance=hello-ms
Annotations: openshift.io/generated-by=OpenShiftNewApp
Latest Version: 1
Strategy: Source
URL: https://github.com/tektutor/spring-ms.git
From Image: ImageStreamTag jegan/openjdk-11:1.20-2.1727147549
Volumes: <none>
Output to: ImageStreamTag hello-ms:latest
Build Run Policy: Serial
Triggered by: Config, ImageChange
Webhook GitHub:
URL: https://api.ocp4.rps.com:6443/apis/build.openshift.io/v1/namespaces/jegan/buildconfigs/hello-ms/webhooks/<secret>/github
Webhook Generic:
URL: https://api.ocp4.rps.com:6443/apis/build.openshift.io/v1/namespaces/jegan/buildconfigs/hello-ms/webhooks/<secret>/generic
AllowEnv: false
Builds History Limit:
Successful: 5
Failed: 5
Build Status Duration Creation Time
hello-ms-1 running running for 25s 2024-10-18 15:55:28 +0530 IST
Events: <none>
[root@server3 ~]# oc describe build/hello-ms-1
Name: hello-ms-1
Namespace: jegan
Created: 38 seconds ago


Activities
Firefox
Oct 18 16:12
Terminal
root@server3:~
root@localhost:~
jegan@localhost:~
root@server3:~
hello-ms-1 running running for 25s 2024-10-18 15:55:28 +0530 IST
Events: <none>
[root@server3 ~]# oc describe build/hello-ms-1
Name: hello-ms-1
Namespace: jegan
Created: 38 seconds ago
Labels: app=hello-ms
app.kubernetes.io/component=hello-ms
app.kubernetes.io/instance=hello-ms
buildconfig=hello-ms
openshift.io/build-config.name=hello-ms
openshift.io/build.start-policy=Serial
Annotations: openshift.io/build-config.name=hello-ms
openshift.io/build.number=1
openshift.io/build.pod-name=hello-ms-1-build
Status: Running
Started: Fri, 18 Oct 2024 15:55:28 IST
Duration: running for 38s
FetchInputs: 1s
Build Config: hello-ms
Build Pod: hello-ms-1-build
Strategy: Source
URL: https://github.com/tektutor/spring-ms.git
Commit: d6f73b7 (Update Dockerfile)
Author/Committer: Jeganathan Swaminathan / GitHub
From Image: DockerImage registry.access.redhat.com/ubi8/openjdk-11@sha256:3079d840963b3416337120e483700a78
529cb2ca0bfbbcdec3049cb86d5b6cb0


Activities
Firefox
Oct 18 16:12
Terminal
root@server3:~
root@localhost:~
jegan@localhost:~
root@server3:~
Author/Committer: Jeganathan Swaminathan / GitHub
From Image: DockerImage registry.access.redhat.com/ubi8/openjdk-11@sha256:3079d840963b3416337120e483700a78
529cb2ca0bfbbcdec3049cb86d5b6cb0
Volumes: <none>
Output to: ImageStreamTag hello-ms:latest
Push Secret: builder-dockercfg-hnmhc
Build trigger cause: Image change
Image ID: registry.access.redhat.com/ubi8/openjdk-11@sha256:3079d840963b3416337120e483700a78529cb2ca0bfb
bcdec3049cb86d5b6cb0
Image Name/Kind: openjdk-11:1.20-2.1727147549 / ImageStreamTag
Events:
Type Reason Age From Message
Normal Scheduled 37s default-scheduler Successfully assigned jegan/hello-ms-1-build to worker
03.ocp4.rps.com
Normal BuildStarted 37s build-controller Build jegan/hello-ms-1 is now running
Normal AddedInterface 37s multus Add eth0 [10.129.2.53/23] from ovn-kubernetes
Normal Pulled 37s kubelet Container image "quay.io/openshift-release-dev/ocp-v4.
0-art-dev@sha256:da2b1cbae4f609b1a5c28e8225a4bf3167626b05baff8eb80e960efd39e1028c" already present on machine
Normal Created 37s kubelet Created container git-clone
Normal Started 37s kubelet Started container git-clone
Normal Pulled 35s kubelet Container image "quay.io/openshift-release-dev/ocp-v4.
0-art-dev@sha256:da2b1cbae4f609b1a5c28e8225a4bf3167626b05baff8eb80e960efd39e1028c" already present on machine
Normal Created 34s kubelet Created container manage-dockerfile
Normal Started 34s kubelet Started container manage-dockerfile
Normal Pulled 34s kubelet Container image "quay.io/openshift-release-dev/ocp-v4.
0-art-dev@sha256:da2b1cbae4f609b1a5c28e8225a4bf3167626b05baff8eb80e960efd39e1028c" already present on machine
Normal Created 33s kubelet Created container sti-build
Normal Started 33s kubelet Started container sti-build



Uploading image.png…

Further references

```
https://kubernetes.io/docs/concepts/cluster-administration/networking/#the-kubernetes-network-model

https://docs.tigera.io/calico/latest/about/kubernetes-training/about-networking

https://docs.tigera.io/calico/latest/about/kubernetes-training/about-network-policy

https://docs.tigera.io/calico/latest/about/kubernetes-training/about-kubernetes-ingress

https://docs.tigera.io/calico/latest/about/kubernetes-training/about-kubernetes-egress

https://docs.tigera.io/calico/latest/about/kubernetes-training/about-kubernetes-services

https://docs.tigera.io/calico/latest/about/kubernetes-training/about-ebpf

https://docs.tigera.io/calico/latest/about/kubernetes-training/about-k8s-networking

https://www.tigera.io/blog/everything-you-need-to-know-about-kubernetes-pod-networking-on-aws/

https://www.tigera.io/blog/everything-you-need-to-know-about-kubernetes-networking-on-azure/
```

```
https://www.tigera.io/blog/everything-you-need-to-know-about-kubernetes-networking-on-google-cloud/

https://docs.openshift.com/container-platform/4.8/windows_containers/understanding-windows-container-workloads.html#understanding-windows-container-workloads
```