**Day 1**

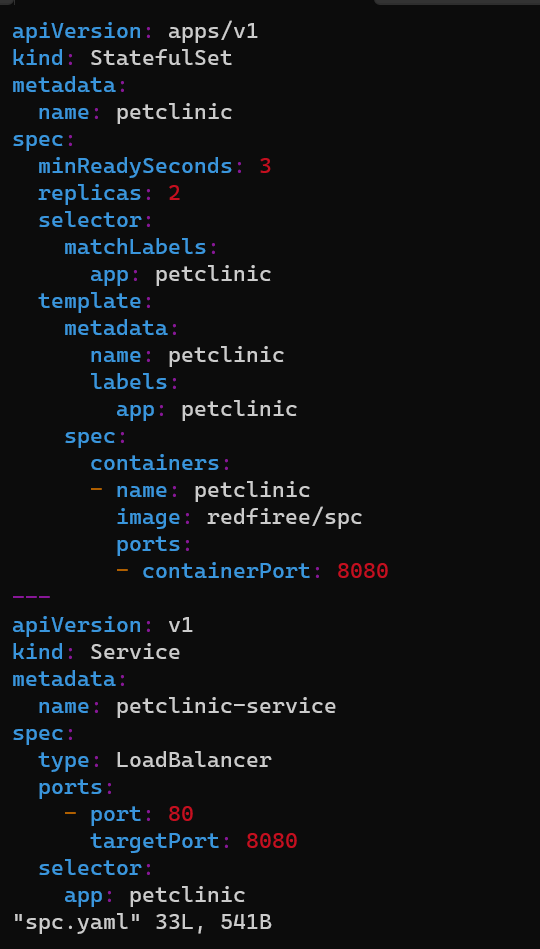
1) Write a Pod Spec for Spring Pet Clinic and   
nopCommerce Applications

1st we have to install Docker in 3 machines then enter

sudo usermod -aG docker ubuntu now exit and relogin machine.

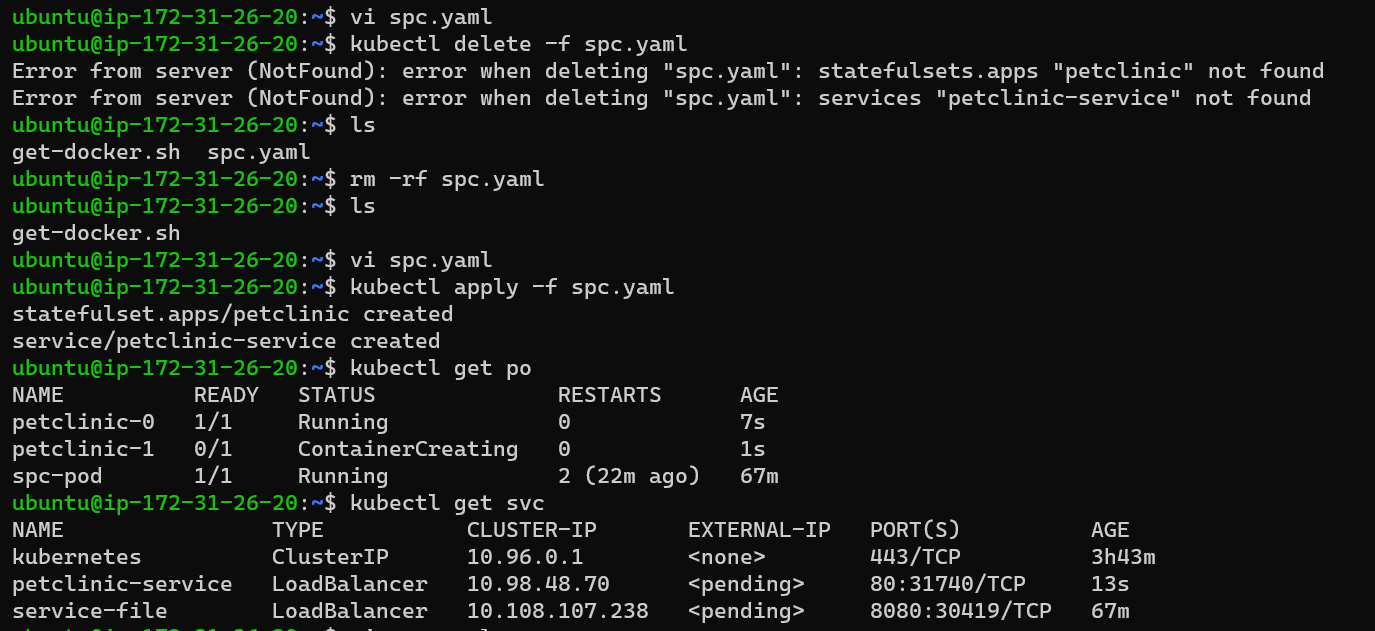
Write yaml file vi spc.yaml

Vi spc.yaml

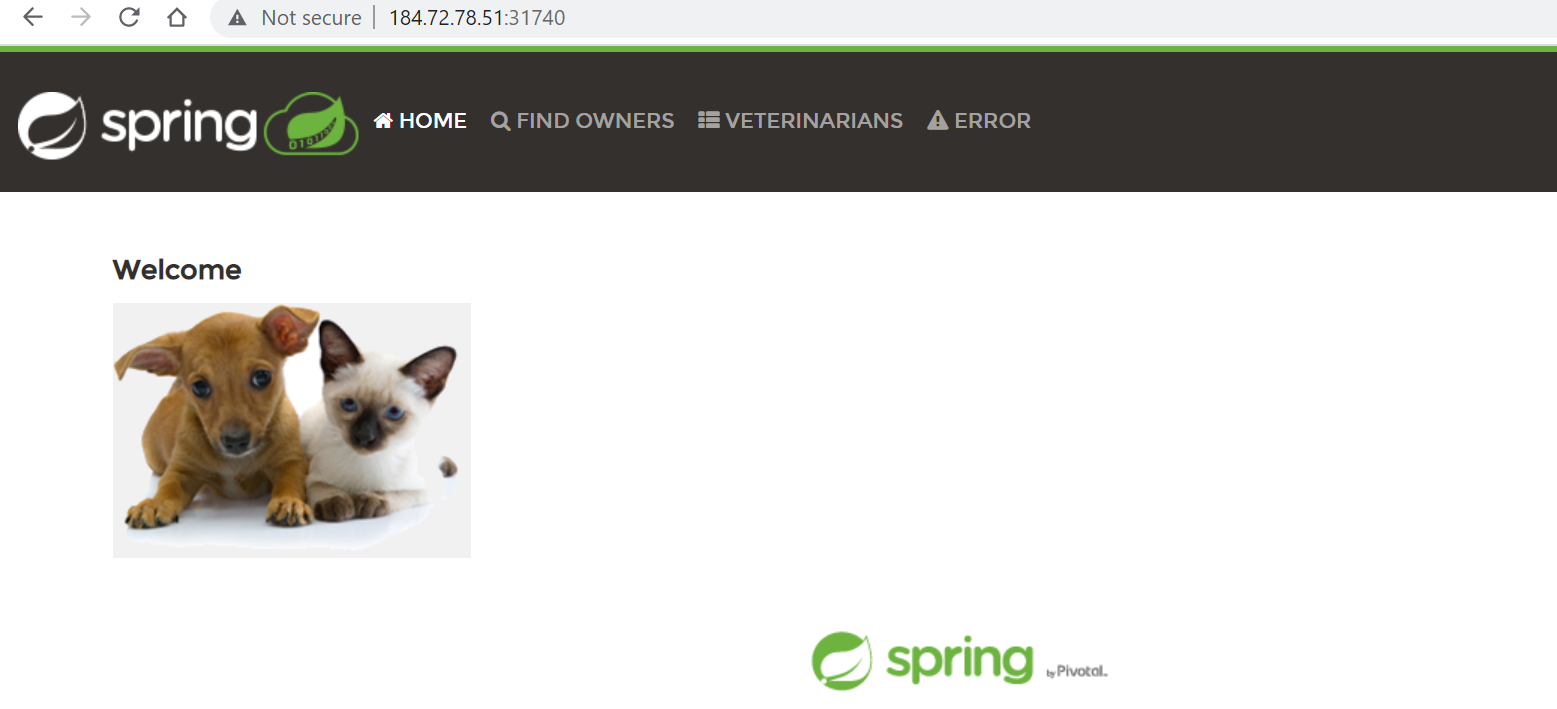


Kubectl get apply -f spc.yaml

Kubectl get po



Now check on server with the help of port



**2**) **Execute the kubectl commands: kubectl get pods   
and describe pods**

**kubectl get pods:**

**:-** *# Get commands with basic output*

kubectl get services *# List all services in the namespace*

kubectl get pods --all-namespaces *# List all pods in all namespaces*

kubectl get pods -o wide *# List all pods in the current namespace, with more details*

kubectl get deployment my-dep *# List a particular deployment*

kubectl get pods *# List all pods in the namespace*

kubectl get pod my-pod -o yaml *# Get a pod's YAML*

**describe pods**

**:- #Describe a pod**

**kubectl describe pods/nginx**

Kubectl Describe Pod is a command that describes any resource in Kubernetes. It is used to show data on a single or even a collection of resources. This command combines a number of API calls to create a thorough description of a resource or set of resources.

A pod is a collection of containers sharing a network, acting as the basic unit of deployment in Kubernetes. All containers in a pod are scheduled on the same node.

Check to see if the pod is running: **kubectl get pods**

**# Describe a pod**

**kubectl describe pods/nginx**

------------------------------------------------------------------------------------

Day 2

------------------------------------------------------------------------------------

1. Explain Kubernetes architecture

The Kubernetes architecture consists of two main components: Master Components and Node Components.

Master Components**:**

Master components are the control plane components of Kubernetes that manage the Kubernetes cluster. The master components include:

Kubernetes API Server:

The Kubernetes API Server exposes the Kubernetes API, which is used by clients to interact with the Kubernetes cluster. It is the primary management point for the Kubernetes cluster and is responsible for validating and processing API requests.

etcd**:**

etcd is a distributed key-value store that stores the configuration data of the Kubernetes cluster. It is used to store the state of the cluster, including the state of all objects (pods, services, deployments, etc.).

Kube-Controller Manager**:**

The Kube-Controller Manager is responsible for running controllers that are responsible for maintaining the desired state of the Kubernetes cluster. The controllers include the node controller, the replication controller, and the endpoint controller.

Kube-Scheduler:

The Kube-Scheduler is responsible for scheduling the pods on the nodes in the Kubernetes cluster. It uses information about the nodes' available resources and the pod's resource requirements to determine the best node to schedule the pod on.

Node Components:

Node components are the worker components of Kubernetes that run on each node in the Kubernetes cluster. The node components include:

**Kubelet**:

The Kubelet is the primary node agent that communicates with the Kubernetes API server and ensures that the containers are running on the node as expected. It is responsible for starting, stopping, and monitoring the containers on the node.

**Container Runtime**:

The Container Runtime is the software that runs the containers on the node. Kubernetes supports several container runtimes, including Docker, CRI-O, and containerd.

kube-proxy:

The kube-proxy is responsible for providing network connectivity to the pods running on the node. It does this by creating network rules that allow traffic to be forwarded to the pods.

Kubernetes Components:

Kubernetes components can be divided into two categories: Control Plane Components and Worker Nodes Components.

Control Plane Components**:**

**kube-apiserver:**

The kube-apiserver is the main management point for the Kubernetes cluster. It exposes the Kubernetes API, which is used by clients to interact with the Kubernetes cluster.

kube-scheduler**:**

The kube-scheduler is responsible for scheduling the pods on the nodes in the Kubernetes cluster.

kube-controller-manager:

The kube-controller-manager runs controllers that are responsible for maintaining the desired state of the Kubernetes cluster.

cloud-controller-manager:

The cloud-controller-manager is responsible for managing the cloud provider-specific resources in the Kubernetes cluster. It provides a way to integrate with the cloud provider's APIs to manage the cloud resources.

Worker Node Components:

**Nodes:**

Nodes are the worker machines that run the containers. They are managed by the Kubernetes master components.

## Container Runtime Engine:

The Container Runtime Engine is responsible for running the containers on the node. Kubernetes supports several container runtimes, including Docker, CRI-O, and containerd.

## kubelet:

Kubelet is one of the main components of Kubernetes responsible for managing individual nodes and their containers. It is essentially an agent that runs on each node in the Kubernetes cluster and communicates with the API server to ensure that the containers running on the node are healthy and running as expected.

The kubelet performs several functions, including:

1. Fetching container manifests from the Kubernetes API server.
2. Ensuring that the containers described in the manifest are running and healthy.
3. Reporting the status of the containers back to the API server.
4. Mounting and unmounting volumes as necessary.
5. Executing container health checks.

## kube-proxy

The kube-proxy component is responsible for managing the network proxy between the Kubernetes services and the pods that are running on the worker nodes. The kube-proxy uses various networking modes to ensure that the communication between the pods and services is efficient and reliable.

## Container Networking

The container networking component is responsible for ensuring that all the containers running on the worker nodes can communicate with each other and with the external networks. Kubernetes provides several plugins for container networking, including Flannel, Calico, and Weave.

1. **Setup k8s on single node using minikube and kind**

Launch an ec2 instance and Docker

curl -fsSL https://get.docker.com -o get-docker.sh

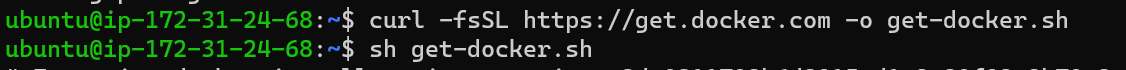
sh get-docker.sh

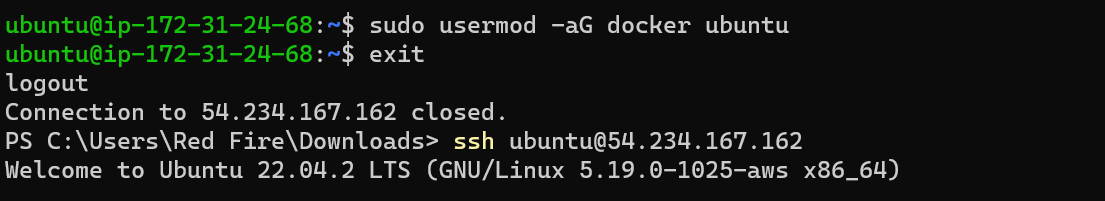
sudo usermod -aG docker <username>

#exit and relogin

docker info

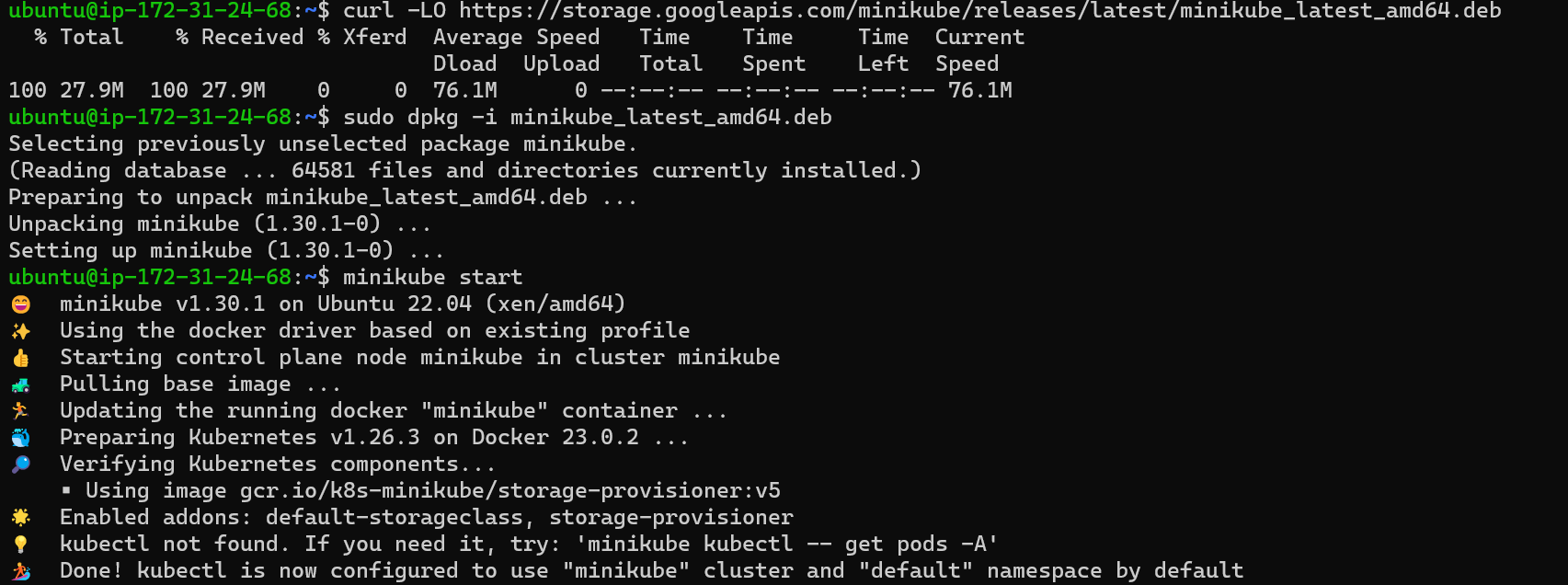
we need to install minikube





curl -LO <https://storage.googleapis.com/minikube/releases/latest/minikube_latest_amd64.deb>

sudo dpkg -i minikube\_latest\_amd64.deb



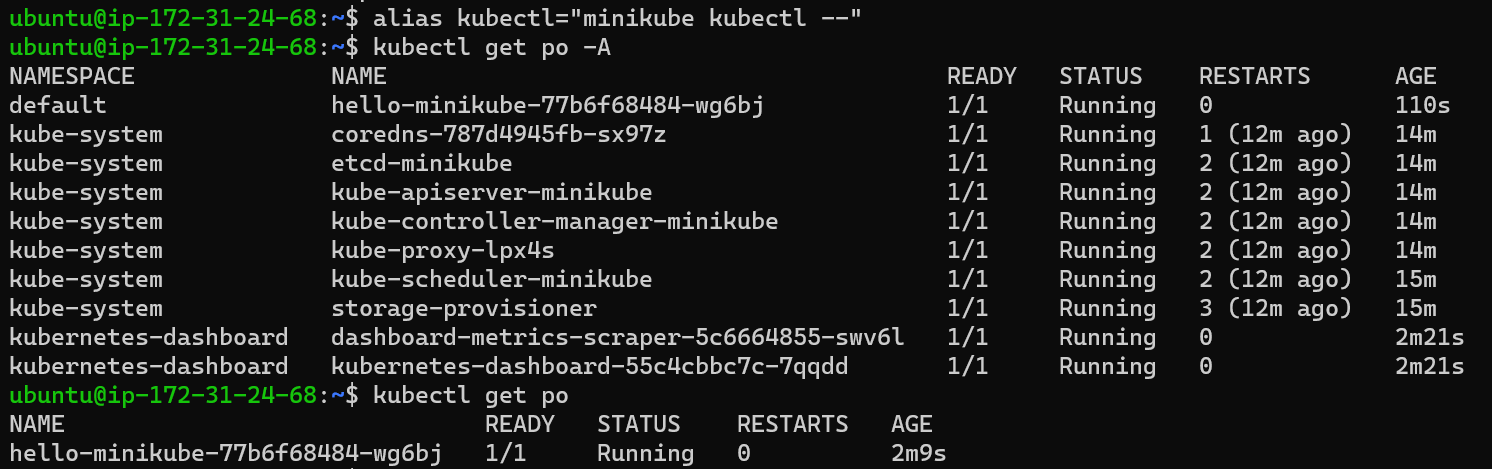
After installations we need to start our cluster

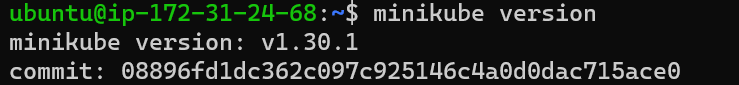
To start cluster we need to run below command

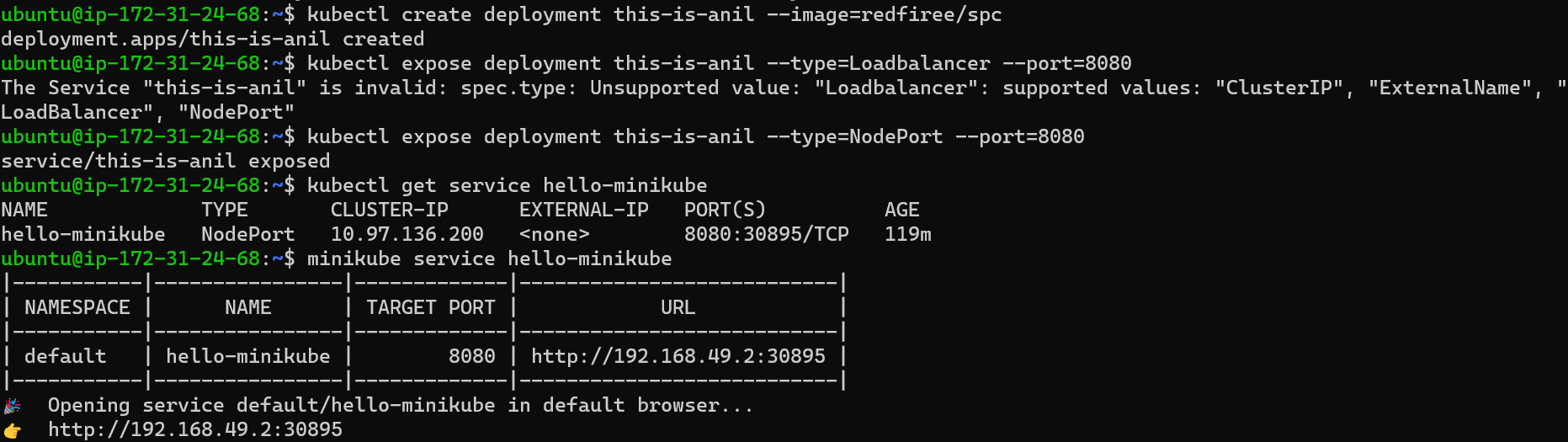
minikube start

once its started we have to intract with our cluster

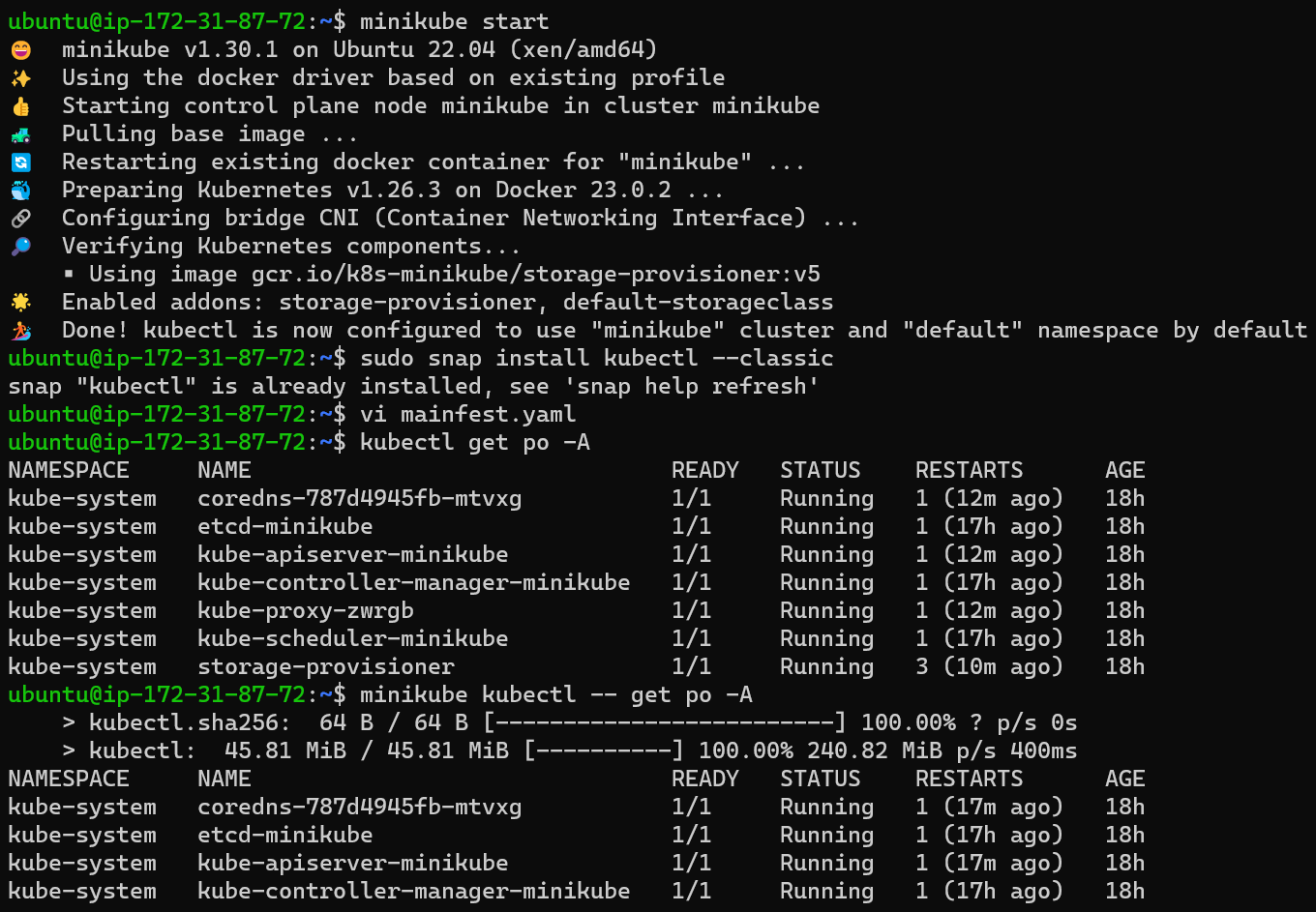
to intract = kubectl get pod -A

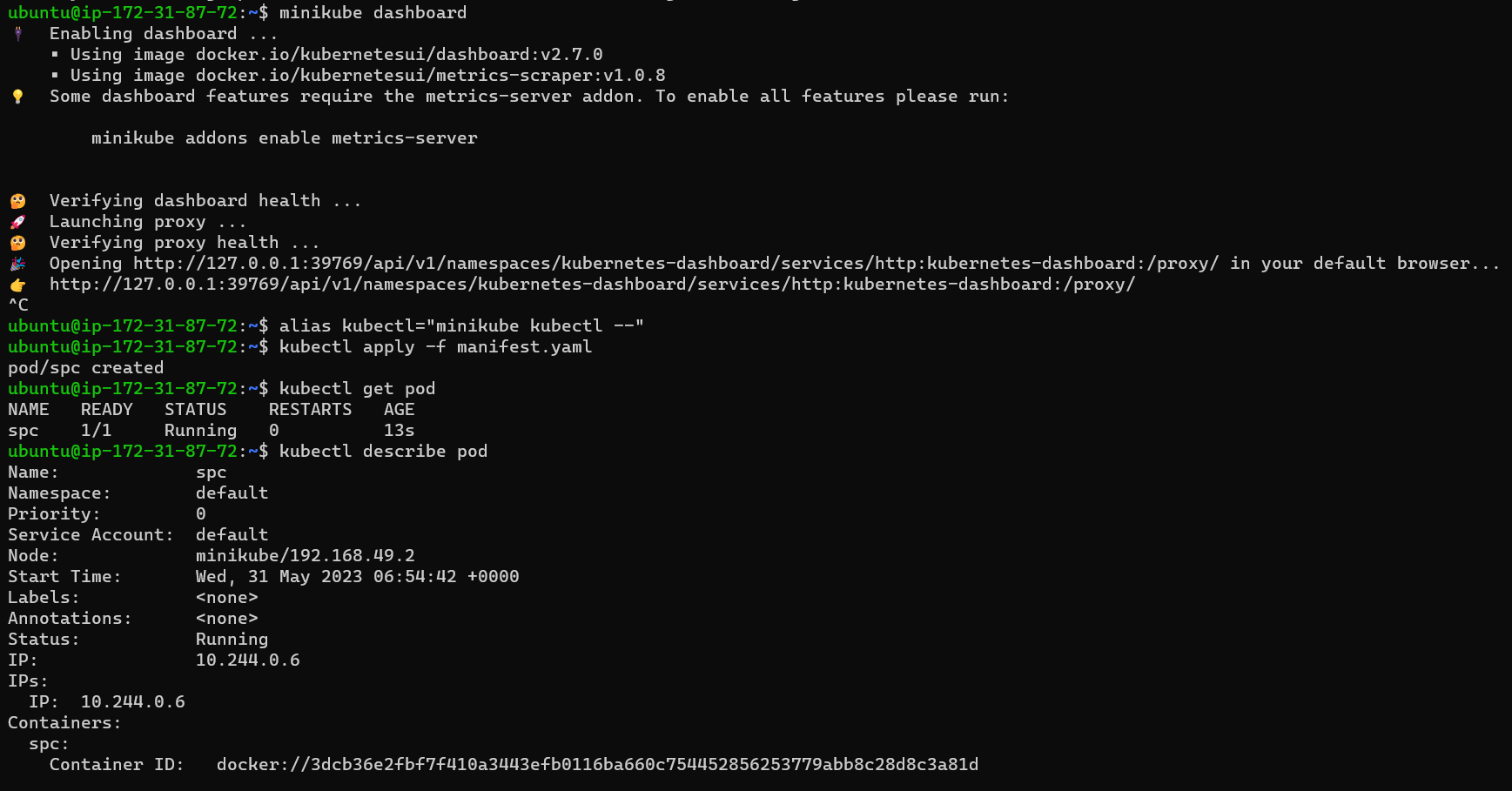


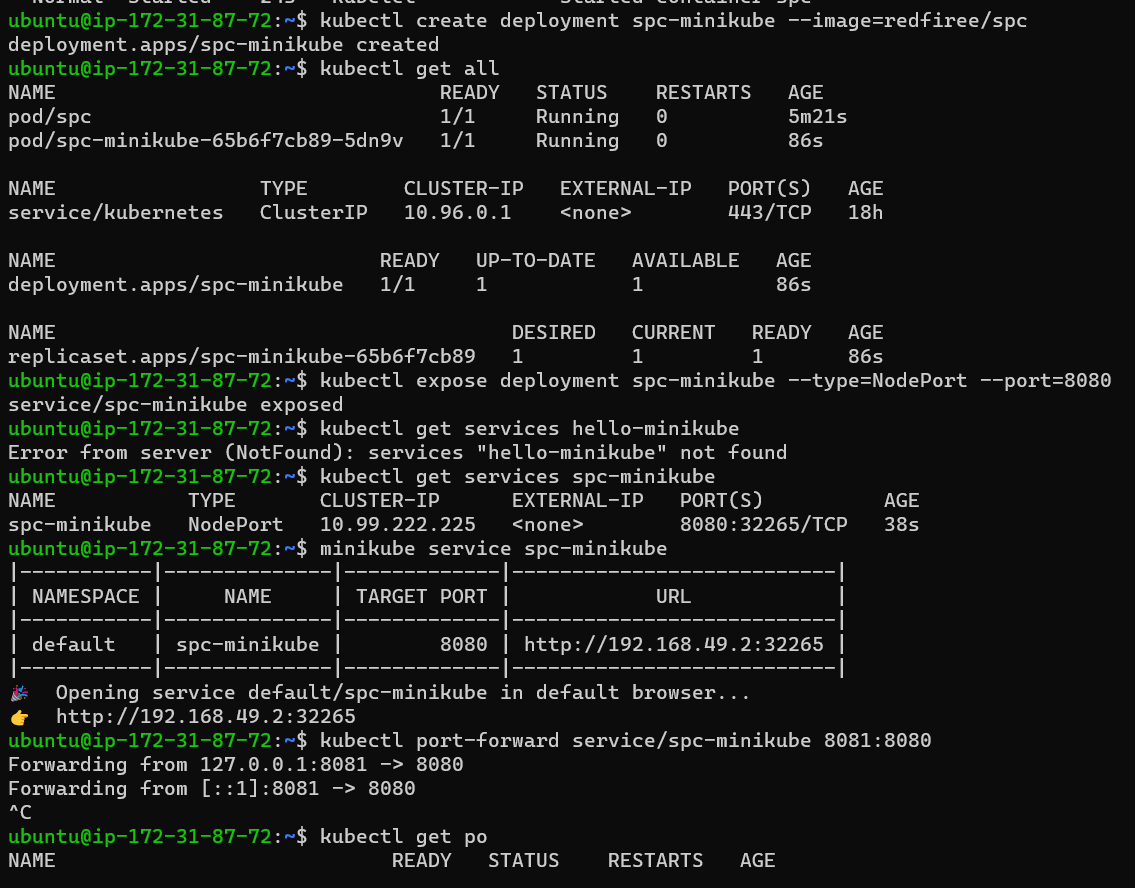


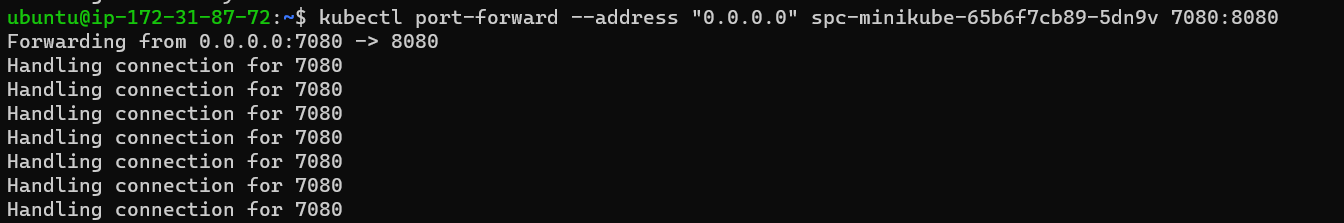


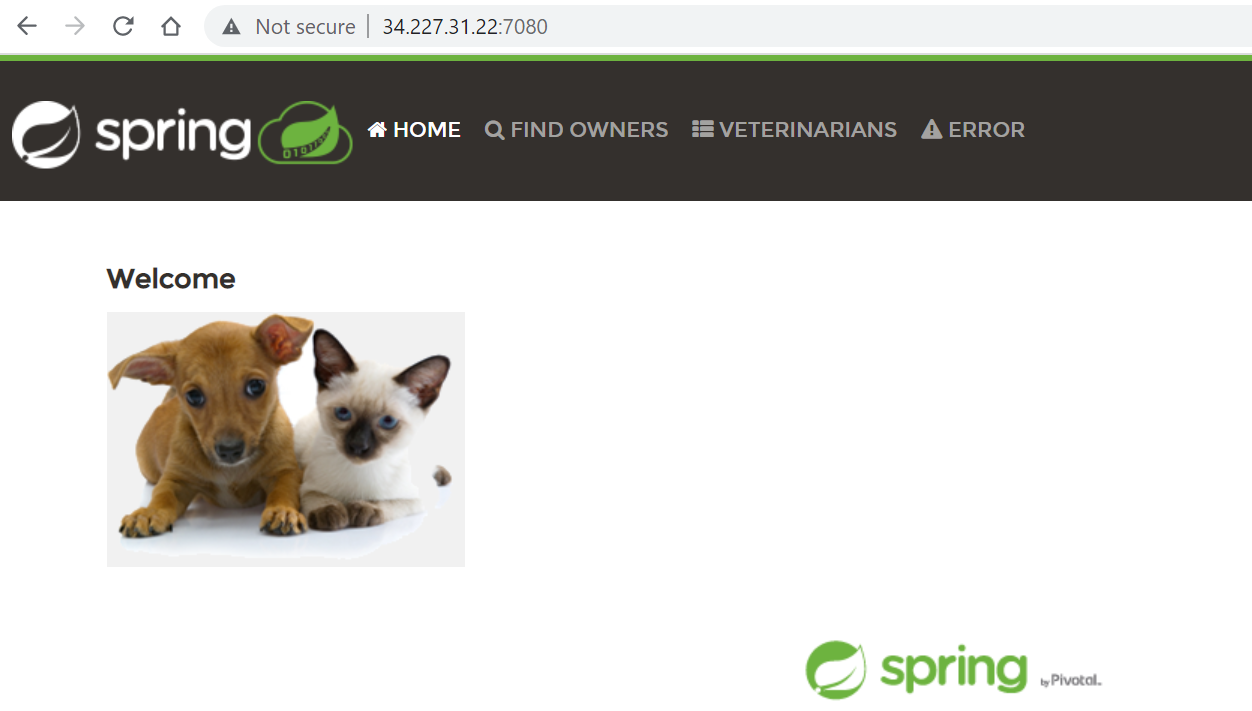
Or











------------------------------------------------------------------------------------

***Kind :- TO install Kind***

Install an ec2 instance

Install docker below commands

curl -fsSL https://get.docker.com -o get-docker.sh

sh get-docker.sh

sudo usermod -aG docker <username>

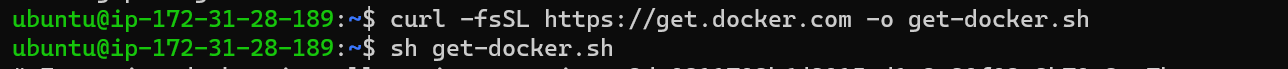
#exit and relogin

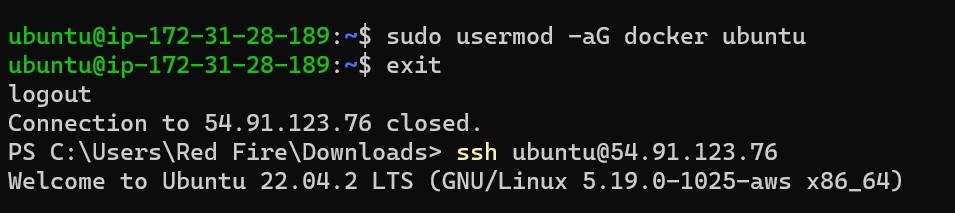
docker info

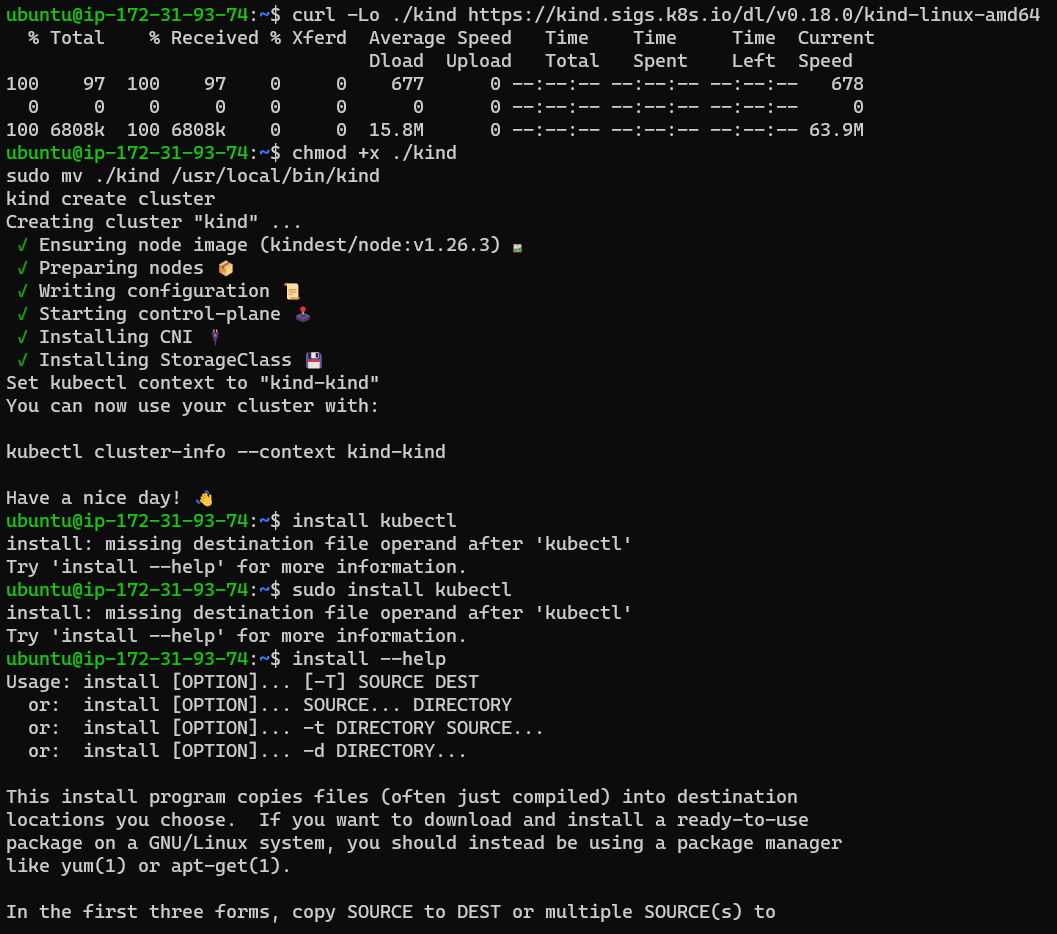
curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.18.0/kind-linux-amd64

chmod +x ./kind

sudo mv ./kind /usr/local/bin/kind

kind create cluster 





Now install kubectl

sudo snap install kubectl –classic

vi manifest.yaml

---

apiVersion: v1

kind: Pod

metadata:

  name: spc

spec:

  containers:

    - name: spc

      image: redfiree/spc

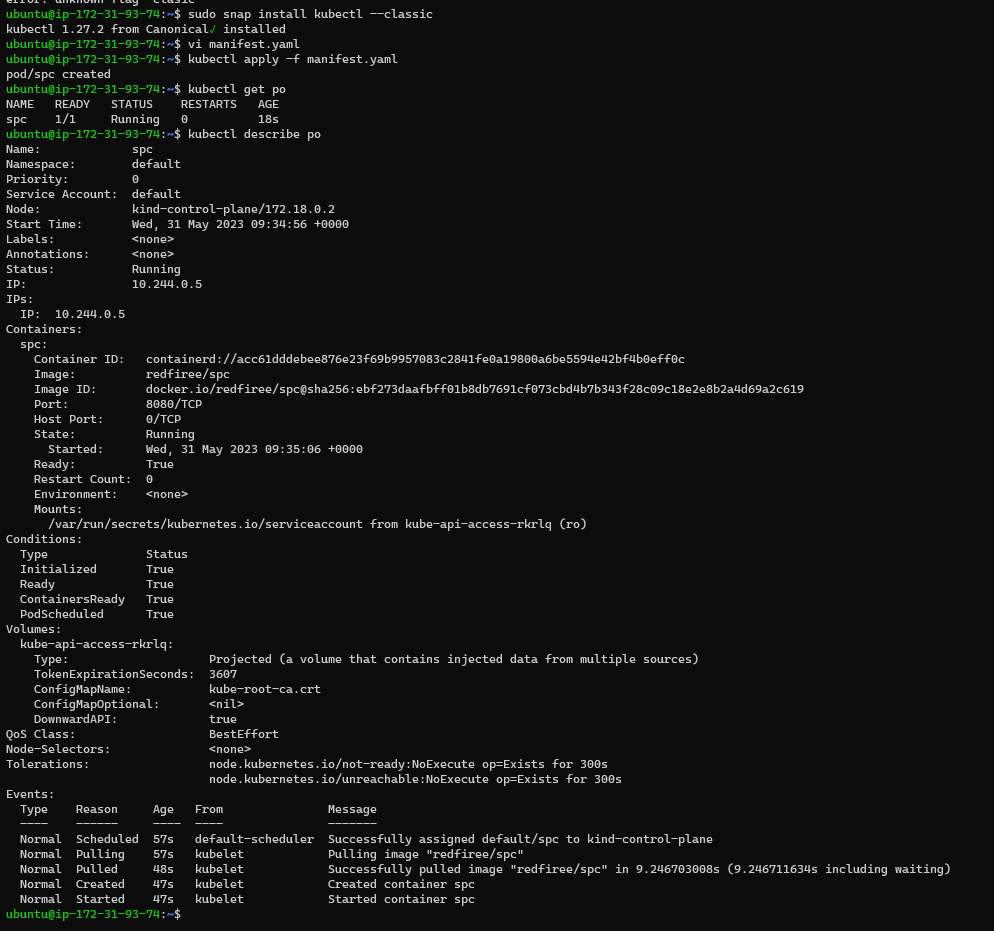
      ports:

        - containerPort: 8080

Kubectl apply -f manifest.yaml

Kubectl get po

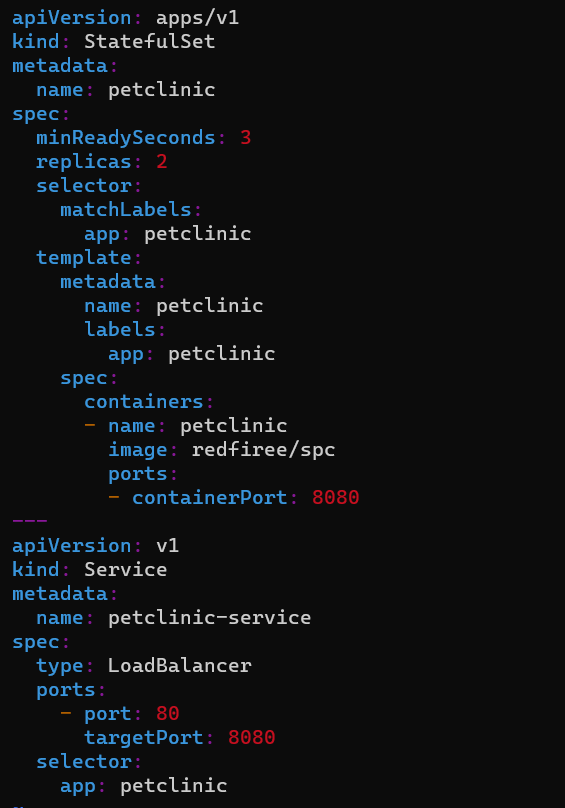
Kubectl get po



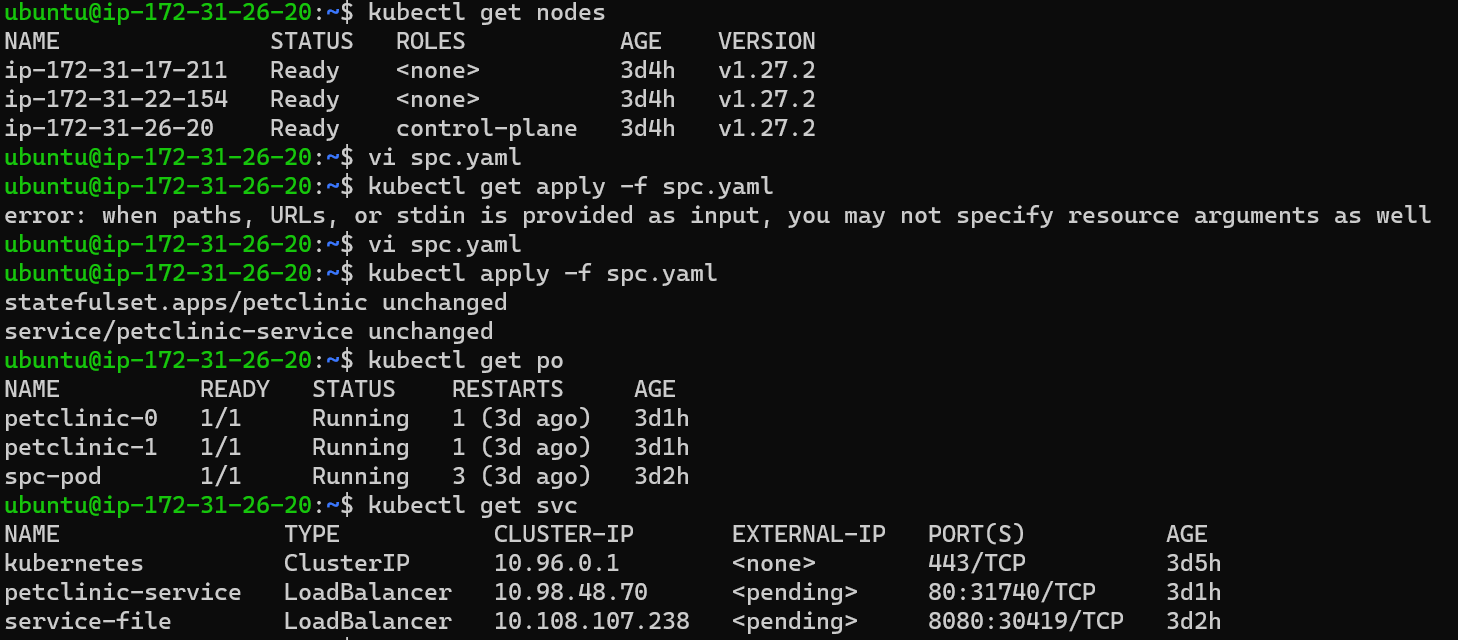
3.Run the Spring Pet Clinic by manifest

1st we have to install Docker then install K8s in our machine

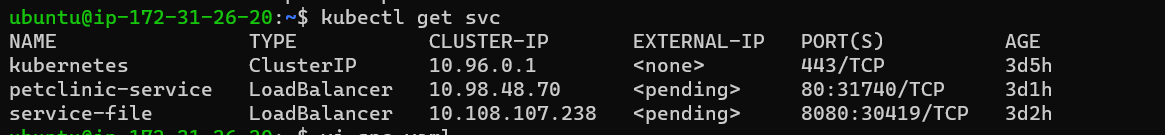
After that we have to write yaml file

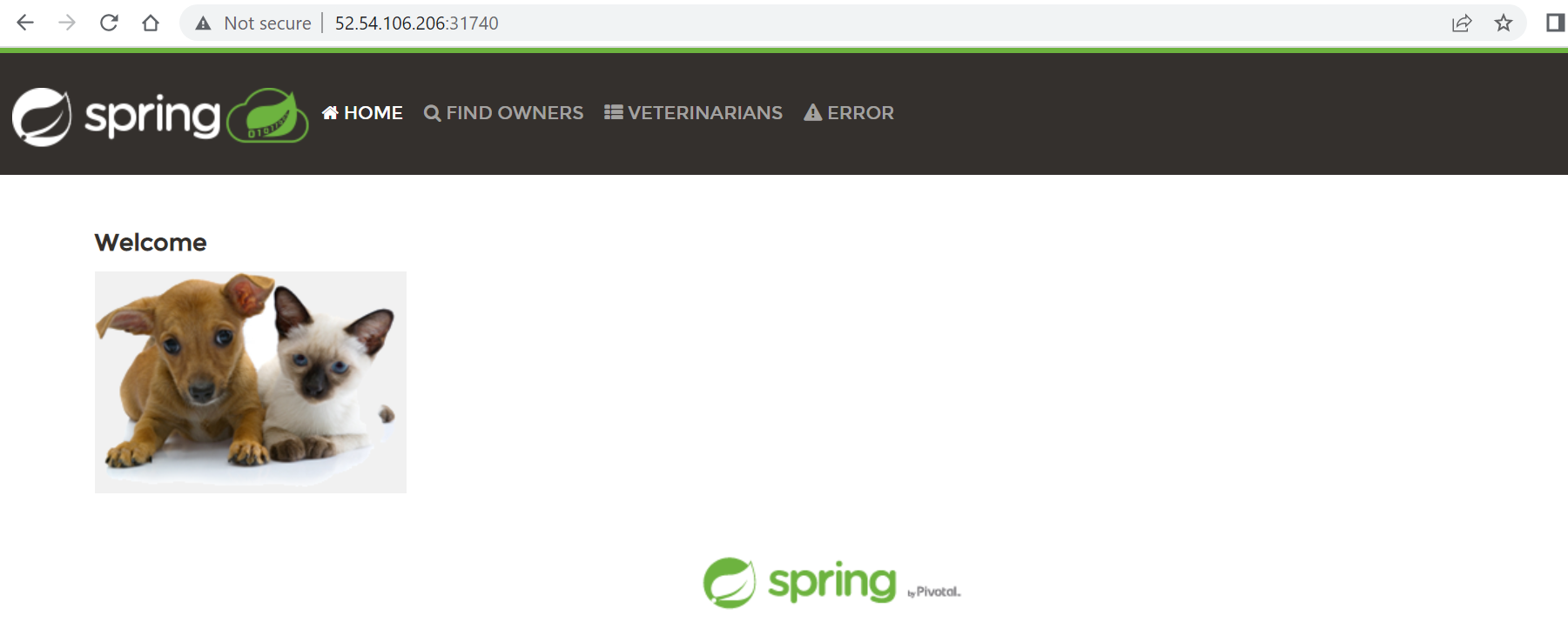


And apply   
{kubectl apply -f spc.yaml}  
Now check service file  
{kubectl get po}



Now Check whether service file is running or not   
{Kubectl get svc}





1. **Writing the Manifest Files for Game of Life App**

Manifest.yaml for game of life.

#### manifest file of game of life

---

apiVersion: v1

kind: Pod

metadata:

  name: game of life

spec:

  containers:

    - name: game of life

      image: shravanipranay/shravani:latest

      ports:

        - containerPort: 8080

------------------------------------------------------------------------------------------------

# Creating the Jobs and CronJobs

### **Launch an ec2 machine // install docker //** Install Docker on ubuntu machines

* Take 3 instance and run below commad

# sudo apt-get update

# curl -fsSL https://get.docker.com -o get-docker.sh

# sh sh get-docker.sh

# sudo usermod -aG docker ubuntu

# exit

# relogin

# # Run these commands as root

# ###Install GO###

# wget https://storage.googleapis.com/golang/getgo/installer\_linux

# chmod +x ./installer\_linux

# ./installer\_linux

# source ~/.bash\_profile

# git clone https://github.com/Mirantis/cri-dockerd.git

# cd cri-dockerd

# mkdir bin

# go build -o bin/cri-dockerd

# mkdir -p /usr/local/bin

# install -o root -g root -m 0755 bin/cri-dockerd /usr/local/bin/cri-dockerd

# cp -a packaging/systemd/\* /etc/systemd/system

# sed -i -e 's,/usr/bin/cri-dockerd,/usr/local/bin/cri-dockerd,' /etc/systemd/system/cri-docker.service

# systemctl daemon-reload

# systemctl enable cri-docker.service

# systemctl enable --now cri-docker.socket

# sudo apt-get update

# sudo apt-get install -y apt-transport-https ca-certificates curl

# curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-archive-keyring.gpg

# echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | 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

# ## node section is completed here, now stop ##

# ## now we have to do in master section extra steps ##

# kubeadm init --pod-network-cidr=10.244.0.0/16

# kubeadm init --pod-network-cidr "10.244.0.0/16" --cri-socket "unix:///var/run/cri-dockerd.sock"

# ----------------------------------------------------------------------------------------------------------------

# -------- after this steps u will get token so copy that token n paste on nodes.....-----

# ------ now paste this token on each nodes-------

# kubeadm join 173.41.26.20:7443 --token ridrvm.jh9687n888sa3a7w \

# --cri-socket "unix:///var/run/cri-dockerd.sock" \

# --discovery-token-ca-cert-hash sha256:f228a307c760dcc513f38cb691d14b31905242ce9d3488f593a096b7e8933a99

# Now login

# Create 1 manifest file

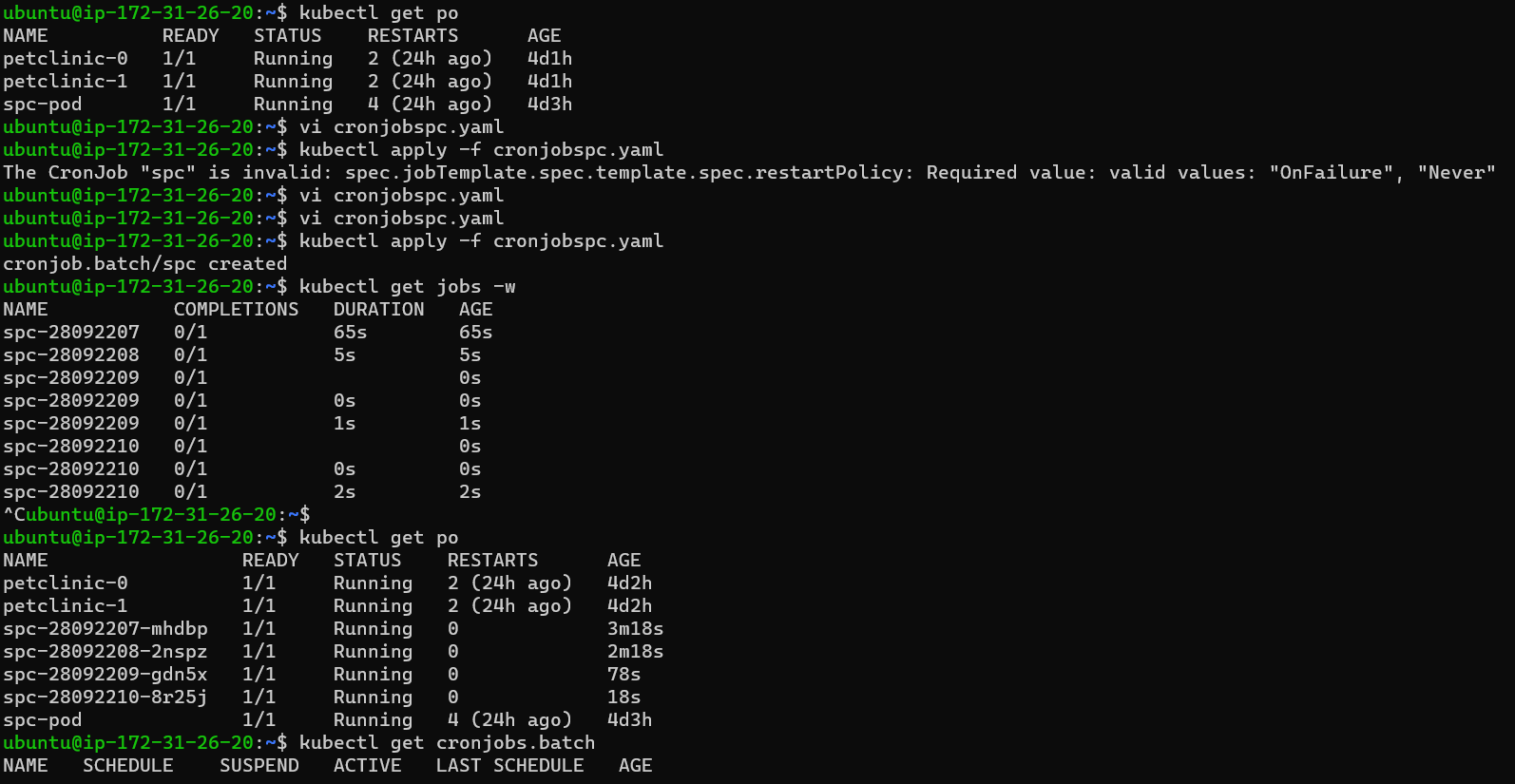
# Vi cronjob.yaml

**Vi cronjob.yaml**

**Paste the above yaml file.**

**kubectl apply -f cronjob.yaml**

**kubectl get cronjobs.batch  // kubectl get cronjobs.batch -w**

****

---

apiVersion: batch/v1

kind: CronJob

metadata:

  name: spc

spec:

  schedule: "\* \* \* \* \*"

  jobTemplate:

    metadata:

      name: spc

    spec:

      template:

        metadata:

          name: spc

        spec:

          restartPolicy: OnFailure

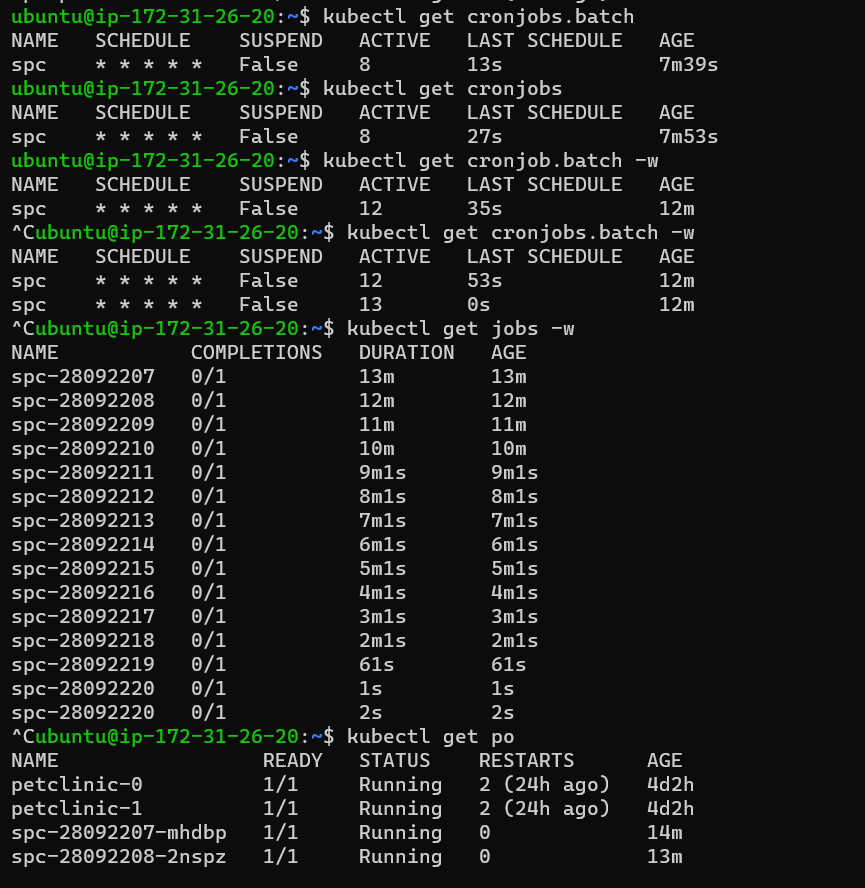
          containers:

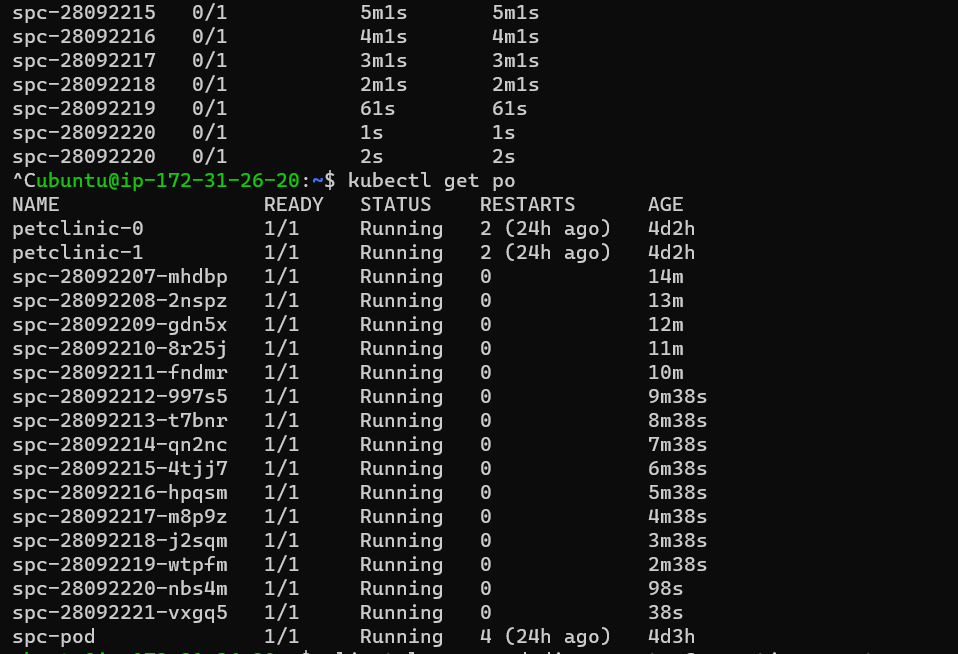
            - name: spc

              image: redfiree/spc

              ports:

                - containerPort: 8080

****

****

**Jobs manifest file**

**Vi jobs.yaml**

---

apiVersion: batch/v1

kind: Job

metadata:

  name: job

spec:

  template:

    metadata:

      name: job

    spec:

      restartPolicy:  Never

      containers:

        - name: job

          image: redfiree/spc

          command:

            - sleep

            - 10s

## Create the ReplicaSet and Replication controller Creating the ReplicaSet && Writing the LABELS and Selecting the LABELS using selector concept.

## This task I will do it tommarow

# 

**Day 4. 04-05-2023**

1. **Theory concepts of k8s**
   1. **Pods / Containers**

* A pod is the smallest execution unit in Kubernetes and Pods are the smallest deployable units of computing that you can create and manage in Kubernetes.
* A Pod (as in a pod of whales or pea pod) is a group of one or more [containers](https://kubernetes.io/docs/concepts/containers/), with shared storage and network resources, and a specification for how to run the containers. A Pod's contents are always co-located and co-scheduled, and run in a shared context. A Pod models an application-specific "logical host": it contains one or more application containers which are relatively tightly coupled. In non-cloud contexts, applications executed on the same physical or virtual machine are analogous to cloud applications executed on the same logical host.

**Containers**

* Each container that you run is repeatable; the standardization from having dependencies included means that you get the same behavior wherever you run it.
* Containers decouple applications from the underlying host infrastructure. This makes deployment easier in different cloud or OS environments
* Each [node](https://kubernetes.io/docs/concepts/architecture/nodes/) in a Kubernetes cluster runs the containers that form the [Pods](https://kubernetes.io/docs/concepts/workloads/pods/) assigned to that node. Containers in a Pod are co-located and co-scheduled to run on the same node.
* A [container image](https://kubernetes.io/docs/concepts/containers/images/) is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.
* Containers decouple applications from the underlying host infrastructure. This makes deployment easier in different cloud or OS environments.
  1. **Jobs / CronJobs**

**Jobs:**

* A Job creates one or more Pods and will continue to retry execution of the Pods until a specified number of them successfully terminate. As pods successfully complete, the Job tracks the successful completions. When a specified number of successful completions is reached, the task (ie, Job) is complete. Deleting a Job will clean up the Pods it created. Suspending a Job will delete its active Pods until the Job is resumed again.
* A simple case is to create one Job object in order to reliably run one Pod to completion. The Job object will start a new Pod if the first Pod fails or is deleted (for example due to a node hardware failure or a node reboot). You can also use a Job to run multiple Pods in parallel.

**CronJobs:**

* A CronJob creates [Jobs](https://kubernetes.io/docs/concepts/workloads/controllers/job/) on a repeating schedule.
* CronJob is meant for performing regular scheduled actions such as backups, report generation, and so on. One CronJob object is like one line of a crontab (cron table) file on a Unix system. It runs a job periodically on a given schedule, written in [Cron](https://en.wikipedia.org/wiki/Cron) format.

## Writing a CronJob spec

### **Schedule syntax**

The .spec.schedule field is required. The value of that field follows the [Cron](https://en.wikipedia.org/wiki/Cron) syntax:

# ┌───────────── minute (0 - 59)

# │ ┌───────────── hour (0 - 23)

# │ │ ┌───────────── day of the month (1 - 31)

# │ │ │ ┌───────────── month (1 - 12)

# │ │ │ │ ┌───────────── day of the week (0 - 6) (Sunday to Saturday;

# │ │ │ │ │ 7 is also Sunday on some systems)

# │ │ │ │ │ OR sun, mon, tue, wed, thu, fri, sat

# │ │ │ │ │

# \* \* \* \* \*

* 1. **ReplicaSets:**
* A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.
* A ReplicaSet (RS) is a Kubernetes object that ensures there is always a stable set of running pods for a specific workload. The ReplicaSet configuration defines a number of identical pods required, and if a pod is evicted or fails, creates more pods to compensate for the loss.
  1. **Deployment**
* A Kubernetes Deployment tells Kubernetes how to create or modify instances of the pods that hold a containerized application. Deployments can help to efficiently scale the number of replica pods, enable the rollout of updated code in a controlled manner, or roll back to an earlier deployment version if necessary.
  1. **Service / Headless Service**

**Service:**

* Kubernetes services connect a set of pods to an abstracted service name and IP address. Services provide discovery and routing between pods. For example, services connect an application front-end to its backend, each of which running in separate deployments in a cluster.
* A Service is an [object](https://kubernetes.io/docs/concepts/overview/working-with-objects/#kubernetes-objects) (the same way that a Pod or a ConfigMap is an object). You can create, view or modify Service definitions using the Kubernetes API. Usually you use a tool such as kubectl to make those API calls for you.

**Headless Service:**

* In Kubernetes, [Services](https://kodekloud.com/blog/kubernetes-services/)provide a stable IP address for clients to connect to [Pods](https://kodekloud.com/blog/kubernetes-terms/). A client makes a request to the Service. Then the Service forwards that request to one of the Pods associated with it. The client does not know which Pod it is connected to, nor does it care about it.
* But what if the client wants to see the Pods’ IP addresses, so that it has complete control over which Pod(s) it can connect to? In such cases, Kubernetes provides the concept of headless Services.
* **A Kubernetes headless Service allows a client to connect to whichever Pod it prefers, *directly***. It doesn’t route the client request like a regular Service does. How to Configure a Kubernetes Headless Service?

apiVersion: v1

kind: Service

metadata:

name: my-web-app

spec:

selector:

app: my-web-app

ports:

port: 80

targetPort: 8080

This creates a ClusterIP Service called "my-web-app" that routes incoming traffic to Pods that have the label "app: my-web-app". The Service listens on port 80 and forwards traffic to the Pods' port 8080.

Note: The default Service type in Kubernetes, if no type is specified, is "ClusterIP".

**Volumes**

* A Kubernetes volume is a directory containing data, which can be accessed by containers in a Kubernetes pod. The location of the directory, the storage media that supports it, and its contents, depend on the specific type of volume being used.

## Types of Kubernetes Volumes..

* Kubernetes supports various volumes, allowing each pod to use several volume types simultaneously. Ephemeral volumes are bound to the pod's lifetime, while persistent volumes can persist beyond the pod's lifetime. It means Kubernetes destroys ephemeral volumes once their pod no longer exists while keeping the data of persistent volumes.

### Persistent Volumes

* Kubernetes provides a PersistentVolume subsystem with an API that abstracts storage provisioning and consumption. It works with two API resources—PersistentVolume (PV) and PersistentVolumeClaim (PVC).

**PersistentVolume (PV)**

* A PV is a storage resource located in the cluster. Administrators can manually provision PVs, and Kubernetes can use storage classes to dynamically provisioned PVs. Like volumes, PVs are plugins, but their lifecycle is independent of any pod using the PV.
* A PV works as an API object that captures the details of the storage implementation, including iSCSI, NFS, and cloud provider storage systems. It works similarly to a node but offers storage resources rather than computing.

**PersistentVolumeClaim (PVC)**

* A PVC is a storage request made by a user. It works similarly to a pod but consumes PV resources rather than node resources. A PVC can request specific storage resources, specifying size access modes such as ReadWriteOnce, ReadWriteMany, and ReadOnlyMany.
* PVCs enable users to consume abstract storage resources, but users typically need PVs with varying properties for different problems. This is why cluster administrators often need to offer varying PVs that differ in more than size and access modes. They can do that without exposing users to implementation details through StorageClass resources.

### Ephemeral Volumes

* Ephemeral volumes do not store data persistently across restarts. These volumes are bound to the pod's lifetime, which means they are created and deleted along with the pod. It enables stopping and restarting pods without limiting them to the availability of persistent volume.
* Ephemeral volumes are simple to deploy and manage. You can specify them inline in the pod spec. Ephemeral volumes are ideal for applications that do not require persistent storage, like caching services

### EmptyDir Volumes

* An emptyDir volume is created when Kubernetes assigns a pod to a node. The lifespan of this volume is tied to a pod's lifecycle existing on that specific node. An emptyDir volume recreates when containers restart or crash. However, the data in this volume is deleted and lost when the pod is removed
* from the node, crashes, or dies.
* After creating an emptyDir volume, you can declare the volume type name as a field in the pod manifest file. It shows under the volume property section with empty curly braces{} as the value. EmptyDir volumes are suitable mainly for temporary data storage. For example, you can use it for scratch space, like a disk-based merge.
* You can store emptyDir volumes on the medium backing the node. For example, you can use network storage or SSD. Alternatively, you can set "memory" in the emptyDir.medium field and Kubernetes will mount a RAM-backed filesystem (tmpfs). Note that Kubernetes clears tmpfs on node reboot.
  1. **Stateful Sets**
* StatefulSet is the workload API object used to manage stateful applications. Manages the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods. Like a Deployment, a StatefulSet manages Pods that are based on an identical container spec.
  1. **Namespaces**
* Namespaces are a way to organize clusters into virtual sub-clusters — they can be helpful when different teams or projects share a Kubernetes cluster. Any number of namespaces are supported within a cluster, each logically separated from others but with the ability to communicate with each other.

**---------------------------------------------------------------------------------------------------------**