Kubernetes Command

Commands to delete the worker node

Kubectl get nodes -o wide

kubectl drain oel75.localdomain(NodeName) --delete-local-data --force --ignore-daemonsets

On the worker node run below command

kubeadm reset

Once done run the below command on master node

kubectl delete node oel75.localdomain

Now run these command on worker node to completely uninstall the Kubernetes on the worker node

sudo yum remove kubeadm kubectl kubelet kubernetes-cni kube\*

sudo yum autoremove

sudo rm -rf ~/.kube

ii) Generate a token

kubeadm token generate

 Generate node join command

kubeadm token create hp9b0k.1g9tqz8vkf78ucwf --print-join-command

Installing Calico Network Utility on Kubernetes Cluster.

Note

Please use below command to enable networking on Kubernetes cluster using calico

Command 1:

kubectl create -f <https://raw.githubusercontent.com/projectcalico/calico/v3.24.5/manifests/tigera-operator.yaml>

Command 2:

kubectl create -f <https://raw.githubusercontent.com/projectcalico/calico/v3.24.5/manifests/custom-resources.yaml>

Check the status of the nodes on the cluster if its ready after running the above command. It might take some time. So, give it a minute or two or it to be enabled.

Command: kubectl get nodes

Command to create the pod

Command: kubectl run pod1(Name of your choice) –image nginx

Command to list the pod on the VM:

Command: kubectl get pods

Command to describe the pod:

Command: kubectl describe pod

Command to check the pod log:

Command: kubectl logs pod1(Name of the pods)

Command to get inside the pod:

Command: kubectl exec -it pod1(NameOfThePod) /bin/bash

Command to delete the pod:

Command: kubectl delete pod pod1(NameOfThePod)

Command to list out the namespaces on the server:

Note: Namespaces are like Git branches.

Command : kubectl get namespaces

Command to get inside a specific namespace:

Command: kubectl get pods -n(DenotesNameSpace) kube-system(NameSpace Name)

Command to create a Namespace

Command: kubectl create namespace dev(NameOfYourChoice)

Command to list out the pods created in the above namespaces

Command: kubectl get pods - -namespace=dev

Command to delete the Namespace:

Command: kubectl delete namespace dev(NameOfTheNameSpace)

Replicaset In Kubernetes:

Before we talk about replicaset in Kubernetes. We will discuss about ‘labels and ‘selector’. Labels are used to group together our pods running across the cluster. For Example let us imagine a scenario where you have frontend and backend for your application, Frontend being the nginx and backend being your database (like Mariadb, mysql etc). As we will multiple instances of our pod running across our cluster for high availability and backup. We can group all these pods in two groups lets say frontend and backend like the example below:

Example: labels:

name: Nginxfrontendapplication /Mysqldb

type: frontend/backend



What is Replicaset:

In order for your Pod to be HA(High Availability) and back up in case of any issue. Using the replicaset concept of Kubernetes we will be able to run multiple instances of same image of the pod across our cluster.

Please find the below script for replicaset save the file with an extension and execute the file as mentioned below. Replicaset will ensure you will have the desired pod running across our cluster even when you try delete the one of the pod it will bring the pod to desired number mentioned in our replicaset. It uses a replicaset controller to achieve this task until unless you delete the replicaset manually as mentioned below

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: example123

labels:

app: replicaapp3

spec:

template:

metadata:

labels:

app: replicaapp3

spec:

containers:

- name: nginxpod

image: nginx

replicas: 3

selector:

matchLabels:

app: replicaapp3

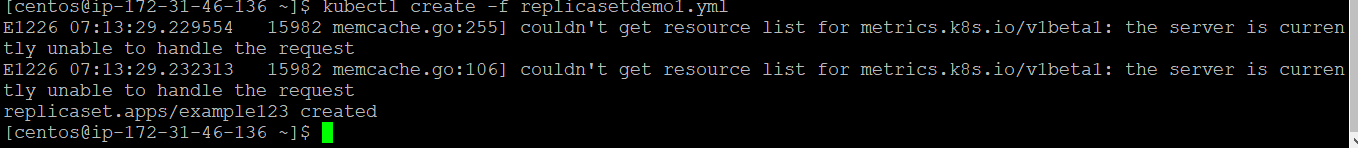
command: kubectl create -f replicasetdemo1.yml ( To execute the file)

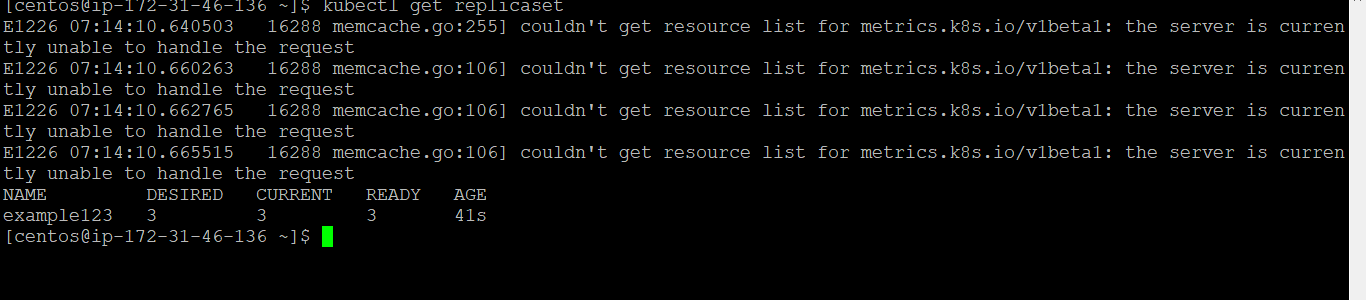
command: kubectl get replicaset ( To view the replicaset created on your VM)

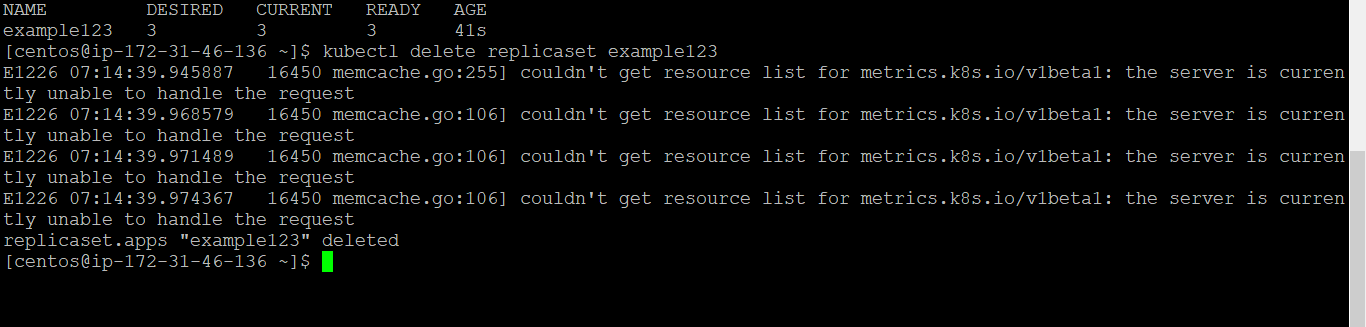
command: kubectl delete replicaset example123 (To delete the replicaset)

command: kubectl scale -–replicas=6 –f replicaset-definition.yml ( Scale up the replicas pod without making any changes to the manifestation file)

command: kubectl replace -f replicasetdemo1.yml( If you make any changes inside the manifestation file run this command to update the changes)

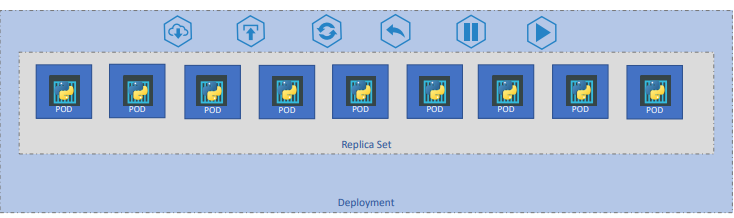






Deployment in Kubernetes:

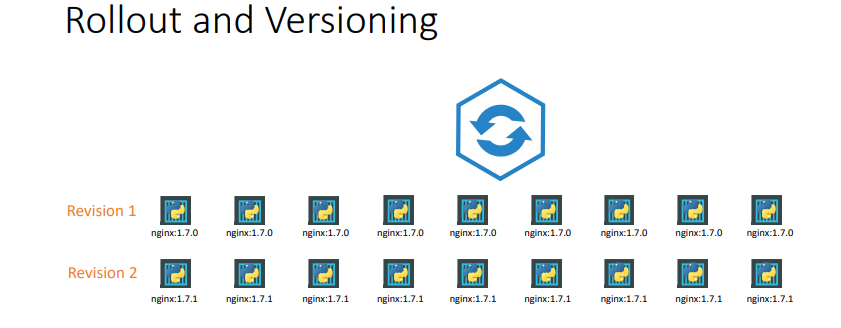
Deployment is the upper most abstraction layer over the pod it is above replicaset as depicted in the image below:



In the production we do deployment instead of creating single pod or replicaset as we were doing till now. Deployment object in Kubernetes follows the rolledout and rollback concept of the deployment.

Rolledout: This rolled out concept of deployment help us achieve zero down time. Let’s say you have ten pods each running nginx container version 1.7.1. You have got the task to upgrade the nginx image to 1.7.5. There are two ways we can achieve this task either we can bring all the pod down and upgrade the nginx version and bring it up which might leads to downtime or we can use rolledout concept of deployment where we will be bring one pod down upgrade the image version and bring up that pod and go to other pod for upgradation in this way we will not have any downtime. Kubernetes deployment object will follow this rolled out process by default.

Roll back: We take the same example as above i.e. we have to upgrade our nginx image version. For instance, let us imagine that the upgraded image version is not compatible with our infrastructure. In those scenarios we want to go back to our previous version. Using Kubernetes object deployment is very easy to roll back to the previous version.



Script for deployment:

apiVersion: apps/v1

kind: Deployment

metadata:

name: example1234

labels:

app: deploymentdemo

spec:

template:

metadata:

labels:

app: deploymentdemo

spec:

containers:

- name: nginxpod

image: nginx

replicas: 3

selector:

matchLabels:

app: deploymentdemo

Save the file and run the below commands to execute the file.

Commands: > kubectl apply –f deployment-definition.yml( command to execute the deployment)

Command: kubectl get deployments ( To view the deployments created on our VM)

Command: kubectl delete deployment (deploymentname) (To delete the particular deployment)

Command: kubectl rollout status deployment (deploymentname) ( To view our rollout status)

Command: > kubectl rollout undo deployment (deploymentname) (To rollback our deployment)

Services in Kubernetes:

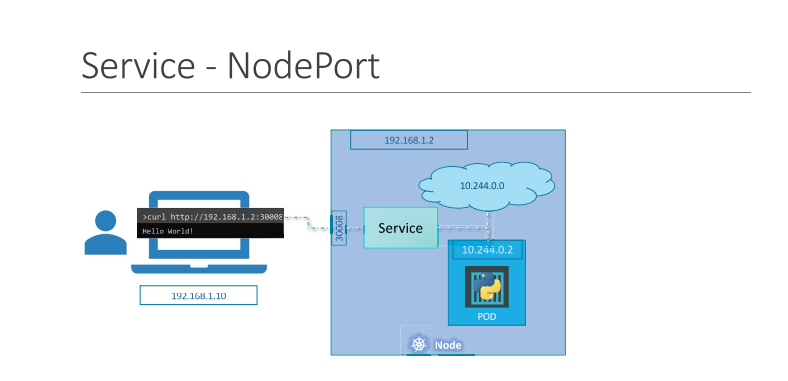
Like we have port forwarding, binding port in docker to make our container accessible to end users or over the internet. We have something called as services in the world of Kubernetes to achieve the same tasks. There are three types of services in Kubernetes.

Services in Kubernetes:

1. NodePort
2. ClusterIP
3. LoadBalancer

NodePort: As illustrated in the image below nodeport will make our pod accessible to the outside world this will be achieve by creating a service as you see in the image below. As you have in the image below the service will be created outside the pod and it will have its own IP address which will be provided by the Kube-Proxy. For example let us consider a Pod running nginx container which has a internal port 80. So the service which gets created will also have the port 80 which is called as just port and the port which is internal to the container i.e. 80 as well is called as targetPort. But still we haven’t make our Pod accessible to the outside world this will be achieve using the NodePort as you see in the image below.

Nodeport will be in a range of 30000-32767. We can chose any range as per our convenience or we can let Kubernetes decide which random port to chose from the range above.



Script for NodePort Service:

apiVersion: v1

kind: Service

metadata:

name: myapp-service

spec:

type: NodePort

ports:

* port: 80(ServicePort)

targetPort: 80 (Internal container 80)

nodeport: 30008 (NodePort in the range

selector:

app: myapp(Label of the pod for which you want to attach this service)

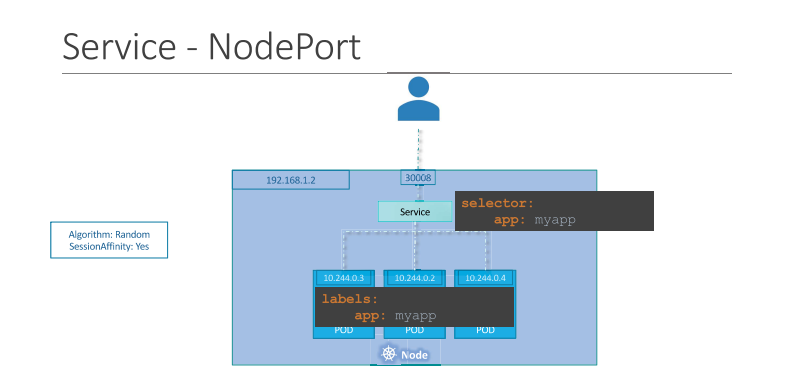
save the file with .yml extension and execute the script using below commands;  
command: kubectl create -f nodeportdemo.yml

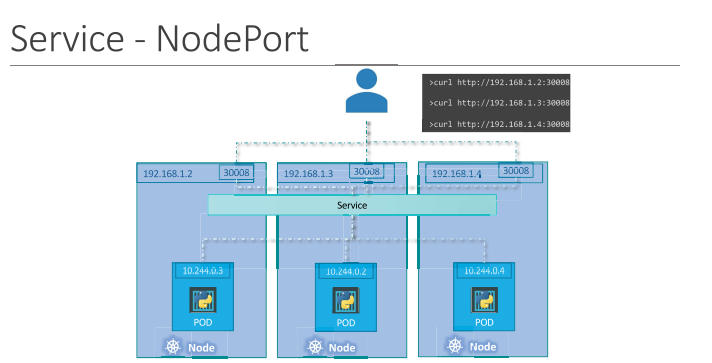
command: kubectl get services ( To view all the services created on our vm)

command: kubectl delete service (servicename) (To delete the service )

To access the above pod use the PublicIPOfVM:30008(Make sure the port is open on your VM)

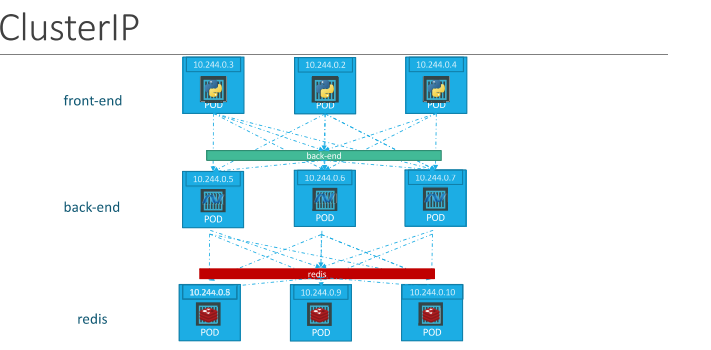
Note: By default, load balancing will be taken care by Kubernetes across all the cluster. That is if we have the same pod running across the cluster, we can access those pod with the same nodeport along with PublicIP of the VM where it is running.





ClusterIP in Kubernetes

The second is CluserIP, and in this case, the service creates a virtual IP inside the cluster to enable communication between different services, such as a set of frontend servers to a set of backend servers.



Script for ClusterIP:

apiVersion: v1

kind: Service

metadata:

name: myapp-service1

spec:

type: ClusterIP

ports:

* port: 80(ServicePort)

targetPort: 80 (Internal container 80)

selector:

app: myapp(Label of the pod for which you want to attach this service)

command: use the same command as Nodeport to execute the script and view the services created

LoadBalancing in Kubernetes

The third type is a Load Balancer, where it provisions a load balancer for our application in supported cloud providers.

A good example of that would be to distribute load across the different web servers in your frontend tier.

Script for Load Balancer:

apiVersion: v1

kind: Service

metadata:

name: myapp-service2

spec:

type: LoadBalancer

ports:

* port: 80(ServicePort)

targetPort: 80 (Internal container 80)

selector:

app: myapp(Label of the pod for which you want to attach this service)

command: use the same command as Nodeport to execute the script and view the services created

Helm Chart:

Helm helps you manage Kubernetes applications — Helm Charts help you define, install, and upgrade even the most complex Kubernetes application.

Charts are easy to create, version, share, and publish — so start using Helm and stop the copy-and-paste.

Helm Installation on Centos:

Note: <https://helm.sh/docs/intro/install/>

**$** curl -fsSL -o get\_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3

**$** chmod 700 get\_helm.sh

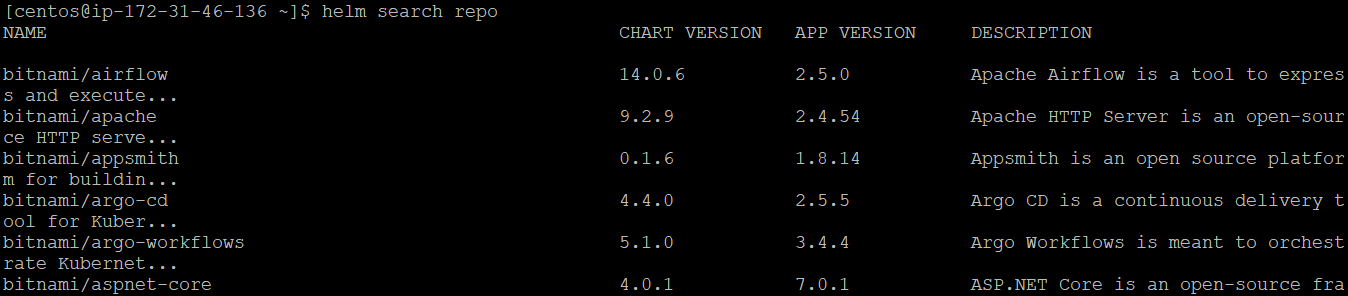
**$** ./get\_helm.sh

We will be able to use helm chart from any one of these github, bitnami, artifacthub, and the standard service office documentation page services like, nginx, Prometheus etc.

Note: <https://github.com/bitnami/charts>

Commands: $ helm repo add bitnami https://charts.bitnami.com/bitnami

$ helm search repo bitnami



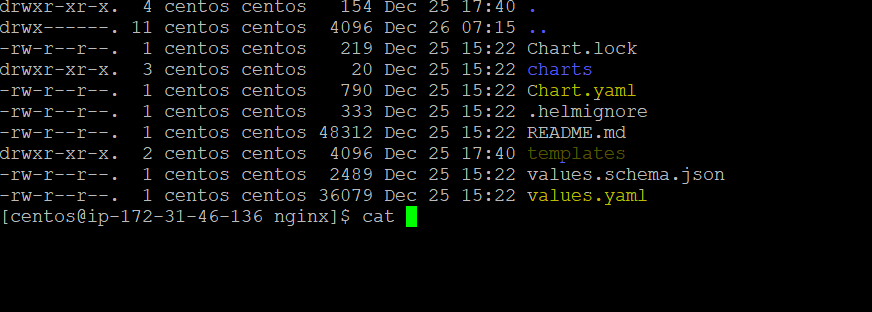
helm pull --untar bitnami/nginx ( It will untar the chart and save on the current directory as nginx directory )

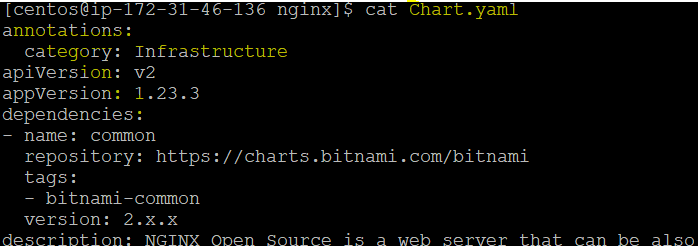
Note: the Helm chart will have three important files values.yml, charts.yml and templatese(directory)

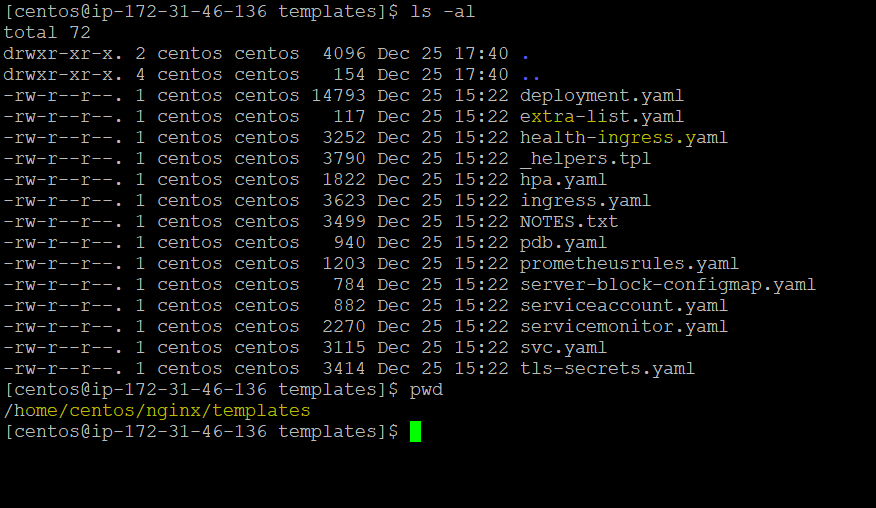
Charts.yml ( It is the file which contains the description regarding that particular helm chart such as developer name,date , what is the use of the chart etc. in a nutshell it will contain only the meta data of that particular chart)

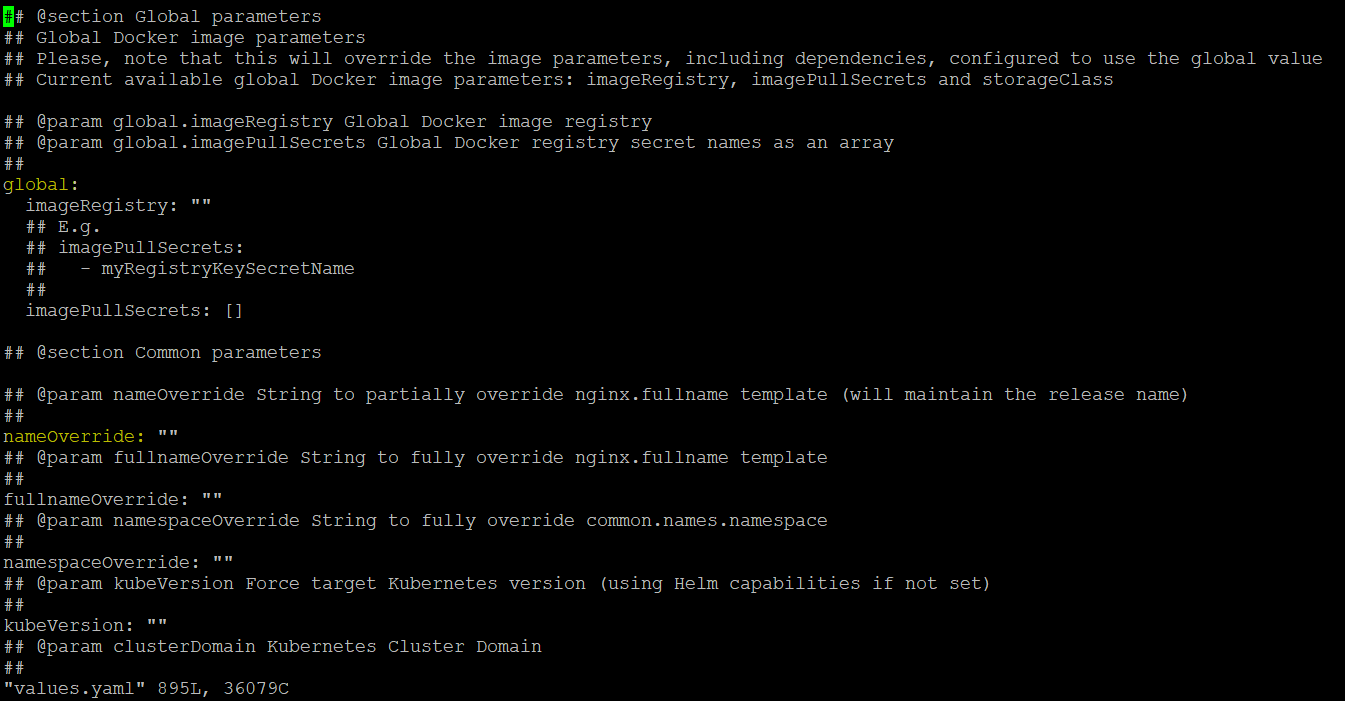
Values.yml( it is like a variable file where we have all the variables store for the configuration file)

Templates(folder) contains all the configuration file for the helm chart



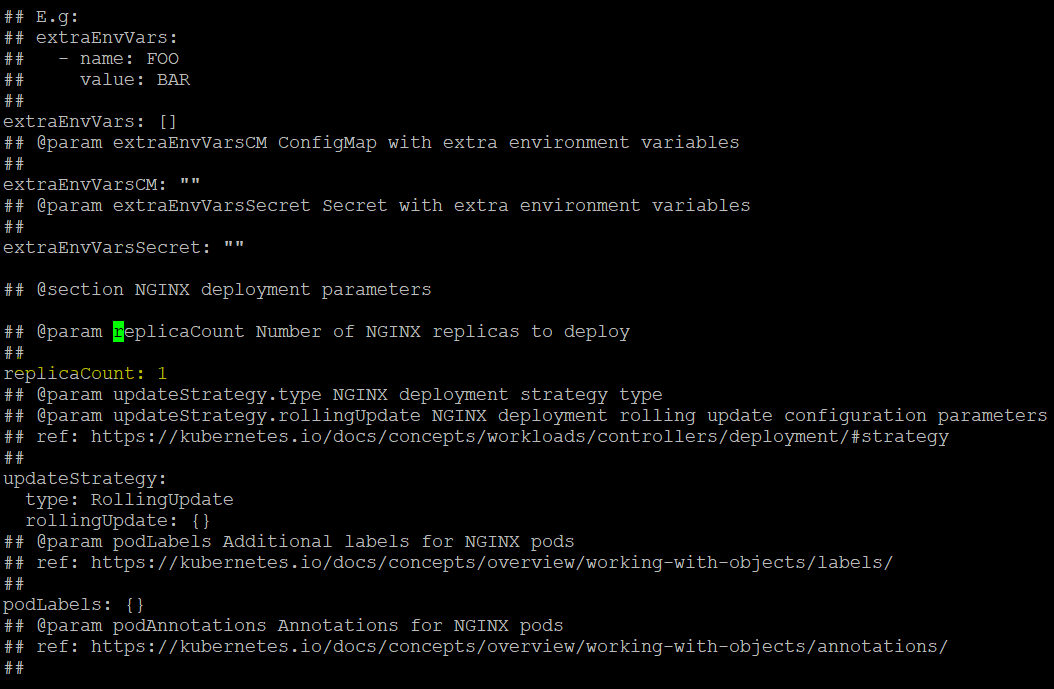


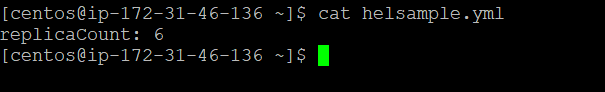




Note: The best way to execute the helm chart is to not make any changes inside this file rather just pick the variables you want to edit and create a separate manifestation and execute as below. The file created will have more precedence than those file.

In this example we are trying to change the replicacount to 6 as we want to have three pods instead of one of nginx. We have created a separate file with .yml as shown in the image below with replicaCount: 6





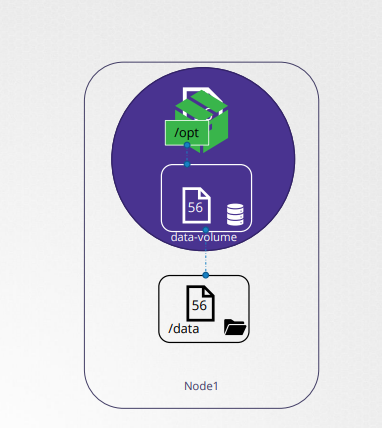
Command> helm install omer (Name of our choice) bitnami/nginx -f helsample.yml

Command: helm list (To list out all the pod created using helm)

Command: helm delete omer(name of the helm) (note: it will delete all the pod and all the object which was created using helm)

Persistent Volumes in Kubernetes :

Persistent volumes in kubernetes , we know from the docker session that once the container is deleted all the information related to that container will be deleted along with it. Same concept is applicable for the pods once the pods are deleted all the data along with that pod will be deleted. So, we can use persistent volume to have a backup for the pod.



Script for Volume:

apiVersion: v1

kind: Pod

metadata:

name: random-number-generator

spec:

containers:

- image: centos

name: centos

command: ["/bin/sh","-c"]

args: ["shuf -i 0-100 -n 1 >> /opt/number.out;"]

volumeMounts:

- mountPath: /opt

name: data-volume

volumes:

- name: data-volume

hostPath:

path: /data

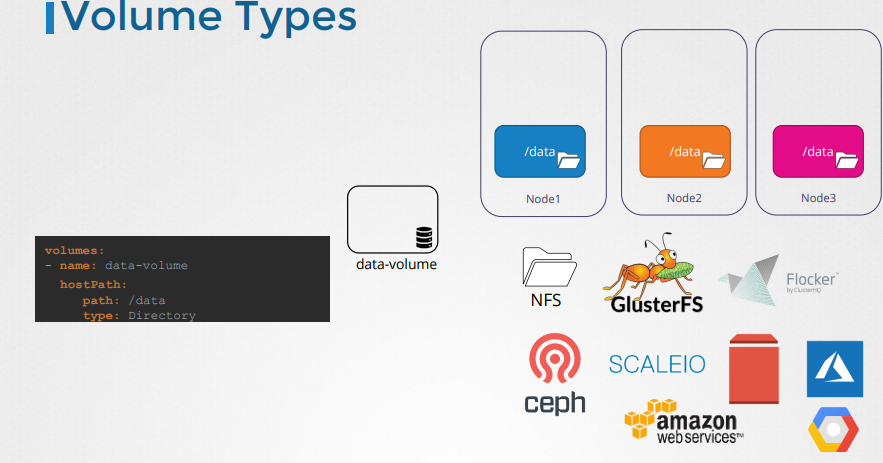
type: Directory

command: kubectl create -f volume.yml ( To execute the script)

command: kubectl get pv (To list the persistent volume created on our VM)

command: kubectl delete pv (volumeName)(to delete the persistent volume)

Note: We can use internal hard disk to store the persistent volume data or we can above remote storage like shown in the image below



In the below example we will use how to use ebs(Amazon elastic block storage as our remote storage for persistent volume data)

Script:

apiVersion: v1

kind: PersistentVolume

metadata:

name: newvol1

labels:

type: ebsvol1

spec:

capacity:

storage: 3Gi

accessModes:

- ReadWriteMany

awsElasticBlockStore:

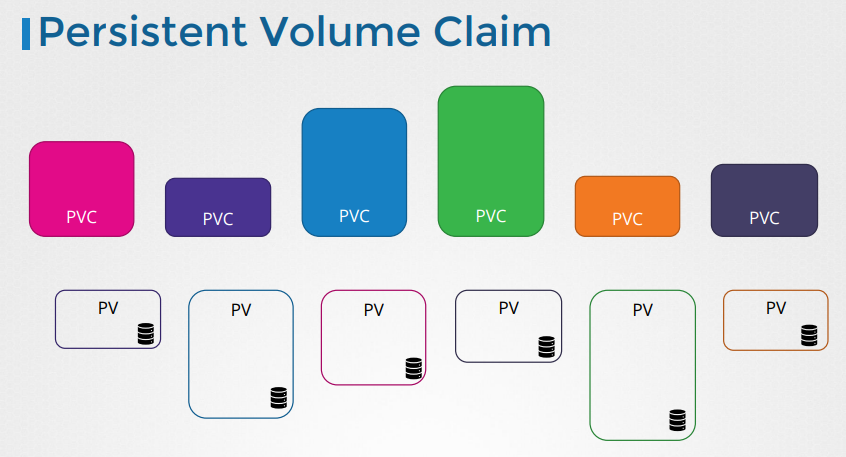
volumeID: vol-0ae25de83035fb1b7 (We have to create elastic block storage on aws once we do that we will volume id )

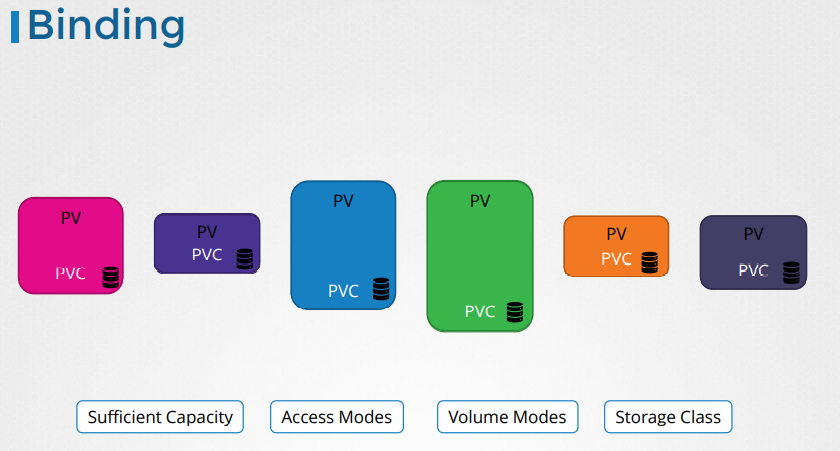
fsType: ext4

command: Use the above command like in first example of persistent volume)

Persistence Volume Claim

After we have created the claim we have to bind the volume with the claim so that it can’t be used by any other pods





Script for persistence volume claim:

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: promvolume

spec:

accessModes:

- ReadWriteMany

resources:

requests:

storage: 2Gi

selector:

matchLabels:

type: local (Label of the persistence volume)

commands: kubectl create -f claim.yml (to execute the file)

command: kubectl get pvc (to list all the persistence volume claims on our vm0

command: kubectl delete pvc (To delete the persistence volume claim)

command: kubectl describe pvc (To describe the claim)