Kubemaster - 192.168.xxx.xxx/24

Kubenode01 - 192.168.xxx.xxx/24

Kubenode02 - 192.168.xxx.xxx/24

For pods we use

10.244.0.0/16

## 1) setup VM

============

Use Vagrant or can setup VM’s separately and assign a static IP address to each VM

## 2) Install Docker

================

Follow the docker installation setup on docker website

## 3) Install kubeadm, kubelet, kubectl

==================================

Follow the instructions on kubernetes.io

## 4) Initialize the cluster

=====================

kubeadm init --pod-network-cidr 10.244.0.0/16 --apiserver-advertise-address=192.168.56.2

[init] Using Kubernetes version: v1.21.0

[preflight] Running pre-flight checks

[preflight] Pulling images required for setting up a Kubernetes cluster

[preflight] This might take a minute or two, depending on the speed of your internet connection

[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'

[certs] Using certificateDir folder "/etc/kubernetes/pki"

[certs] Generating "ca" certificate and key

[certs] Generating "apiserver" certificate and key

[certs] apiserver serving cert is signed for DNS names [kubemaster kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 192.168.56.2]

[certs] Generating "apiserver-kubelet-client" certificate and key

[certs] Generating "front-proxy-ca" certificate and key

[certs] Generating "front-proxy-client" certificate and key

[certs] Generating "etcd/ca" certificate and key

[certs] Generating "etcd/server" certificate and key

[certs] etcd/server serving cert is signed for DNS names [kubemaster localhost] and IPs [192.168.56.2 127.0.0.1 ::1]

[certs] Generating "etcd/peer" certificate and key

[certs] etcd/peer serving cert is signed for DNS names [kubemaster localhost] and IPs [192.168.56.2 127.0.0.1 ::1]

[certs] Generating "etcd/healthcheck-client" certificate and key

[certs] Generating "apiserver-etcd-client" certificate and key

[certs] Generating "sa" key and public key

[kubeconfig] Using kubeconfig folder "/etc/kubernetes"

[kubeconfig] Writing "admin.conf" kubeconfig file

[kubeconfig] Writing "kubelet.conf" kubeconfig file

[kubeconfig] Writing "controller-manager.conf" kubeconfig file

[kubeconfig] Writing "scheduler.conf" kubeconfig file

[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"

[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"

[kubelet-start] Starting the kubelet

[control-plane] Using manifest folder "/etc/kubernetes/manifests"

[control-plane] Creating static Pod manifest for "kube-apiserver"

[control-plane] Creating static Pod manifest for "kube-controller-manager"

[control-plane] Creating static Pod manifest for "kube-scheduler"

[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"

[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests". This can take up to 4m0s

[kubelet-check] Initial timeout of 40s passed.

[apiclient] All control plane components are healthy after 68.504463 seconds

[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace

[kubelet] Creating a ConfigMap "kubelet-config-1.21" in namespace kube-system with the configuration for the kubelets in the cluster

[upload-certs] Skipping phase. Please see --upload-certs

[mark-control-plane] Marking the node kubemaster as control-plane by adding the labels: [node-role.kubernetes.io/master(deprecated) node-role.kubernetes.io/control-plane node.kubernetes.io/exclude-from-external-load-balancers]

[mark-control-plane] Marking the node kubemaster as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule]

[bootstrap-token] Using token: 6d1d5y.omq273340jdgj2fp

[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles

[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to get nodes

[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials

[bootstrap-token] configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token

[bootstrap-token] configured RBAC rules to allow certificate rotation for all node client certificates in the cluster

[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace

[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key

[addons] Applied essential addon: CoreDNS

[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

export KUBECONFIG=/etc/kubernetes/admin.conf

**You should now deploy a pod network to the cluster.**

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:

https://kubernetes.io/docs/concepts/cluster-administration/addons/

## 5) Setup pod Network

=====================

Run the below command to setup weave network

kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"

## 6) Join the worker nodes to kube masternode

===========================================

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.56.2:6443 --token tvyvyc.odf4xwkvu8awq0yb \ --discovery-token-ca-cert-hash sha256:3fa3797095c3f1ca9b813c660d20265a212e645c4b58b7c2fd59a7132ff50a6c

the token indicated in blue will change every 24 hrs. After initializing the cluster need to generate a new token if you want a new worker node to join the cluster after 24 hours.

## 7) Service Kubernetes

Used to communicate between different microservices (PODS/Containers) or to interact outside pod network

Terminology used in kubernetes services

**Target Port** - Port on the POD

**Port** - Port on the service

Node Port - Port on the actual NODE (VM) - 30000 to 32768

To expose a service to a pod/deployment

kubectl expose pod <Pod Name> --type="NodePort" --port 80

kubectl expose deployment <deployment name> --type="NodePort" --port 80 –namespace=<namespace name>

===================================

## 8) How to Login to POD shell

kubectl exec --stdin --tty <Pod\_Name> -- /bin/bash

kubectl -n <Namespace> exec -it <pod name> cat /path/ (log file path )

## 9) Replicate sets

Replication sets has a selector to identify labels for the pods. Replication controller doesn’t have labels

If a pod is running on a cluster using a label and a replica set of 5 replicas is created with a selector using same label, it will only create 4 extra pods instead of 5. As the existing pod also using the same label

## 10) IP Address of the POD and Container and Node

Kubectl get pods -o wide (To get the POD details including IP address and assigned Nodes)

docker ls (To get the container ID)

docker inspect --format '{{ .State.Pid }}' <ContainerID> (To get the PID of the container)

nsenter -t dc937d623598 -n ip addr (To find the veth and IP add of that container)

## 11) Kubectl commands

Kubectl get all –all-namespaces – to see all pods, services, deployments, replicasets across all namespaces

Kubectl run <pod name> --image=<imagename> --namespace=<namespace name> - to create a pod in a particular namespace. If not, it will get created in default namespace

Kubectl create -f <filename>.yml – alternatively can create a pods, services, deployments, replicasets using a YAML files

Kubectl edit pod <pod-name> --image=<updated image name> - to edit a running pod with a new image

Or can edit the YAML file and can apply the changes

kubectl describe node <nodename> - to check the details of the nodes

kubectl get nodes –show-labels – to see all the nodes labels

## 11) create a deployment/pod/services YAML files using DRYRUN – to verify if the format is correct – Very useful during the exam

kubectl run deployment nginx-deploy-demo --image=nginx --port 80 --dry-run=client -o yaml

kubectl create deployment nginx-deploy-demo --image=nginx --port 80 --replicas=3 --dry-run=client -o yaml

kubectl run pod nginx-deploy-demo --image=nginx --port 80 --dry-run=client -o yaml

kubectl run replicaset nginx-replicademo --image=nginx --dry-run=client -o yaml

kubectl expose deployment.apps/nginx-deploy --type="NodePort" --port 80 --namespace=dev --dry-run=client -o yaml

## 12) Labels Selectors and annotation

Labels under metadata – to group a certain type of pods example frontend, DB, webtier,

Selectors: under the specs to identify the pods generally used in a service.yml

Annotation: under metadata to refer to a particular build

**kubectl get pods --selector env=dev**

kubectl get pods --selector tier=frontend,bu=finance,env=prod – its like a AND operator

kubectl label

## 13) Taints and Tolerations

NODE – will have taints

PODS – will have toleration

Scheduler tries to place the pods on available nodes – by default PODS have no tolerations if there is a taint applied on a NODE, no pods can be placed there unless they are configured with tolerations

Taints and tolerations won’t restrict a pod to go to one particular node. It will only add tolerations to a pod for a specific node with a taint. This pod which has a toleration set can also be placed in any pod without taints.

Taints and tolerations doesn’t tell a POD to go to a particular node, it will just tell the NODE to accept certain PODS with tolerations to the taint. For a POD to go to a particular NODE Kubernetes uses affinity rules

By default all kubernetes master nodes have a taint by default - Taints: node-role.kubernetes.io/master:NoSchedule

Kubectl taint nodes <NodeName> key=value:taint-effect

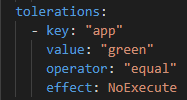
Types of taints effects on Nodes –

a) PreferNoSchedule (Schedulers will try to avoid placing a POD on that NODE)

b) NoSchedule (Pods won’t be placed on the Node – Default for master NODE),

c) NoExecute (New Pods will not be placed on the POD and existing pods without tolerations will be removed from the NODE)

Adding Pod definition files with tolerations



## 14) Node Selectors & Node Affinity

NODES have labels

PODS will be configured with nodeSelector

Nodes will be set with labels – kubectl label node <nodename> key:value

Then can use node Selector in a the POD/Deployment config yaml file to select a specific node for the PODS



but node selectors have limitations – if a pod should be placed on Node with an OR/NOT/AND operator condition. Then need to user node affinity rules

TIP: If you want a specific PODs to be placed on a specific NODE and that node should only have those PODS – a combinations of taints/tolerations and Node affinity rules (Labels) can be used together.

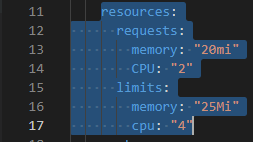
## 15) Resource Limits

CPU – MEM – DISK – Minimum requirement

0.5 – 256Mi

By default Kubernetes sets a limit of 1 CPU and 512Mi of cpu and memory limits respectively. If you need to change the limits can add to the Deployment or pod yaml file to set the limits

Can limit by adding under container



## 16) POD editing Highly important

Cannot edit an existing POD except the below parameters it will allow to edit – image, tolerations

But still need to edit an existing POD can run the below command

Kubectl edit pod <pod\_name>

Make changes to required parameters, and it wont allow you to change it will just give error and will save the file saved to a temporary location (Take note of that location) – delete the existing pod and then create a new POD from the file saved in the temporary location

Or can save the exiting pod using -o YAML

Kubectl run pod <pod name> -o yaml >> pod.yaml – this will save the pod in yaml format and can edit and delete the exiting pod and create as new

In case the pods are created using a deployment, can edit the deployment by using below command

Kubectl edit deployment <deployment-name> - once edit all pods will be created again

to change the current namespace

kubectl config set-context --current --namespace=<insert-namespace-name-here>

## 17) Deamon sets

Deamon sets are similar to replica sets

REPLICA sets ensure that required number of replicas running across all nodes, but deamonset ensure that one particular POD is running on each NODE in the cluster

Used cases – like a monitoring agent, network agent, kubeproxy – these pods created using deamon sets are required to run on each node – Deamonset ensures that this is met.

## 18) Static POD

Creating a static pod on a worker node. Kubelet should be able to manage the container on its own

Need to create a yaml file in that particular path where the kubelet looks for the pod.yaml files

If cluster setup using kubeadm tool the static pod path is under config.yaml

Check here first - /lib/systemd/system

For config.yaml check this path /var/lib/

Executable - /usr/local/bin

Creating a yaml file kubectl run static-busybox --image=busybox --command sleep=1000 --dry-run=client -o yaml >> busybox.yml

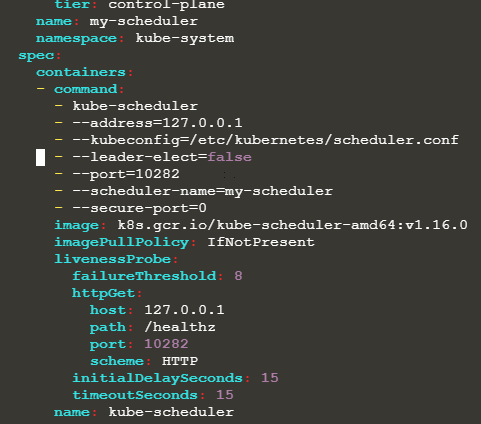
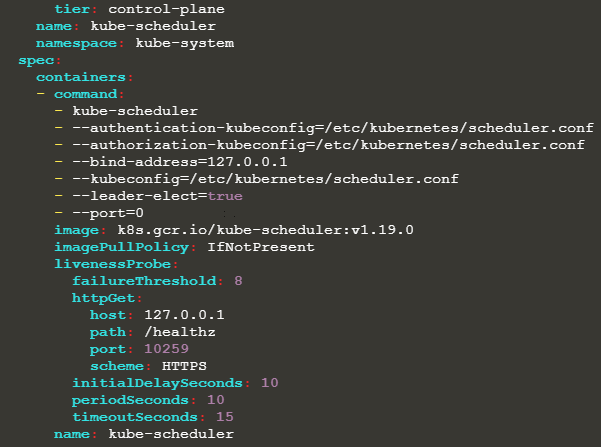
## 19) Multiple schedulers

If you want a custom scheduler can setup custom scheduler based on the requirement. But need to create

* --scheduler-name=<Scheduler name> - to differentiate between the default scheduler and custom scheduler
* --leader-elect=true – this should be set to false in case of multiple schedulers (In HA setup)
* --lock-object-name=<Custom schedulername> this wont allow the custom scheduler to participate in leader elect process

For a POD to use the custom scheduler, need to configure the schedulerName on the POD

Custom Scheduler Original scheduler

## 20) Monitoring and logging

Kubelet uses Cadvisor which is used for monitoring. Kubelet runs on each node to manager the pods. For monitoring need to deploy pods from metric server

<https://github.com/kodekloudhub/kubernetes-metrics-server.git>

Kubectl top node

Kubectp top pod

Kubectl log -f <pod name> - to tail the logs for a particular pod

## 21) Application Life Cycle Management

Rolling updates and rollback

Two methods –

1. Recreate - all containers will be updated same time – Downtime required
2. Rolling Update – One container at a time – no downtime – this is the default strategy if the nothing is mentioned

Can use declarative way **kubectl apply -f <deployment.yaml>** - this is changing the yaml file and it will be a permanent upgrade – kubectl edit deployment.yaml (to edit the image file before rollout)

Can use imperative way – **kubectl set image deployment/<my deployment> \<current version>=<New version>** - this way it wont modify the yaml file and yaml file will still have old image mentioned

Rollback command – kubectl rollout undo deployment/<my deployment>

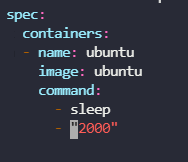
To check status

Kubectl rollout status <deployment>/<deployment name>

Kubectl rollout history <deployment>/<deployment name>

## 22) Configure applications

Should know – commands and arguments, environment variables, configuring secrets



**Config map definition file**

Create a config map and map that config map in a POD to use the corresponding environment variables

kubectl get configmaps

Kubectl create configmap <configmap-name> --from-literal=<Key>=Value –dry-run=client -o yaml

To pass the config map in a pod just use envFrom

Secrets – similar to config files but store data in a encrypted format esp for **passwords**

Kubectl create secret generic <secret name> --from-literal=<key>=<value> --from-literal=<key>=<value> --dry-run=client -o yaml

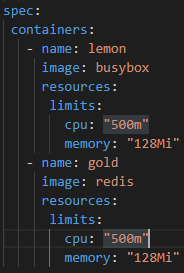
Kubectl get secrets

Kubectl describe secrets

## 23) Multiple container pods

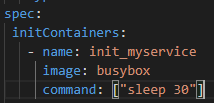
Types of Multiple container - Sidecar, Ambassador, Adapter

This Multi container pods are used if one container needs a helper. Mostly for forwarding the logs.



## 24) init containers

These are used inside a POD along with main container, kubernetes will execute the init containers first and once its successfully completed then it will run the main container. Until the init container doesn’t succeed kubernetes will try to restart but wont start main container.



## 25) cluster maintenance

OS upgrades of the nodes, Kubernetes software upgrade, backup and restore of ETCD

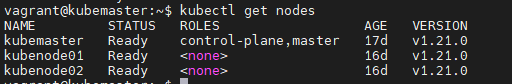
Kubectl drain <node name> - this is useful to drain the pods from this node before shutdown during schedule activity like OS upgrade or system reboot

1. Need to pass this argument for the above command --ignore-daemonsets as it wont evict kubeproxy and network solution which are deployed using daemonsets.
2. Cannot drain a POD which is not part of any replica set, if there a standalone pod need to force the pod to drain from the node using the argument --force

Kubectl uncordon <Node Name> - this is used to when the node comes back up need to run this so the scheduler can deploy the pods on this node.

Kubectl cordon <Node Name> - command is helpful to make sure the scheduler doesn’t deploy any pods on this NODE. It wont drain the existing pods but wont schedule any new ones.

**kubernetes version – below displays the kubelet version**



to upgrade using the kubeadm run the command kubeadm upgrade plan it will display the current version and the latest version

Upgrade path (one minor version at a time) –

**Below two on master node**

1. kubeadm first – apt-get upgrade -y kubeadm=<latest version>
2. upgrade cluster components – kube api server, etcd, controller and scheduler

kubeadm upgrade apply <latest version>

Below on master node first – then on each worker node one at a time – need to drain the node before upgrade as it requires restart.

1. update the kubelet apt-get upgrade -y kubelet=<Latest Version>
2. HINT – if upgrade doesn’t work because of dependency- can use **apt-get install**

## 26) backup and restore strategy

Backup using kubeapiserver

Kubectl get all –all-namespaces -o yaml >> all\_config.yml

Backup etcd - /etc/manifest/kubernetes/etcd.yaml take note of –data-dir directory

Run the below command to take snapshot (For more details on ETCD backup refer to Section 38)

ETCDCTL\_API=3 etcdctl snapshot save snapshot\_4may2021.db --endpoints=https://127.0.0.1:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key

To check status of the backup file

ETCDCTL\_API=3 etcdctl snapshot status snapshot\_4may2021.db

Restore DB

1. Stop kubeapiserver service
2. ETCDCTL\_API=3 etcdctl snapshot restore <Path of the restore file> --data-dir <Path of the dataDIR from the etcd.service file or the yaml file(if installed suing kubeadm>

## 27) Security (Very Important)

Symmetric encryption and asymmetric encryption

Key pair – private key and public key

Certificate authority – ca.key and ca.crt

Kube-apiserver – Server.key and server.crt || client.key and client.crt

Etcd - Server.key and server.crt

Kubelet - Server.key and server.crt

Scheduler – client.key and client.crt

Controller – client.key and client.crt

User/service account – client.key and client.crt

openssl x509 -in server.crt -text -noout – to read the certificate content using opessl

docker ps -a – to list all the containers

docker logs containerID – this will help to find when kubernetes is down to see whats the issue on the kubernetes pods

## 28) Certificates – API

Certificates signing is done by controller managaer components called **CSR-Approving** and **CSR signing**

Certificate signing request is a yaml file

Master node itselt is a ca server – as the ca certs are saved on the master node.

Create a .CSR file from the key

Then create a yaml file with the base64 encoded csr in the yaml and create a certificate signing request.

# to convert the csr content to base64 to add to the yaml file

cat myuser.csr | base64 | tr -d "\n"

Once request is created check the status using kubectl get csr it will be in poending state

Use kubectl certificate approve <csr name> - to approve the pending certs

kubectl get csr agent-smith -o yaml to check the contents of the csr file in yaml format

kubectl certificate deny agent-smith – to deny any request

kubectl delete csr agent-smith – to delete the csr

## 29) Kubeconfig

Everytime you query a apiserver you have to pass the cert information for authorization. This authorization is handled by the kubeconfig file under the home directory $HOME/.kube/config

Kubectl config view command – this will show the default kubeconfig under the &HOME/.kube/config

Kubectl config –kubeconfig=<custom config file> view to view the contents of custom config file

Kubectl config use-context <context name> - to switch the context of the default config file

kubectl config --kubeconfig=my-kube-config use-context <Context\_Name> if you want to switch the context of a custom config file

to make the custom config file as the default config file just replace the file in the default path $HOME/.kube/config

## 30) API Groups

## 31) Authorization modes

Node access, Access based access controls, Role based access controls, webhook (Third party) Always allow, always deny

These are configured on the kube-apiserver --authorization-mode – if not specified it will be by default always allow.

For example

If –authorization-mode is set to NODE, RBAC, WEBHOOK it will check the authorization mode one after the other

ROLE BASED ACCESS

Two Yaml files ROLE and ROLEBINDING

For example – Role – developer, administrator, operator, tester ( access to certain resources)

Role Binding – Binding the users to that roles to give them access

Kubectl get roles

Kubectl get rolebindings

Kubectl auth can-i delete pods –as <username> --namespace <namespace name>

Can even restrict a user to access only particular pods by adding a resourceNames in the role.yaml file

Cluster scoped – NODES, Namespaces, clusterroles, certificate signingrequests, Persistent volumes,

Namespace scoped – pods, replicasets, roles, rolebindings, deployments

For resources within namespace we use roles and rolebindings to give user a particular access using the yaml files

For resources within cluster we use cluster roles and cluster rolebinding –

NOTE: The cluster roles can grant access to namespace resources as well. It will have access that resources across all namespace.

## 32) Image security

--image-nginx or image = nginx

The actual name is nginx/nginx – first is user account and second is image repository.

Docker repository, google registery, aws repository.

You can even pull the images from the private registry. For that need to login. Can use a secret file and user imagePullsecrets to login to docker and pull the image from a private repositiry

Security context

kubectl create secret docker-registry private-reg-cred --docker-username=dock\_user --docker-password=dock\_password --docker-server=myprivateregistry.com:5000 [--docker-email=dock\_user@myprivateregistry.com](mailto:--docker-email=dock_user@myprivateregistry.com)

## 33) security context

It is to give permission on a pod level or container level – if its on pod level it will be applied to all containers running within the POD, but if its in the container level it will be applied only to that container.

To see which kubectl exec <Containername> -- whoami

## 33) Network Policy (Very important)

Its another object in kubernetes

Network policies are applied on a POD to allow a certain traffic to that POD. By default, in kubernetes all pods can talk to each other, but if you want to block a certain traffic to reach the POD you can use network policy and apply to that pod.

Create a network policy and add a section in the yaml for podSelector and match labels option to apply to a particular pod.

Policy type – ingress – incoming

Egress – outgoing

Kubectl get network policy

Kubectl describe networkpolicy

## 34) Storage (Very Important)

CRI – container runtime interface – to make it independent on what ever container runtime is used – docker, rocker, CRI-O

CNI – Container network interface – To be independent with any vendor – flannel, weave, ACI

CSI – container storage interface – To be independent of the storage – portworX, Nutanix, amazon EBS, DELL EMC, HPE

CSI is not Kubernetes specific standard – it will work with any supported plugin – it supports other orchestration tools like mesos, couldfoundry

The storage unit should have this CSI plugin

RPC – create volume, delete volume, controller publishvolume

If create volume Remote procedure call is made by the orchestration engine to create a volume and the storage driver will implement this RPC and will allocate the required storage.

## 36) Networking

POD Networking

CNI in Kubernetes

CNI weave

IP Address Management - IPAM

Service Networking – ClusterIP Service, NodePort Service, Loadbalancer

This below range is defined to all the services created across the cluster. Kubernetes will make sure this service range wont overlap with the POD network range.

For the IP range can see the kube-apiserver process or can check the yaml file /etc/kubernetes/manifest/kube-apiserver.yml it will have the IP range given to service

--service-cluster-ip-range=10.96.0.0/12

## 35) PV and PVC (Very Important)

Persistent volumes (PV) is created by administrator

Persistent volume claims (PVC) are created by user

When ever a container is created or deleted its storage is empheral – means the data is also deleted along with the container – to avoid this we need persistent volumes

This could be any storage – your local host storage (Not recommended for PROD) or some cloud storage like AWS EBS, Google persistent storage

For example, 10GB of persistent storage is allocated the administrator will create persistent volumes based on application requirement, this volumes can be claimed by PVC’s when they are assigned in a POD under spec 🡪 under volumes 🡪 under persistent volume claims

If a persistent volume claim is created it will be in pending state until a persistent volume is assigned to it, the assignment is done if the capacity matches or the capacity is higher than the required capacity.

When a claim is deleted the persistent volume is by default retained. It can be deleted to free up storage or can be recycled.



Create a persistent volume – the status will be available if not claims – kubectl create -f pv.yml

Create a persistent volume claim it will be in pending state if it doesn’t match the access mode of the persistent volume. Kubectl create -f pvc.yml

Assign the persistent volume claim in the POD specs container section under volumes will and this persistent volume will be assigned to that container

Once the PVC is deleted it will be in terminating state until the pod is killed. Once pod is killed it will be deleted also

PV will still be available based on what policy it uses – RETAIN – RECYCLE – DELETE – by default its retain.

Storage class

Storage class comes in picture when you are using a cloud-based storage solution instead of regular volumes. If storage class is defined you will have to use that storage class in the PVC instead of resources.

Kubectl describe storageclasses

Kubectl create -f storageclass.yml

Kubectl get storageclasses

## 37) Ingress (Very Important)

INGRESS Controller – GCP https loadbalancer, NGINX ingress – type deployment

INGRESS resources – type ingress

## 38) ETCD BACKUP and CLUSTER

For backup

ETCDCTL\_API=3 etcdctl --endpoints=<url from etcd.yml> - port 2379> --cacert=<path of ca.crt from etcd.yml> --cert=<path of server.crt from etcd.yml> --key=<path of server.key from etcd.yml> snapshot save <path to save the .db file>

For restore

ETCDCTL\_API=3 etcdctl snapshot restore <path of the .db file to restore> --data-dir=<new directory for etcd - /var/lib/etcd> --cacert=<path of ca.crt from etcd.yml> --cert=<path of server.crt from etcd.yml> --key=<path of server.key from etcd.yml> --name=kubemaster --initial-cluster=<info from etcd.yml Port 2380> --initial-advertise-peer-urls=<info from etcd.yml port 2380> --initial-cluter-token=<New name>

After restore completed successfully – verify the directory created under /var/lib and

Edit the etcd manifest file

1. add an entry for initial-cluster-token
2. Change --data-dir to new path
3. Volume mount of the ETCD container
4. Volume hostpath of the etcd POD

Do a systemctl restart docker – this is required because the old etcd container is also using the same name else the etcd status will be in pending state.

ETCDCTL\_API=3 etcdctl snapshot save <path> --endpoints=https://[127.0.0.1]:2379 --cacert= --cert= --key=

ETCDCTL\_API=3 etcdctl snapshot restore <path of the backup file> --data-dir= --cacert= --cert= --key= --intial-advertise-peer-url= initial\_cluster= initial\_cluster\_token= –name=

## 39) JSONPATH commands

kubectl get pods -o=custom-columns="POD\_NAME:.metadata.name, POD\_STATUS:.status.containerStatuses[].state"

## 40) Stateful Sets

Similar to deployment – but used mainly used for stateful applications – stateful applications are like those which involves DB – which depends on the previous record before processing the latest request

Stateful applications usually deployed using STATEFULSETS

Stateless applications are those which just process data on the flow – doesn’t depend on the previous record. Usually they are deployed using DEPLOYMENTS

Main difference – Replications

* Both deployments and statefulsets can be replicated, but unlike deployments, stateful replications are not interchangeable between PODS and each replicated POD is given a unique identity – this is a requirement for a DB application because of its data inconsistency
* So, when a DB application is created using a stateful sets – one POD will be the master POD which can read and write, and rest of the POD will be slaves because of this feature each POD will be given a unique identity.
* Each POD will have its own volumes, they don’t share volume, and they sync with the master to keep their volumes updated. When a new POD joins it will sync the data from previous POD – always recommended to use persistent storage for a DB application (Storage Class – remote storage)
* The reason to use a storage class in case a POD is terminated and reschedules on a different NODE the remote storage will have the info of the POD identifier and it gives the same identification as before, this is not possible if we use a local storage (hostpath)
* Unique identifier for example you use a mysql in a statefulset with three replicas – the replicas will have mysql-0 (MASTER), mysql-1, mysql2 – master and slaves will be in the order of creating
* It will create one by one if first one doesn’t come up it won’t create the second one – unlike deployment all replica PODS are created at the same time.
* Replicas in a stateful set will have permanent name and permanent DNS entry <POD-NAME>.src

If the replica is deleted and recreated IP will change but same pod and DNS name is assigned

* Even kubernetes help to keep the data consistency still lot of things like backup/restore, configuring the data synchronization should be done, normally stateful applications is not a ideal for a containerized deployment because it requires the previous data to process new tasks