INTRO/INSTALLATION

Tuesday, September 20, 2022 7:29 PM

Pod:
- State- is the pod up and running
- Health - is the app in the pod running
- Probes - to test if a pod is running by a test

Controller:
- Defines the desired state
- Create,manage pods for you
- Respond to pod state and health

ReplicaSet to decide number of replicas to run
Deployment creates the / manage rollout of replicasets if specified in file
Services adds persistency to the ephemeral world by giving networking abstraction for pod access
IP and DNS name for a service

Storage
Volume,Persistent Volume,Persistent Volume Claim

Cluster Components:
- Control Plane Node(master(PI server,scheduling etc..))
- Node(having pods)

Control plane node only has the etcd,scheduler,controller manager,api server(RESTFUL))
Kubectl connects to api server to manage stuffs

API server updates the etcd for management and that etcd is for persisting state of api objects in key value pairs

Scheduler watches api server for unscheduled pods and schedules them and sees the resources required for them to be scheduled that respects constraints like all

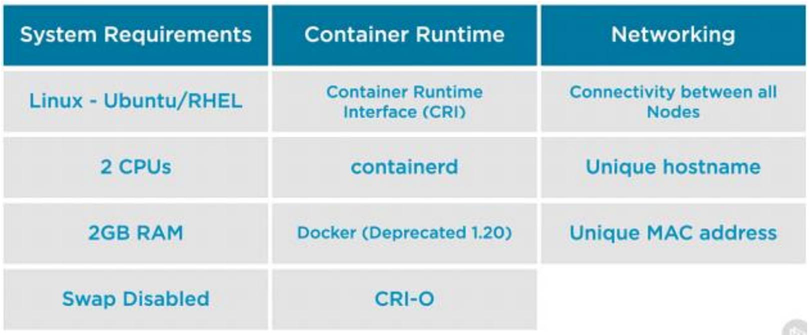
Controller manager is a controller loop that manages lifecycle fucntoins and its desired state
Watch and update the api server

Node:
- Kubelet(starting up pod,monitors api server for change,execute pod probes)-------communicate with api server
- Kube-proxy(pod networking using iptables,implement services,route traffic to pods(load balancing)--------communicate with api server
- Container runtime((downloading gr's and execute it)runtime for containers pulling from gcr and execution env)(containerd-runtime)

Pods on a node can communicate with all pods on all nodes without nat,agents on a node can communicate with all pods on that node

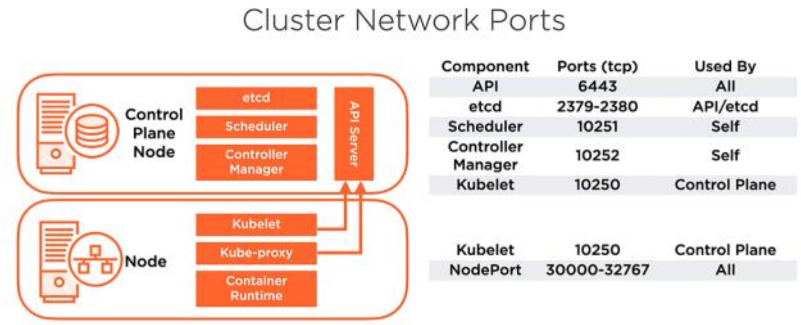
Installation methods:
- In desktop
- Using kubeadm
- Cloud

The container runtime should be containerd since for previous versions is been deprecated for docker support
Installation requirements:

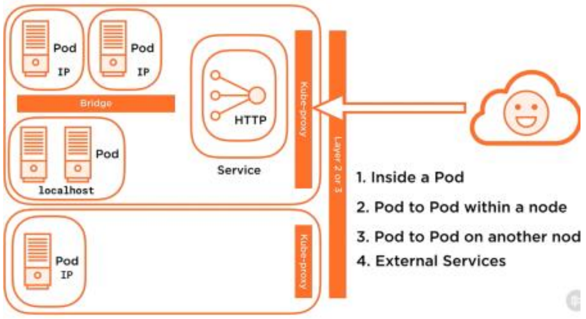


System Requirements	Container Runtime	Networking
Linux - Ubuntu/RHEL	Container Runtime Interface (CRI)	Connectivity between all Nodes
2 CPUs	containerd	Unique hostname
2GB RAM	Docker (Deprecated 1.20)	Unique MAC address
Swap Disabled	CRI-O	

Networking requirements:



Component	Ports (tcp)	Used By
API	6443	All
etcd	2379-2380	API/etcd
Scheduler	10251	Self
Controller Manager	10252	Self
Kubelet	10250	Control Plane
Kubelet	10250	Control Plane
NodePort	30000-32767	All



1. Inside a Pod
2. Pod to Pod within a node
3. Pod to Pod on another nod
4. External Services

Installation methods:

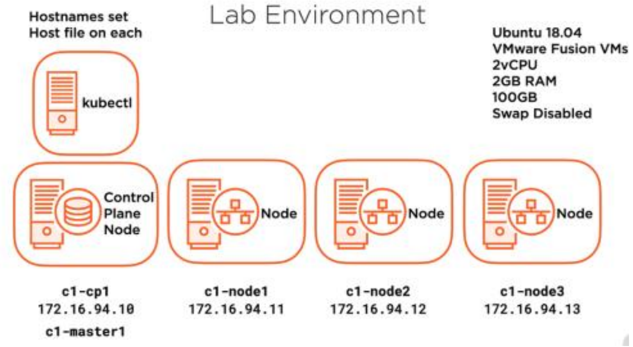
- Desktop app
- Kubeadm
- Cloud scenarios

We can get kubernetes either through github
Https://github.com/kubernetes/kubernetes
Else linux distribution repos
Yum and api

Building cluster:

- Install and configure packages(containerd,kubelet,kubeadm,kubectl) - on all nodes in the cluster
- Create cluster
- Configure pod networking
- Join nodes to your cluster

Installing packages:
sudo apt-get install -y containerd
sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
(adding the gpg key (to trust the repo)for the kubernetes app respository where they live and add it to local repos list)
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl containerd
(used for not to update this packages and gives full control to us)



Hostnames set
Host file on each

Ubuntu 18.04
VMware Fusion VMs
2vCPU
2GB RAM
100GB
Swap Disabled

systemd is a system and service manager for Linux operating systems. When run as first process on boot (as PID 1), it acts as init system that brings up and maintains userspace services. Separate instances are started for logged-in users to start their services.
In systemd , a unit refers to any resource that the system knows how to operate on and manage
Installation process:

- ssh to control plane node
- swapoff -a (checks if swap is enabled or not if no output is displayed then it is disabled)
  We can see vi /etc/fstab to view if swap.img is present or not if present comment

Step1 :

Step 2:

Installation process:

- ssh to control plane node
- swapoff -a (checks if swap is enabled or not if no output is displayed then it is disabled)
  We can see vi /etc/fstab to view if swap.img is present or not if present comment
- sudo modprobe overlay
- sudo modprobe br_netfilter
  cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf
  overlay
  br_netfilter
  EOF
- cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf
  net.bridge.bridge-nf-call-iptables = 1
  net.ipv4.ip_forward = 1
  net.bridge.bridge-nf-call-ip6tables = 1
  EOF
- sudo sysctl --system
- sudo apt-get update
- sudo apt-get install -y containerd
- sudo mkdir -p /etc/containerd
- sudo containerd config default | sudo tee /etc/containerd/config.toml
  We need to modify toml file to have SystemdCgroup=true inside the
  [plugins,"io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
- sudo systemctl restart containerd
  sudo bash -c 'cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
  deb https://apt.kubernetes.io/ kubernetes-xenial main
  EOF'
- sudo apt-get update
- apt-cache policy kubelet | head -n 20 =>to se different veriosn of the kubelet in repo
- VERSION=1.24.0-00
- sudo apt-get install -y kubelet=$VERSION kubeadm=$VERSION kubectl=$VERSION
- sudo apt-mark hold kubelet kubeadm kubectl containerd
- sudo systemctl status kubelet.service
- sudo systemctl status containerd.service
- sudo systemctl enable kubelet.service
- sudo systemctl enable containerd.service

Bootstrapping a cluster with kubeadm:

- Kubeadm init
  - Pre-flight checks
  - Creates a certificate authority
  - Generates kubeconfig files
  - Generates static pod manifests
  - Wait for control pkane to start
  - Taints the control plane node
  - Generates a bootstrap token
  - Starts add-on componens:dns,kube-proxy
- Kubeadm created kubeconfig files
  - Usdd to define how to connect to your cluster
  - Client certificates
  - Client api server network location
  - /etc/kubernetes
  - Admin.conf,kubelet.conf,controller-manager.conf,scheduler.conf
- Static pod manifests
  - Manifest describes a configuration
  - /etc/kubernetes/manifests
  - Etcd , Api server, Controller manager , Scheduler watched by kubelet started automatically when the system starts and over time
  - Enables the startup of the cluster

Pod Networking:

- Single un NATed IP address per pod
- (optional)Direct routing - configure infrastructure to support IP reachability between nodes,pods
- Overlay netowrking
  - Flannel - Layer 3 virtual network
  - Calico - Layer 3 and policy based traffic management
  - Weave net - multi host network

Creating a control plane node:

- wget https://docs.projectcalico.org/manifests/calico.yaml
- kubeadm config print init-defaults | tee ClusterConfiguration.yaml
  In this yaml file there will be a place where the api server endpoint needs to be changed as per wish,also the sock from docker to container d following key
  kind: InitConfiguration
  localAPIEndpoint:
    advertiseAddress: 1.2.3.4
    bindPort: 6443
  sed -I 's/ advertiseAddress: 1.2.3.4/ advertiseAddress: 172.16.94.10/' ClusterConfiguration.yaml
  sed -i 's/ criSocket: \/var\/run\/dockershim\.sock/ criSocket: \/run\/containerd\/containerd\.sock/' ClusterConfiguration.yaml

Step1 :

4-CreateGKECluster

3-CreateAKSCluster

2-CreateNodes-conta...

1-CreateControlPlan...

0-PackageInstallatio...

Step 2:

2-deployingapplications

1-workingwithyourclu...

```
cat <<EOF |cat >> ClusterConfiguration.yaml
 ---
 apiVersion: kubelet.config.k8s.io/v1beta1
 kind: KubeletConfiguration
 cgroupDriver: systemd
 EOF
```

Also we can change the node name under criSocket to the control plane name,also the kubernetes version

- sudo kubeadm init --config=ClusterConfiguration.yaml --cri-socket /run/containerd/containerd.sock

  After running above command the kubernetes generates the following commands and tells you to sexecute these
- mkdir -p $HOME/.kube
- sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
- sudo chown $(id -u):$(id -g) $HOME/.kube/config
- Kubectl apply -f calico.yaml

<u>Adding a node to the cluster:</u>

- Install packages
- Kubeadm join
- Download cluster info
- Node submits a CSR
- CA signs the CSR automatically
- Configures kubelet.conf

Kubeadm join ip_of_api_server(controlplane):port --token token --discovery-token-ca-cert-hash sha256:hash

CONCEPTS

Friday, September 23, 2022 8:07 PM

For outputting use
Kubectl with wide,yaml,json as parameter for output format

Dry-run - print an object without sending it to the api server

kubectl get all --all-namespaces | more (to list everytihng

Kubectl api-resources | more (lists the different types of entity and its shortnames

For documentation on each resource:
kubectl explain pod --recursive | more
kubectl explain pod | more
kubectl explain pod.spec | more

kubectl describe nodes node1 | more

For to able to have a bash completion while typing commands we can instaqll bash completion
sudo apt-get install -y bash-completion
echo "source <(kubectl completion bash)" >> ~/.bashrc
Source ~/.bashrc
Eg: kubectl get pods --[tab] will list all attributes available

Imperative cli:

- Kubectl create deployment nginx --image=nginx
- Kubectl run nginx --image=nginx

Declarative cli:

- Define our state in code(yaml/json)
- manifest
- Kubectl apply -f deploy.yaml

Basics:

```
apiVersion: apps/v1
Kind: Deployment
Metadata:
    Name: hello-world
Spec:
    Replicas:1
    Selector:
        matchLabels:
            App: hello-world
    Template:
        Metadata:
            Labels:
                App.: hello-world
        Spec:
            Container:
              □ Image: gcr.io/image
              □ Name: hello-app
```

Generating manifests with dry-run

Kubectl create deployment hello-world --image=gcr.io/image --dry-run=client -o yaml > deployment.yaml

sudo crictl --runtime-endpoint unix:///run/containerd/containerd.sock ps (in node if we run this will give us containerd containers)

kubectl exec -it hello-world-pod -- /bin/sh (to execute a shell commands inside container)

kubectl describe service hello-world (will give us the greater detail including endpoint which will not be given in get services)
kubectl get endpoints hello-world

MANAGE API SERVER/PODS

Saturday, September 24, 2022 11:33 AM

Kubernetes API objects:

- Organized by
  - Kind: Pod,Service,Deployment,Persistent Volumes
  - Group - core,apps,storage
- Use dry run to generate api objects
  - Server side- processed as a typical request,not persisted in storage
  - Client side - writes the object to stdout,validate suntax

Kubectl diff is used to generate differences in the objects running in the cluster and the objects defined in the new mainfest file (kubectl diff -f deployment.yaml)

kubectl config get-contexts
Kubectl config use-context kubernetes-admin@kubernetes
Kubectl cluster-info

Api groups:

- Core api(legacy group)
- Named api groups

Can be given as part of the api object request

Core:

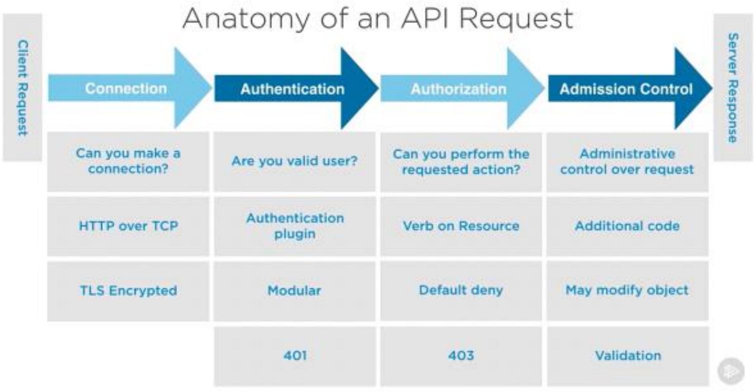
- Pod
- Node
- Namespace
- PersistentVolume
- PersistentVolumeClaim

Named api:

- Apps - Deployment
- Storage.k8s.io - storageclass
- Rbac.authorization.k8s.io - Role

Api Versioning:

- Alpha/experimental
  - V1alpha1
  - Early release
  - Diabled by default
  - For testing only
- Beta
  - V1beta1
  - Thoroughly teste
  - Considered safe
- Stable
  - V1



Anatomy of an API Request

Client Request

Connection	Authentication	Authorization	Admission Control
Can you make a connection?	Are you valid user?	Can you perform the requested action?	Administrative control over request
HTTP over TCP	Authentication plugin	Verb on Resource	Additional code
TLS Encrypted	Modular	Default deny	May modify object
	401	403	Validation

Server Response

For running the get pod in verbose mode
Kubectl get pod podname -v 5(verbose lvevel)

kubectl proxy &
curl https://10.128.0.53:6443/curl http://localhost:8001/api/v1/namespaces/default/pods/hello-world | head -n 20

To request api server the details of the pod through rest request with kube proxy running so that it takes the authentication from kubeconfig

To stream output for the object changes use
Kubectl get pods --watch -v 6 & (runs in the background as a streaming process by establishing tcp session open with the server waiting for data (netstat -plant | grep kubectl))

Namespaces are used to subdivide a cluster and its resources(virtual cluster), for resource isolation/organization   (not a linux namespace)

Resources are namespaced like pods,controllers,services but not the physical things like persistentVolumes,Nodes

Common namespaces:
- Default
- Kube-public
- Kube-system
- User-defined

Kubectl create namespace name(-o yaml to output and apply using yaml file)
Use namespace: namespacename inside metadata key when deploying apps

For creating a label
- Kubectl label pod nginx tier=PROD app=v1
- Kubectl label pod nginx tier-DEBUG app-v1 --overwrite
- Kubectl label pod nginx app-(deletes the label)

To query using label
- Kubectl get pods --show-labels
- Kubectl get pods --selector tier=prod
- Kubectl get pods -l 'tier in (prod,qa)'

Defining deployemnt and services to match selector labels
Eg: Deployment

```
Kind:Deployment
---
Spec:
    Selector:
        matchLabels:
         run: hello-world
     template:
        Metadata:
            Labels:
                Run: hello-world
        Spec:
            Containers:
---
```

For scheduling pods to nodes using labels selector:
- kubectl label node node4 nodename=node4
- In the pod use nodeSelector: straight to continers key with the label of the node

In service also we give selector in spec withthe labels/matchLabels

Containers in pod:
- Single container pod
- Multi container pod
- Init container

Static pods:
- Managed by the kubelet on nodes
- Static pod manifests
- staticPodPath in kubelet's configuation in /etc/kubernernetes/manifests
  Find in /var/lib/kubelet/config.yaml
- staticPodPath is watched
- Creates a mirror pod
- We can delete the pod from only the manifests directory where we created the pod

Kubectl exec -t pod1 --container container1 -- /bin/bash (--container required if it is a multicontainerpod)
Kubectl logs pod1 --container container1

Kubectl port-forward pod pod1 LOCALPORT:CONTAINERPORT (for forwarding the port that app is using to the local port where the kubectl is executed)

If we want to wait for some files or some before process that needs to be executed then we can use initContainers and execute commands on it so that after initcontainers execute the main containers will start to spin up

Stopping/terminating pods(lifecycle):

- Grace period 30 sec default
  Kubectl delete pod name --grace-period=seconds
- Pods change to terminating
- SIGTERM
- Service endpoints and controllers updated
- If > grace period SIGKILL
- API and etcd updated
  Forecefully deleteing
  Kubectl delete pod name --grace-period=0 --force

Also we can put terminationGracePeriodSeconds: 10 inside the spec: of manifest to do thisi

We can kill all the processes running In the container by
Kubectl exec -it hellow-world-pod -- /usr/bin/killall helloapp

Defining pod health:

- Using container probes we can ad additional intelligence to pod's state and health
  - Liveness probe
  - Readinessprobe
  - Startupprobe

Liveness probe:

- Runs a diagnostic check on a container
- Per container setting
- onFailure,kubelet restarts the container(container restart policy)

Readiness probe:

- Runs a diagnostic check on a container
- Per container setting
- Wont receive traffic from service until the readiness probe is succeeds
- Onfailure removes pod from load baalncing
- Apps that temporarily cant respond to a request
- Prevents users from seeing errors

Startupprobe

- Runs a diagnostic check on a container
- Per container setting
- Ensuring all containers in a pod are ready
- On startup all other probes are disabled until startupprobe succeeds
- onFailurre,kubelet restartsaccording to restart policy
- Apps have long startup times

Types of diagnostic check for probes(to search into or examine something):

- Exec(process exit code)
- tcpSocket(successfully open a port)
- httpGet(retrun code 200=> and <400 when trying urls

Status for these are success,failue,unknown

managing-kubernetes...

initialDelaySeconds - number of seconds after the container has started before running container probes, default 0
periodSeconds - probe interval, default 10 seconds
timeoutSeconds - probe timeout 1 seconds
failureThreshold - number of missed checks before reporting failure, default 3
successThreshold - number of probes to be considered succsesfull and live ,default 1

Inside the containers section adda a livenessProbe/readinessProbe key

```
Spec:
    Containers:
        livenessProbe: / readinessProbe / startupProbe
            tcpSocket:
                Port: 8080
            initialDelaySeconds: 15
            periodSeconds: 20
```

Managing Kubernetes Controllers and Deployments

Sunday, September 25, 2022   11:25 AM

Controller manager:
- Kube-controller-manager
- Cloud-controller-manager

Pod Controllers:
- ReplicaSet
- Deployment
- DaemonSet
- StatefulSet
- Job
- CronJob

Other Controller:
- Node
- Service
- Endpoint

Deployments:
- Creating,updating,scaling are operation performed by deployments

In deployment manifest file we specify a selector key inside spec: and give it a matchLabels and the corresponding labels and in the template key straight to tht selector we have the pod spec's with metadata matching the metadata in tht selector

ReplicaSets:
- Allows for more complex set based selectors
- matchExpressions as the selector
- Operators(In,NotIn,Exists,DoesNotExist) like keys and values type

Sudo shutdown -h now (running this inside the node vm will close the connection between control plane and this node)

If we shutdown the node the pod will be restarted until this transient (temp)error(till a specific amount of time) is not there and the node is back

If there is a permanent failure then the kubectl will spin up a new pod in place of that pod in other node

Kubectl set image deployment deploymentname name=hello-app:2.0 --record(will record the changes and will be easier to investigate through it)(used to see change cause in revision history

UpdateStrategy:
- RollingUpdate(default) - a new replicaset starts to scale up and old one starts scaling down
- ReCreate - terminates all pods in current replicaset set prior to scalingup the new ReplicaSet

Controlling the rollingupdate strategy
- maxUnavailable(20%)(ensures only a certain number of pods are unavailable being updated)
  It means that when carrying out deployment kubernetes will altleast terminate/scale down (eg:20%) of the replicas
- maxSurge(ensure that only a certain number of pods are created above the desired number of pods)
  This is like when deployment rollout happens can only increase upto (eg: 5) pods/replicas

eg:

```
Kind: Deployment
Spec:
    Replicas: 20
    Strategy:
        Type: RollingUpdate
        rolingUpdate:
            maxUnavailable: 25%
            maxSurge: 5
```

To pause.resume deployment using kubectl
Kubectl rollout pause deployment name
Kubectl rollout resume deployment-name

The revision history is default to 10

Kubectl rollout history deployment name
Kubectl rollout history deployment name --revision=1
Kubectl rollout undo deployment name (--to-revision=1)

Kubectl rollout restart deployment name

DaemonSet:
- Ensures all or some nodes run a pod
- Effectively an init daemon inside your cluster

- Kube-proxy is a daemonset that serves the network services
- Eg include log collectors,metric servers,resource monitoring agents,storage daemons
- Can also specify nodeSelectors in order to run on specific nodes

DaemonSet has two updatestrategy like RollingUpdate,OnDelete

When a pod has labels and we want to overwrite it
Kubectl label pods mypod app=hello --overwrite (if previously it was app-nothello it will overwrite it)
Kubectl label pods mypod app- (removes label with key pod)

- We can also label nodes and define a nodeselctor with labels of the nodes and daemonsets will automatically sin up pods in those nodes

Jobs:

- Creates one or more pods
- Runs a program in a container to completion
- Ensure that the specified number of pods complete successfully

Here we give a pod template in the manifest

Restartpolicy in job manifest should be either never or onfailure should not be always

backoffLimit - number of jobretries before it is marked failed
activeDeadlineSeconds - max execution time for the job
Parallelism - max number of running pods in a job at point in time
Completions - number of pods that need to finish successfully

CronJob:

- The resource is created when the object is submitted to the API server
- When it's time , a Job is created via the Job template from the CronJob object

Schedule - a cron formatetd schedule
Suspend - suspends the CronJob
startingDeadlineSeconds - the Job hasn't started in this amount of time it has Failed
concurrencyPolicy - handling concurrent executions of a Job,(Allow,Forbid,Replace)

Here we give a job a s the template instead of pod inside jobTemplate:
Inside the jobTemplate spec.template.spec.contianers will have the pod which will be ru a s a job

We can also specify backOffLimit in Job to stop execution / create pod in event of filure

<u>StatefulSets:</u>

- Examples that will use statefulsets are Database workloads,caching servers,app state for web forms

Capabilities:

1) Naming(unique naming)
2) Storage (persistent storage)
3) Headless Service (services without clusterIP /load balancer to locate objects using cluster dns)

deployment .probes-3

ReplicaSet-matchExp...

job

ParallelJob

StatefulSet

CronJob

DaemonSet

DaemonSet WithNode...

Kubernetes Storage and Scheduling

Monday, September 26, 2022 4:34 PM

Storage API objects in kubernetes:

- Volume
- Persistent Volume
- Persistent Volume Claim
- Storage Class

PersistentVolumes:

- Is a administrator defined storage in the cluster
- Implementation details for your storage
- Its managed by the kubelet that maps the storage in the node and exposes the PV as a mount inside the container

Types:

- Networked(NFS,azureFile)
- Block(Fibre Channel,iSCSI)
- Cloud(awsElasticBlockStore,azureDisk,gcePersistentDisk)

PersistentVolumeClaim:

- A request for a storage by a user given
  - Size
  - Access mode
  - Storage class
- Cluster will map a PVC to a PV

Access Modes:(Node level,not pod level)

- ReadWriteOnce(RWO)
- ReadWriteMany(TWX)
- ReadOnlyMany(ROX)

Static Provisioning Workflow:

- Create a persistent volume
- Create a persitent volume claim
- Define volume in pod spec

Storage lifecycle:
Binding(PVC created,control loop,matches PVC->PV)
Using(pod's lifetime)
Reclaim(PVC deleted,Delete(default),Retain)

Defining a PV:

- Type{nfs,fc,azuredisk}
- Capacity
- accessModes
- persistentVolumeReclaimPolicy
- Labels

EG: for NFS

```
...
Kind: PersistentVolume
...
Spec:
        Capacity:
                Storage: 10Gi
        accessModes:
            - ReadWriteMany
 nfs:
        Server: ipaddress
        Path: "/export/volumes/pod"
```

Defining a PVC:

- accessModes
- Resources
- storageClassName
- Selector

EG:PVC

```
Kind: PersistentVolumeClaim
...
Spec:
        accessModes:
                -   ReadWriteMany
 resources:
        Requests:
                Storage: 10Gi
```

Inside the pod we will provide the volume type as

```
persistentVolumeClaim:
        claimName: pvc-nfs-data
```

We have to create a persistetn volume first and create a persistent volume claim and it will automatically bind the pvc to the pv and the pvc name can be used to mount the file inside the pod

We can create a nfs server by installing a package with two-three commands and use that vm's ip as a nfs server
When we delete the PVC the PV status becomes released and if the reclaim policy is retain then the PVC goes pending and the all the pods which use that PVC goes pending so we need to delete the deplloyments,PVC,PV and recreate the PV
When we put reclaimpolicy to delete(default) when we deletepvc pv will be deleted
Till now we are statically provisioning PV's but if we want to dynamically provision it we use storage classes where we define tiers/classes of storage

Dynamic provision workflow:

- Create a storage class
- Create a PVC
- Define volume in pod spec
- When the pod starts PV is dynamically created

EG:StorageClass

```
apiVersion: storage.k8s.io/v1
Kind:StorageClass
...
Parameters:
        Kind: Managed
        Storageaccounttype: Premium_LRS
Provisioner: kubernetes.io/azure-disk
```

Inside the PVC spec:

```
                        storageClassName: name
                        Resources:
                                ...
```

For eg GCP:

- Create a PVC with the storageclassname with the name from the available in the gcp console gke
- It will create a pv based on that pv and all the pods will be running
- If the pvc is deleted the pvc and its corresponding pv will be deleted

For custom storage class we need to see the parameters available for the cloud providers and create the custom classes and use them in PVC

Configuring applications in pods:

- Command line arguments
- Env vars
  - User-defined - defined in name/value| valueFrom defined inside each container
  - System defined - (printenv | sort (run inside container))names of all services available at the time the pod was created
- Config maps

Secrets are base64 encoded and there are only referenced y the pods with the same namespace as the secrets

Secret types:

- Generic
- Docker-registry(for accessing private docker registry)
- Tls

```
Kubectl create secret generic app1 --from-literal=USERNAME=app1login --from-literal=PASSWORD='something'
```

We can create a secret as a immutable but can be updatable

```
Spec:
        Containers:
         - name: hellow
           ...
         env:
            -   Name: app1password (ENV KEY)
                valueFrom:   (ENV VALUE)
                        secretKeyRef:
                                Name: app1 (SECRETNAME)
                                Key: PASSWORD (SECRETNAME'S KEY)
```

(THIS WILL AUTOMATICALLY STORE THE KEY/VALUE PAIRS IN SECRETS IN ENV INSTEAD OF SPECYING ALL KEYS ONE BY ONE)

```
envFrom:
 - secretRef:
        Name: app1(SECRETNAME)
```

We can mount the secret as files with volume mounts

```
Secret:
        secretName: app1(inside the volumes key)
```

If we see the mounted path it will be like /mountpath/SECRETKEY (we can use this path to access the value

To access the value from cli

```
Echo $(Kubectl get secret app1 --template={{.data.USERNAME}} | base64 --decode)
```

```
apiVersion: v1
kind: Secret
metadata:
  name: app2
stringData:
  USERNAME: app2login
  PASSWORD: S0methingS@Str0ng!
```

```
Sudo ctr(containerd) images pull gcr.io/___/
Sudo ctr images list
Sudo ctr images tag gcr.io/__/ docker.io/privatereponame(ours)/dd:ps
Sudo ctr images push docker.io/privatereponame/app:ps --user $USERNAME(WILL ASK FOR THE PASSWORD NEED TO TYPE THAT)
```

Pulling images from a private registry:

- Kubectl create secret docker-registry secretname
  --docker-server=https://index.docker.io/v2/
  --docker-username=$USERNAME

```
--docker-password=$PASSWORD
--docker-email=$EMAIL
```

- In the pod spec we need to specify the imagePullSecrets

```
Spec:
        Container:
          - name: hello-world
            image: privatereponame/hello-app:ps
            ports:
                Containerport: 8080
        imagePullSecrets:
         - name: secretname
```

CONFIGMAPS:

- Key/value pairs exposed into a pod used app config settings
- Define app or env specific settings
- Decouple app and pod config
- Env vars or files
- Volume configmaps can be updated
- Can be marked immutable

Imperative:

- Kubectl create configmap configname
  --from-literal=KEY=VALUE
  --from-literal=KEY=VALUE .....
- Kubectl create configmap configname
  --from-file=appconfigqa (that has the key=value\nkey=val)
- Yaml

```
Kind:ConfigMap
...
Data:
        KEY: VALUE
        KEY: VALUE
```

When creating a pod instead of valueFrom: secretKeyRef we needd to use valueFrom: configMapKeyref: name: configname key: KEY
Or envFrom: configMapRef: name:configname
Or as volumes configMap: name: configname

Scheduling Process:

- Scheduler watches the api server for unscheduled pods
- Node selection(kube-scheduler)
- Update nodeName in the pod object
- Node's kubelets watch api server for work
- Signal container runtime to start container's

Node Selection:

- Filtering(can a node run pod)
- Scoring(appropriate place to run the pod)
- Binding(update API object,nodename is updated I pod)

Setting requests will cause the scheduler to find the node to fit the workload /pod
Requests are guarantees (CPU,memory), pod will go pending if there are not enough resources

Note: kubectl config get-contexts (will list all the clusters we have with their contexts we can use one at a time which we need)
Kubectl config use-context (to use that cluster)

Controlling Scheduling:

- Node Selector(using labels (of the node)and selectors)
  If no nodes are matching then pods will remain pending

```
Spec:
   nodeSelector:
        Key: value
```

- Affinity
  nodeAffinity - uses labels on nodes to make a scheduling decision with matchExpressions
  requiredDuringSchedulingIgnoredDuringExecution - will not be scheduled if roles does not evaluate to true
  preferredDuringSchedulingIgnoredDuringExecution - will be scheduled somewhere even if this is not evaluating to true
  podAffinity - schedule pods onto the same node,zone as some other pod(using matchlabels,expressions)
  podAntiAffinity - schedule pods onto the different node,zone as some other pod
- Taint and tolerations
- Node Cordoning
- Manual Scheduling

EG: POD AFFINITY

```
Spec:
 containers:
.....
 affinity:
      podAffinity:
```

nfs.pvc

secret.encoded

nfs.pv

deployment-secrets-e...

requests

CustomStorageClass

deployment-tolerations

deployment-private-r...

DeploymentsToNodes

deployment-antiaffinity

AzureDisk

deployment-configma...

deployment-affinity

```
 containers:
.....
 affinity:
        podAffinity:
                requiredDuringSchedulingIgnoredDuringExecution:
                    -   labelSelector:
                            matchExpressions:
                            -   Key: app
                                Operator: In
                                Values:
                                - hello-world-web
                        topologyKey: "kubernetes.io/hostname"
```

AzureDisk

deployment
-configma...

deployment
-affinity

deployment
-configma...

deployment
-configma...

Taints and tolerations:

- Taints - ability to control which pods are scheduled to nodes
- Tolerations - allows a pod to ignore a taint and be schedduled as normal on tainted nodes
- Useful in scenarios where the cluster admin needs to influence scheduling without depending on user
- Eg: kubectl taint nodes node1 key=MyTaint:NoSchedule
- To remove taint kubectl taint nodes node1 key:NoSchedule-

In pod we define tolerations

```
Containers:
.....
tolerations:
  -  key: "key"
     operator: "Equal"
     value: "MyTaint"
     effect: "NoSchedule"
```

The topology key is the key for the node name in the specific node's labels
kubernetes.io/hostname is the key and the value is the nodename

If we define the taint ona node as NoSchedule then no pods will be scheduled on that node unless specified tolerations on the node

Node Cordoning:

- Marks a node as unschedulable
- Prevents new pods form being scheduled to that node
- Will not affect any existing pods on that node
- Useful when eant to do a reboot or maintenance of that node

Kubectl cordon nodename
Kubectl uncordon nodename
If we want to gracefully evict your pods from a node
Kubectl drain node1 --ignore-daemonsets (evict our pod and if replicas are set then In that place another pod will be there in other node)(if no replicaset/some controller that creates the same pod it will not be deleted unless forecefully deleting a pod / manually deleting that pod)

For manually schedluing a pod we can define the **nodeName: 'node1'** in the pod manifest
If we use nodeName atribute we are bypassing scheduler and if any cordon is made on that node still it will be scheduled on that node since the schedduler will add this attribute when scheduling process

- Can create our own scheduler

- Run multiple schedulers concurrently
- Define which scheduler to use in pod spec
- We can deploy your scheduler as a system pod

configuring-managing-kubernetes-networking-services-ingress

Wednesday, October 5, 2022 11:55 AM

Network types:

- Pod Network
- Node Network
- Cluster Network

For connectivity between pod we use localhost to connect
For pod to pod connectivity kubernetes uses a bridge/tunnel and communicates through the pod's eth0
For pod node to another pod node connectivity it uses layer 2/3 or overlay network

Kubernetes uses container network interface plugin to do the ntworking stuffs
For local cluster - Calico CNI plugin
For Azure kubernetes service - kubenet

To see the node's info put kubectll describe/explain nodemasternode | more

To view routing info inside of the node (route)

Ip addr - for seeing the interfaces

Node to node traffic is happening via tunnel interfaces inside each node

For defining a codedns in cluster DNS using configmap

```
Kind:ConfigMap
…
Metadata:
        Name: coredns
        Namespace: kube-system
Data:
        Corefile: |
          .:53 {
        …
        Kubernetes cluster.local in-addr.arpa ip6.arpa {
        Pods insecure
        Fallthrough in-addr.arpa ip6.arpa
        Ttl 30
        }
        Forward . 1.1.1.1(anyip we want)
        }
```

____

Inside pod configuring dns is as follows:

```
---
Spec:
        Containers:
        …
        dnsPolicy: "None" (because by default it takes clusters dns policy)
        dnsConfig:
```

Nameservers:
  - 9.9.9.9

Searches:
  - Db1.ns1.svc.cluster.local

____

Services:

- Matches pods using labels and selectors
- Creates and registers endpoints in the service(pod IP and port pair)
- Implemented in the kube-proxy on the node in iptables
- Kube-proxy is a pod running in all nodes as a daemonsets
- Kube-proxy watches the API server and the endpoints

ClusterIP:
- The ip is taken from the range allotted to the service from the cluster

NodePort:
- The ip is taken from the node's actual ip assigned instead of cluster's service ip,uses clusters service ip for node to node internally

LoadBalancer:
- Provisions the load balancer from the public provider and assigns a real public ip and that is used for service access through creation of both nodePort and CLusterIP ip's

Example:
Kubectl create deployment hello --image=gcr.image
Kubectl expose deployment hello --port=80(service port) --target-port=8080 (container's port) --type NodePort

Kubectl get service hello -o jsonpath='{ .spec.clusterIP }' (like this for all the kind's we can extract the details

This will have 80/TCP as the port

Kubectl get endpoints servicename (this will give the endpoints in which the pods is running with the target port)

For nodeport this will have 80:32245/TCP as the port that is 32245 is the nodeport and the 80 us the clusterport

For LoadBalancer is done only using cloud provider
Loadbalancerip >> nodeportip >> clusterip >> pods

____

Service discovery:

- *Services get DNS records in cluster DNS*
- 'Normal' Services get A/AAAA records(clusterip,nodeport,loadbalancer)
- Format will be <servicename>.<namespaceofservice>.svc.<clusterdomain>
  Example: helloworld.default.svc.cluster.local
  Nslookup helloworld.default.svc.cluster.local clusterip(kube-dns)(to get the clusterip of the service helloworld)
- *Namespaces get DNS subdomains*
- Format will be <namespace>.svc.<clusterdomain>
  example: ns1.svc.cluster.local
- *Environment variables*
- Defined in pods for each service available at pod startup

Other types of services:
- ExternalName
  - Service discovery of external services
  - CNAME to resource
- Headless
  - DNS but NO ClusterIP
  - DNS records for each endpoint

    - Stateful apps(statefulsets)
- Without selectors
    - Map to specific endpoints
    - Manually create endpoint objects
    - Point to any ip inside or outside cluster

____

INGRESS:

- Ingress resource
- Ingress controller (implements the rules defined in the ingress resource, has many types)
  ingresClassName:
  Pods in a cluster - **nginx**
  Hardware external to the cluster - **Citrix,F5 and more**
  Cloud ingress controllers - **AppGW,google load balancr,AWS ALB ingress**
  Have a defined spec
- Ingress class

(**FOR AZURE FOR INGRESS CONTROLLER WE CAN ENABLE A CHECKBOX IN AKS CLUSTER FOR CREATING AN INGRES CONTROLLER AND THE CONTROLLER WE CAN ANNOTATE THAT IN THE METADATA FIELD**

```
 annotations:
  kubernetes.io/ingress.class: azure/application-gateway
```

AND GIVE THE INGRESS RESOURCE HAVING THIS AS THE ANNOTATION AND IT WILL CREATE A IP ADDRESS AND USE THAT ONE TO DO THE PATH BASED OR ANYTHING)

(WHEN WE CREATE MANUALLY NODES WE CAN DEPLOY THE PODS THROUGH NODEPORT AND EXPOSE THEM BY CREATING AN APPLICATION GATEWAY IN AZURE WITH AN PUBLIC IP THAT FORWARDS TRAFFIC FROM PORT 80 TO PRIVATE/PUBLIC IP OF NODE IN THE NODEPORT(30200))(when specifying backend settings we should provide override path of / if the app that is deployed serves only on main path else not required (then only when we define path based routing it will point to the specific backend setting(nodeip:port)

Defines rules for external access to services
Namebased virtual hosts
Path-based routing
TLS termination

It is layer 7 and it is directly point to apps

Flow:

- The service is created and the ingress resource created with rules, ingress controller is created ,
- The ingress controller creates a nodeport ip or load balancer ip depending on the service
- When an http request comes it forwards to nodeport/public(loadbalancerip) and goes to ingress controller and that goes to the correct backend defined

Path based routing:
In the resource rules we can specify multiple services like for a host with a path as /red we expose the service redservice ad with /blue we expose the blue service so that http request comes in it will be directd accordingly and also we can define the default backend

ingress-path

Namebased virtual hosts:
In before we used a single host in the rules section here we use as many hosts we need to migrate/route

ingress-namebased

ingress-namebased

Tls certificates for https ingress:

We have a tls section inside the spec: where we define the host and the secret where the tls cettificate is stored

ingress-tls

Kubectl describe ingressclasses nginx
Kubectl get ingress

Curl http:// ingressip/nodeport --header "Host: path.exmaple.om" -- for testing using host defined in ingress

ingress-path-backend

configuring-managing...

MAINTAINING ,MONITORING,TROUBLESHOOTING

Thursday, January 5, 2023 11:19 AM

Backing up etcd:

- Backup with snapshot using etcdctl
- Secured and encrypted to protect info stored on it
- Can schedule backups using CronJob

Inside of the etcd pod it will be in the /var/lib/etcd directory(data dir)

hostPath mounted into a Pod

etcdctl:

- Download from github (binary)
- Exec into an etcd pod
- Start a container

```
ETCDCTL_API=3 etcdctl --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 snapshot save /var/lib/dat-backup.db
```

```
ETCDCTL_API=3 etcdctl --write-out=table snapshot status /var/lib/dat-backup.db
```

Restoring etcd with etctl:

- Restore backup to another location
- Move the original data out of the way
- Stop etcd
- Move the restored data to /var/lib/etcd
- Kubelet will restart etcd

(etcdctl version should match the version of etc running inside the etcd pod(kubectl exec -it etcd-c1-cp1 -n kube-system -- /bin/sh -c 'ETCDCTL_API=3 /usr/local/bin/etcd --version' | head)

```
ETCDCTL_API=3 etcdctl snapshot restore /var/lib/dat-backup.db
```

```
mv /var/lib/etcd /var/lib/etcd.OLD
```

```
sudo crictl --runtime-endpoint unix:///run/containerd/containerd.sock ps
```

```
sudo crictl --runtime-endpoint unix:///run/containerd/containerd.sock stop $CONTAINER_ID
```

```
Mv ./default.etcd /var/lib/etcd
```

Upgrading the cluster:(controlplane)

- Update kubeadm package
- Drain the control plane/master node
- Kubeadm upgrade plan
- Kubeadm upgrade apply
- Uncordon the control/master node
- Update the kubelet and kubectl

```
sudo apt-mark unhold kubeadm
sudo apt-get update
sudo apt-cache policy kubeadm
sudo apt-get install kubeadm=$TARGETVERSION
sudo apt-mark hold kubeadm
```

```
kubectl drain controlplane --ignore-daemonsets
```

```
sudo kubeadm upgrade plan
sudo kubeadm upgrade apply v$TARGETVERSION
kubectl uncordon controlplane
```

```
sudo apt-mark unhold kubectl kubelet
sudo apt-get update
sudo apt-cache policy  kubectl/ kubelet
sudo apt-get install -y kubectl=$TARGETVERSION kubelet=$TARGETVERSION
sudo apt-get hold kubelet kubectl
```

Upgrading the workder node:

- Update kubeadm
- Drain the node
- Kubeadm upgrade node

- Update kubelet kubectl
- Uncordon node

```
kubectl drain node --ignore-daemonsets
```

```
sudo apt-mark unhold kubeadm
sudo apt-get update
sudo apt-get install -y kubeadm=$TARGETVERSION
sudo apt-mark hold kubeadm
```

```
sudo kubeadm upgrade node
```

```
sudo apt-mark unhold kubectl kubelet
sudo apt-get update
sudo apt-cache policy  kubectl/ kubelet
sudo apt-get install -y kubectl=$TARGETVERSION kubelet=$TARGETVERSION
sudo apt-mark hold kubelet kubectl
```

```
kubectl uncordon node1
```

Cluster topologies:
https://bit.ly/3cOdWqi
Building an HA cluster with kubeadm
https://bitly/37dyMOL
Building an HA etcd cluster
https://bit.ly/3dOrRxH

LOGGING:

- stdout,stderr
- Logging driver(/var/log/containers)
  Retained on the node
- Log aggregation
- Kubectl logs
- Log rotation

```
Kubectl logs podname
Kubectl logs podname -c containername
```

If the api server is not running then we cant get the logs through kubectl instead we use crictl

```
Crictl --runtime-endpoint unix:///run/containerd/containerd.sock logs $CONTAINERID
```

Also if container runtime is donw then we can go to the node machine and give
Tail /var/log/containers/$CONTAINER_NAME_$CONTAINER_ID

To get the logs for the kubelet for a systemd service
Journald command
Journalctl kubelet.service

For non systemd service
/var/log/kubelet.log

For kube-proxy
Kubectl logs
/var/log/containers
/var/log/kube-proxy

/var/log/kube-apiserver.log
/var/log/kube-scheduler.log
/var/log/kube-controller-manager.log

For events in kubernetes
Kubectl get events
Kubectl describe $type $name

```
Kubectl logs podname --all-containers --follow(like--watch) --tail 5 (bottom top 5)
```

```
Journalctl -u kubelet.service --since today --nopager(wordwrap)
```

```
Kubectl get events --field-selector type=Warning,reason=Failed
```

```
--sort-by='.metadat.name.CreationTimestamp'
```

JSONPATH ACCESSING DATA:

- Kubectl get pods -o jsonpath='{ .items[*].metadata.name }'
- Kubectl get pods --al-namespaces -o jsonpath='{ .items[*].spec.containers[*].image |'

# get all internal ip addresses oif nodes

Kubectl get nodes -o jsonpath="{ .items[*].status.addresses[?(@.type=='InternalIP')].address }{"\n"}" - for new line also in stdout

--sort-by=.metadata.name
For displaying custom output

…. --output=custom-columns='NAME:metadata.name,CREATIONTIMESTAMP:metadata.creationTImestamp'

We can use kubernetes metric server for providing resource metrics for pods ,nodes,collects resource metrics from kubeletes and exposes via api server, pcpu,memory

Kubectl top pods kubectl top nodes

wget https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.6.0/components.yaml

```
#Add these two lines to metrics server's container args, around line 132
# - --kubelet-insecure-tls
# - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
```

```
#Deploy the manifest for the Metrics Server
kubectl apply -f components.yaml
```

Kubectl top pods --containers --sort-by=memory

TROUBLESHOOTING:

- Kubectl logs
- Kubectl events
- Systemctl
- Journalctl
- System logs

Nodes:
Server online
Network reachability(firewall,etcc)
Systemd
Container runtime
Kubelet
Kube-proxy
Controlplane:
Extra static pod manifest reachability

```
    Systemctl status kubelet.service
    Systemctl enable kubelet.service
    Systemctl start kubelet.service
    Systemctl stop kubelet.service
```

#Systemd unit configuration
/etc/systemd/system/kubelet.service.d/10-kubeadm.conf (need to see the path of the kubeconfig.yaml file specified here)

#kubelet componentconfig
/var/lib/kubelet/config.yaml

Node failures:

- Stopped kubelet
- Inaccessible kubelet config.yaml
- Misconfigured kubelet systemdunit

Controlplane:

#kubelet componentconfig
/var/lib/kubelet/config.yaml(inisde this the sttaispod path willl be defined)

Staticpodpath: /etc/kubernetes/manifests

Kubectl get componentstatuses (status for etcd,controllermanager,schedulr)

maintaining
-monitori...

maintaining-monitori...

2-ClusterUpgrade

1-etcd-containerd

3-monitoring

2-jsonpath

1-logging

3-TroubleshootingWo...

2-TroubleshootingCo...

2-TroubleshootingCo...

We can use networkpolicy when a network plugin like calico is installed we can block or grant access to namespace or pods(calico is required to run coredns)

2-TroubleshootingCo...

1-TroubleshootingNo...

1-TroubleshootingNo...

Configuring and Managing Kubernetes Security

Friday, January 6, 2023 7:15 PM

Securing the api server:

- Authentication
- Authorization
- Admission control

Requests to api server comes from kubectl users,pod's service account(CA certs,token,namespace are stored as a secret(mounted inside the pod in /var/run/secrets/kubernetes.io/serviceaccount) s,scheduler,controller manager

Authentication plugins:

- Client certs(commonly used (default using kubeadm)
- Authentication token(http authorization header in the client request(used by service accounts,bootstrap token and static file))
- Basic http auth(static password file (simple to use and setup)
- OpenID Connect

Authorization plugins:

- RBAC
- Node
- Attribute based access control

```
# getting the certificate info
kubectl config view --raw -o jsonpath='{ .users[*].user.client-certificate-data }' | base64 --decode > admin.cert
```

```
# seeing the info in that cert
openssl x509 -in admin.cert -text -noout
```

```
$ authenticating to api server
Curl --cacert $CACERT --header "Authorization: Bearer $TOKEN" -X GET https://kubernetes.default.svc/api/
```

```
# to check if we have authorization or not
Kubectl auth can-I list pods --as=system:serviceaccount:Default:mysvc
```

```
# to create authorization
Kubectl create role demorole --verb=get,list --resource=pods
Kubectl create rolebinding demorb --role=demorole --serviceaccount=default:mysvc
```

Public key infrastructure(pki)

Kubernetes uses certificates to provide TLS encryption

The kubeadm based clusters use a self signed certificate authority and generates certs for system components but we can also use an external CA
https://bit.ly/39KLm9j

Inside /etc/kubernetes/pki there will be ca.key ca.crt

Ca.crt:

- Distributed to clients to trust the CA
- Present /copied to
  - nodes during build
  - Kubeconfig files
  - Serviceaccount(secrets)
- CN=kubernetes means it is selfsigned

CN=COMMONNAME(USERNAME) O=ORGANIZATION(GROUPS)
The kube-proxy has a configmap instead of storing the kubeconfig as a file

#creating certificates with certificates API for new users

1- Submit and sign certificate signing requests via the api server for x509 cert
2- Used for encryption and authentication in cluster

1) Create a private key with openssl or cfssl
   ```
   Openssl genrsa -out demouser.key 2048(2048 bit private key)
   ```
2) Create a certificate signing request with openssl or cfssl
   ```
   openssl req -new -key demouser.key -out demouser.csr -subj "/CN=demouser" (O for group)
   ```

   (base64 encoded(header,trailer pulled out)
   ```
   cat demouser.csr | base64 | tr -d "\n" > demouser.base64.csr
   ```

3) Create and submit CertificateSigningRequest Object
   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: certificates.k8s.io/v1
   kind: CertificateSigningRequest
   metadata:
           name: demouser
   spec:
           groups:
       -   system:authenticated
           request: $( cat demouser.base64.csr)
           signerName: kubernetes.io/kube-apiserver-client
           usages:
       -   client auth
   EOF
   ```
4) Approve the CertificateSigningRequest
   ```
   Kubectl certificate approve demouser
   ```
5) Retrieve the cert
   ```
   Kubectl get certificatesigningrequests demouser -o jsonpath='{ .status.certificate }' | base64 --decode > demouser.crt

   Openssl c509 -in demouser.crt -text -noout  | head -n 15
   ```

Kubeconfig file:

- Cluster access info
- Context is a clusters location and credentials
- Multiple configuration contexts
- Multiple kubeconfig files
- Used by users,systemcomponents

Users: credentials,username,certifictae/token/password
Clusters: network location,ca,crt
Context: access parametrs to cluster,cluster,user in this kubeconfig file

Creating kubeconfig filesmanually:

#define cluster
kubectl config set-cluster kubernetesdemo --server=https://10.1.0.4:6443 --certificate-authority=/etc/kubernetes/pki/ca.crt --embed-certs=true --kubeconfig=demouser.conf

#define credential
kubectl config set-credentials myuser --client-key=myuser.key --client-certificate=myuser.crt --embed-certs=true --kubeconfig=myuser.conf

#define context
kubectl config set-context name --cluster=kubernetsdemo --user=demouser --kubeconfig=demouser.conf

#setcontext
kubectl config use-context demouser@kubernetes-dmo --kubeconfig=dd.config

kubectl create clusterrolebinding demouserclusterroelbinding --clusterrole=view(scoped to cluster(node,pv) --user=demouser

# create a demouser linux
Sudo useradd -m demouser

Mkdir -p /home/demouser/.kube
Cp -I demouser.conf /home/demouser/.kube/config
Chown -R demouser:demouser /home/demouser/.kube/

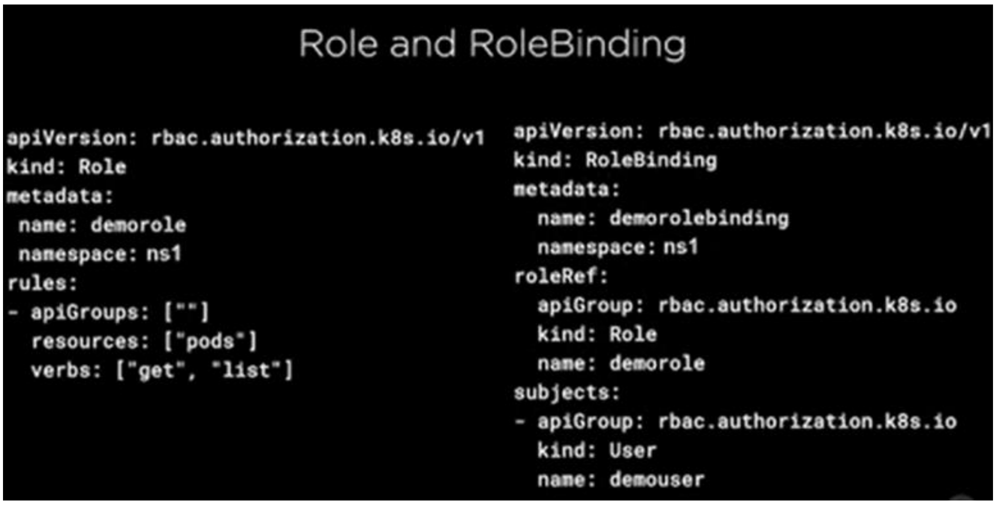
Sudo su demouser(switch to demouser)

Default clusterroles:

- cluster-admin(superuser)
- Admin(withing a namespace)
- Edit(no rolebinding,roles)
- View(no secrets,role,rolbinding

Defining rules:

- apiGroups - an empty string designates the core api group
- Resources - pod,services,deployments,nodes etc..
- Verbs - get,list,create,update,patch,watch,delete,deletecollection



Role and RoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
 name: demorole
 namespace: ns1
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: demorolebinding
  namespace: ns1
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: demorole
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: demouser
```