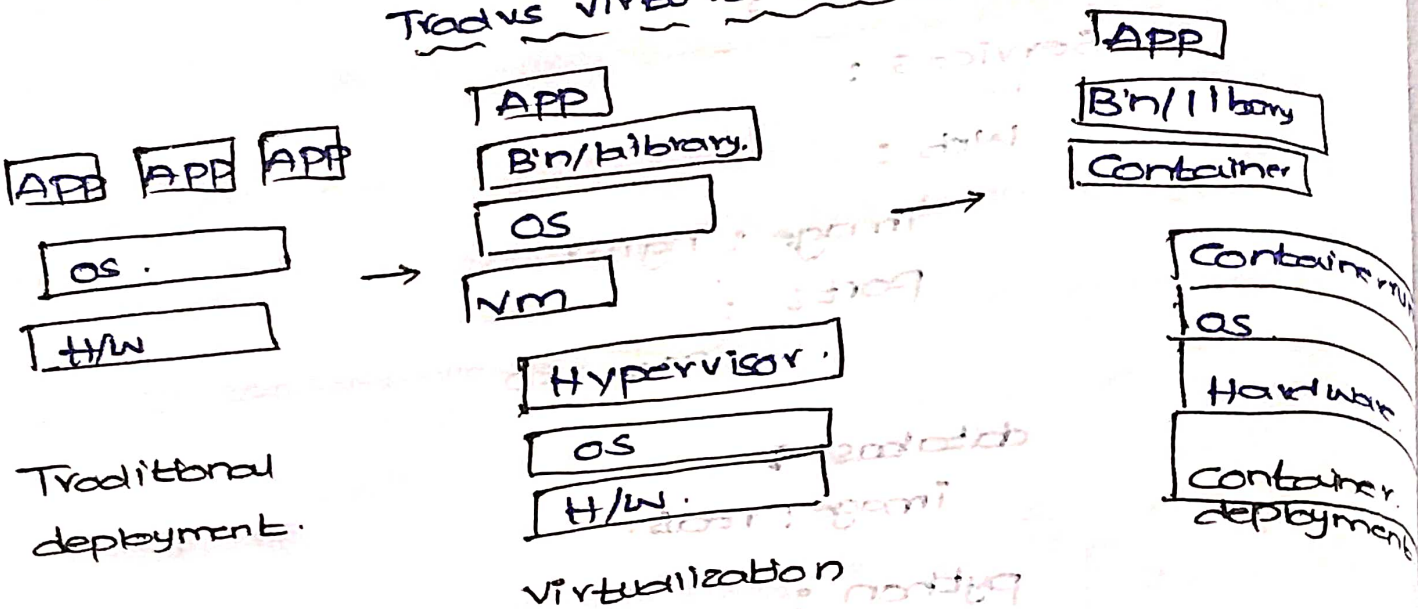
30/12/22

Kubernetes

Trad vs virtu vs container



KUBERNETES :

→ Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services that facilitates both declarative configuration & automation.

→ Kubernetes also known as K8S (or) "kube"

→ Kubernetes cluster can span hosts across on-prem public, private or hybrid clouds.

Features of kubernetes: [

→ Automated rollouts and rollbacks.

Service discovery and load balancing.

Storage orchestration.

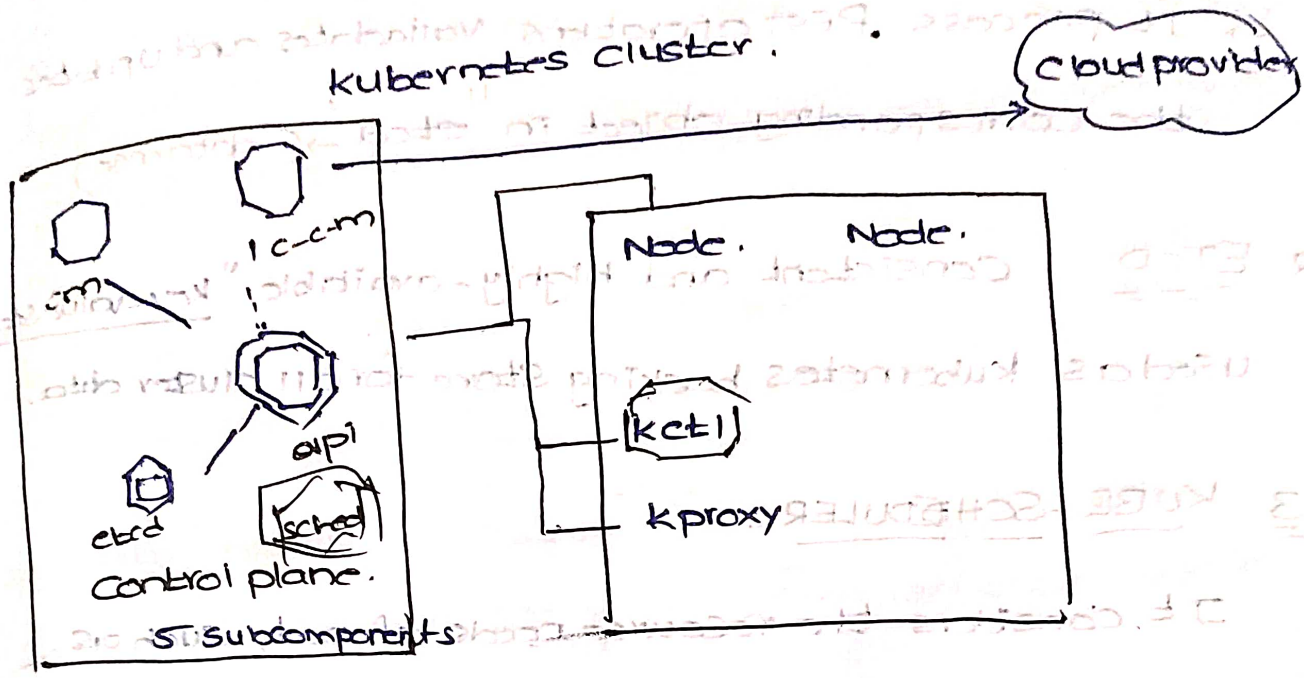
Secret and configuration managment
passwd username

Automatic bin packing

Ipv4 & Ipv6 dual stack.

Self-healing [scale up and down]

Architecture & components.



Node → vm

* A kubernetes cluster consists of a set of worker machines called "nodes", that run containerized application. Every cluster has at least one worker node

→ The worker node(s) host the "pods" that are the components of the application workloads.

→ The control plane manages the worker nodes and pods in the cluster.

→ Cluster usually runs multiple nodes; providing "Fault-tolerance & High Availability"

* Control plane Components:-

1. KUBE-APIServer :- It is the main managment P of the entire cluster, handling internal and extern requests.

→ It process .REST operations, validates and updates the corresponding object in "etcd" → (database)

2. ETCD: Consistent and highly-available "key-value" used as kubernetes backing store for all cluster data.

3. KUBE-SCHEDULER:

It considers the resource needs of pod, such as: cpu (or) memory along with the health of cluster

→ It Schedules the pod to an appropriate "compute node"

4. kube-controller manager:-

It runs the control process. A control process is a loop that focus on making the desired State equal to the current State for any application in any given instance of time.

5. Cloud-controller-manager:

It link your cluster in to your cloud provider

API and separates out the components that interact with that cloud platform from components that only interact with your cluster.

* Node Components *

1. KUBELET :- An agent that run on each node in the cluster. It makes sure that containers are running in a pod.

→ When the control plane needs something to happen in a node, the kubelet execute the action.

2. KUBE-PROXY :

→ It is a Network proxy that runs on each node in your cluster

→ It maintains network rules on nodes. These rules allow network communication to your pods of your cluster.

3. Container Runtime :

It is a software that is responsible for "running Containers"

→ Kubernetes support : Docker, containerd, CRI-O, and kubernetes CRI. (Container runtime Interface)

Note: Every "Node" we have to install docker on it

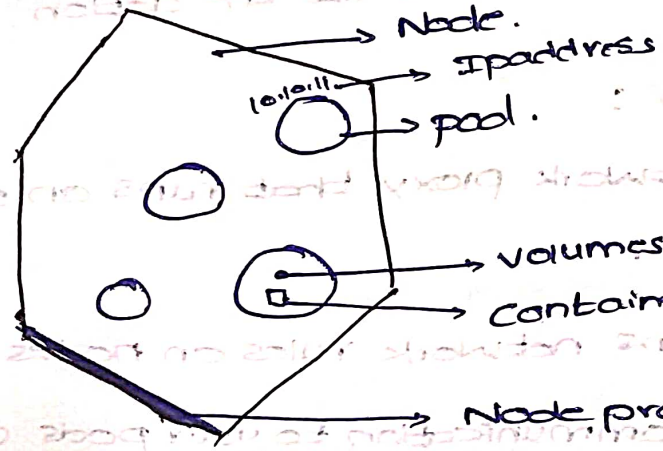
Kubernetes Nodes & pods :-

→ A node is a worker machine in k8s and may be vm (or) physical machine, depending on the clust

→ k8's runs on your workload by placing containers in pods to run on Nodes. A Node can have multiple po

→ A Node include kubelet, a container runtime, kube

→ Node names are uniqueness in the cluster.



PODS

→ A pod is the smallest & simples k8's object. It represent a single instance of a running process in cluster.

→ pods contain one (or) more containers, such as. Docker containers

→ pods run multiple containers, the containers are managed as a single entity & share the pod resources

→ pods also contain shared Networking and storage resources for their container.

Note: Running multiple containers in a single pod is an advanced use case.

2/1/23.

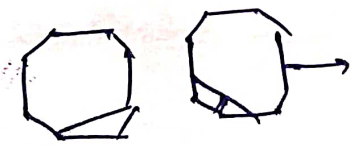
MINIKUBE

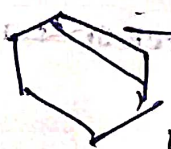
→ minikube is a utility you can used to run kubernetes (k8s) on your local machine.

→ It is a light weight kubernetes implementation that creates a vm on your local machines and deploys a simple cluster containing only "one node".

→ minikube CLI provide basic bootstraping operations for your cluster, including Start, Stop, Status & delete.

mini kube Architecture





Control plane :- It is responsible for managing the

→ The control plane cordinates all activities in your cl

Such as scheduling application, maintaining applicatio

desired state & scaling applications and rolling out

update.

Node :- It is a vm or a physical computer that serves

as a worker machine in a kuberretes cluster.

→ Each node has a "kubelet and container runtime" for mana

the node and handling container applications.

* minikube installation :-

PRE-Requestes 1. 2 cpu or more.

2. 2GiB of free memory.

3. 20GiB of free disk space. & Good Inter

4. Container (or) vm manager (docker, virtualbox,

vmware, kvm, Hyper-V]

* Go to aws Instance [Ec2] → Launch Instances

MINIKUBE

Redhat Server.

t2. medium [per hour → 4 rupees]

Create SG :-

1 x 20 GiB, → [Launch Instances]

→ Select Instance → Security → click on Security Group

Edit Inbound rules → Add rule → open all TCP, anywhere.

Save.

2. Connect Instance.

→ To change hostname: hostname minikube.

→ vi /etc/hostname. [to update hostname]

→ keep computer name (minikube) as it is!

exit. → for update of computer name.

→ sudo -i

→ yum update -y

→ cat /etc/redhat-release [Server version]

→ yum install net-tools vim -y

↓

for ifconfig command.

→ ifconfig [inet: 172.31.56.87]

→ vim /etc/hosts [mapping ipaddress to hosts]

name → Ipaddres
Ipaddres → name } mapping

Ipaddress Server hostname.

172.31.56.87 minikube as it is!

→ Setenforce 0 [to stoping the firewalls]

getenforce [to check status of firewall]

* Docker Installation:

1. yum install -y yum-utils.

2. cd /etc/yum.repos.d/

   vim docker-ce.repo [create a repo file]

   ↓

   add six lines. : wq!

3. yum install docker ---------- plugin -y.

4. systemctl start docker;
   " enable. "
   " status. "

5. Go to google → minikube install (minikube start)

   Select Installation Step · [Linux] stable, rpm package

→ exit

→ $ [curl -Lo https://storage.---minikube-lat

→ Run the s/w: sudo rpm -Uvh minikube-latest

2. Start your cluster:
   ↳ sudo + command ↳ docker driver

   → minikube start --driver=docker --force.

   → minikube

   → minikube status.

   Creating docker container

sudo minikube kubectl -- get po -A. [Running]

Manage your cluster:

→ sudo minikube pause.

→ sudo minikube unpause.

Stop the minikube :- sudo minikube stop.
" " start.

3/1/23.

KUBEADM Installation

kubeADM:- Kubeadm is a tool used to build multinode kubernets (k8s) cluster.

→ kubeadm provides kubeadm init and kubeadm joins as best-practices "fast paths" for creating kubernet's cluster.

→ kubeadm provides the actions necessary to get a minimum viable, clusterup & running.

prerequisites:

→ A compatible Linux host (debian (or) redhadt).

server → master node should have 2GB RAM & 2CPU.

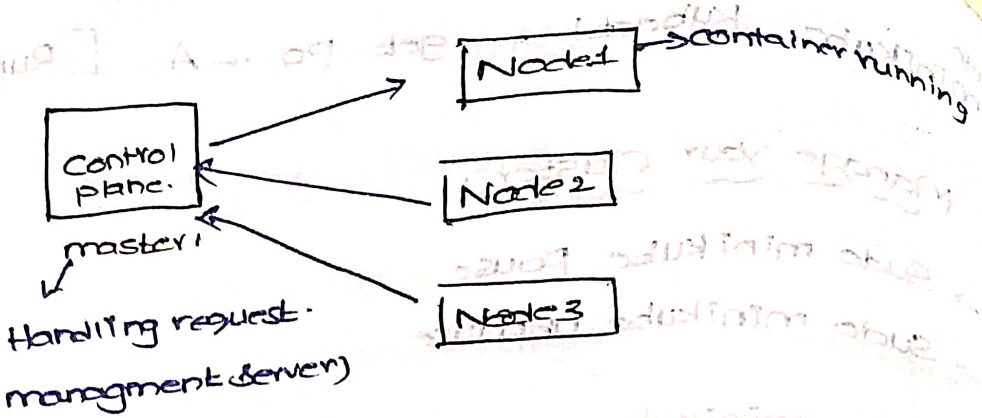
client → worker node " " 1GB RAM & 1 cpu

→ Full N/W connectivity b/w all machines in the cluster

→ Unique "hostname, MAC address and product_uuid" for every node.

→ certain ports are open to your machine.

→ Must disable Swap(virtual memory)



* Step1: Creating Ec2 Instance (3) with one master and 2 Nodes.

→ Launch Instance, the No of Instance "3", "Linux"

1 Instance → Linux, free tier.

N/w setting edit → SG → Kubernetis-SG

→ open All Tcp & anywhere address.

→ Storage 20GiB. → Launch Instances.

1. master 2. Node1 3. Node2 [Instances]

→ Step2: Stop the master machine. | Enlarge Instance vertically.

Action → Instance Setting → Chang Instance types →

(t2.medium) → Apply. → Start Instances.

Note:

→ Master Configuration:- Connect master Instance →

putty → Login

Step1: change hostname → hostname master.

vi /etc/hostname [master] : wq! (type hostname in a file)

exit.

sudo -i

Step2: yum update -y [Server update]

Step3: yum install net-tools vim -y

ifconfig

Step4: vim /etc/hosts

172.31.80.129 master. : wq!

Step5: Installing Docker.

1. vim /etc/yum.repos.d/docker-ce.repo

↓

2. add six lines. [:wq!]

3. yum install ------plugin -y

→ Systemctl start docker.

" enable "

" stop "

→ Steps: Go to google → kubeadm Install.

Installing kubeadm, kubelet and kubectl

Redhat based distribution [commands]

→ copy all commands // Systemctl enable kubelet

→ Step2: Creating a cluster on master → ifconfig

[root@master] kubeadm init --pod-network-cidr=

10.10.0.0/16 --apiserver-advertise-address = ip

31.80.129 [master Node Ipaddress] " Error resolve

2. → root user → export KUBEconfIG = /etc/ kbs's/admin Config

→ Token → Joins command copy.

→ How do you known nodes Join in master.

kubectl get nodes.

→

Installing pod N/W

kubectl apply -f https://github.com/coreos/flannel/raw/master/Documentation/kube-flannel.yml

portIssue:- kubectl get nodes

* Node configuration:

Connect Node1 Instance (Ipvu+ key) => Login. [both Nodes should be connected parallely. [2 terminals]

Step1. yum update -y

Step2. yum install nettools vim -y

Step3. host name Node1

vi /etc/hostname [Node1] : LN2!

ifconfig.

vi /etc/hosts.

172-31-81-181 Node1

172-31-87-254 Node2 : LN2!

Step4. ~~AA~~ cat /etc/hosts.

Adding Node machine address in to master file [root@master]

mastermachine: 172-31-80-129 master

node1

node2

Steps: Install docker on Node1 Node2.

1. vi /etc/yum.repos.d/docker-ce.repo

↓

2. add six lines

3. yum install --- plugin ---Y

→ systemctl start docker | both Nodes

" " enable "

Steps:

* Swapoff -a [to stop vm]

→ Google → Redhat distrubution → commands. [4 com add

→ Delet a file → rm /etc/containerd/config.toml

Kubeadm Join Commands: [copy]

Errors on master clusterNode

→ kubeadm reset . | AA Container runtime not run

1. rm /etc/containerd /config.toml [on master]
2. Systemctl restart containerd

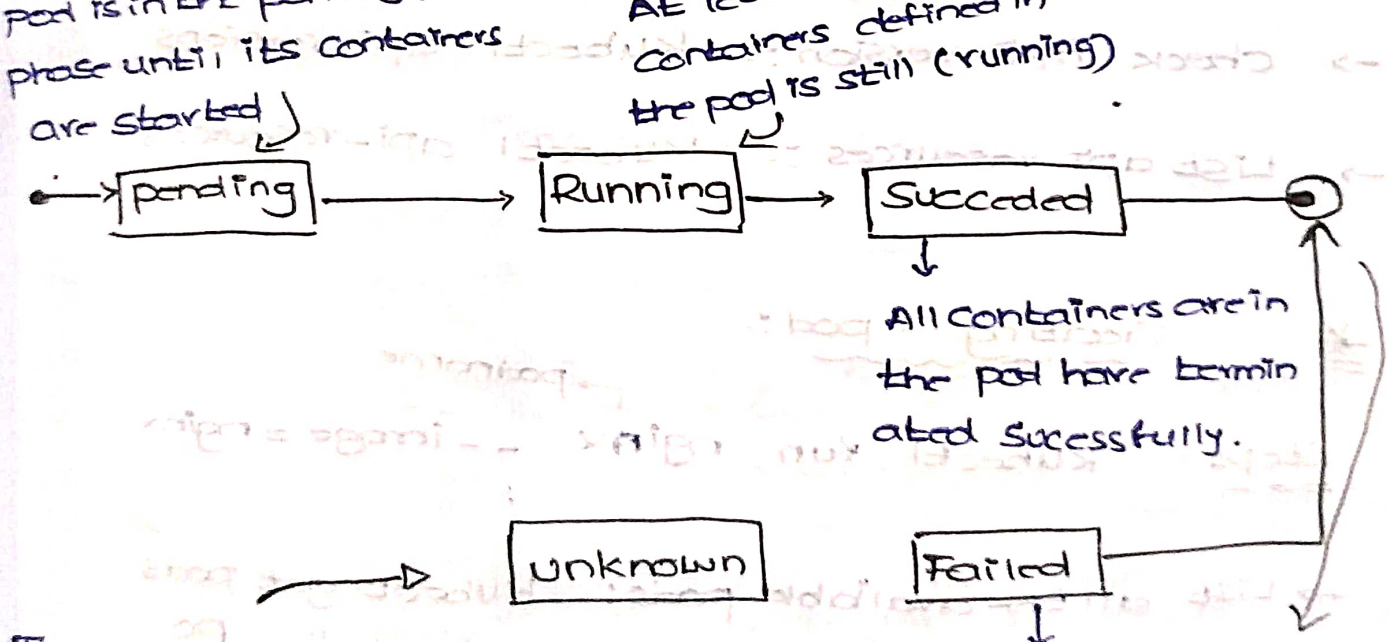
4/11/23

Pods :- A pod is the smallest & simplest kubernetes object.

- pods represent a "single Instance of a running process in your cluster".
- pods contain one or more containers, such as Docker containers
- When a pod run multiple containers, the containers are managed as single entity & share the pod's resource

Note: Running multiple containers in a single pod is an advanced usecase.

Pod Lifecycle



* Creating a pod :-

→ kubectl get nodes.

→ Complete details of node :- kubectl get nodes -o

→ 3-service need to running state. While working with cluster.

Sudo systemctl restart docker.

" " " restart containerd

" " " restart kubelet.

→ How to check the resources ?

Syntax: kubectl explain resources (pod, N/W, volume)

kubectl explain pod

→ Check Specification :- kubectl explain pod.spec (→ field)

→ Check API-version :- kubectl api-versions

→ List api resources :- kubectl api-resources

* Creating a pod :-

Step1: kubectl run nginx --image=nginx (→ podname)

→ List all the available pods: kubectl get pods

" " po

→ Check complete pod details :-

1. kubectl describe pods nginx.
2. kubectl get pods -o wide.

→ create a pod with the help of .yaml file

Step1: Create a pod with yaml-file.

3-services → Running.

pods write in microsoft visualcodes [yaml]

Step2: Install microsoft visual studio ..→ VS download free,,

Newfile:-

```
apiVersion : v1
kind : pod
metadata :
  name : my-pod.
  labels :
    app : myapp
Spec :
  containers :
  -name : myapp-container
    image : nginx
    command : ['sh', '-c', 'echo Hello Raju...
              && sleep 3600']
```

→ Create a file ..→ Vim pod-yml → Samecode,,

kubectl create -f pod-yml

kubectl get pods.

kubectl get describe pods my-pod.

* Creating multiple containers in a pod.

→ vi mpods.yaml

```
apiVersion : v1
kind : pod
metadata :
    name : my-site
    labels :
      app : web
Spec :
  containers :
   -names : front-end.
    image : nginx  → c1
    ports :
      - containerport : 80.

   - name : rss-reader.                    → c2.
     image : nickchase/rss-php-nginx:v1
     ports :
       -container ports : 88.
: wq!
```

→ kubectl create -f mpods..yml

kubectl get pods

⊛ Delet a pod.

kubectl delete pods my-pod

nginx

* Labels: It is in the under metadata resources

→ Labels are key-value pairs that are attached to objects, such as pods.

→ It is used to specify "identifying" attributes of objects that are meaningful and relevant.

→ Label can be used to organize and to "select subset of objects.

Ex: 1000 pods

400 - pro - team
200 - dev - team
:

```
"metdata": {
    "labels": {
        'key1' : "value1",
        "key2" : "value2"
    }
}
```

* ANNOTATIONS.

It is also "key-value" pair for connecting non-identifying metadata with object.

→ These are not used to identify and select object

```
"metadata": {
    "annotations": {
        'key1' : "value1"
    }   "key2" : "value2"
}
```

* Example of label.yml

→ vim label.yml

```
apiVersion : v1
kind : pod
metadata :
  name : label-demo
  labels :
    environment : production
    app : nginx
    ver : 1.4
    proj : airtel

Spec :
  containers :
    -name : ngix
     image : nginx : 1.14-2.
     ports :
       containerport : 80
```

:wq!

→ Kubectl Create -f label-yml

→ Kubectl describe pods ~~pod~~ label-demo.

5/1/23

To Connect a pod ?

kubectl exec -it my-pod --bash

↳ podname.

[my/pod] ---exit.

Label Selectors:

→

1. Labels doesn't provide uniqueness, In general, we expect many objects to carry the same labels.

→ The label selector is the "core grouping primitive in kubernetes.

→ It support two-types:

Equality Based Requirement:-

→ Equality (or) Inequality-based requirements allows filtering by label keys & values.

→ matching objects should satisfy all the specified labels

→ Supported operators are: =, ==, !=.

Set-Based Requirement:

→ It allow filtering keys according to a set of values

→ Supported operators are: in, notin, exists.

* How to filter pods with labels :-

kubectl get pods.

kubectl get pods --show-labels → display labels

→ to list labels for filtering

EQR → kubectl get pods -l environment = production

SBR :- kubectl get pods -l 'environment in (production)'

→ kubectl get pods -l 'environment in (production, development)'

* Annotation Example :

vim annotation-yml

```
apiVersion : v1
kind : pod
metadata :
  name : annotation-demo :
  annotation :
    provider : raju.
    mailid : rnrajuuu@gmail.com.
    imageregistry : "https://hub.docker.com/"
Spec :
  containers :
    -name : nginx
     image : nginx : 1.14.2.
  ports :
  -container port : 80 → : wq !
```

Example

→ Given object, metadatalls → label

↓

container details.

prod - Airtel

prod - produ

Web - Apache

Annotation: Url from where webserver download.

---

* kubectl create -f annotation.yml

kubectl get nodes.

kubectl describe pods annotations-demo

↓

detail → It run on Node1 decided by master node

* NAMESpaces: [Important]

→ Namespaces provide a mechanism for isolating group of resources within a single cluster.

[or]

→ It is a way to organize cluster into virtual sub-clusters --- they can be helpful when different teams or projects share a kubernetes cluster.

→ Names of resources need to be unique with in a namespace, but not across namespaces.

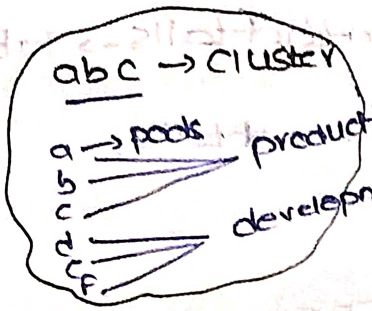
→ Kubernetes starts with "four initial namespaces".

→ default

→ kube-node-lease.

→ kube-public

→ kube-system



→ (namespace for each environment)

→ podnames are unique within namespace.

→ kubectl get ns [namespaces] → To list namespaces

→ How to create a namespace :

→ Nameofns.

kubectl create namespace development

kubectl get nodes.

→ TO switch namespace :

kubectl config set-context --current --namespace

= development.

kubectl get pods. [No resource (or) pods not found.

→ Create a pod under development :-

vim pod.yml

kubectl creat -f pod.yml

kubectl get nodes.

Switch back :

kubectl config set-context --curent --namespace

= default.

→ Switch to development.

* Create a pod on development namespace :-

vim namespace.yml

apiVersion : apps/v1
kind : Deployment.

Ex. apiVersion : v1
kind : pod
metadata:
name : label-demo.
namespace : development.
labels :
environment : production.
app : nginx.
proj : airtel

Spec :
containers :
- name : ngnix.
image : nginx : 1.14.2.
ports :
container.port : 80 : W2!

→ kubectl create -f label.yml
kubectl get nodes

Switch to : kubectl config set-context --current
--namespace = default

→ kubectl get all -A

* To delete a namespace :-

→ kubectl get ns.

→ kubectl delete ns development.

→ kubectl delete pods ↘ kubectl get pods.

→ kubectl delete pods annotations-demo label-demo my-pod [deleting multiple pods]

→ kubectl get nodes

6/1/23.

→ maintain application H.v

Kubernetes Deployment

→ It is used to create (or) modify instance of the pods, that hold a containerized application.

→ A deployment provides declarative updates for "pods and Replicaset".

Usecase of deployment :-

→ Create a Deployment to rollout a Replicaset.

→ Declare the new state of the pods.

→ Rollback to an earlier deployment reversion.

→ Scale up the deployment to facilitate more load

→ pause the rollout of a deployment.

→ Use the status of the Deployment.

→ Clean up older replicasets.

* ReplicaSETS :- (RS)

→ A Replicaset (RS) is a kubernetes object used to maintain a stable set of replicated pods running within a cluster at any given time.

→ It is often used to gurantee the availability of a specified number of identifical pods.

Example :- vim deployments.yml

```
apiVersion : apps/v1
kind       : Deployment.

metadata :
  name : nginx-deployment.
  labels :
    app : nginx

Spec :
  replicas : 2.
  Selector :
    matchLabels :
      app : nginx.

  template :
    metadata :
      labels :
        app : nginx

    Spec :
      Containers :
      - name : nginx
        image : nginx : 1-14-2.
        ports :
        - container : 80
```

:wq!

→ Kubectl create -f deployments.yml → it gives pod only.

Kubectl get pods.

→ Connect a pod :-

Kubectl exec -it nginx-deployment-<tab>....

--bash.

→ exit.

* To check the roll out status :- → Name

→ Kubectl get deployments

→ Kubectl rollout status deployment/ nginx-deployment.

→ Kubectl get rs.

→ Label name : Kubectl get pods --show-labels

*. To modify the Replicaset value :

Goto → vi deployments.yml → under specification change the value "replicas: 5"

Note: Kubectl apply -f deployments.yml

↓

only Warning message.

→ Kubectl get deployments.

→ Kubectl get pods. → (5 pods displays)

*. How to rollout to back (5-2) :-

kubectl get rs.

→ To rollout replicasets :- [yaml fixed]

kubectl rollout undo deployment/nginx-deployment

* Rolling back a deployment:

Sometimes you may want to rollback a deployment

For Example: When the deployment is not Stable, such as "Cross-looping".

→ To check history of deployment :-

kubectl rollout history deployment/nginx-deployment

↓

Revision = 1 [1 time rollout]

→ kubectl rollout history depolyment/nginx-deployment

--revision=1

→ Rollback to a specific revision :- [--to]

kubectl rollout undo deployment/nginx-deployment

--to-revision = 1

→ To verify deployment description :-

kubectl describe deployment nginx-deployment

* Scaling a Deployment :- [scale]

→ kubectl Scale deployment/nginx-deployment

--replicas=10

→ kubectl get deployments

→ (only updation, set limation cpu limits,)

* pause & Resuming a deployments :-

→ kubectl rollout pause deployment/nginx-deployment

kubectl get deployments.

* Change the image of the deployment. [set image]

→ kubectl set image deployment/nginx-deployment

nginx = nginx : 1.16-1

→ kubectl rollout history deployment/nginx-deployment

→ kubectl get rs.

→ kubectl describe deployment nginx-deployment.

Tradition method	Kubernetes
Dep → Web 2.2 ↓ 2.4.	Set Image = 2.4 //
Command → yum update --y	--cpu=8.
Web: 2.4	-- memory = 512
Increase, (cpu (or) memory	
takes time	

* Resource: kubectl set resources deployment/ nginx -c=nginx --limits=cpu=200m, memory=512Mi

→ kubectl describe deployment nginx-deployment

* Resume the deployment rollout and observe a new replica set coming up with all the new updates.

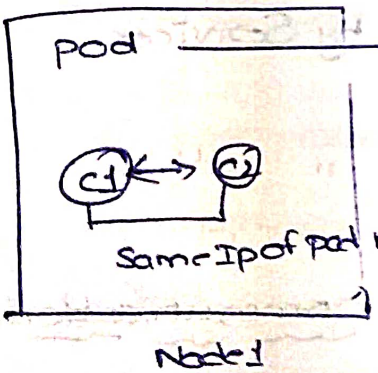
→ kubectl rollout resume deployment/nginx-deployment.

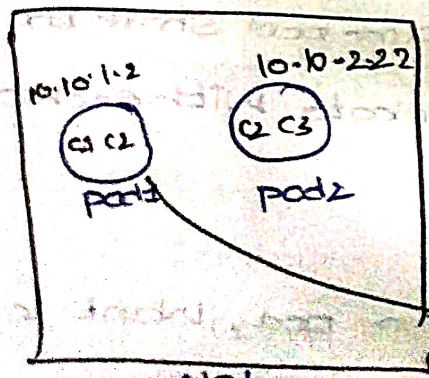
→ kubectl get rs -w.

→ kubectl get rs.

→ kubectl rollout status deployment/nginx-deployment

7/1/23:- KUBERNETS NETWORKING [cluster N/W]





→ Every pod maintain different IP addrs

pod1 → pod2 → communication
Node1 → Node2 →
overlay N/W

external use (service) expose outside of N/W (user)

K8's N/Working :

→ It allow kubernets components to communicate with each other and with other application.

→ K8s Networking is based on a "Flat Network" structure that eliminates the need to map host ports to container ports.

→ 2 types of Communication :-

1. Inter-node communication
2. Intra-node communication.

4 Networking problem To Address :-

→ Container to Container Communication : Sloved by pods and localhost cmu

→ pod to pod " :- The primary focus of this document.

→ pod to Service " :- Covered by Services.

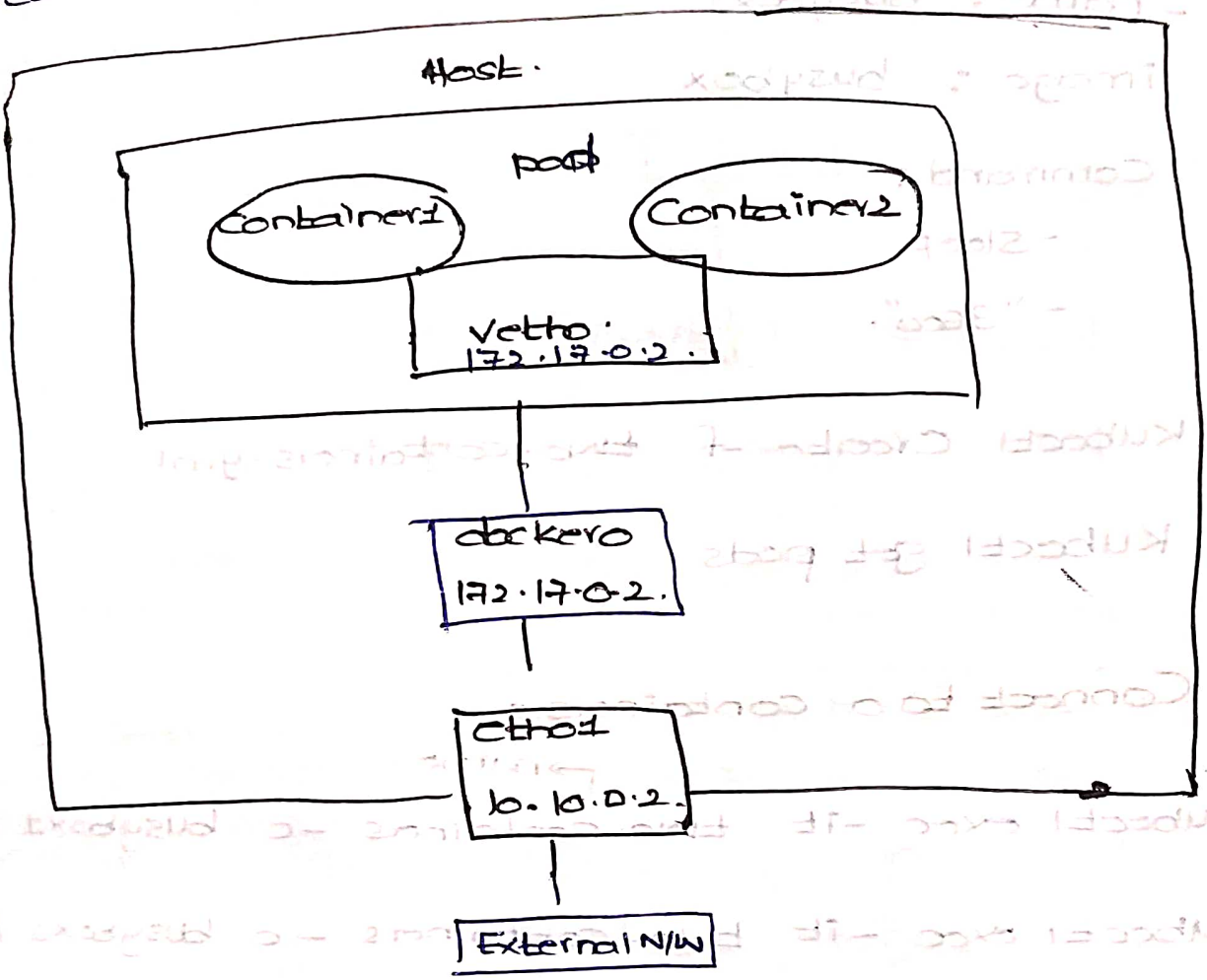
→ External to Service " :- " " "

Container to Container communication :-

→ multiple containers in the same pod share the same Ipaddress. They can communicate with each other by addressing localhost.

For Example, if a container in a pod, want to reach

another container in the same pod on port 8080, it can use the address localhost:8080. You couldn't have two containers in the same pod which expose Port 8080, because there would be a conflict.



* Create a pod with two containers ?

```
vi two-containers.yaml

apiVersion : v1
kind : pod
metadata :
    name : two-containers.
Spec :
    Containers :
    - name : busybox1
```

```
image : busybox
command :
  - sleep
  - "3600"

- name : busybox2
image : busybox
command :
  - sleep
  - "3600".
```

→ kubectl create -f two-containers.yml

kubectl get pods

1. Connect to a containers :-

kubectl exec -it two-containers -c busybox1 /bin/s (-c → name)

kubectl exec -it two-containers -c busybox2 /bin/s

⇓

Result verify Ipaddress are same

2. pod to pod communication :

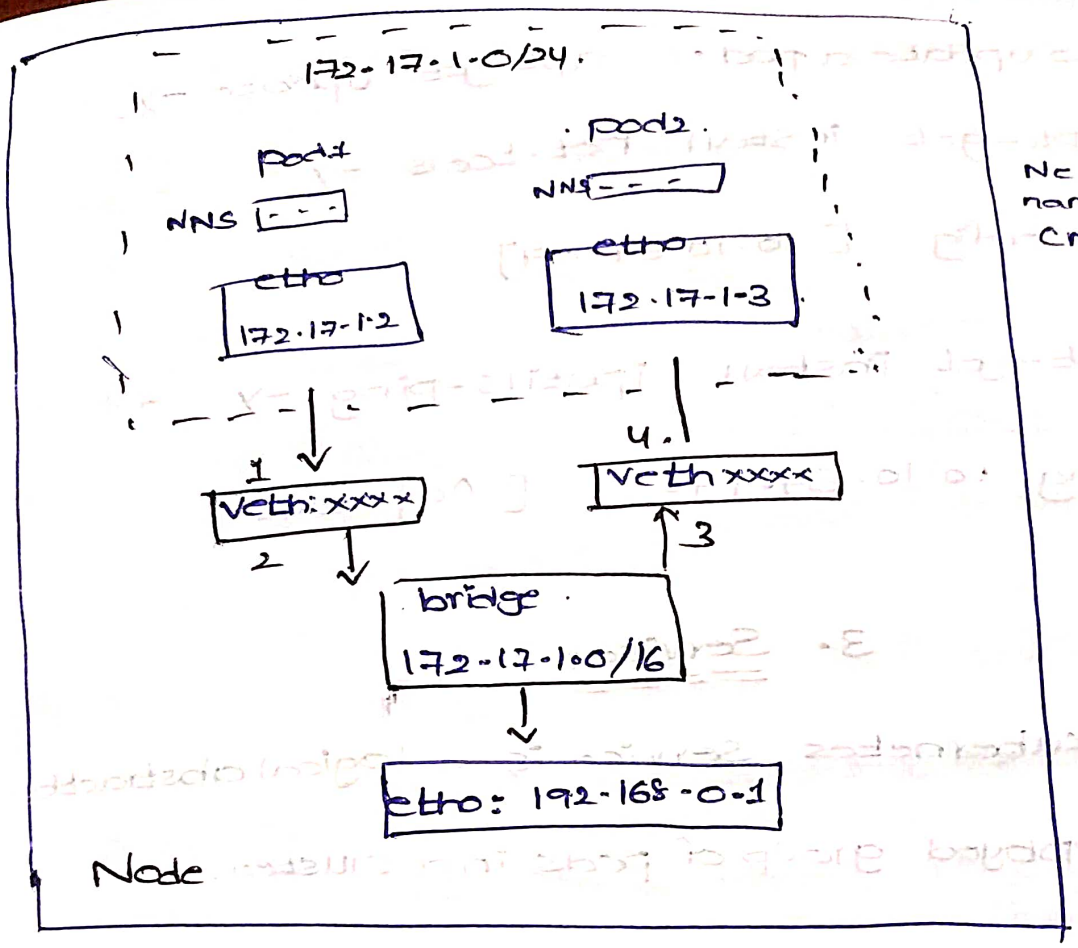
→ · In kubernetes, each pod has its own Ipaddress.

At a very primitive level, pods can communicate with each other using their Ip address.

→ pod to pod communicate requirments :

Layer2 (Switching) solution.

Layer3 (Routing), overlay solution.



*. Create two pods ?

→ Create a pod 1 : kubectl run nginx1 --image=nginx

2. kubectl run nginx2 --image=nginx.

→ kubectl get pods.

→ To check Ip address of pod:-

kubectl describe nginx1 | grep -i ip [Ip = 10.10.69.197

kubectl describe nginx2 | grep -i ip [10.10.69.198]

→ Connect to pod1:-

kubectl exec -it nginx1 --bash

* To update a pod :- apt-get update -y

→ apt-get install net-tools -y

→ ifconfig [10.10.69.197]

→ apt-get install iputils-ping -y

→ ping 10.10.69.198 [^c] exit

3. Service :-

→ A kubernetes service is a logical abstraction for a deployed group of pods in a cluster.

→ It is exposing an application running on a set of pods as a Network service. Which can be reached by a single fixed DNS name (or) Ip address.

→ The set of pods targeted by a service is usually determined by a "Selector".

→ There are various Service types options for exposing a services outside of your Cluster Ip address.

Service types :

1. Cluster Ip : Expose a service which is only accessible from "within the cluster"

2. Nodeport : Expose a service via a static port on each node's Ip

Node port must be in the range of 30,000-32,767

LoadBalancer: Exposes the service via the cloud provider's load balancer.

ExternalName: maps a service to a predefined external name field by returing a value for the CName record (Canonical)

8/1/23

ClusterIp:

It expose a service which is only accessible from within the cluster.

Ex:- Create a deployment clusterdeploy-yaml

```
apiVersion : apps/v1
kind : Deployment
metadata:
  name : nginx-Deployment.

Spec:
  replicas : 2
  Selector :
    matchlabels :
      app : nginx.

  template :
    metadata:
      labels :
        app : nginx
```

```
Spec:
  Containers:
  - name : nginx.
    image : nginx-latest.
    ports:
    -containerport: 80        : 1\12!
```

→ kubectl creat -f cluster-deploy.yml

kubectl get pods.

* Create a Cluster Ip for the depbyment;

vim Cluster-Service.yml

```
apiVersion : v1
kind : Service.
metadata:
  name: my-service.
Spec:
  type: ClusterIp.
  Selector:
    app: nginx.
  ports:
  - protocol : Tcp
    port : 8080
    targetport : 80          : W12!
```

→ kubectl create -f Cluster-Service.yml

Kubectl get service.

Kubectl describe service my-service ← Name

3. Nodeport :- Expose a service via a static port on each node Ip address

→ Nodeport must be in the range of "30000 to 32,767"

create a deployment :-

vim nodeport deployment.yml

apiversion : apps/v1

kind : Deployment.

metadata :

name : nginx_Deployment.

labels :

app : nginx-app.

Spec :

replicas : 1.

Selector :

matchlabels :

app : nginx-app.

template :

metadata :

labels :

app : nginx-app

Spec :

containers :

- name : nginx-np-container

image : nginx-latest.

container :

- containerport : 80

* kubectl create -f nodeport.yml

kubectl get pods.

* Create a service:

vim nodeport-service-yaml

```
apiVersion : V1
kind : service.
metadata:
  name : my-service-nodeport.
  labels :
    app : nginx-app.
spec :
  type : NodePort
  ports :
  - protocol : Tcp
    port : 80.
    targetPort : 80.
    nodeport : 31000
  Selector :
    app : nginx-app
```

→ kubectl create -f nodeport-service-yaml

kubectl get service

kubectl get describe service my-service-nodepo

* Node1 → IpAddress
  Node2 → IpAddress

→ Go to Web browser, then add NodeIPaddress : 3000.

*. Kubernetes Storage :-

Kubernetes volumes: Volumes in a Kubernets is a directory which is accessible to the container in a pod.

→ volumes allow user to Store Data outside of the container.

→ These are not limited to any container. It supports any or all the containers deployed inside the pod of K8's

→ It support different kind of Storage Wherein the pod can use multiple of them at the same time.

Volume types in K8's.

→ aws Elastic block Store
- azure file.
- cephfs.
- emptydir.
- fc (fibre channel)
- gitRepo.
- iscsi
- nfs.
- gcePersistentDisk
- azureDisk
- cinder
- hostpath.
- flocker.
- glusterfs
- local
- vSphere volume - etc

EMPTYDIR :- [Tmp directory]

→ Volume is created when a pod is assigned to a Node. It is active as long as the pod is running on that node.

Volumes is initially "empty" and the containers in the Pod can read & write the files in the "emptyDir volume"

-> When the pod is removed from the node, the data in the "emptyDir" is Erased.

* kubectl explain pod.spec.volumes [command]

1. Create a pod for Empty directory.

vim emptydir-pod.yml

```
apiVersion: v1
kind : pod
metadata:
  name : emptydir-one-container.
Spec:
 containers:
 -image : Centos:7
  Command:
    Sleep
    "3600"
  name : test-container
  VolumeMounts:
    mountpath: /tmp.
     name : tmp-volume.

 volume-Mounts:
 - mountpath:

 volumes:
    name: tmp-volume.
    emptyDir : {}
```

kubectl creat -f emptydir-pod.yml

→ Kubectl describe pod emptydir-one-container

→ connect to pod:

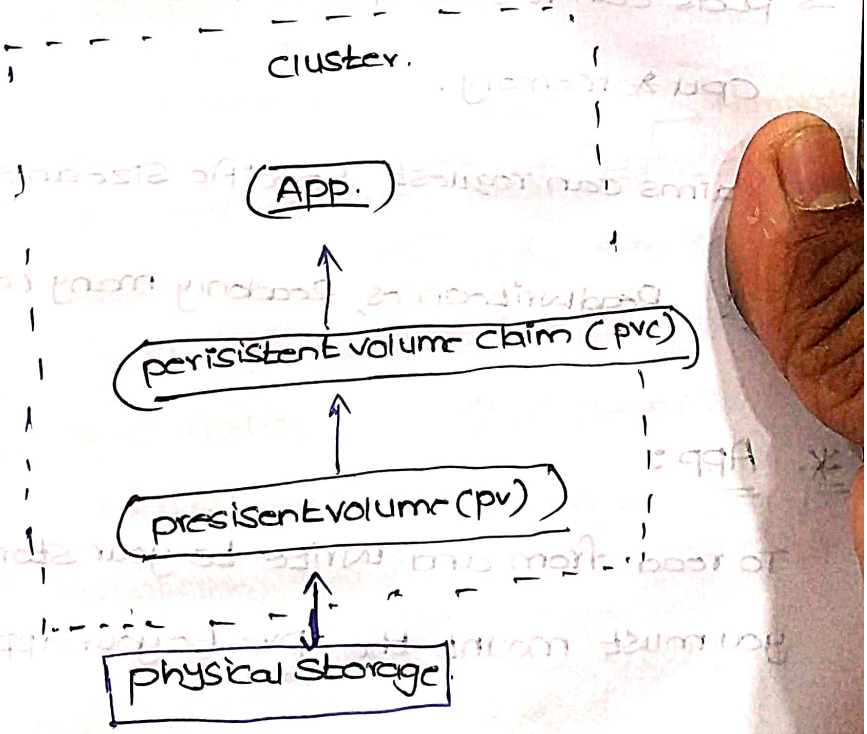
Kubectl exec -it empdir-one-container -- bash

Hostpath: [Important]

→ A hostpath volume mounts a file (or) directory from the hostnode's filesystem into your pod.

→ The hostpath volume mounds a resource from the host node filesystem.

→ pods running on the same node and using the same path in their host path volume.



* physical storage:-

A physical storage instances that you can use to persist your data.

* persistent volume (pv):

→ pv is a piece of storage in the cluster that has been provisioned by an administrator (or) dynamically provisioned using storage class.

→ It is a resource in the cluster just like a node is a cluster resources.

* persistent volume claims (pvc):-

→ pvc is a request for storage by a user.

→ It is similar to a pod. pods consume node resources and pvc consumes pv resources.

→ pods can request specific levels of resources like cpu & memory.

→ Claims can request specific size and accessmodes

Ex: ReadWriteonces, Readonly many (or) R/W many.

* App:

To read from and write to your storage instance you must mount the pvc to your app.

* Cluster:

By default, every cluster is setup with a plug-in to provision file storage.

→ you can choose to install others add-one, such as the one for block-storage.

→ To use storage in a cluster, you must create a persistent volume claim (pvc), a peristant volume and a physical storage instance.

Note:- Different storage types have different read-write rules.

* Create a persistant volume (pv) :-

Step1:

vim pod-pv-yml

```
apiVersion: v1
kind : persistentVolume.
metadata :
   name : my-pv.
Spec :
   StorageClassName : local-storage.
   Capacity :
      Storage : 2Gi
   accessModes :
    - ReadWriteOnce.
   hostPath :
      Path : "/mnt/data"
```

→ kubectl create -f pod-pv-yml

kubectl get pods.

kubectl get pv

* With the help of pv we need to create pvc,

Step2: vim pod-pvc.yaml

```
apiVersion: v1
kind: persistentVolumeClaim.

metadata:
  name: my-pvc

Spec:
  storageClassName: local-storage.
  accessModes:
    - ReadWriteOnce

  resources:
    requests:
      Storage: 1Gi
```

Kubectl create -f pod-pvc.yml

kubectl get pods.

kubectl get pvc

* Create a pod :-

Step3:- vim pod-hostpath.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: pvc-pod

Spec:
  containers:
    -name: busybox
```

image : busybox.

Command : ["/bin/sh", "-c", "while true;
do sleep 3600; done"]

VolumeMounts:
- mountPath : "/mnt/storage"
  name : my-storage.

mount to this location.

volumes:
- name : my-storage.
  persistentVolumeClaim:
    claimName : my-pvc → 1GiB : W2!

→ kubectl create -f pod-hostpath.yml

kubectl get pvc.

kubectl get pods.

Kubectl describe pods pvc-pod

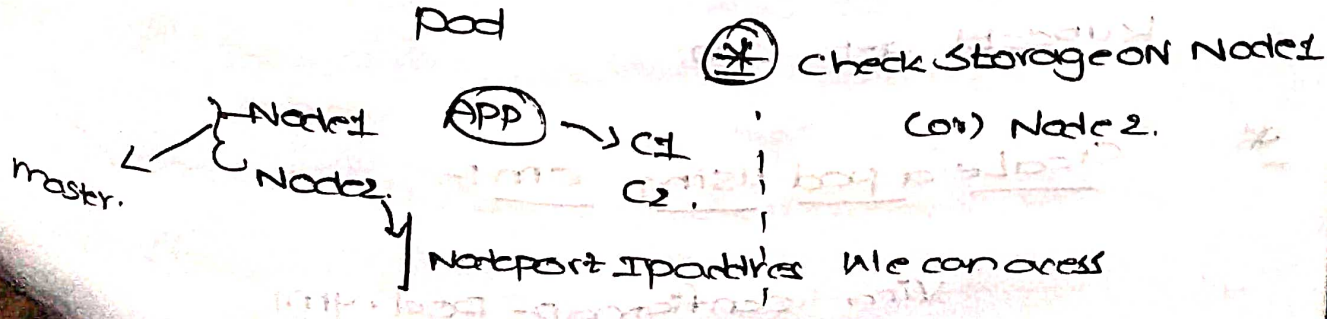
Steps:-
⇒ connect to a pod

kubectl exec -it pvc-pod -- bash

cd /mnt/Storage.

* Delete: kubectl delete pod pvc-pod.

kubectl delete pod my-pv.



9/1/23: K8's ConfigMaps & Secrets : [dynamic=running]

→ A configMap is an API object used to store non-confedential data in key-value pair.

→ It can bind configuration files, command-line arguments environment variabls, port numbers and each other config-uration artifacts to your pod's container and System components at runtime (dynamically) at the pod runti...

→ These are useful for storing and sharing Non-Sens... unencrypted configuration information.

Create a configmap:- vim configmap-yml

[data]

```
apiVersion : v1
kind : configMap
metadata :
    name : my-configmap
data :
    mykey : myvalue
    anotherkey : anothervalue
```

→ kubectl create -f configmap-yml

kubectl get cm

* Create a pod using cm :-

vim configmap-pod.yml

```
apiVersion : v1
kind : pod
metadata :
  name : my-configmap-pod
spec :
  containers :
  - name : my-container
    image : alpine
    command : ['sh', '-c', "echo $(my-var) && sleep
                                3600"]
env :
- name : my-var
  valueFrom :
    configMapKeyRef :
      name : my-configmap
      key : mykey                    : w2!
```

1st way :- kubectl create -f configmap-pod-yml
kubectl get pods.

2nd way : kubectl logs my-configmap-pod.

SECRETS :

→ A Secret is an object that contain a small amount of sensitive data such as password, a token or SSH keys, authentication tokens etc.

→ Secrets are similar to ConfigMaps but are specifically intended to hold confidential data.

→ User can create secrets and the system also create some secrets.

User for Secrets:

→ As files in a volume mounted on one or more of its container.

→ As container environment variables.

→ By the kubelet

* managing a Secret:. Several options to create a Secret:

1. using kubectl
2. use a configuration file.
3. use the kustomize tool

* Create a Secret using kubectl command :.

Step1: Encode a password

↳ It will not take any newline of text.

echo -n 'admin123' | base64

o/p: YWRtaWUxMjM=

Step2: Creat a Secret

-: kubectl create Secret generic db-pass \

--from-literal = username = admin \

--from-literal = password = 'YWRtaWUxMjM='

→ kubectl get Secrets. (type) opaque = encrypted

Decode a password :-

```
echo -n 'YWRtaWUxMjM=' | base64 --decode
```

* Create a Secret using Files :-

Step1 : create 2 files, with user and password

```
echo -n 'admin' > ./username.txt
echo -n 'YWRtaWUxMjM=' > ./password.txt
ls. -> cat username.txt
       cat password.txt
```

Step2 :- Create a secret from files :-

```
Kubectl Create Secret generic db-user-pass
--from-file = username = ./username.txt \
--from-file = password = ./password.txt
```

-> kubectl get Secrets.

-> To verify the Secrets : kubectl describe db-user-pass.

* To modify (or) Edit Secrets :

```
kubectl edit Secrets db-user-pass
```

* Create a pod :-

```
vim Secrets-pod-yml
```

```
apiVersion : v1
kind : pod
metadata :
   name : mypod-volumeSecret.

spec :
 containers :
 -name : volumeSecret.
  image : redis
  volumeMounts :
  -name : foo
  mountpath : "/opt/secrets"
  readonly : true

 volumes :
 -name : foo
  secret :
   SecretName : mySecret-manifest : lvl2
```

-> kubectl create -f secrets-pod.yml

-> kubectl get pods.

* To describe pod :

kubectl describe pod mypod-volumeSecret.

* Connect to a pod :-

kubectl exec -it mypod-volumeSecret --bin/sh

cd /opt/secrets.

ls

Certifications :- docker

Certified kubernetes Adminstrator.

" " Applications developer.

" " Security Specialist "

19/01/23.

ANSIBLE

→ An ansible is a open-source software provisioning, configuration management and application deployment tool

→ Ansible is simple to use for system adminstrators & developers because it is built on python.

→ It is included in as a part of Fedora distrubution of linux (Redhat, centos, debian, OEL) , etc

→ Ansible's architecture is "Agentless". Work is push to remote host" When ansible executes

Ansible Automation :

→ It is used by thousands of organization globally to help them automate IT task, such as configuration managment, provisioning, Workflow orchestration, application deployment and life-cycle managment