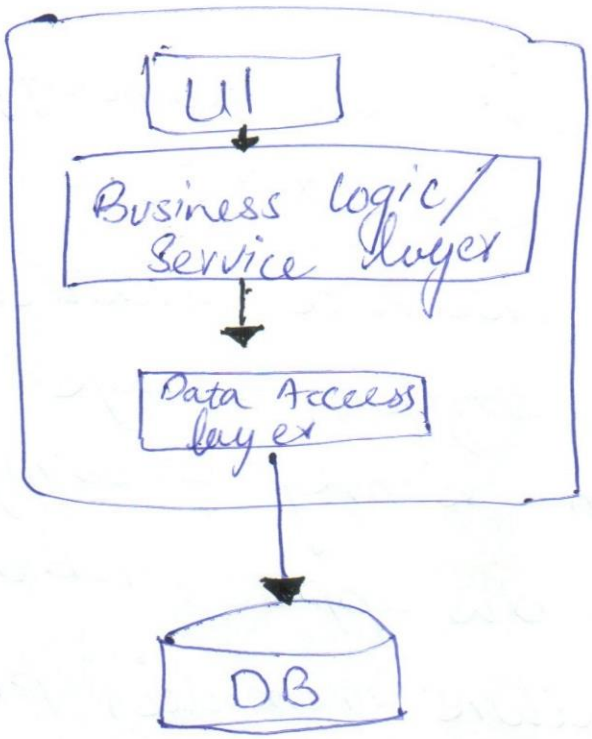
CNC
Part III
By Aamir Pinger

KUBERNETES

10:53 PM
9-JUN-2020
TUESDAY

* Traditional Architecture :-

- Monolithic : composed all in one piece



- Monolithic software is designed to be self-contained meaning having all that is needed in itself.

PROBLEM :-

⊖ slow release cycle and are updated relatively infrequently.

⊖ Once new release get ready, developers package up the whole system and hand over to the ops team, who then deploy and monitor it.

⊖ In case of hardware failures, the ops team manually migrate it to the remaining healthy servers.

* MICROSERVICES :-

- It is breaking down big monolithic component. application into smaller independent app.

- Decoupled
- Scaled individually.
- Easy to adopt new technology.

MICROSERVICE COMPLEXITIES :-

- In bigger microservice architecture it is very difficult to configure, manage and keep the whole system running smoothly.
  - ① Data linking b/w apps. problem
  - ② hardware failure remedies problem

What is kubernetes :-

1. Automated deployment process
2. No Ops team requirement
3. Help Ops team to transfer/migrate from
   - One server/hardware to another.
4. One or thousand servers seems as a single server. to provide ease in deploying.

What is kubernetes Continued

1. kubernetes select from range of servers and decide how micro service talk to other micro service.
2. Provide ease for cloud providers

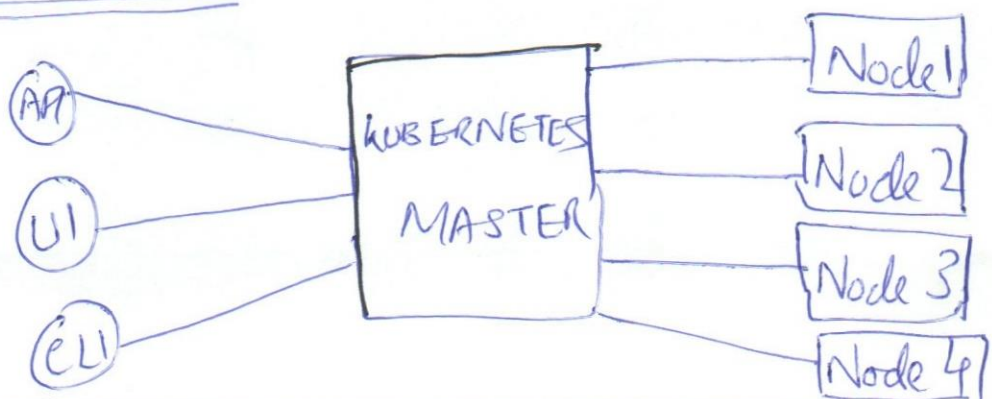
KUBERNETES ORIGIN :-

- Google created a system for their internal use with the name "Borg"
- later name changed with "Omega"
- 2014, Google introduced kubernetes an open source system based on the experience

BROADER PERSPECTIVE :-

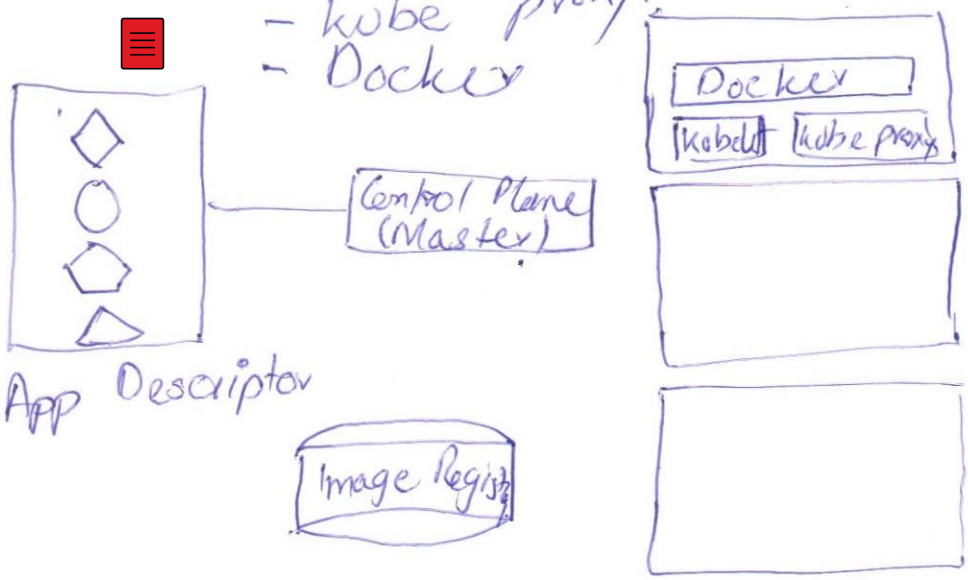
- Simplify the infrastructure layer.
- Resource allocation.

* KUBERNETES ARCHITECTURE



KUBERNETES BROADER VIEW

1. App descriptor.
2. Control Plane (Master Node)
3. Worker Nodes
   - kubelet
   - kube proxy
   - Docker



LESSON 2

PODS

18-06-2020
10:58 PM
Thursday

1) MINIKUBE & KUBECTL :-

Minikube creates single node cluster of kubernetes.
(Master + Worker node at a same time)

```
# minikube start
```

⊙ To check status :-

```
# minikube status
```

⊙ To check kubectl cluster info :-

```
# kubectl cluster-info
```

2) NODES

⊙ To check list of worker & master nodes

```
# kubectl get nodes
```

⊙ To describe nodes

```
# kubectl describe node minikube
```

3) ALIAS :-

To short command kubernetl get nodes

```
# alias kgn = "kubectl get nodes"
```

* Pods it is basically a like a wrapper or a seperate machine with its own IP, hostname, processes, and so on.
* Pods deploy single containerised app. on single worker node.

WHY PODS :-

* Pod have capability to group supporting containers.

E.g One container is creating logs
Second container is reading & manipulating that logs.

ABOUT PODS ISOLATION :-

- It is necessary if we share volumes then both nodes must be on same worker node.

MULTI - TIER APPS :-

Multiple container in a single pod always runs on a single worker node.

7 24

27-JUN-2020
SATURDAY
9:58 PM

GROUPING PODS

All dependent container must be grouped to run in a particular worker node

* YAML FILE (CREATING A POD)

vi myfirstpod.yaml

```
kind: Pod
apiVersion: v1
meladata:
  name: myfirstpod
spec:
  containers:
  - name: container1
    image: aamirping/helloworld:latest
    ports:
    - containerPort: 80
```

kubectl create -f myfirstpod.yaml

* POD LISTING & INSIGHT :-

kubectl get pods

POD LISTING & INSIGHT :-

① # kubectl get pods

② # kubectl get pods myfirstpod -o yaml

③ # kubectl get pods myfirstpod -o json

④ # kubectl describe pods myfirstpod

* PORT FORWARDING :-

kubectl port-forward mysecondpod 6500:80

E.g localhost: 6500

CREATING POD FROM COMMAND LINE

kubectl run mysecondpod -- image = aamirpinger/helloworld:latest --port=80 -- restart = Never

* Note:
--restart = Never is not written then kubernetes will create other type of resource

LABELS :-

- kubernetes organizes pod with the help of label
- key/value pair

vi myfirstpodwithlabels.yaml

```
- kind: Pod
- apiVersion: v1
- metadata:
    name: myfirstpod with labels
    labels:
      type: backend
      env: production
- spec
    containers:
    - image: aamirpinger/helloworld:latest
    name: container1
    ports:
    - containerPort: 80
```

kubectl create -f myfirstpod with labels.yaml

kubectl run anotherpodwith label --image=
aamirpinger/helloworld --port=80
--restart=Never --label=type=frontend,env=development

kubectl get pods --show-labels

kubectl get pods -L env, type, run
To seperate by rows

LABELING POD AT RUNTIME :-

kubectl label pods myfirstpod app=helloworld type=frontend

* To change label during runtime

kubectl label pods anotherpod withlabel env=pro --overwrite

* To remove label

kubectl label pod myfirstpod app-

* LABEL SELECTOR :-

kubectl get pods -l type=frontend

Not equal to

kubectl get pods -l type!=frontend

kubectl get pods --show-labels.

kubectl get pods -l type!=frontend,env=production --show-labels

kubectl gets pods -l 'env' --show-labels

kubectl get pods -l 'type in (frontend, backend

kubectl get pods -l 'type notin (frontend, backend)' --show-labels

* PODS SCHEDULING WITH NODE SELECTOR

nano podwithnodeselector.yaml

```
kind: Pod
apiVersion: v1
metadata:
    name: podwithnodeselector
spec:
    nodeSelector:
        type of harddisk: ssd
    containers:
    - image: aamirpinger/helloworld:latest
      ports:
      - containerPort: 80
```

kubectl create -f podwithnodeselector.yaml

- Assigning type of harddisk

kubectl label node minikube type of harddisk=ssd

ANNOTATION:

- Annotation are basically words that explaination or comment on something.

E.g creater name, contact

But it can not be used to group or filter like label

```
# nano podwithannotation.yaml

kind: Pod
apiVersion: v1
metadata:
  name: podwithannotation
  annotation:
    createdBy: "Zohaib Ahmed"
    email : zohaib4986@gmail.com

spec:
  containers:
  - name: container-new
    image: aamirpinger/flag: latest
    ports:
      - containerPort: 80

# kubectl create -f podwithannotation.yaml
# kubectl annotate pod myfirstpod app-describe
    = "this is annotation example"
```

* DESCRIBING POD'S INSIGHTS :-

kubectl describe pod myfirstpod

* OVERLAPPING LABELS

It may be possible that different pods have same label which may create difficulty for your learn while deployment or management

* NAMESPACE :-

To solve overlapping issue name space is introduced

- it is virtual box which isolate self contain resources with other namespace

CREATING NAMESPACE :-

kubectl create namespace production

kubectl get ns

POD INSIDE NAMESPACE :-

```
# kind: Pod
# apiVersion: v1
  metadata:
      name: myfirstpod
      namespace: production

  spec:
      containers:-
      - name: container1
      * image: aumirpinge/helloworld:latest
```

10:12 PM
MONDAY

REPLICASETS, JOBS AND CRONJOB :-

* REPLICA :-

- It is also a resource
- It helps managing multiple copies of application (replica) in kubernetes.

REPLICASET PARTS :-

It has three essential parts

1) Label Selector. Scope
   Determine what pods are in replicaset
2) Replica Count
   desire no. of pods must be running
3) Pod template
   ReplicaSet uses to create new pod

* CREATING REPLICASET :-

```
# nano rs.yaml

kind: Pod
apiVersion: apps/v1beta2
metadata:
  name: myrs
spec:
  replicas: 3
  selector:
    matchLabels:
      app: rsexample
  template:
    metadata:
      label:
        app: rsexample
```

Spec:
  containers:
  - name: rscontainer
  - image: aamirpinger/helloworld:latest.
    ports
      - containerPort: 80

kubect create -f rs.yaml

"LISTING RS":-

kubectl get rs myrs -o yaml

kubectl describe rs myrs

kubectl label pod myrs-sq5zz app=rsexample --overwrite.

DELETING PODS AND REPLICASET:-

kubectl delete pods -l app=rsexample
-criteria

kubectl delete rs myrs
To delete pod rs but individual pods will exist

kubectl delete rs myrs --cascade=false

- Note:- After deleting rs, if we recreate so it will function old deleted pods.

OPERATORS IN MATCH EXPRESSIONS :-

There are 4 operators in matchExpression

- In
- notin
- Exist
- Doesnot Exist.

* If match label & match expression

nano rs.yaml

```
kind: ReplicaSet
apiVersion: apps/v1beta2
metadata:
  name: myrs
spec:
  replicas: 3
  selector:
    matchexpressions:
      - key: app
        operator: In
        values:
          - vsexample
  template:
    metadata:
      label:
        app: vsexample
    spec:
      containers:
      - name: vscontainer
        image: aamirpinger/helloworld
        ports:
        - containerPort: 80
```

kubectl delete rs myrs --cascade=false

MODIFYING REPLICA SET:-

- ReplicaSet can be modified at runtime
- Only changes to the replica count will affect existing pods.
- label selector and pod template in replicaSet will only affect new containers.

kubectl edit rs myrs

SCALING REPLICA SET:-

kubectl scale rs myrs --replicas=5

JOB RESOURCE:-

- Job is another in kubernetes.
- It is basically a Pod which we create under the type (kind) of Job.

JOB RESOURCE :-

```
kind: Job
apiVersion: batch/v1
metadata:
  name: Jobexample
spec
  template:
  spec:
    containers:
    - name: jobcontainer
      image: docker/whalesay
      command: ["cowsay", "This is job resource example"]
    restartPolicy: Never
  backoffLimit: 4
  activeDeadlineSecond: 60
```

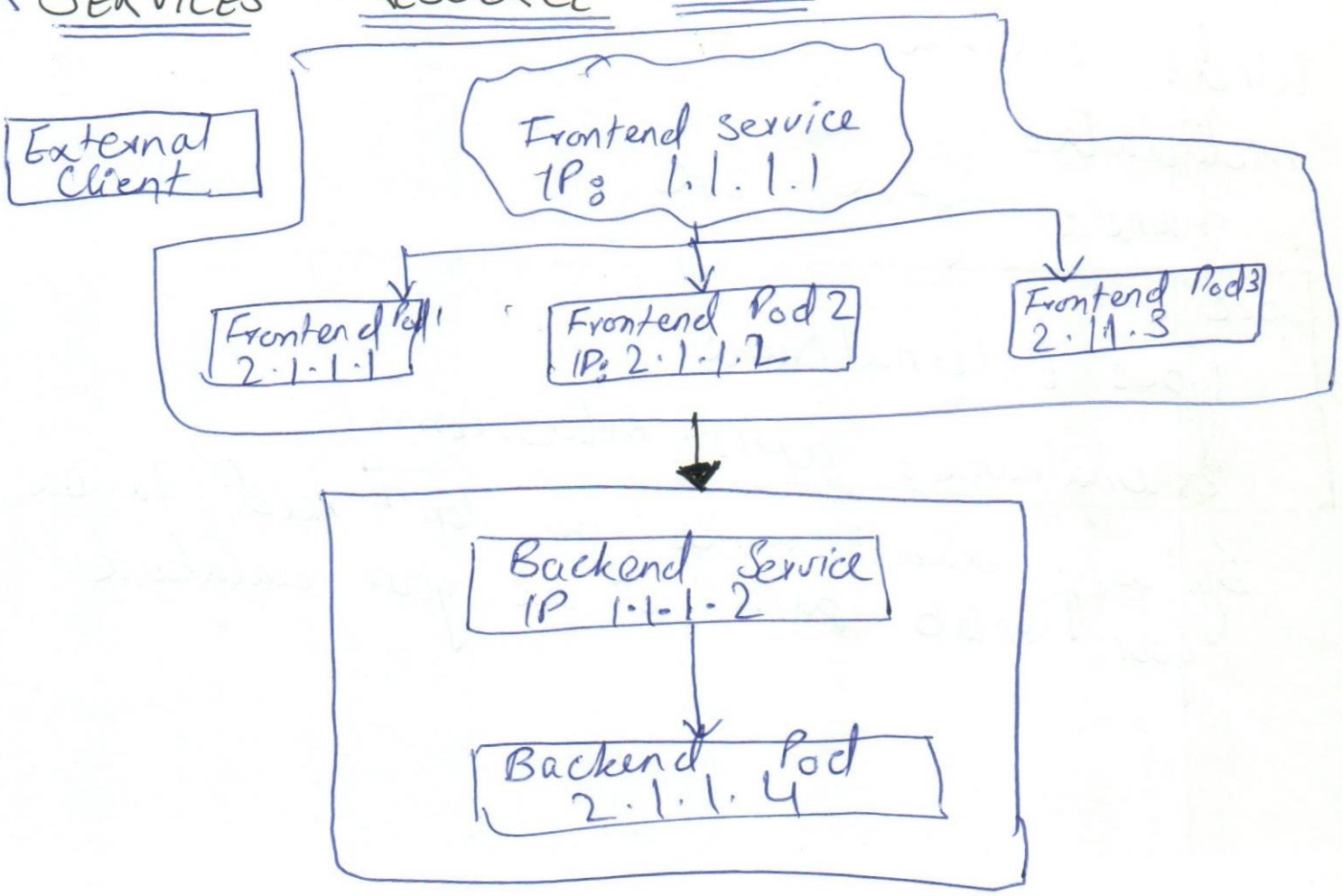
```
# kubectl create -f job.yaml

# kubectl logs jobexample-b25z9
```

Kubernetes:

"SERVICES & PROBES" :-

- Since k8s assign every pod to individual IP.
- It provides static IP to grouped of resources.
- It provides IP so that we can easily access
- It keeps the records of changing IPs if any changes & occurs at worker node.

*"SERVICES RESOURCE FLOW" :-



SERVICE RESOURCE TYPE :-

- Cluster IP
- Node Port
- load Balancer
- External Name.

EXTERNAL NAME EXAMPLE :-

```
#
apiVersion: v1
kind: Service
metadata:
  name: my-ext-svc
```

```
Spec :
  type: ExternalName
  externalname: api2.mlab.com
```

By using external service you don't need to use your mlab path to use your database

LOADBALANCER SERVICE EXAMPLE (YAML)

```
# nano mysvc.yaml

kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  ports:
  - port: 8080
    targetPort: 80
  type: LoadBalancer
  selector:
    app: rsexample.
```

```
# kubectl create -f mysvc.yaml

# kubectl get svc.
NAME          TYPE   CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes
my-service    -      -            -             8080:32468/TCP   -
```

```
# kubectl expose rs myrs --name=mysvc-lb
  --selector=app=rsexample --port 8000
  --target-port=80 --type=LoadBalancer.
```

LIVENESS PROBE :- (HEALTH CHECK)

① HTTP GET :

It checks container IP & port and if it is not get any response then it will restart that container.

② TCP SOCKET :

It tries specified port and if it is not get any response then it will restart.

③ EXEC PROBE :

If status code is 0 then container is good otherwise upon receiving other codes it will restart the container.

my-ln-http.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: myapp-ln-http
spec:
  containers:
  - name: myapp
    image: aamirpinger/hi
    ports:
    - containerPort: 80
    livenessProbe:
      httpGet:
        port: 80
        path: /
```

my-ln-tcp.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: myapp-ln-tcp
spec:
  containers:
  - name: myapp
    image: aamirpinger/hi
    Ports:
    - containerport: 80
    livenessProbe:
      tcpSocket:
        Port: 8080
```

my-ln-exec.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: my-ln-exec
spec:
  containers:
  - name: aamirpinger/hi
    Ports:
    - containerPort: 80
    livenessProbe:
      exec:
        command:
        - Cat:
        - /tmp/healthy
```

LIVENESS PROBE ADDITIONAL PROPERTIES:-

- failure Threshold: 3
- period Seconds: 10
- success Threshold: 1
- timeout Second: 1
- Initial Delay Second: 15

READINESS PROBE

The main difference b/w liveness probe & readiness probe is readiness probe check the pod wheather it is ready to entertain traffic or not.
- ht mark containes so that

READINESS PROBE TYPES

1. Http GET
2. TCP Socket
3. Exec probe.

READINESS PROBE EXAMPLE

```
apiVersion: v1
kind: Pod
metadata:
  name: vpod
spec:
  containers:
  - name: container-vp
    image: aamirping/helloworld:latest
    ports:
    - containerPort: 80
    readinessProbe:
      httpGet
        port: 80
        path: /
```

kubectl create -f vpod.yaml

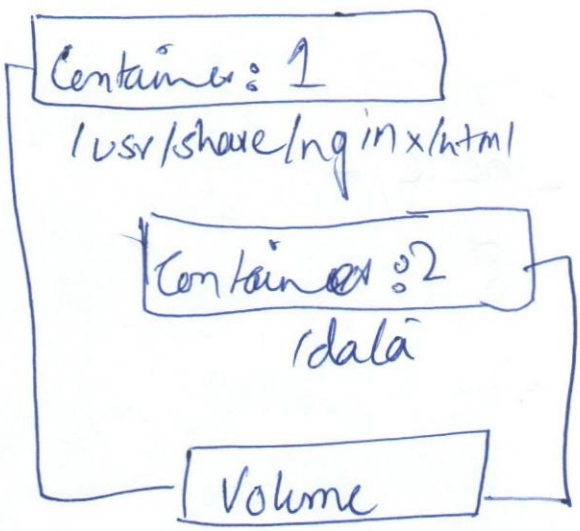
```
httpGet:
  port: 90
  path: /
```

09-July-2020
10:52PM

VOLUME

- It provide a shared folder type storage.
- It is basically a shared folder.

VOLUME FLOW :-



VOLUME TYPES

- emptyDir
- configMap, secret, downwardAPI
- persistentVolumeClaim
- gitRepo
- gcePersistentDisk
- AwsElasticBlockStore
- azure disk

nano vol.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: my-pod-with-vol
spec:
  volumes:
  - name: shared-dir
    emptyDir: {}
  containers:
  - name: container one
    image: aamirpinger/logfile_nodejs
    ports:
    - containerPort: 3030
    volumeMounts:
    - name: shared-Dir
      mountpoint: /data
  - name: container two
    image: nginx
    ports:
    - containerPort: 80
    volumeMount:
    - name: shared-dir
      mountpath: /var/c-two
```

kubectl create -f vol.yaml

8:20 PM
11-July-2020

Persistent Volume :-

- In case of pod termination data in the volume will be lost.
- To solve the issue kubernetes provide us option of Persistent Volume.
- P.V add a volume at a cluster level instead pod level.

Persistent Volume Claim :-

- It is a kind of request for claiming a persistent volume by pod.
- kubernetes bound persistent volume request by match persistent volume claim configuration

Persistent Volume Specs :-

- Access Mode
  - Read/Write Once. (RWO)
  - Read Only many. (ROX)
  - Read Write Many. (RWX)
- Reclaim Policy
  - Delete
  - Recycle
  - Retain (Default)

- DELETE

  PVC is deleted but data will persist

- Recycle

  . data / volume's contents will be deleted.

  . Persistent volume will be available to be claim again.

- Retain (default)

  . k8s will retain the vol. and its content after it released from its claim.

  . We can manully delete and recreate the persistent Volume resource.

```
# nano pv.yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv
spec:
  accessModes:
    - ReadWriteOnce.
  capacity:
    storage: 100M
  hostpath:
    path: /tmp/pvexample
  PersistentVolumeReclaimPolicy: Delete.
```

kubectl create -f pv.yaml

nano pvc.yaml

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc
spec:
  accessModes:
  - ReadWriteOnce.
  resources:
    requests:
      storage: 100M
  StorageClassName: "  "
```

kubectl create -f pvc.yaml

kubectl get pv, pvc

nano pod.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: pod-pv
spec:
  volumes:
  - name: pv-vol
    persistentVolumeClaim:
      claimName: pvc
  containers:
  - name: container1
    image: aamirpinger/logfile-nodejs
    volumeMounts:
    - name: pv-vol
      mountpath: /data
```

kubectl create -f pod.yaml

Persistent Volume Example Part 4:-

kubectl exec pod-pv -it sh.

16/7/2020 7:28 PM

CONFIGURATION APPLICATION AND MOBILE DEPLOYMENTS :-

- ConfigMaps allow you to seperate your application configuration from your application code, which helps keep your containerized application portable.
- ConfigMaps are useful for storing and sharing non-sensitive, unencrypted configuration information which you can change at runtime.
- It helps makes their configuration easier to change and manage and prevent hardcoding configuration data to pod specification.

CONFIGMAP FROM LITERALS

kubectl create configmap myfirstcm
--from-literals=fname=Aamir --from-literals
=lname=Pinger.

kubectl get cm

CONFIGMAP FROM FILE :-

nano user.txt

fname = Aamir
Lname = Pinger

kubectl create cm myfirstfromfile
--from-file = user.txt.

If we want the key to be used personalized

kubectl create cm cmfromfile --
from-file = bio = user.txt

Describe:

kubectl get cmfromfile -o yaml

kubectl describe cm cmfromfile.

CONFIGMAP FROM ENV FILE :-

nano cm.env
CREATED BY = Zohaib

kubectl create cm cmfromfile
--fromenvfile = cm.env

CONFIGMAP AS VOLUME :-

```
# nano  podwithcm.yaml

kind : Pod
apiVersion : v1
metadata :
    name : podwithcmvol
spec :
    volumes :
    - name : cmvol
      configMap : myfirstcm
    containers :
    - name : container1
      image : nginx
      volumeMounts :
      - name : cmvol
        mountpath : /data/cm
      imagePullPolicy : IfNotPresent
```

```
# kubectl create -f podwithcm.yaml

# kubectl exec podwithcmvol -c
    container1 -it sh.
    ↳ cd /data/cm
    ↳ cat lname
```

CONFIGMAP AS ENV VARIABLE

```
# nano podemenv.yaml

kind: Pod
apiVersion:
metadata:
  name: podemenv
  labels:
    app: envemexample
spec:
  containers:
  -name: container1
  image: aamirpinger/node-app-image
  imagePullPolicy: IfNotPresent
  envFrom:
  - configMapRef:
      name: cmfromenvfile
```

```
# kubectl creati -f podemenv.yaml

# kubectl exec podemenv -it sh
```

FOR SERVICE RESOURCE

```
# kubectl expose pod podemenv
    --port=8080 --target-port=8080 --type
  =loadbalancer
```

SECRET: FROM LITERALS :-

kubectl create secret generic myfirstsecret --from-literals=username= ramirpinger --from-literals=api=test.db.aws.com

kubectl describe secret myfirstsecret.

kubectl get secret myfirstsecret -o yaml

echo _____ | base64 -d

SECRET FROM FILE :-

nano keyval.txt

fname: aamir
lname: pinger

kubectl create secret generic secret fromfile --from-file=bio=keyval.txt

kubectl get secret

kubectl describe secret secret fromfile

SECRET FROM ENV FILE

```
# kubectl create secret generic secretfrom
  envfile --from-env-file=keyval.txt

# kubectl get secret

# kubectl describe secret secretfromenvfile
```

SECRET AS VOLUME:-

```
# nano podsecret.yaml

kind: Pod
apiVersion: v1
metadata:
    name: podwithsecret
spec:
    volumes:
    - name: secvol
      secret:
        secretName: myfirstsecret
    containers:
    - name: container1
      image: nginx
      imagePullPolicy: ifNotPresent
      VolumeMounts:
      - name: secvol
        mountpath: /secretdata
```

kubectl create -f podsecret.yaml

kubectl get po

kubectl exec podwithsecvol -it sh

#↳ cd secretdata

api username

#↳ cat api

test.db.aws

SECRET AS ENV VARIABLE :-

nano secenv.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: secenv
spec:
  containers:
  - name: container1
    image: nginx
    imagePullPolicy: IfNotPresent
  envFrom:
  - secretRef:
      name: myfirst-secret
```

kubectl exec secenv -it sh

env

ENVIRONMENT VARIABLE :-

nano podwithenv.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: podwithenv
spec:
  containers:
  - name: container1
    image: nginx
    imagePullPolicy: IfNotPresent
    env:
    - name: USERNAME
      value: aamirpinger
    - name: COURSENAME
      value: kubernetes.
```

To Create Pod:

kubectl create -f podwithenv.yaml

kubectl exec podwithenv env

DEPLOYMENT :-

- How to group containerized app into pods.
- Ways to provide them with temporary & permanent storage.
- How to pass both secret & non secret config data to them.
- How to run a full fledged system composed of independently running smaller components (microservices)

DEPLOYMENT USE CASE

- If you created your pod with a container image aamirpinger/hello-world:v1
- If you upgrade your image with aamirpinger/hello-world:v2
- You can't change existing pods image after the pod is created, you need to remove the old pods and replace them with new ones running the new image

DEPLOYMENT STRATEGIES PROS

PROS

- You can delete pods manually and replica set will recreate it.

- Change service resource but it will consume double resource meanwhile/ if you have deleted older pods/replica set. (Rolling update)

WHAT STRATEGY TO CHOOSE :-

- To resolve rolling update kubernetes gives us solution by providing deployment resource

PART 1

```
# name deploy.yaml
kind: Deployment
apiVersion: apps/v1 beta1
metadata:
  name: deployexample
spec:
  replicas: 4
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        -name: container1
         image: aamirping/helloworld
         imagePullPolicy: IfNotPresent
```

```
#kubectl get deploy

#kubectl get rs

#kubectl expose
deployment example --port=2000
  --target-port=80
    --type=loadbalancer
#kubectl get svc
```

PART 2 :-

# kubectl set image deploy deployexample
container=aamirpinger/flag.

kubectl get deploy, rs, pod

PART 3 :-

- Default strategy of kubernetes is rolling update
If we define strategy in spec.

nano deploy.yaml

same as above

```
spec:
  replica: 4
  =
  =
  =
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 0
      maxSurge: 1
```

OR

```
type: Recreate
```

HELPER COMMAND

kubectl rollout pause deploy deployexample

" " resume " "

kubectl rollout status deploy deployexample

kubectl rollout history deploy deployexample

same as above --revision=1 or --revision=2

Revert back:

kubectl rollout undo deploy deployexample
--to-revision=1

To check pod

kubectl get po

kubectl describe po deployexam- no - no

BEST PRACTICES

- Use diffrent tag policy like V1, V2, V3 rather use latest
- Image policy: (imagepullpolicy) should be used wisely. (always/Never)
- Use label of multi-dimension.
- Annotation as we can give bigger discription.
- Application logs.