KUBERNETES

Preethi Dovala

105) ## What is K8s?

K8s is an open source project developed by Google now maintained by CNCF

⇒ K8s is a container orchestration tool which can manage containers on a cluster of Nodes.

⇒ Other top container orchestrators

⇒ Docker, Swarm

⇒ Apache Mesos

Container Orchestrators:-

⇒ Helps in scheduling the containers

⇒ It provides HA for apps

⇒ Resilient systems

⇒ Scaling solution

⇒ Self Healing

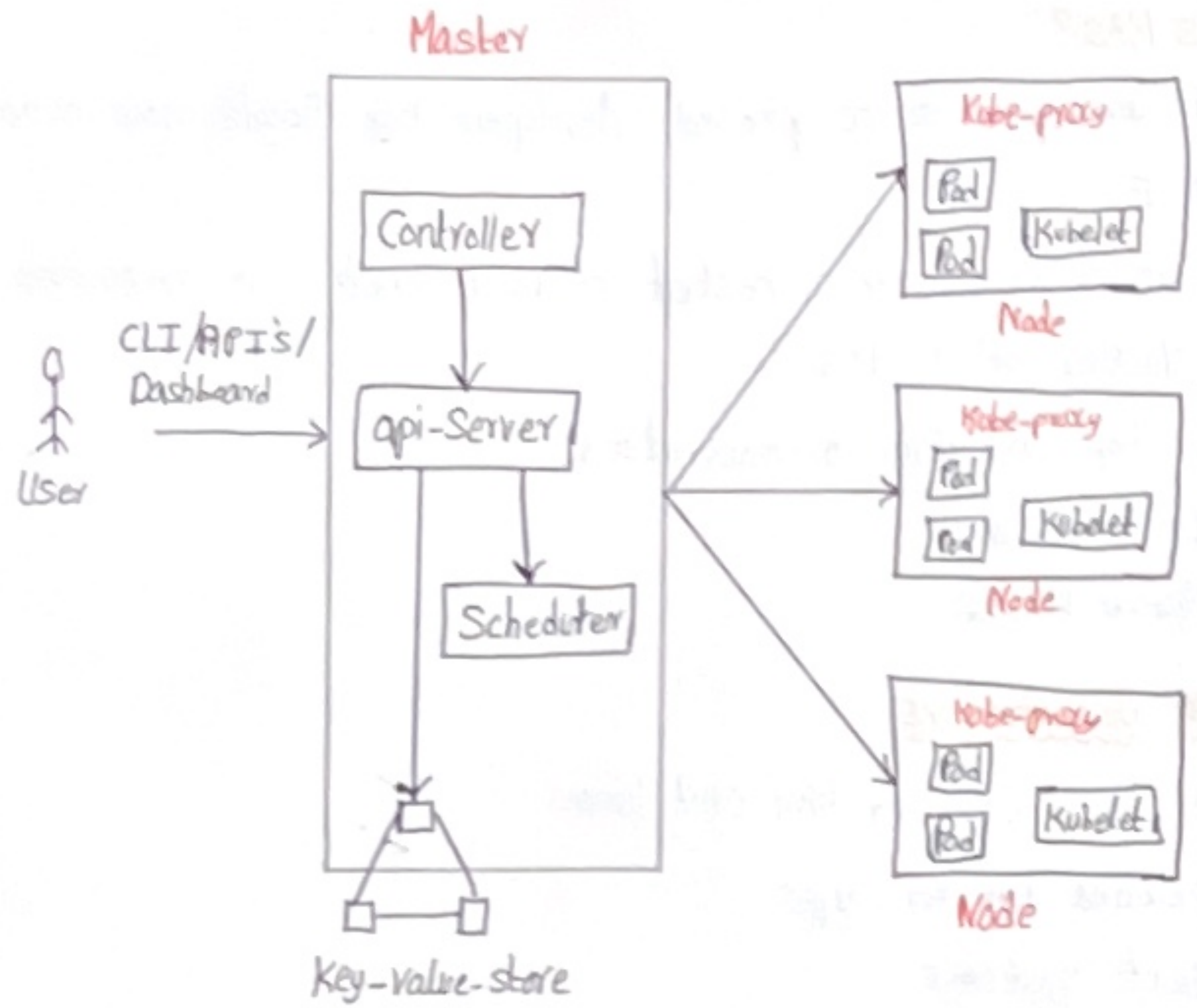
⇒ Automatic Rolling updates and Rollbacks

106) ## K8s Architecture

⇒ K8s cluster can have any number of nodes

⇒ K8s master

⇒ K8s nodes/minions



107) Amazon EKS

=> Amazon EKS is Amazon Kubernetes Service

=> It is a managed service by AWS that helps in running K8S on AWS by managing the Control plane of the cluster

Amazon EKS Control plane Architecture

=> Amazon EKS runs a single tenant K8S control plane for every cluster

=> Amazon EKS provides HA for the cluster nodes.

=> Amazon EKS is highly reliable and secure.

108) K8S cluster setup on AWS vs GCP and AWS EKS pricing

109) K8S cluster setup with Amazon EKS

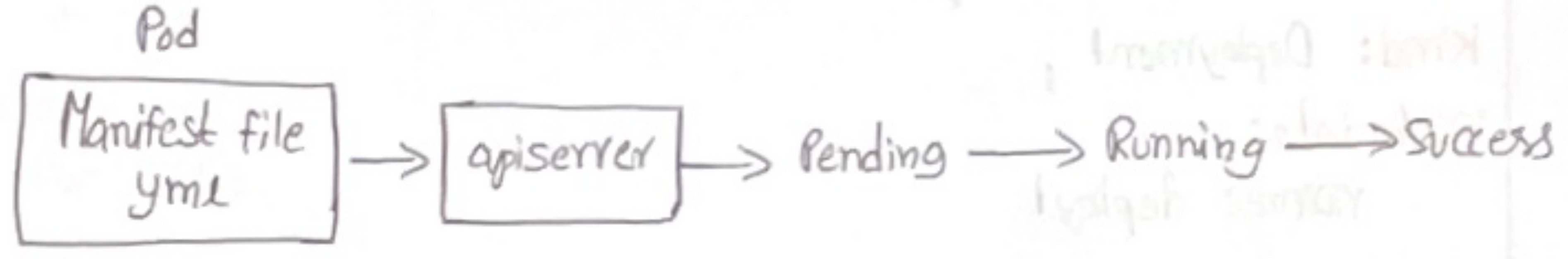
Windows / Linux / Mac

Install Linux instance on AWS

1) pip
2) AWSCLI
3) eksctl
4) kubectl
5) aws-iam-authenticator

110) Pods in K8S

Pods are defined in manifest file in yaml language



Pod Lifecycle

111) Manifest file for Pod

=> Kubectl get nodes

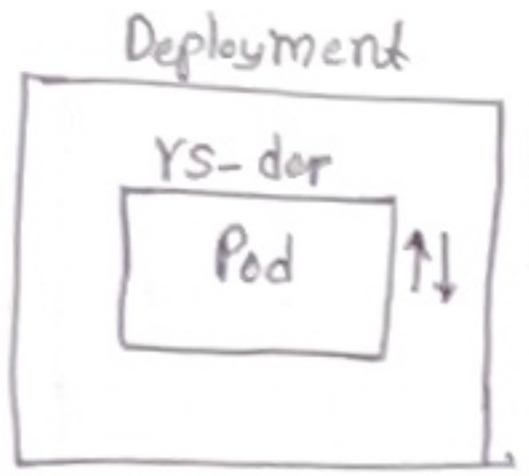
```
# pod-yml
apiVersion: v1
Kind: Pod
Metadata:
   name: Sample-pod
   labels:
      zone: prod
      Version: v1
spec:
   containers:
      - name: my-cont
        image: nginx
```

```
image:
ports:
   - ContainerPort: 80
#kubectl create -f pod.yml
#kubect get pods
#k describe pod podname
```

Kubectl get pods -o wide (To see where pod in the Node)

112) Deployment objects with yaml

Deployment → Rest object API



For each and evey deployment we will get rs

```
#deploy1.yml
apiVersion: extensions/v1beta1
Kind: Deployment
metadata:
    name: deploy1
spec:
   replicas:4                              → selector:
   template:                                   matchLabels:
     metadata:                                    app: app-v1
        labels:
           app: app-v1
     spec:
        containers:
        - name: sample-ctr
          image: devopstrainer/deploy:v1
          ports:
          - containerPort: 80
```

K create -f deploy.yml

If we want update. To apply changes

K apply -f ~~deployml~~ deploy.yml

113) Exposing Application with Service object in K8S

Service [Rest object]

[VIP
DNS
Port]

Create service object and attach to the Pods through labels

114) Nodeport Service in Kubernetes

Kubectl get deploy

```
#service.yml
apiVersion: v1
Kind: Service
metadata:
   name: svc1
   labels:
      app: app-v1
spec:
   ports:
   - port: 80
     nodePort: 32000
     protocol: TCP
   selector:
     app: app-v1
   type: NodePort
```

K create -f service.yml

K get svc svc1

There is one drawback in this service. If node goes down then IP of node will change.

115) Load Balancer Service

```
#svc2.yml
apiVersion: v1
Kind: Service
metadata:
    name: svc1
    labels:
        app: app-v1
spec:
   ports:
   - ports: 80
     targetPort: 80
     protocol: TCP
   selector:
     app: app-v1
   type: LoadBalancer
```

```
# K create -f svc1.yml
# K get svc
```

Now we will get external-IP with Load Balancer and we can acces through Load Balancer url

116) Rolling updates and Rollbacks

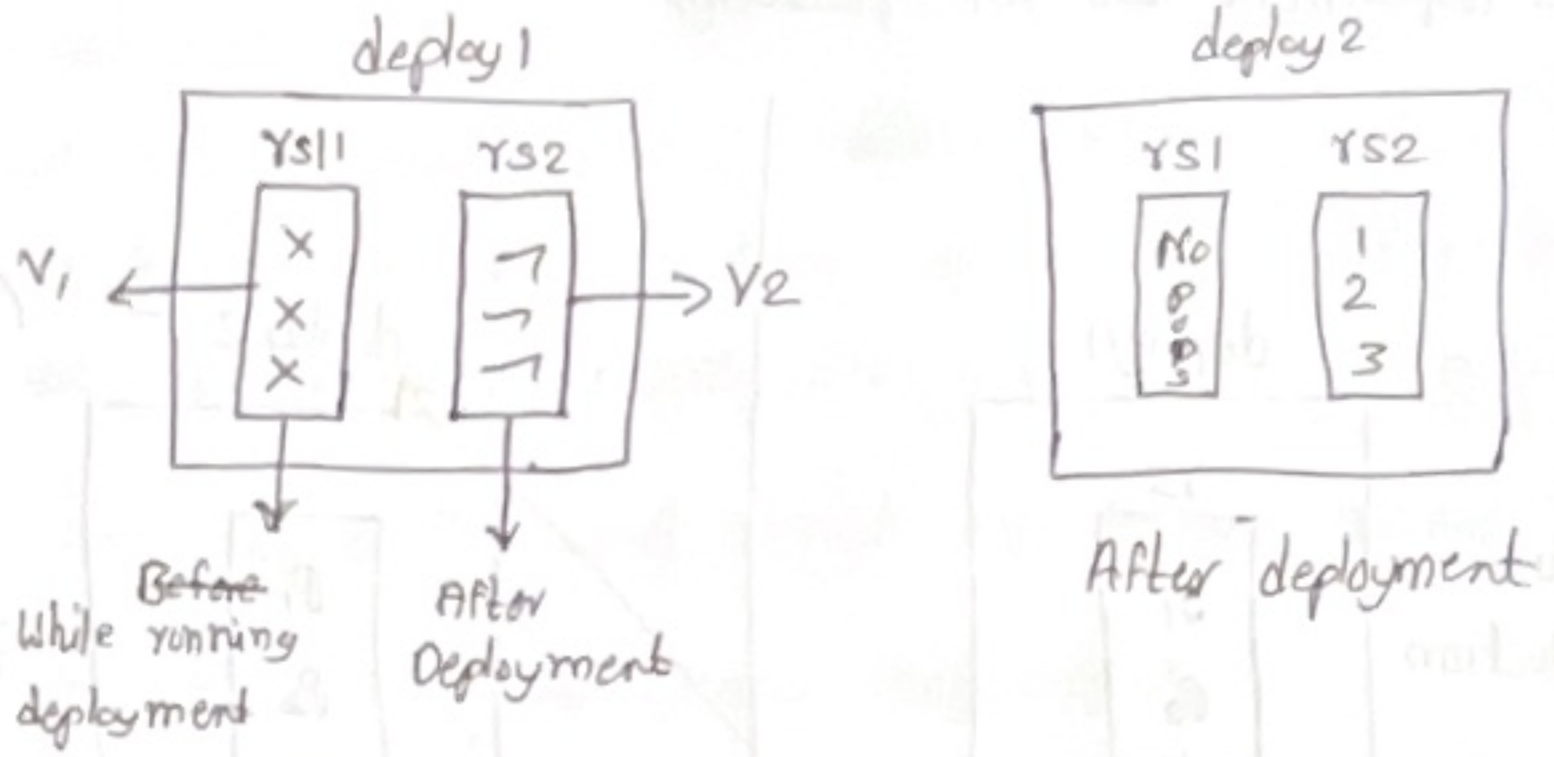
If we want to upgrade my application without any downtime.

Default deployment strategy -> Rolling update

=> Change the image in deploy.yml

=> K apply -f deploy.yml --record -> To save the record

If i apply this command new replicasets will come with new pods.



To see the status of the deployment

```
# Kubectl rollout status deployment deploy1
```

Now

```
# K get rs
```

Now we can see new replicaset with new pods and old replicaset without pods.

To see the rollout history

```
# K rollout history deployment deploy1
```

To revert back to the previous version

```
# K rollout undo deployment deploy1 --to-revision=1
```

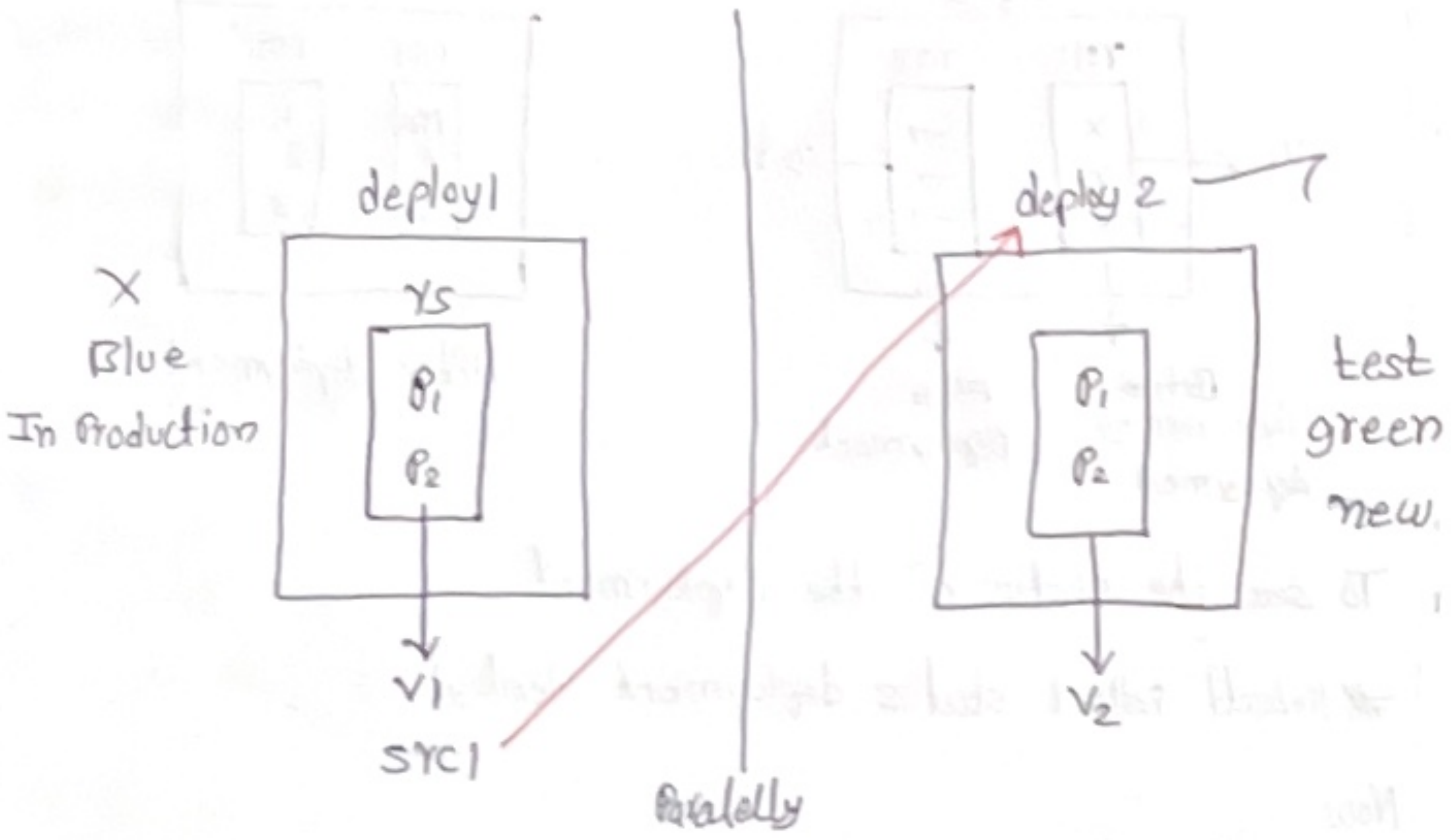
```
# K describe deployment deployname
```

```
# K get rs
```

Now we can see new rs will creates with new pods

117) Blue Green Deployment in Kubernetes

This deployment will run parallelly.



```
# K get deploy
```

Normally deploy one application as usual with deploy.yml. This is Blue deployment.

Now cp deploy2.yml and change image and labels to next version and create deployment parallelly.

```
# K create -f deploy.yml
# K get rs
```

Now we can see

deploy1 2 2 →rs1
deploy2 2 2 →rs2

Pods also created

⇒ Now cp svc.yml svc1.yml

Change name, labels and app to V2. Don't chang any ports

```
# K create -f svc1.yml
# K get svc
```

Now we can see two services. Another LB got created. Hit the LB2 url and check in the browser. Now two parallell deployments are running Now we have double resources. Blue and Green Deployments are there.

IMP Now svc1 should flip and attach to deploy2 as per fig.

⇒ Goto svc.yml and chang app: app-v2

⇒ K apply -f svc1.yml

Now we can see Blue which is in Prod is updated with latest version

```
#K delete svc svc2
```

⇒ Now we can ~~see the~~ delete the unused deployments & services.

118) Autoscaling in Kubernetes cluster with EKS

Adjusting the number of nodes in the cluster and the pods in the deployment as per the need automatically is called Autoscaling in Kubernetes. This can be achieved with different Autoscalers.

Autoscaling Configurations:-

1) Cluster Autoscaler
2) Horizontal Pod Autoscaler

1) Cluster Autoscaler

The K8s Cluster Autoscaler automatically adjusts the number of nodes in your cluster when pod fail to launch due to lack of resources or when nodes in the cluster are underutilized and their pods can be rescheduled on to other nodes in the cluster.

2 nodes ⟶ 20 Pods

This will need as per the resources.

Horizontal Pod Autoscaler:-

The K8S Horizontal Pod Autoscaler automatically scales the number of pods in a deployment, replication controller, or replicaset based on that resources CPU utilization

119) Cluster autoscaler with EKS

```
# eksctl create cluster --name k8sdemo --version 1.14 --nodegroup--name standard-workers --node-type t2.micro --nodes 2 --nodes-min 1 --nodes-max 6 --managed
```

```
# K get nodes
```

Now deploy any application with 2 Pods

```
# K create -f deploy.yml
# K get deploy
# K get pods -o wide
```

Now scale out pods to 8 Pods

```
# K apply -f deploy.yml
```

Pods are in pending state bcz of resources lacking in nodes

*** Now deploy K8S autoscaler**

```
# K apply -f https://raw.githubusercontent.com/kubernetes/autoscaler/master/cluster-autoscaler/cloudprovider/aws/examples/cluster-autoscaler-autodiscover.yml
```

Now cluster autoscaler deployed

Here setting some annotations

```
# K -n kube-system annotate deployment.apps/cluster-autoscaler cluster-autoscaler.kubernetes.io/safe-to-evict="false"
```

Now make some changes by editing deployment

```
# K -n kube-system edit deployment.apps/cluster-autoscaler
```

Change cluster name

/K8sdemo

```
    image: k8s.gcr.io/cluster-autoscaler:v1.14.7
```

Now goto IAM → roles → eksctl-k8sdemo-nodegroup-standard → Permission → JSON and add the JSON script from Udemy.

```
# kubectl delete deploy deploy1
```

Now re-deploy the deployment

```
# K create -f deploy1.yml
# K get pods
```

Now check the logs

```
# K -n kube-system logs -f deployment.apps/cluster-autoscaler
```

⇒ Now we can see new nodes are created in EC2 console.

```
# K get nodes
```

Initially we have 2 nodes but now it is '6'.

⇒ Now scaling in to 1 Pod

```
⇒ K apply -f deploy.yml
```

Now it will scale down the Nodes.

120) Horizontal Pod Autoscaler

Depends on CPU utilization

Deploy metric server

⇒ Delete pods, deployments

```
# wget -O v0.3.6.tar.gz https://codeload.github.com/Kubernetes-
-sigs/metrics-server/tar.gz/v0.3.6

# tar -xzf v0.3.6.tar.gz

# K apply -f metrics-server-0.3.6/deploy/1.8+/

# K get deployment metrics-server -n kube-system
```

Now we can see metric server deployed sucessfully.

```
# K run httpd --image=httpd --requests=cpu=100m
--limits=cpu=200m --expose --port=80
```

Now create Horizontal Pod autosaler for httpd deployment

```
# K autoscale deployment httpd --cpu-percent=50 --min=1
                                               --max=10
                                                  ↓
                                                 Pods

# K get deploy
```

We can see horizontal Pod autosaler deployment

```
# K get hpa/httpd
```

Lets create load on this deployment

```
# K run apache-bench -i --tty --rm --image=httpd -- ab
-n 500000 -c 1000 http://httpd-default.svc.cluster.local/

# K get hpa/httpd -w
```

Now we can see pods are increasing as per CPU Load. Also we can see Node variation in Nodes.