BASIC OF K8s PART II

1. Services (BASIC)
2. Cluster IP
3. Node Port
4. Load Balancer
5. External Name
6. Labels in Kubernetes (Label-Selector)
7. Lifecycle of pods
8. STATES of Pod
9. Lifecycle Hooks - (post start, pre-stop)
   handlers (Exec, HTTP)

Section 39. Basic of Kubernetes part II

[250]. Services in Kubernetes

Services

(1) Service is used to Access the Application Running on pods.

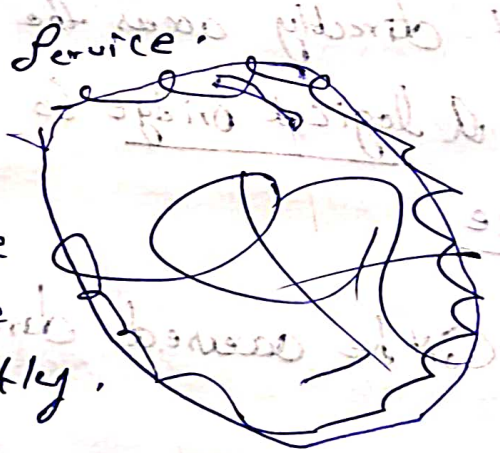
We have already seen that we was executing the application on the Containers.

Container was managed by pods & pods was managed by the Replica Set & Replica Set was manged by deployment.

But whenever we need to acces that particular application on the pod, we need to use the Create Service.

We have see how we can expose the node type port, how we can expose the load balancer.

So we were creating the Service.



Now the Question Comes in when the application is running on the pods, why Cannot we access the application on the pod directly.

So there are multiple reasons that we can't access the application on the pods directly so that we need a logical bridge blw the pods and the user,

So let's discuss about the reasons.

(1) pods are Dynamic in Kubernetes, pods Created and terminated On Demand.

pods were terminated as per the demand, as per the Configuration or as per the load.

(2) using Replication Controller, pods are Created & Terminated during the Scaling.

(3) using Deployments, pods are terminated & new pods are take place during the Image Version upgrade.

So there are the some reasons why it can't be accessed the application directly On the pods & we need to use the Service.

So due to their nature of pods inside the Community Cluster, we can't directly access the application On the pods. we need a logical bridge to access the pods, which is called Service

(4) Pod's Can't be accessed directly, but thru a Service.

(6) How do we Create Services: in last chapters we have Created pods with Expose Command.

(1) Kubectl Expose: Command Created a Service for pods so that they can be accessible Externally.

(2) By using Expose Command we are Creating a Service will Create the end-point for the pods.

(3) we are Creating a logical gate. or we can say we are Creating a logical bridge by which the external world or the user can access the pods.

Now the Question is when we are Creating the Service, which kind of pods should be accessible on that particular Service?

(4) the set of pods targeted by a Service is usually determined by a selector in manifest file.

So inside the Service manifest file, we will define a tag selector & inside that particular selector we will define the name of the ports which can be accessible through that particular Service.

Sample manifest:-

```
apiVersion: v1
Kind: Service
metadata:
  name: my-Service
Spec:
  Selector:
    app: myApp
  Ports:
    - protocol: TCP
      Port: 80
      targetPort: 9376.
```

Now let's discuss about the publishing, the Service, what kind of Service type we have during the publish of the Service.

① ClusterIP: Exposes the Service on a cluster-internal IP. Choosing this value makes the Service only reachable from within the cluster.

it means you can access your Service within the cluster only if you are choosing the Service type ClusterIP then you can access your Service within the cluster only.

② NodePort: Exposes the Service on each Node's IP at a static port. you'll be able to contact the Nodeport Service, from outside the cluster, by requesting <NodeIP>:<Nodeport>

③ Load Balancer: Expose the Service externally using a cloud provider's load balancer. Nodeport and ClusterIP Services, to which the external load balancer routes, are automatically created.

(it means whatever cloud provider you are using, it will spin up a load balancer in that particular cloud provider and that cloud provider access your Service on a cluster IP & the node port

So whenever you will access your load balancer that will route the traffic to the node port & Cluster IP.)

④ External Name:- Maps the Service to the Contents of the External name field, by returning a CName record.

we have not seen that particular thing directly, but indirectly we have seen that particular thing maps the Service to Contents of the External name field. By returning the CE name record.
It meand when we use the External name that will Create a Canonical name which is called C name

[251]: Lab: Services in Kubernetes:-

① Go to the directory where we store the Scripts.

Commands

i) Get Service URL
minikube service <Service-name> --url

for Aws, Gcp. no need to Execute these Command.

ii) To Describe Service
Kubectl svc <Service-name>

③ List All Services.
Kubectl get svc

2)
```
apiversion : V1
Kind: Pod.
metadata :
  name : nginx&webproxy
  labels :
    app : nginxpod
Spec :
  Containers :
  - name : nginx
    image : nginx:1.16.0
    ports :
      -name: nginxport
       Containerport : 80
```

Save & exit

③ Kubectl · ~~apply~~ create -f podmanifest.yml.

④ Kubect get pods

⑤ Now we Create Services for this a Pod manifest file.

```
apiversion : v1
Kind: Service
metadata :
    name : nginxservice
Spec :
    Selector :
      app: nginxpod
```

ports:
- protocol: TCP
  port: 80
  nodePort: 31010
  targetport: nginxport
protocol: TCP

Save & Exit.

what do this Service will do, it will create a Service and Expose on port 31010.

⑥ Kubectl create -f <podService.yml>

⑦ Kubect get svc.

⑧ To access the Service minikube Service nginx-Service --url.
will get the url.

⑨ Curl <http link> Get from Step 8.

⑩ To delete the Service.
Kubectl delete Service nginx-Service.

⑪ To describe the Service
Kubectl describe Service nginx Service

[25] Labels in Kubernetes :-

① Labels are Key/value pairs that are attached to objects.

② Labels are intended to be used to specify identifying attributes of objects that are meaningful & relevant to users.

```
apiVersion: v1
Kind: Pod
metadata:
    name: nginxwebProxy
    labels:
      app: nginxpod
```

app: nginxpod.
↳ value, we can define any name here.
Key, we can define any name here, like app1, pod1 — etc.

③ Labels are like Tags in cloud providers like AWS, GCP.

④ Labels can be attached to objects at Creation time & subsequently added & modified at any time.

for Any Kind of object you are Creating inside the K8s, you are Creating the pods. you are Creating the deployment, or you are Creating the ReplicaSet: or any Kind of objects you are Creating.

You can add the label at the time of initialization & you can edit the labels any time whenever it is required for labels.

(5) For Labels you can follow Key-Value Pair structure like:

Key: environment - Value: dev/da/UAT/prod.
Key: department - Value: Engineering/Cloudops/QA.

There is no hardcode rule that you need to define the specific keys & specific values.
, you can define anything inside the key which is relevant for your work & you can define anything inside the value which is also relevant for your work.

(6) we can attach the labels with any kind of object present inside the K8S to identify that particular object & work with that particular Object.

(7) If you have Configuration file for a pod that has two labels Environment: production & app: nginx.

```
apiVersion: v1
kind: Pod
metadata:
  name: label-demo
  labels:
    Environment: Production
    app: nginx.
```

If two labels we can one or more labels.

(8) Labels are not Unique & multiple Labels Can be added to One object.

(9) Once Labels are attached to object, we Can filter the results On Labels.

(10) Above approach is Called Label-Selector.

(11) Using Label Selectors User Can use matching Expressions to match Labels.

(12) Simple Matching: (4 four way)

Environment = production

tier != backend

Environment in (production, QA, UAT)

tier notin (frontend, backend)

(13) User Can use Labels to tag Nodes.

(14) Once Nodes are tagged, User Can use label Selectors to run Pods Only One matching Nodes.

Suppose you have defined Some nodes inside the Kubernetes Cluster & at Runtime you want to Specify that Particular application or Particular pod should actually Execute On a particular node.

Then you can do that Particular thing with the help of the label & we Can use the label Selector to fulfill this Particular Requirement for this.

(15) If you want to Tag a Node, use these Commands.

Kubectl get nodes

Kubectl label nodes <your-node-name> disktype=ssd

Kubectl get nodes --show-labels.

(16) Run pods on Specific Nodes by nodeSelector:

```
apiVersion: v1
Kind: Pod
metadata:
  name: nginx
  labels:
    env: test
Spec:
  Containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd.
```

} nodeSelector.

[253] Lab: Labels in Kubernetes

Commands.

1) Get all Nodes in Kubernetes Cluster

Kubectl get nodes.

2) Label Nodes with Specific Label

Kubectl label nodes <your-node-name>
disktype=ssd.

3] Verify your node has the Label.

kubectl get nodes --show-labels

4] Create pods & Verify the Status

① Go to directory where Script in Store

② vi Label-pod.yml.

```
apiVersion : V1
Kind: Pod
metadata :
  - name : nginx-web
    labels :
      app: nginx-pod
Spec :
  Containers :
    - name: nginx
      image : nginx 1.16.0
      ports :
        - name: nginxPort
          ContainerPort: 80
  Node Selector:
    diskType: SSD
```

③ Kubectl get nodes

④ Kubectl get nodes --show-label.

⑤ Kubectl Create -f Label-pod.yml.

(6) Kubectl get pods:

we can see that Status in pending, because diskType =ssd Label on node is not Configured.

(7) Kubectl describe pod nginx-web.

(8) Configure Label on nodes:

Kubectl get nodes.

Kubectl label nodes minikube diskType=ssd

(9) Kubectl get nodes --show-labels

(10) Kubect get pods:

as soon has we added Label, pod move from pending to Running Status.

[254] :- Lifecycle of pods in Kubernetes.

① A pod is the smallest unit of work which can be scheduled in Kubernetes.

pod is lower level. Compare to deployment. which is used to schedule any kind of task on the K8s Cluster.

② although the pod is basic block. A pod encapsulates an application Container(s), storage, resources, unique n/w IP & options that govern how a Container should run.

③ We have already discussed like a pod & the applications which are generally deployed with the higher level of abstraction level, which could be the deployment or the replica set.

We will not directly deploy pods on the K8s cluster.

④ Interaction with pods is generally used to troubleshoot issues, hence understanding of pods is important.

⑤ if Executing deployment or Replica Set/Controller to deploy. In base pod is running.

STATES OF A POD :-

① **Pending** :- The pod is accepted by the Kubernetes System but its Containers is/are not Created Yet.

it means there could be the possibility that your image is being downloaded.
Although the Kubernetes Cluster has accepted the pod
So that the download image process will be initiated.

or label's are not matching -- etc.

② **Running** :- The pod is Scheduled on a node & all its Containers are Created & at-least One Container is in Running State.

it means if we are executing ten Containers in a pod then we will get the running state. if any one Container will start.

Right now we are not guarantee about the start of the application.
we are just talking the guarantee of the start of the Container.

③ **Succeeded** :- All Container's in the pod have Exited with Status 0 & will not be restarted.

it mean's they all are being Created Successfully.

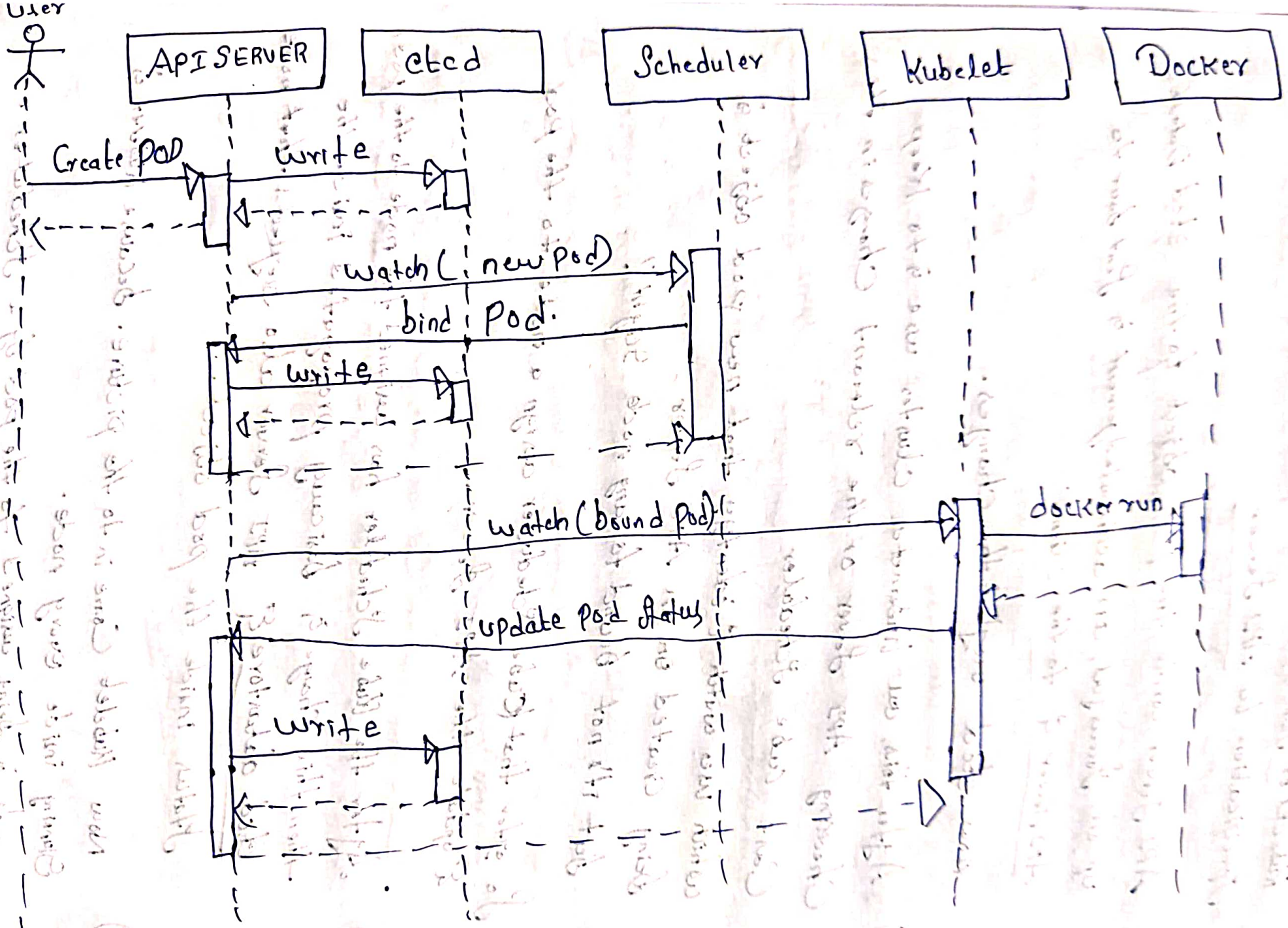
④ Failed :- All Container (s) of the pod have exited & at least one Container has returned a non-zero status.

i] we are executing multiple Containers inside the pod & at least one Container will fail, then the Complete pod state will be marked as a failed. State :

⑤ Crash loop back :- The Container fails to start and is tried again & again.

Container inside the pod, that is not a starting, but it trying again & again, so there is a loop. (Infinite loop).

Let's start the pod life cycle with this diagram :-


User
API SERVER
etcd
Scheduler
Kubelet
Docker
Create POD
write
watch (new Pod)
bind Pod.
write
watch (bound Pod)
docker run
update Pod status
write

(1) First Command is either from Kubectl or some other API client submit for pod Creation Request.

(2) Kubectl or some other API client submit the pod specification to API Server

(2) API Server write the pod object to the etcd Datastore up on successful, an acknowledgement is sent back to API Server & to the client.

now etcd registers the changes.

(3) After this, our Kubernetes cluster use it to keep checking API Server or the relevant changes in this case cube Scheduler

which uses watch & identify that new pod object is being created on an API Server.

but it's not bound to any node right?

So after that Kube Scheduler assign a node to the pod & update the API Server.

(4) After the Kube Scheduler has assigned the node to the pod, then this change is basically propagated inside the etcd database & API Server also registers that node status inside the pod object.

(5) Now Kubelet Come in to the picture. because Kubelet is running inside every node.

node is being assigned to the pod by a Scheduler.

So Kublet will use their watch toward the API Server & it will identify that a new pod is assigned to it.

(8) Once it will identify, that node is assigned to it. Kubelet starts the pod on it node by calling Docker & update the Content listed by back to the API Server.

(9) The API Server persists the pod into the ETCD state, So every Component which is running inside this diagram.

(10) After Completion of the work they are sending the request to the API Server & it is basically writing the desired data

(11) and at last Once it sends the acknowledgment of a success full write.

(12) The API Server sent back an acknowledgement to the Kubelet indicating that event is accepted.

So this is the Complete lifecycle of the pod inside the K8s cluster.

Now let's discuss a few Components of this particular Life cycle.

(1) first activity happens is Init Container :-
Init Containers are Containers which are run before the main application Container gets started.

2] Before creating a main Container. Init Container will start.

③ Init Container have two important characteristics.

① they always run to Completion.

It Can be, Run, terminated or failed it will do its process.

Init Container Running any Job or Script before deploying Pod.

② Each init Container must Complete before the next one is started.

if we have multiple init Containers. it will create Securely. (not parallel) i.e one by one.

④ Init Containers Support all the fields & features of app Containers, including Resource limits, Volumes & Security Settings.

⑤ Init Containers Can be useful when Some initial actions need to be run before the main Container in the pod starts.

Lifecycle Hooks (Important life cycle)

hooks or break Point in the Lifecycle

(1) Lifecycle Hooks allows the user to run Specific Code during Specific events of a Containers lifecycle.

So if we are defining the lifecycle hook inside the manifest file or the pod, then basically we want to Execute Some Special Command going to execute. Some Specific thing before a Specific event inside the lifecycle of my pod or Container. we have two kind of hooks inside the pod lifecycle.

(a) PostStart :- this hooks gets executed upon Container Creation but there is no guarantee that it will run after the Container ENTRYPOINT.

(b) pre_Stop :- This hook gets Executed just before a Container is terminated. This is a blocking Call which means the hook execution must Complete before the Call to delete a Container can be sent. it means before the deletion / termination call of the Container can be sent this hook or the Commands which is mentioned inside this Particular hook. They must be Executed now to Execute these Particular hooks.

We have two handlers :-

There are two types of handlers which can be implemented in the hook implementation:

① Exec :- Runs a specific command inside the container & the resources consumed by the command are counted against the container.

② HTTP :- Executes an HTTP request against a specific endpoint on the container.

Manifest :-

```
apiVersion: v1
kind: Pod
metadata:
  name: lifecycle-demo
spec:
  containers:
  - name: lifecycle-demo-container
    image: nginx
    lifecycle:
      postStart:
        exec:
          command: ["/bin/sh", "-c", "echo Hello"]
      preStop:
        exec:
          command: ["bin/sh", "-c", "nginx -s quit"]
```