**Kubernetes**

**Master: -** has all the control services.

**Api server**: - The Kubernetes API server validates and configures data for the api objects which include pods, services and others.

**Etcd**: - it is a db used to store all the info related to kubernetes like current status, access info and any other config info.

**Scheduler:** - is responsible to decide which payload is run to in which machine, it decides to which node need to deploy the container.

**Controlmanager:** - it controls the state of the cluster.

**Kubeproxy:** -

* It is responsible for networking for node.
* Node having 1 Ip address, and pod has its own Ip address. By using Kubeproxy we can manage all networking components inside the node.

**Kubelet:** acts like an agent for master to communicate to the node, like a client-server communication.

**Pods**: A pod is the smallest execution unit in Kubernetes. A pod contains one or more applications(containers). Any containers in the same pod will share the same storage volumes, network resources, and communicate using localhost.

**kubectl**: It allows you to run commands against kubernetes cluster. You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs.

**Ex**: -

kubectl run nginx: - used to deploy an application on the cluster.

kubectl cluster info: - used to view information about the cluster.

kubectl get nodes: - To list all nodes.

kubectl create -f ngnix.yml: - To create a pod

**kubectl apply -f pod definition.yml**: - if manifest file is changed/updated after deployment and need to re-deploy the pod again.

**kubectl delete pod nginx-pod**: - To delete a pod using name.

kubectl create namespace dev

**kubectl get ns**: - to see list of name spaces

**kubectl get po -n dev**: - to see the resources in dev namespace.

**imperative cmd - kubectl run nginx-pod --image=nginx:alpine** :- it will create in default location.

**kubectl run nginx-pod --image=nginx:alpine -n dev** :- it will create in dev namespace.

**kubectl get pods --all-namespaces** :- to view all deployments

**kubectl get all --all-namespaces**:- it displays all the resources and namespaces.

**kubectl get po nginx-pod -o yaml > nginx.yml** :- to create a yaml file for an image

**kubectl edit po nginx-pod -o yaml** :- we can edit the yaml file and add extra data

**kubectl edit deployment nginx -o yaml** :- edit the resource in deployment file

**kubectl describe po nginx-pod** :- it will show data inside particular pod

**kubectl logs -f pod\_name** :- to see the logs of the pod .

**kubectl get svc --all-namespaces** :- to get all services.

**kubectl get cm** :- to get all config maps

**kubectl get secrets** :- To get all secrets.

**kubectl get secrets regcred -o yaml** :- To see the details of data in that yaml file

**kubectl get svc** :- to get all service details.

**kubectl get svc nginx-1 -o yaml** :- Get all the details.

**kubectl get po memory-demo -o wide:-** To see in detail about pod**.**

**switch master node to worker node: -**

* kubectl get nodes -o wide
* copy the external Ip address
* ssh external Ip

Note:-

if image pull backoff occurs we have to see in describe and see the message as "can't pull image from docker".

**1.ngnix.yml file**

apiVersion: v1

kind: Pod

metadata:

name: memory-demo

spec:

containers:

- name: memory-demo-ctr

image: ngnix

2. **nginxsvc.yml** - ngnix service file

apiVersion: v1

kind: service

metadata:

name: my-service

spec:

type: NodePort

selector:

app: web

ports:

-port: 80 (port on pod)

targetPort: 80 (port on service)

nodePort: 30007

* pod port will connect to service port (target port) and service port will connect to external port (node port).
* If target port and port address will be same their will chance of hacking, to keep more security we used to give different Ip’s outside visible Ip is target port inside communication Ip is 'port'.
* node port range will be b/w 30000 to 32500.

we can create 1 pod by using “kubectl create -f nginxsvc.yml”,after successful creation lauch url with ip:nodeport Ip.

ex: - 192.168.171.167:30007.

**Difference b/w pod and service**

**Pod**- To deploy app we use pod.

**service** - To access the app.

**Daemon set concept:-**

A Daemon Set ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected.

Suppose we have 7 replica sets and each node should deploy 1pod then we can use daemon set.

Some typical uses of a Daemon Set are:

* running a cluster storage daemon on every node
* running a logs collection daemon on every node
* we need antivirus or metrics in each node server.

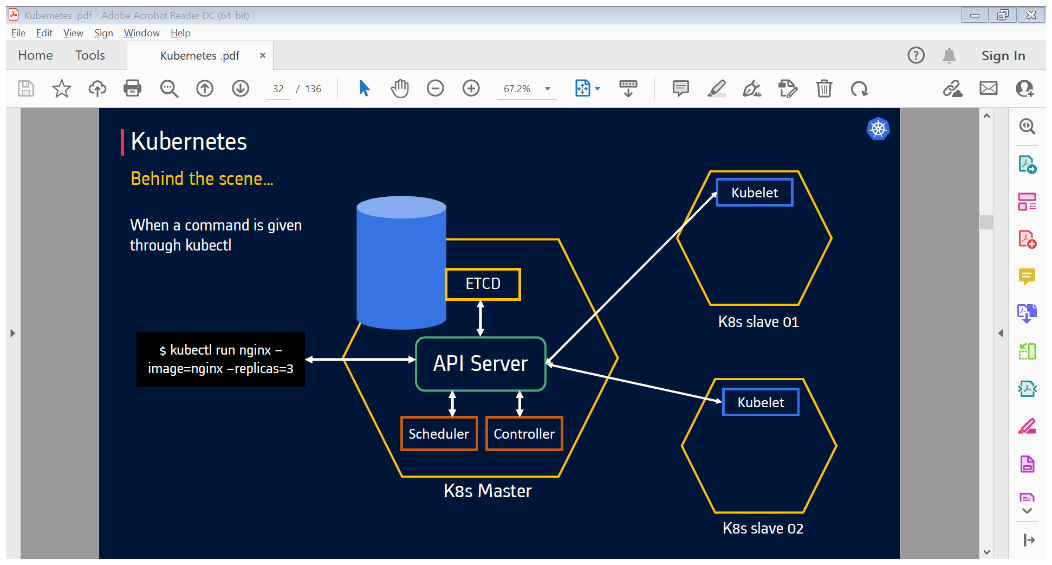
Text

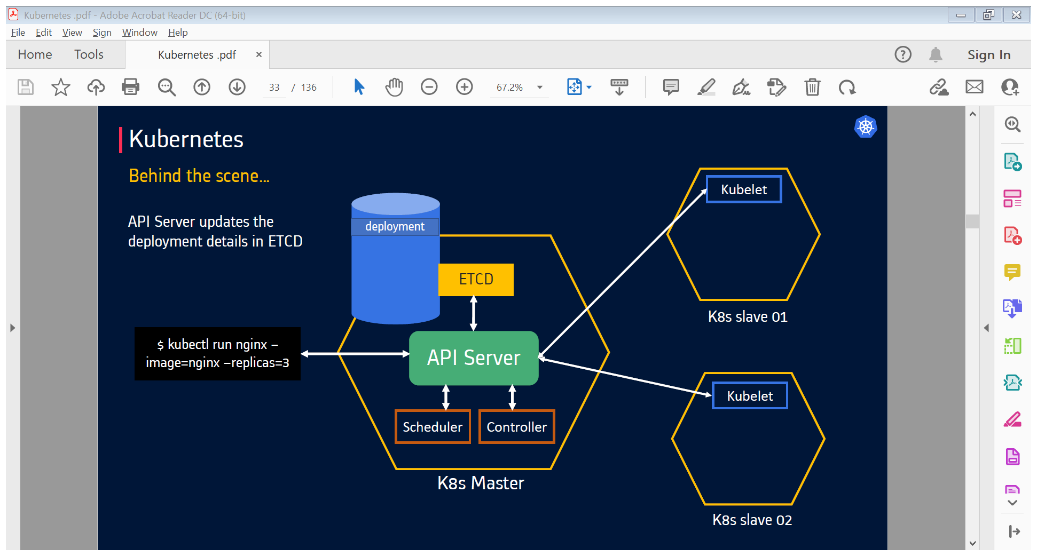
Description automatically generated

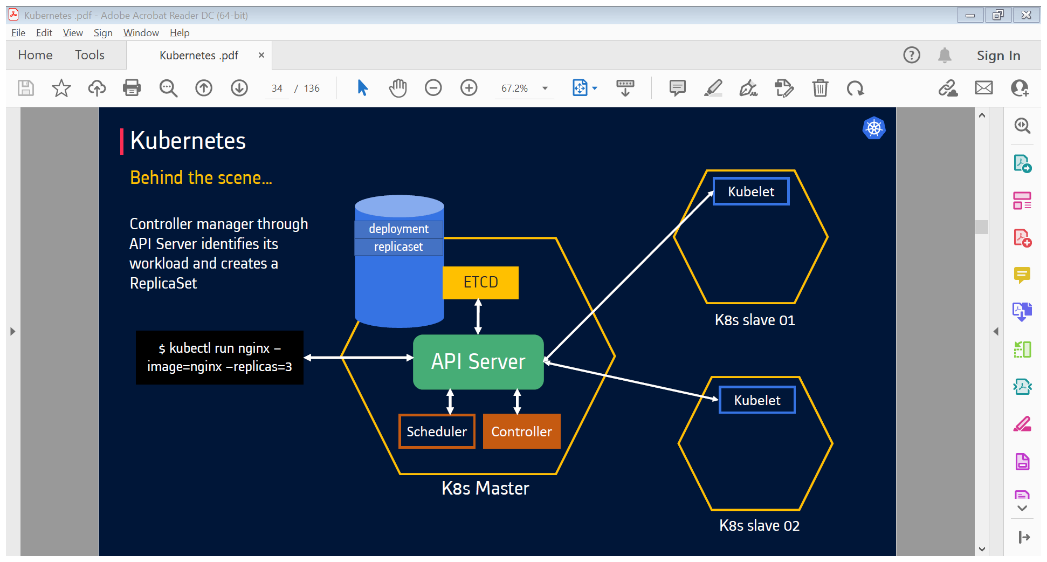
**Replica set** :-

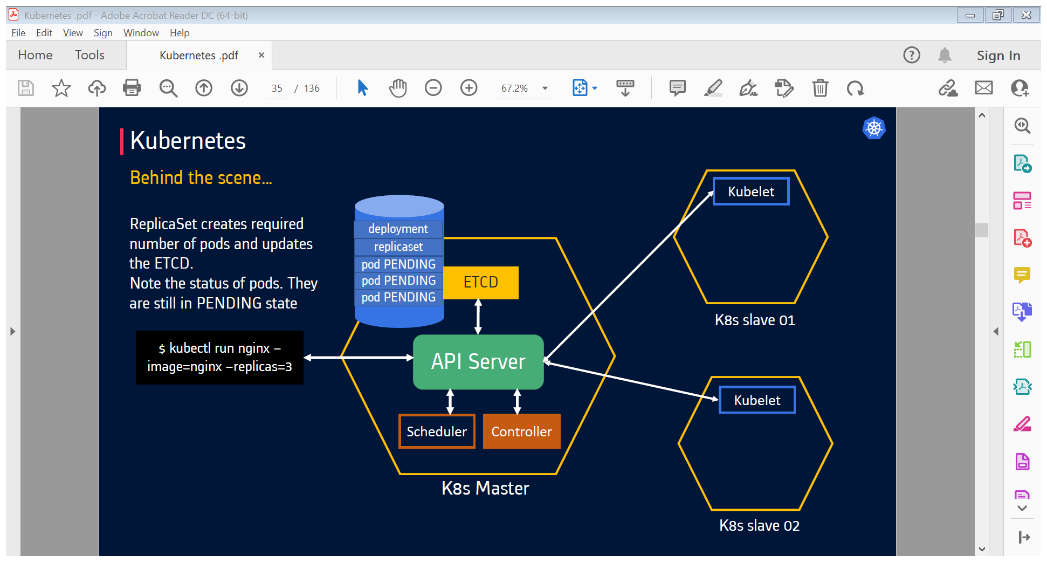
A single pod can’t maintain heavy traffic ,if pod goes down then everything will lost, in order to avoid these, we have to maintain one(or)more instance running at same time using replicasets we can maintain multiple pods.

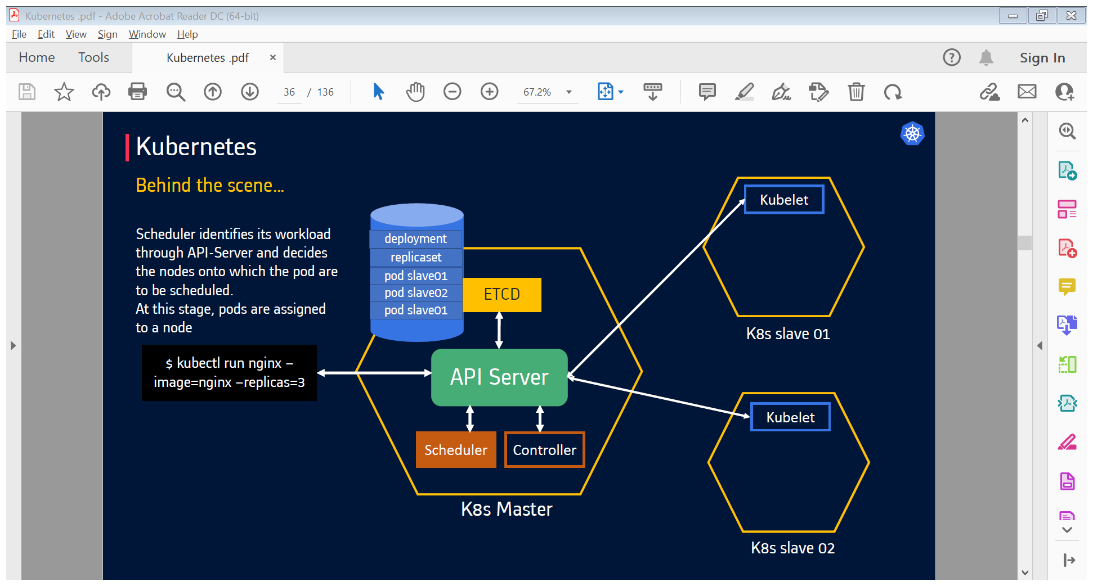
**Working of replicasets**

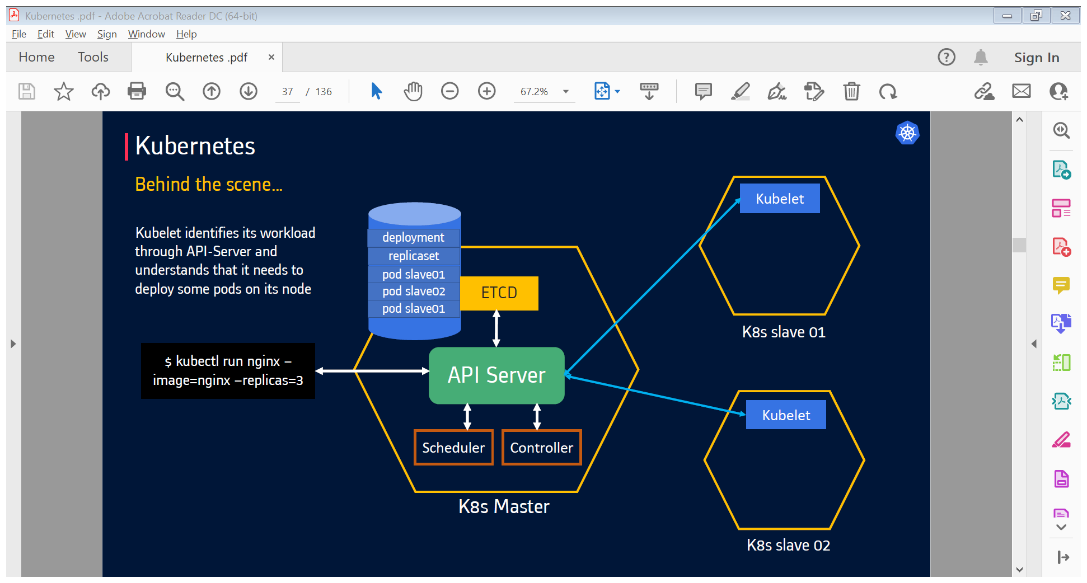
****

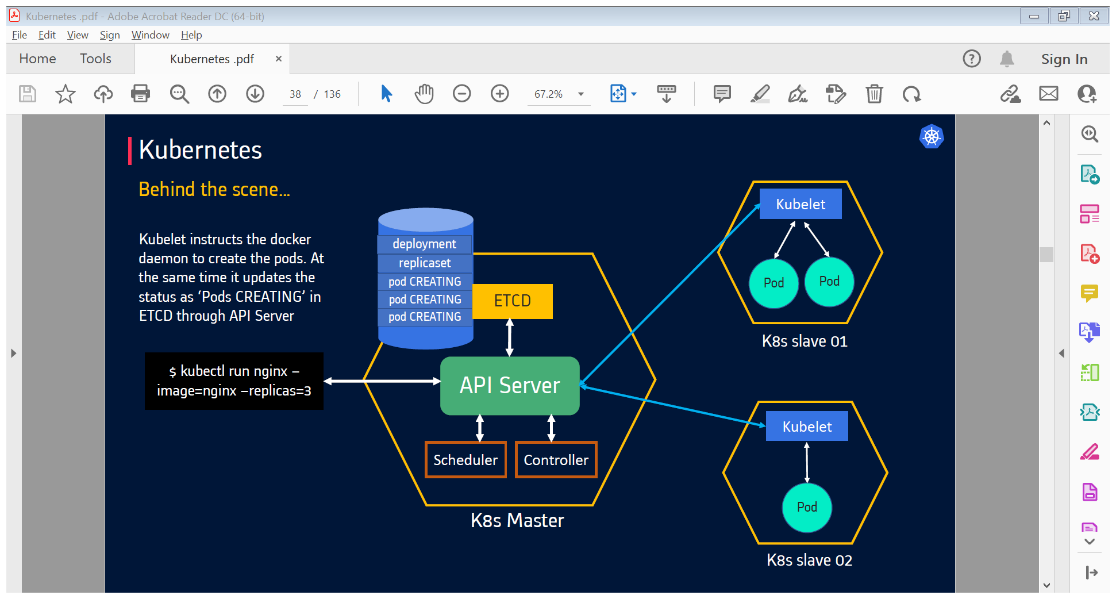
****

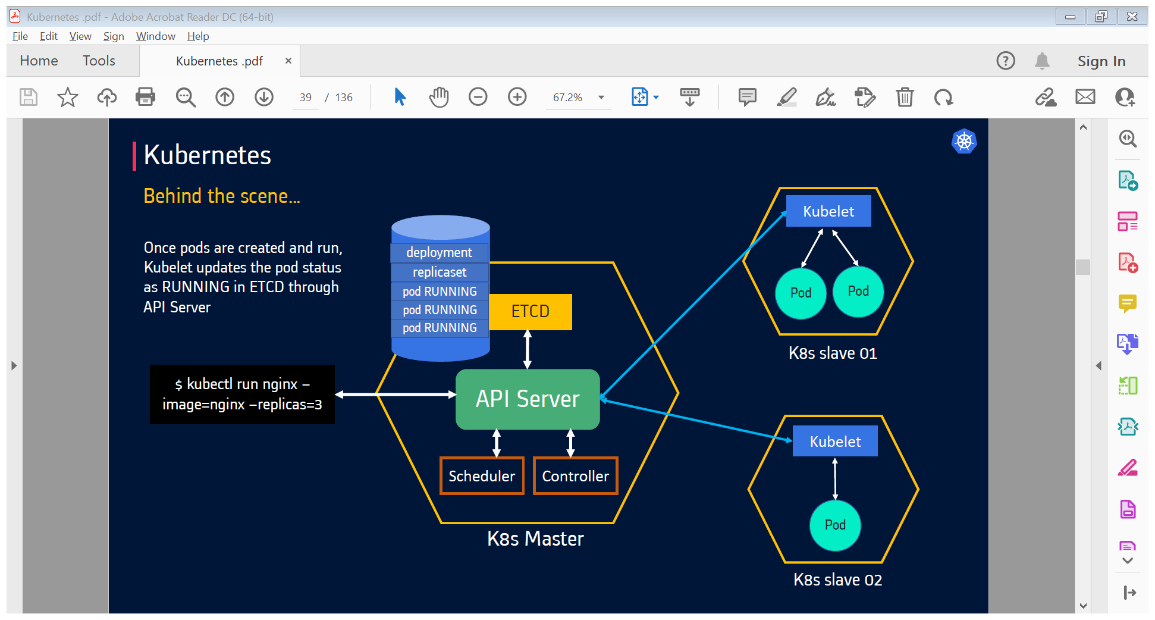
****

****

****

****

****

****

**label**: - name is for identification (if we declare label to communicate with service file we need to provide same name in selectors of service.yaml file).

**Node** **Selector** **concept**:

we have 7 nodes, suppose we got requirement as all pod deployments should happen in 6th node then we have to use 'node selector' concept.

**kubectl get nodes --show-labels :-** To see list of default labels

* we can add labels like

kubectl label node knode1 iam=hero

Text

Description automatically generated

**kubectl get nodes --show-labels | grep iam=hero :-** grep command is to highlight the particular string.

-> so, after mentioning label all the deployed pod files will be seen in node1 only, see with below cmd.

#Kubectl get po -o wide

-> To remove label to node1 we use below one

#**Kubectl label node knode1 iam** - it will unlabel

**Taints and Tolerations concept :-**

It is a mechanism that ensures that pods are not placed in inappropriate nodes.

Suppose we have 3 nodes, node1 have less memory space and pods are getting deployed in node1 then its performance will decrease. To avoid these situations, we will use T&T concept, in these we will use some conditions if it satisfies then we will deploy into the node. Or it won’t get deploy.

Here taints are used in nodes and tolerant are used in deployment file.

Ex: -

You add a taint to a node using kubectl taint.

kubectl taint nodes nodename key=value: effect

kubectl taint nodes knode2 app=Exists: Noschedule

Text

Description automatically generated

->To remove taint to nodes we use below one

#Kubectl taint nodes knode2 app-

We can use multiple taints on same node and multiple tolerations on same pod.

kubectl taint nodes node1 key1=value1:NoSchedule

kubectl taint nodes node1 key1=value1:NoExecute

kubectl taint nodes node1 key2=value2:NoSchedule

<https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/>

**NoSchedule**: Pods will not be scheduled on the node unless they are tolerant. Pods won’t be scheduled, but if it is already running, it won’t kill it. No more new pods are scheduled on this node if it doesn’t match all the taints of this node.

**PreferNoSchedule**: Scheduler will prefer not to schedule a pod on taint node but no guarantee. Means Scheduler will try not to place a Pod that does not tolerate the taint on the node, but it is not required.

**NoExecute**: As soon as, NoExecute taint is applied to a node all the existing pods will be evicted without matching the toleration from the node.

**ConfigMaps: -**

* To declare env variables we use ConfigMaps

kubectl get configMaps coc-configmap -o yaml

**Secrets**:

* To store secrets data, we use secrets by using encryption.
* To encrypt username and password we use secrets.

**Syntax**: -

kubectl get secret

kubectl get secrets git-user-credentials -o yaml

**Readiness and liveness probe: -**

When we launch a new pod, cluster will send request to pod application, whether it is up or not, if pod is not launched application may fails, to avoid these we use this concept. Both will be used to test the health checks.

**Readiness: -**

It will check in initial stage whether pod got up or not. If the pod fails, it will check every particular seconds. if any request comes to this pod, it won't enter bcz application is not in ready state.

Ex: -

readinessProbe:

tcpSocket:

port: 80

initialDealySeconds: 10

periodSeconds: 15

->check for the status until it will come to ready state by using the below command.

Ex: - kubectl get po -o wide

* suppose we declare port as '81'

Ex: -

readinessProbe:

tcpSocket:

port: 81

initialDelaySeconds: 10

periodSeconds: 15

**InitialDelayseconds** - first check after 10 sec

**periodSeconds:** if it fails then it will check after every period seconds.

**->**Application won't start even though we create pod it will be in fail state, if any request comes to this pod application. it won't enter into this becoz application is not ready to enter as mismatch of port while navigating to certain application it sends the rqst which is in ready state so finds for the application up and running.

**Realtime senario:**

if you launch URL then it will take time to start the webserver bcz of heavy traffic then it tries to restart the pod in this scenario we use readiness prob.

**Liveness Prob: -**

After getting pod up, live-ness will check in the middle whether pod gets failed or not but still it live-ness will do health checks for particular time interval. if any pod goes down or dead or slow live-ness it will automatically create 1 new pod.

**livenessProbe:**

tcpSocket:

port: 8081

initialDealySeconds: 10

periodSeconds: 15

**InitialDelayseconds** - first check will occur 10 sec later.

**periodSeconds** - still it didn’t find in initial delay it comes to period sec every 15 sec it checks until it's up.

Text

Description automatically generated

**Horizontal-pod-autoscale:**

To reduce down time while launching applications we use autoscale method.

In these we are using up to 10 replicas one after other, if we mention targetCPUutilisation method as '50%' if its utilised 50% then it invokes to new pod and another replica also utilises 50% then new replica, this process will do up to max replicas. if its below 50% then one by one it will be deleted.

End used should not find down time - In festival seasons so many io operations r done to reduce the risk we use this

Text

Description automatically generated with medium confidence

-> This process will works/target based on kind & name.

-> Latest version of kubernetes is 1.21

Diff b/w node port and clusterip?

**ClusterIp** :- it is the default kubernetes service .it will create an ipaddress within the cluster service, it can’t be accessible over the internet.

apiVersion**:** v1

kind**:** Service

spec:

selector:

app**:** my-app

type**:** ClusterIP

ports:

- name**:** http

port**:** 80

targetPort**:** 80

protocol**:** TCP

-> In the above example ,we can see that traffic to port 80 on the clusterip will be forwarded to targetport.

->You can see the IP address that’s been assigned by running kubectl get services. Other workloads in your cluster can use this IP address to interact with your service.

**Node Port: -**

It will expose our service on fixed port number. we can access the outside of the cluster by using the ipaddress:nodeportnumber.

-> Nodeport number can range from 30000-32767.

spec:

ports:

- name**:** http

port**:** 80

targetPort**:** 80

nodePort**:** 32000

protocol**:** TCP

This will route traffic on port 32000 to port 80 in your pods.