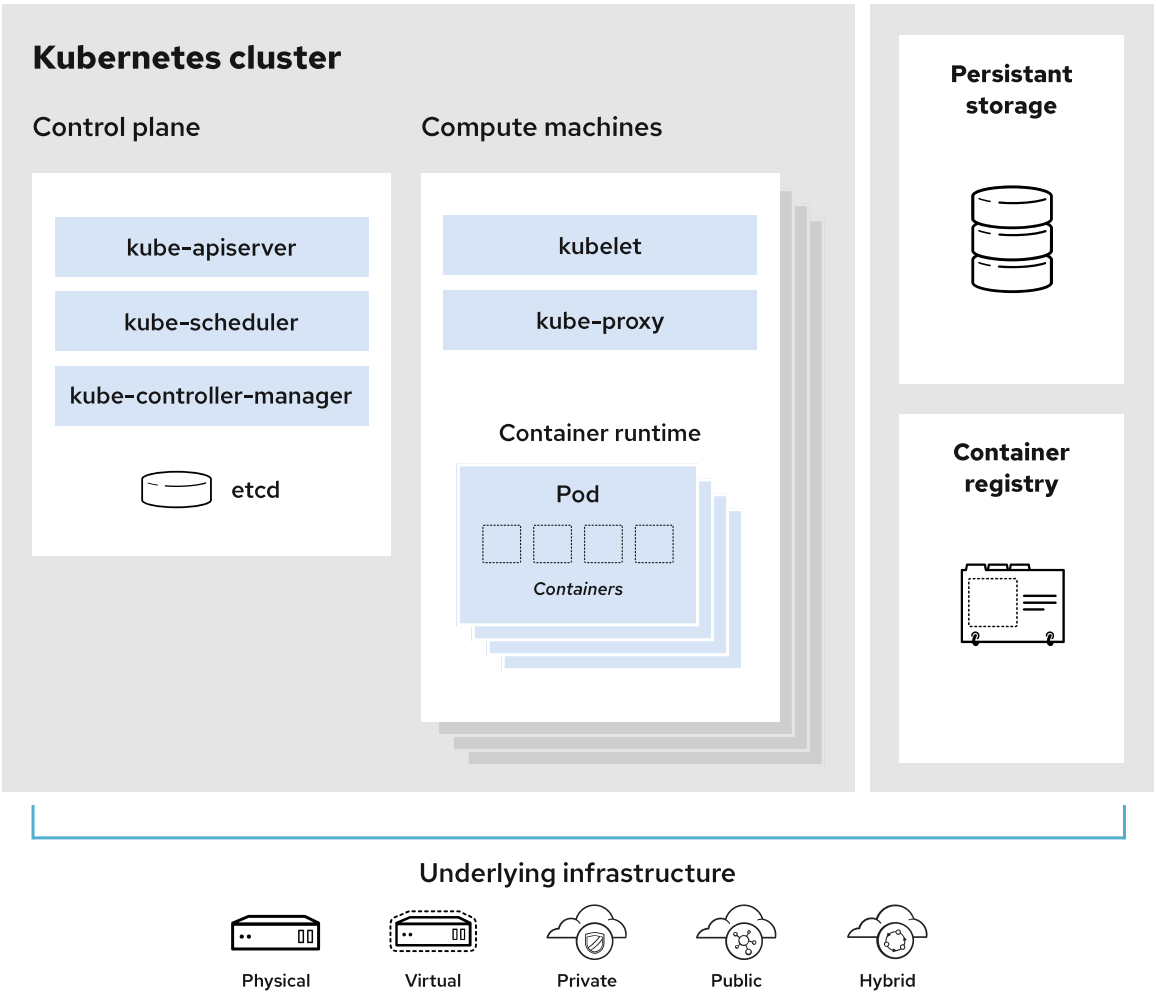
Kubernetes

Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications. It helps automate the process of managing containers.

it was google’s project.

- Control plane manages the work load
- Compute plain computes the work load

Architecture of Kubernetes:



Introduction to Kubernetes architecture

If you know only the basics of Kubernetes, you know it's an open source container orchestration platform designed for running distributed applications and services at scale. But you might not

https://www.redhat.com/en/topics/containers/kubernetes-architecture

Components are:

- Etcd
- kube-scheduler

- Controller manager
- kube-apiserver

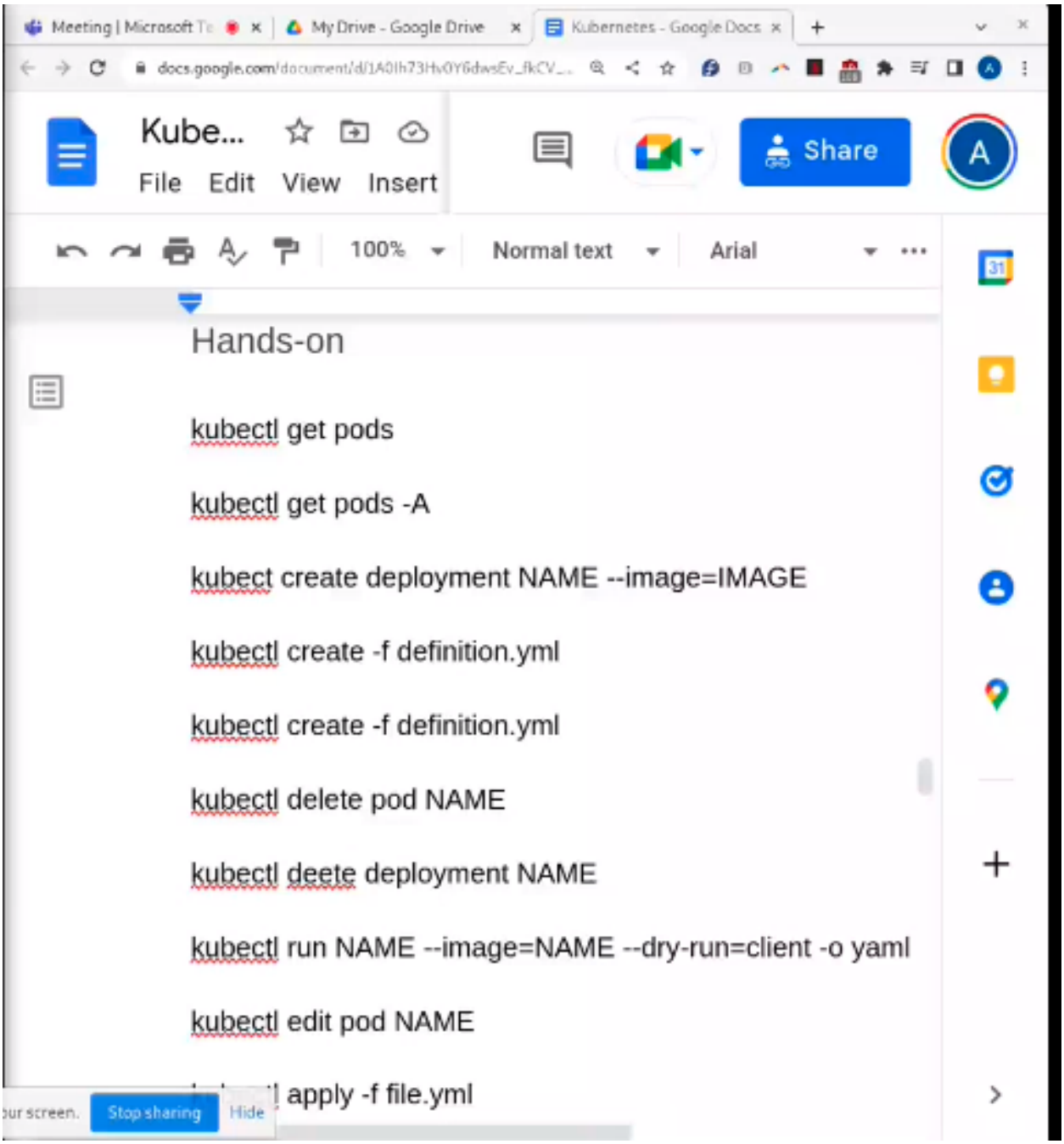
Worker node:

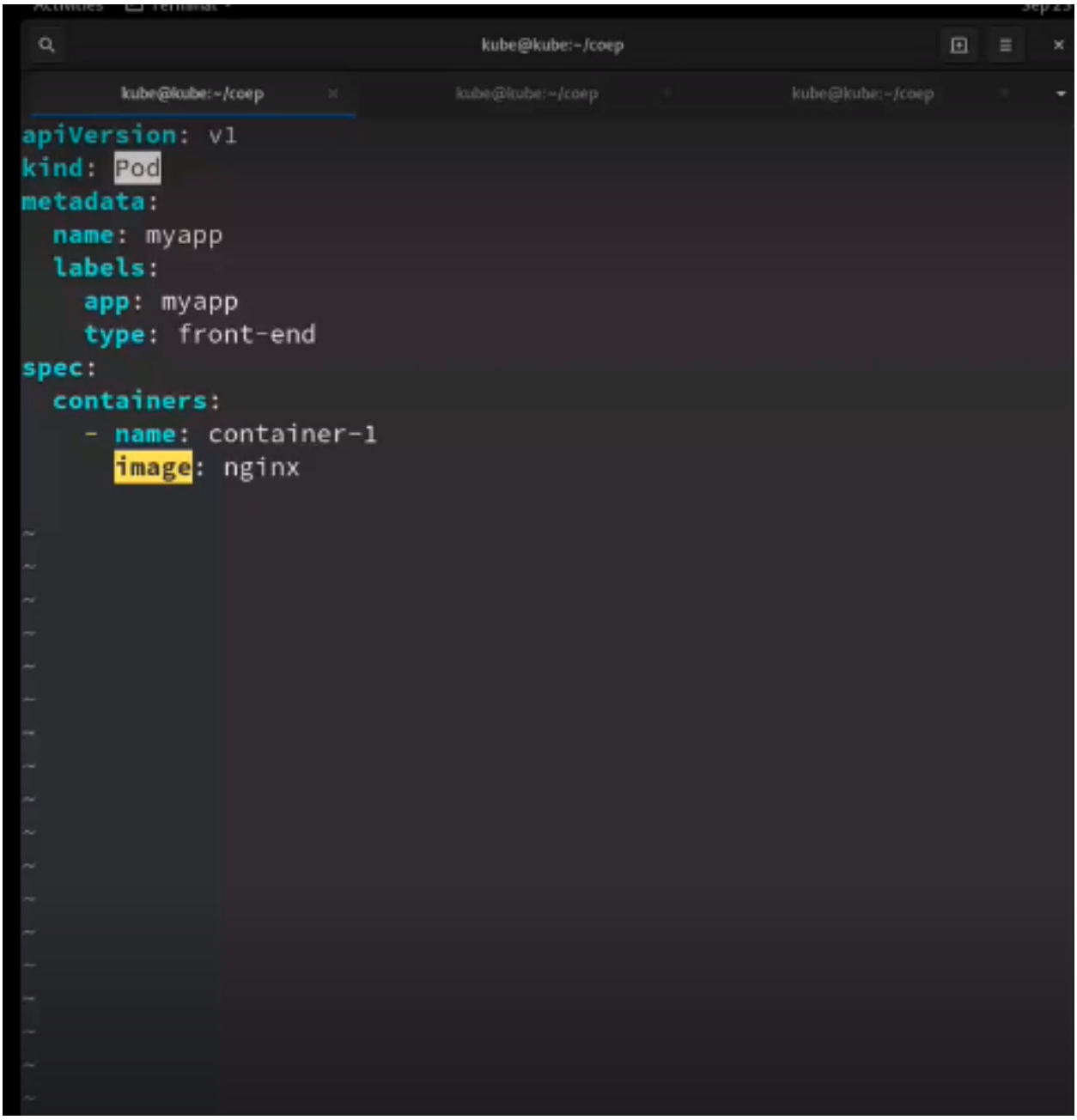
- controller runtime engine
- kubelet
- kube-proxy (for network setup)

kube-apiserver will contact the kubelet for something to be done, and kubelet will take care of that.

pods help for abstraction of container engine.

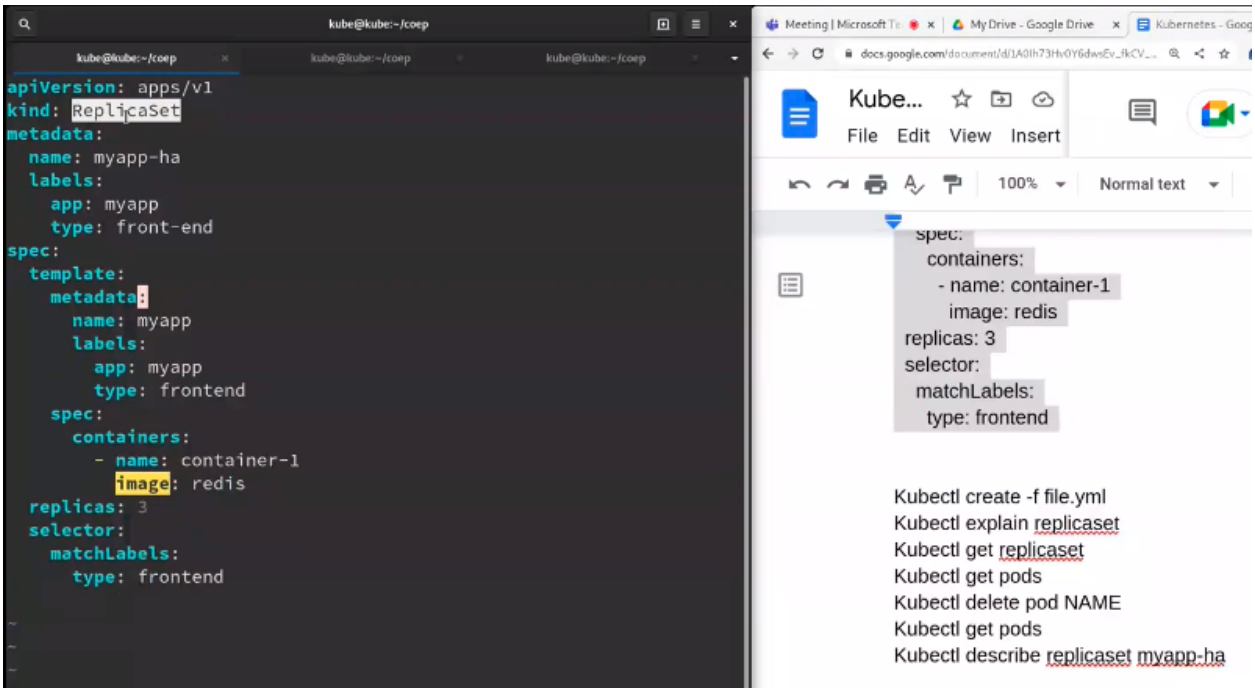
using pods the underlying container engine can be easily changed.
kubectl describe pod mypod | less


Meeting | Microsoft Te
My Drive - Google Drive
Kubernetes - Google Docs
docs.google.com/document/d/1A0Ih73Hv0Y6davsEv_fkCV_...
Kube...
File Edit View Insert
Share
A
100%
Normal text
Arial
Hands-on
kubectl get pods
kubectl get pods -A
kubect create deployment NAME --image=IMAGE
kubectl create -f definition.yml
kubectl create -f definition.yml
kubectl delete pod NAME
kubectl deete deployment NAME
kubectl run NAME --image=NAME --dry-run=client -o yaml
kubectl edit pod NAME
l apply -f file.yml
ur screen.
Stop sharing
Hide

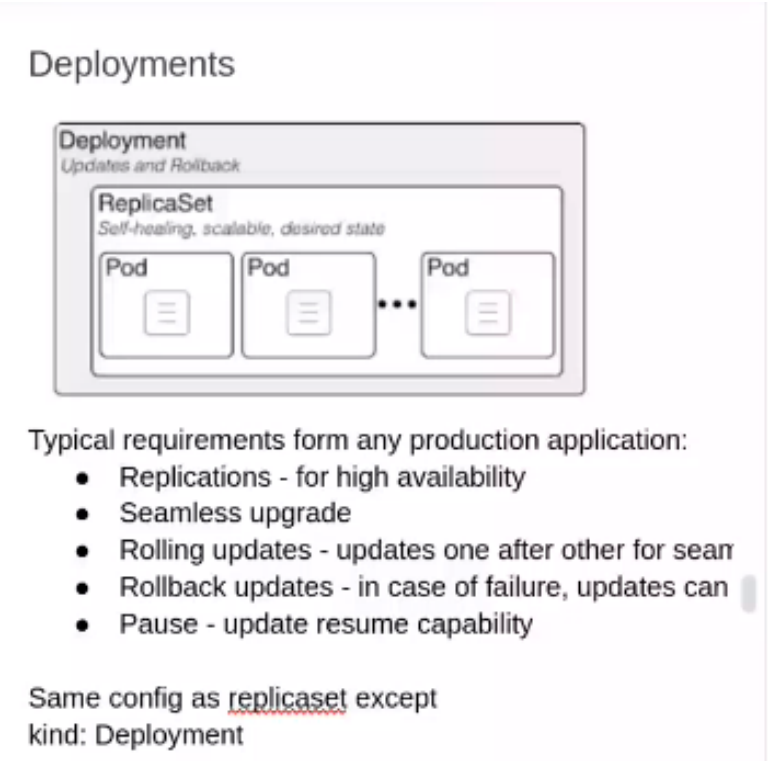


labels are custom things, but the others are kubernetes specific

```
kubectl describe pod mypod
```

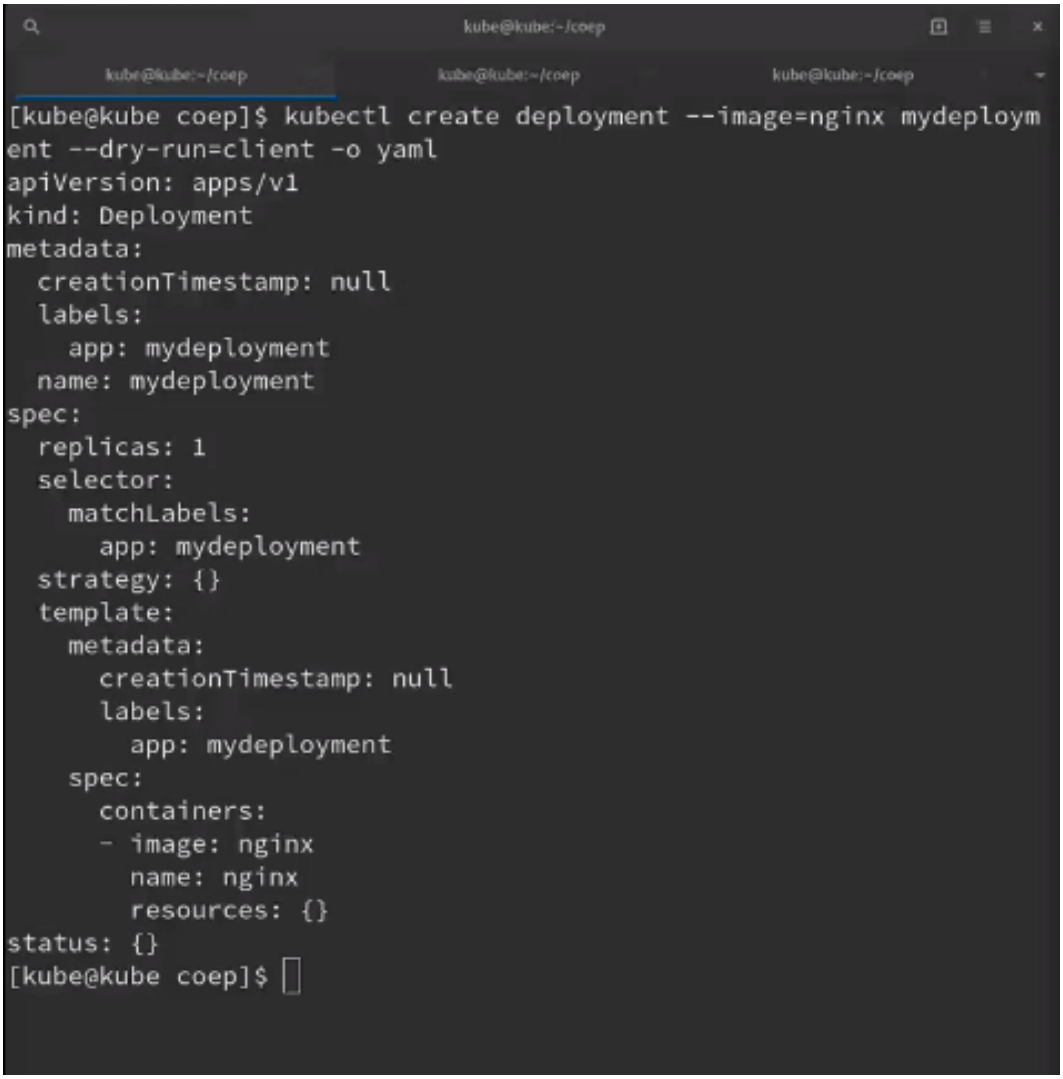


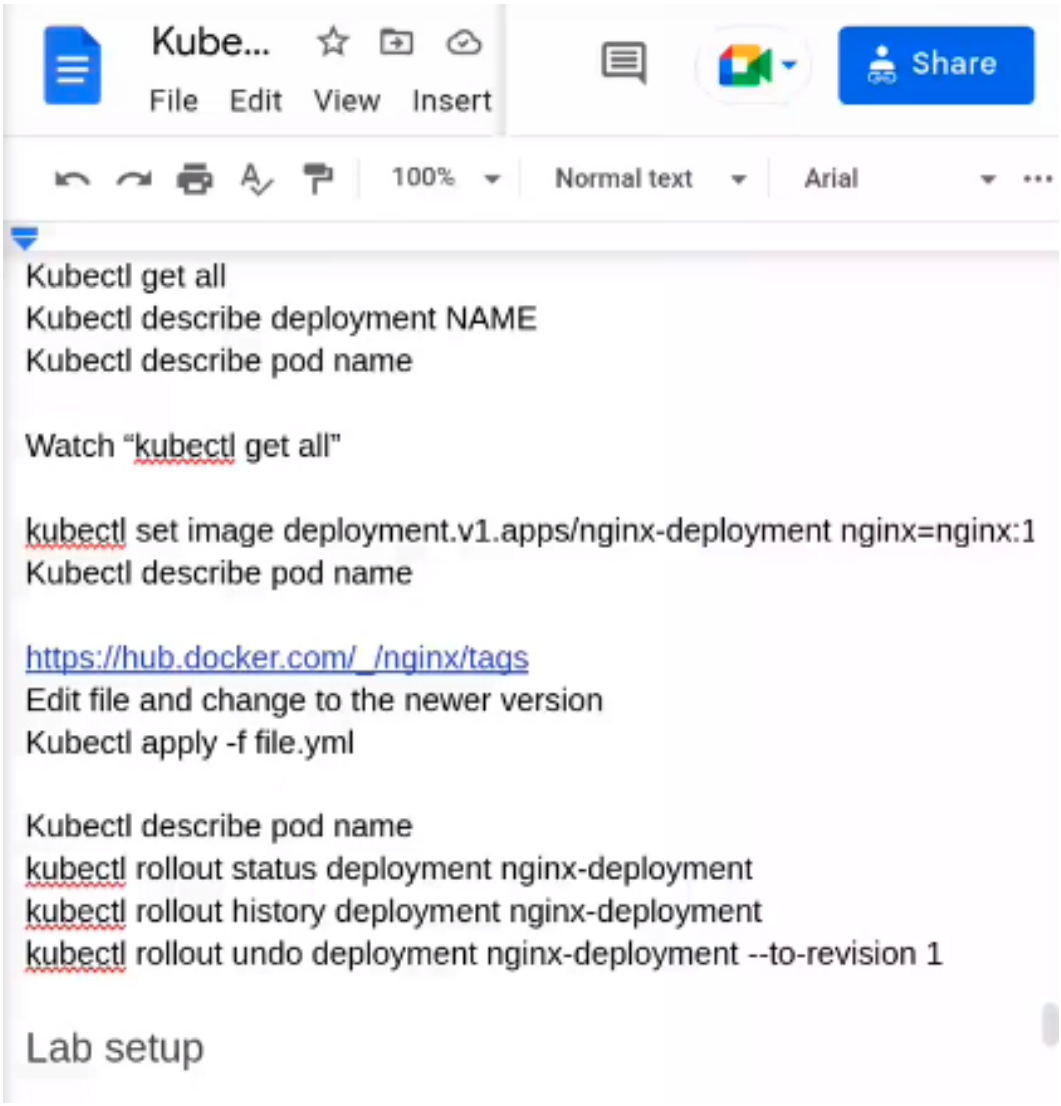

```
kubectl get replica
kubectl describe replicasets.apps myapp-ha
kubectl delete replicasets.apps myapp-ha
```



```
kubectl get deployment.apps
kubectl get all
```

To generate own yml file:





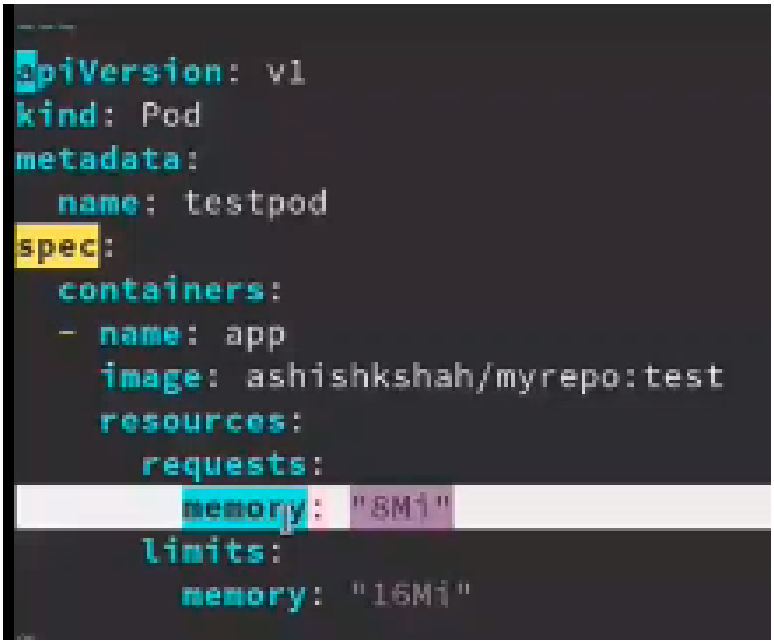
Namespaces:

```
kubectl create namespace myspace
kubectl get pods --namespace=myspace
kubectl create -f file.yml --namespace=myspace
kubeclt get namespaces
kubectl get pods --all-namespaces
kubectl config current-context
kubectl config set context minikube --namespace=myspace
kubectl get pods --watch
kubectl get pods -o wide
```

From custom docker image:

```
kubectl run --image ashishkshah/myrepo:test myfedora
kubectl exec -it myfedora -- bash
```

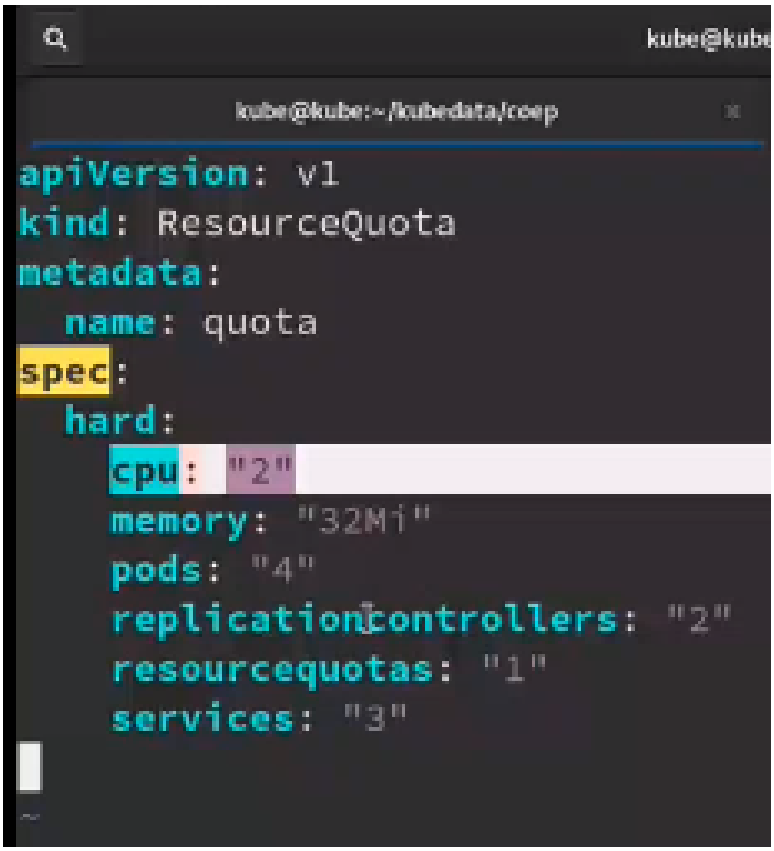
Prevent an image from taking the entire memory of the system



requests memory is the memory alloted at the beginning.

limits memory is the maximum memory and the pod gets killed if it exceeds this limit.

This can also be applied to namespaces



ClusterIP

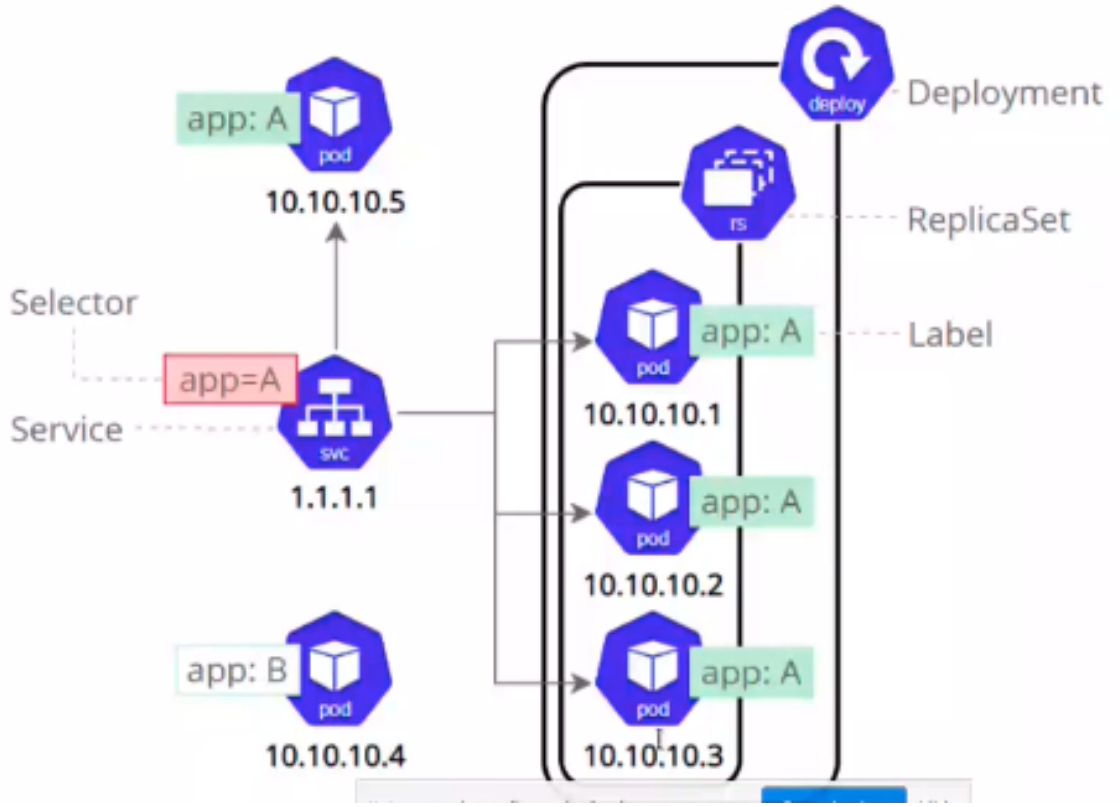
ClusterIP is the default service type.
This is used for inter service communication within the cluster.
Consider two deployment sets, front-end and back-end.
Communication between these two deployment sets is established using clusterip service.

Why do we need it? or why do we prefer this for inter pod communication within the cluster?

If we are using a pod's ip address for communication among the pods, when any pod is deleted or crashed due to some reason, the deployment set or replica set will spawn another pod to maintain the replica count.
The new pod spawned will have different ip addresses and hence it will not be easy to maintain the communication among the pods with new pods running with different ips.

To resolve this, we are creating a clusterip service which attaches itself to pods or replicas or daemonsets using labels and selectors.
In our example we will have two clusterip services for frontend and backend each.
The IP address of clusterip service is now being used for communication among the services instead of pods ip address.
Now even if the pods in replicaset

ip address, communication



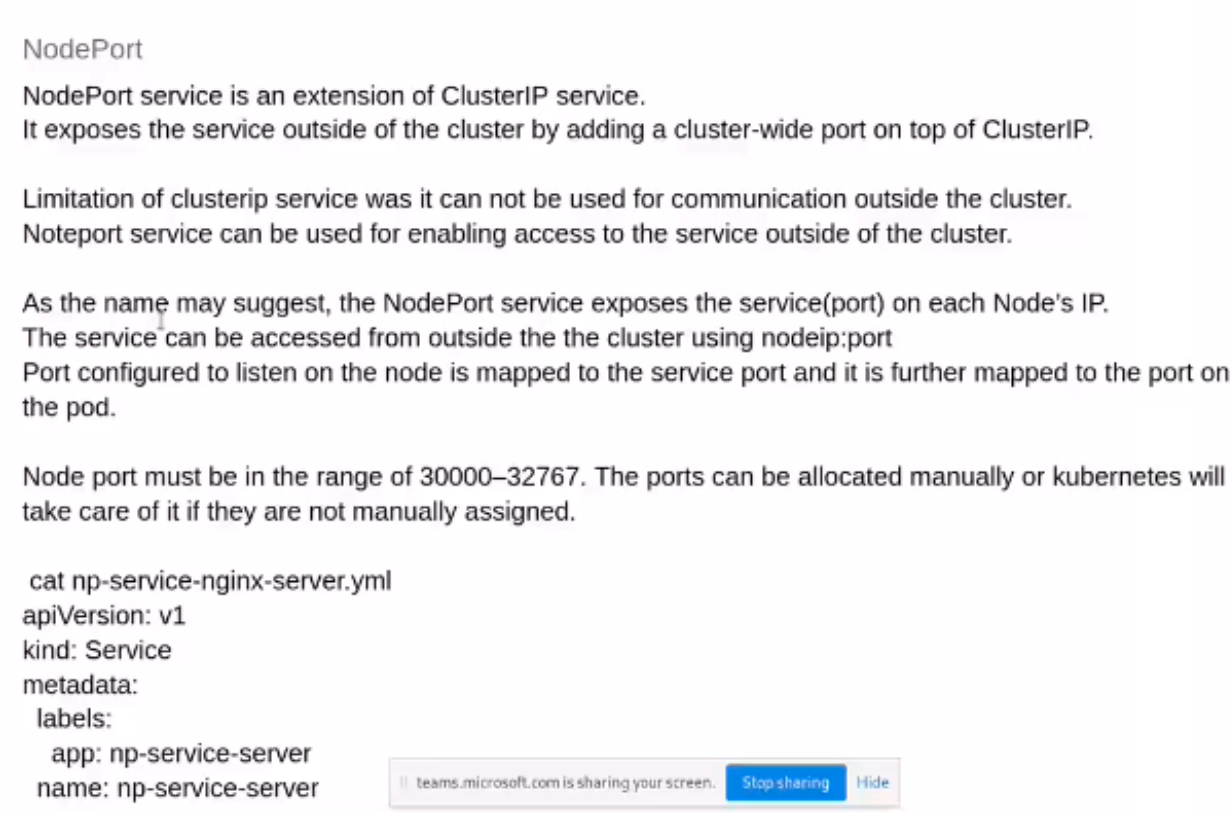
Service does load balancing and uses random algorithm for distribution

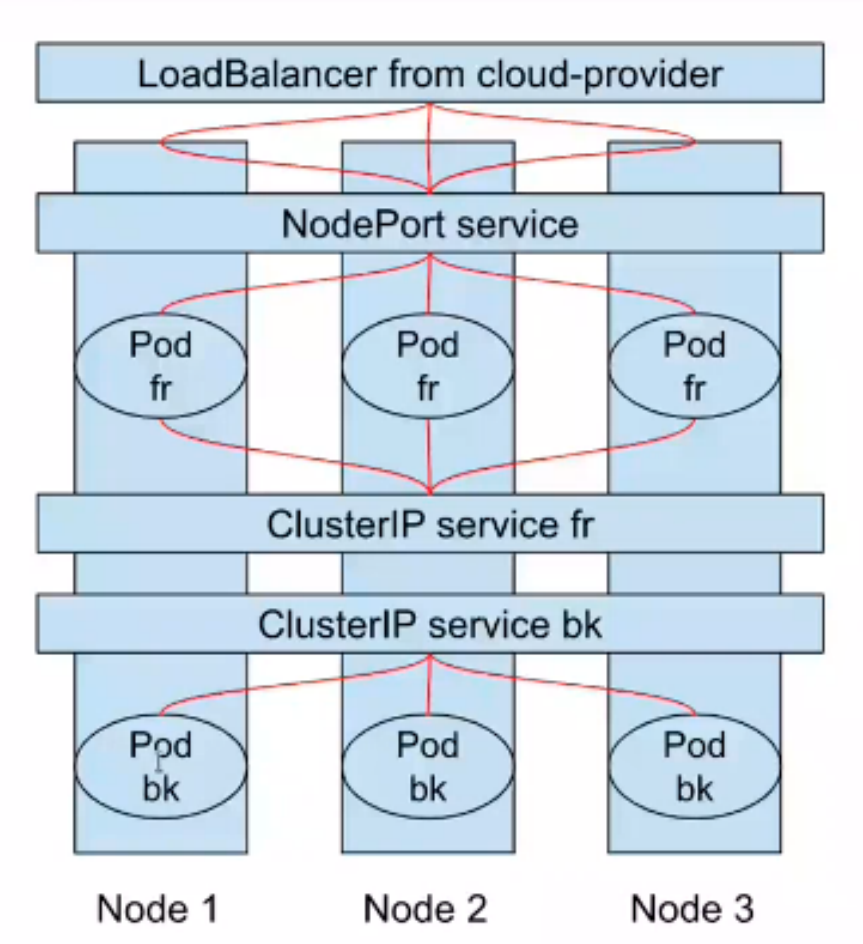
```
[kube@kubemaster service]$ kubectl apply -f cip-service-nginx-
cip-service-nginx-client.yml  cip-service-nginx-dep.yml     cip-service-nginx-server.yml
[kube@kubemaster service]$ kubectl apply -f cip-service-nginx-dep.yml
service/cip-nginx-server-dep created
[kube@kubemaster service]$ cat cip-service-nginx-dep.yml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: cip-nginx-server-dep
  name: cip-nginx-server-dep
spec:
  ports:
  - name: "80"
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx-server
  type: ClusterIP
[kube@kubemaster service]$
```

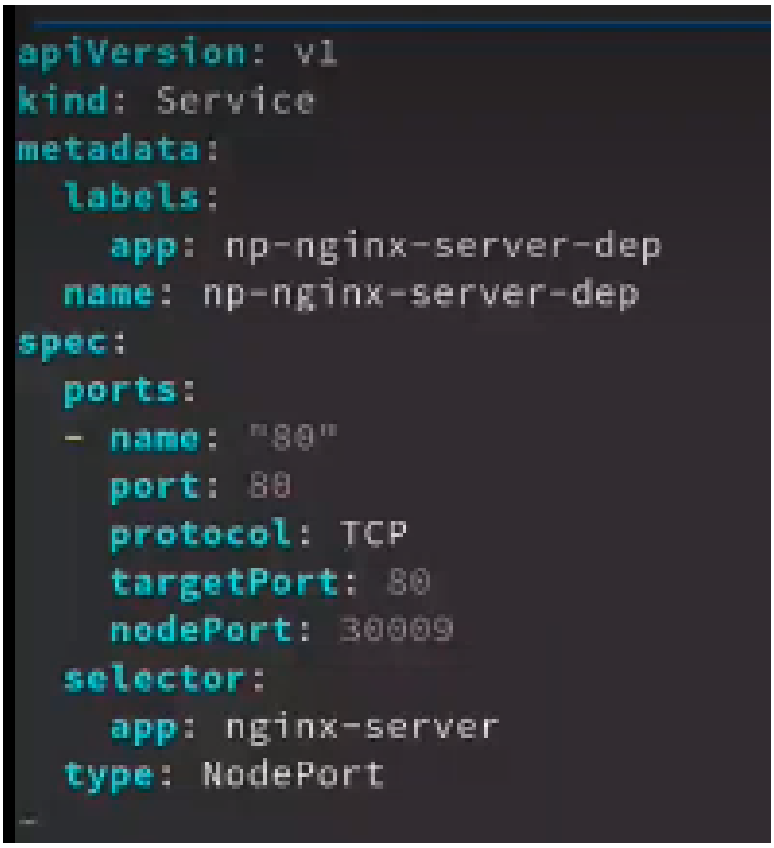
Target port is the port on the node, and port is the port on which it listens

Connectivity between service and pods will be through targetport and an external system will connect to the port.

NodePort




LoadBalancer from cloud-provider
NodePort service
Pod
fr
Pod
fr
Pod
fr
ClusterIP service fr
ClusterIP service bk
Pod
bk
Pod
bk
Pod
bk
Node 1
Node 2
Node 3



```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: np-nginx-server-dep
  name: np-nginx-server-dep
spec:
  ports:
  - name: "80"
    port: 80
    protocol: TCP
    targetPort: 80
    nodePort: 30009
  selector:
    app: nginx-server
  type: NodePort
```

The above is the yaml for NodePort configuration

Load Balancer:

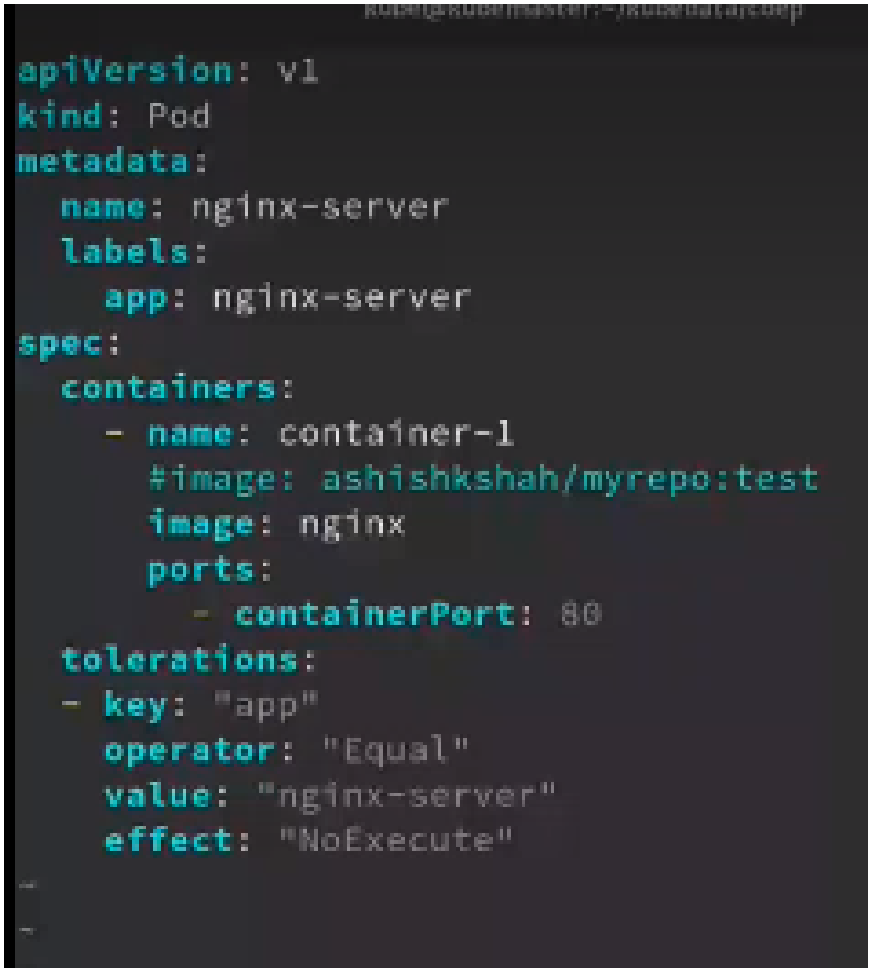
LoadBalancer

The nodeport service has enabled external connectivity for the service. But still there is one problem with it.
The service is listening on the specified port on node's ip. That means if the pods in your replicaset are distributed among different nodes, all the nodes ip addresses will listen on the nodeport. Hence there will be a number of IP:PORT combinations available to access your service.
Needless to say there is no server side distribution of the traffic with this approach of accessing service. This approach is not practically used
for providing external access to the service but is used as an intermediate step.

LoadBalancer service is an extension of NodePort service.
Loadbalancer integrates NodePort with cloud-based load balancers and provides a single ip:port combination for accessing service from external networks.
The load is then distributed by load balancer service to different nodes underneath.

```
kubectl taint node kubeworker1 app-nginx-server:NoExecute
```

Following yaml will still start pod at the tainted worker, its tolerance to the taint



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-server
  labels:
    app: nginx-server
spec:
  containers:
    - name: container-1
      #image: ashishkshah/myrepo:test
      image: nginx
      ports:
        - containerPort: 80
  tolerations:
  - key: "app"
    operator: "Equal"
    value: "nginx-server"
    effect: "NoExecute"
```