


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.

What is Kubernetes?

Kubernetes (also known as k8s or "kube") is an open source container orchestration platform that automates many of the manual processes involved in deploying, managing, and scaling

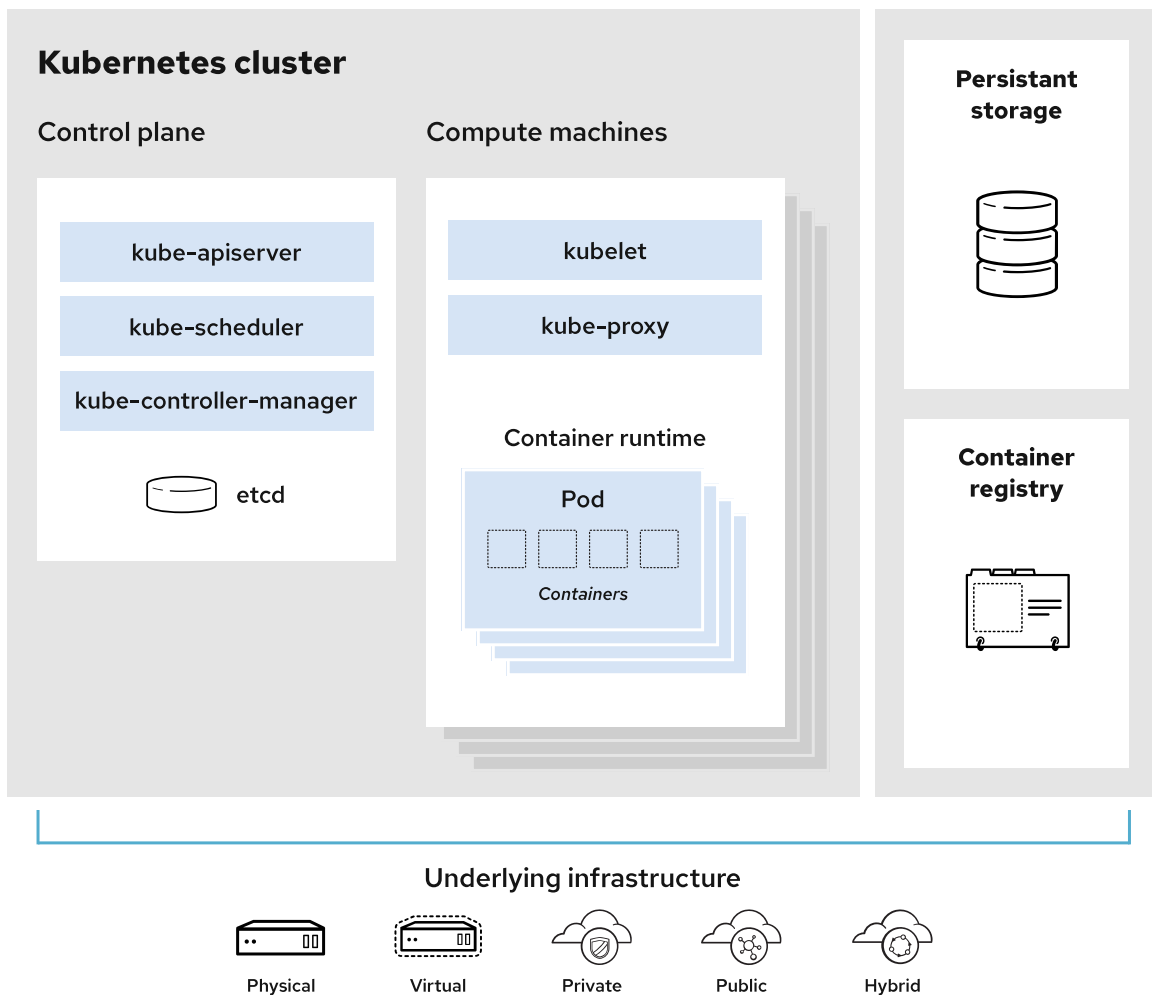
 <https://www.redhat.com/en/topics/containers/what-is-kubernetes>



it was google's project.


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

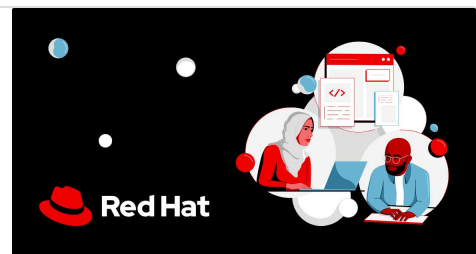
Architecture of Kubernetes:



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:

- container 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 To... x | My Drive - Google Drive x | Kubernetes - Google Docs x

docs.google.com/document/d/1A0lh73Hv0Y6dwsEv_fkCV...

Kube... ☆ 📁 ☁️

File Edit View Insert

100% Normal text Arial

Hands-on

- `kubectl get pods`
- `kubectl get pods -A`
- `kubectl create deployment NAME --image=IMAGE`
- `kubectl create -f definition.yml`
- `kubectl create -f definition.yml`
- `kubectl delete pod NAME`
- `kubectl delete deployment NAME`
- `kubectl run NAME --image=NAME --dry-run=client -o yaml`
- `kubectl edit pod NAME`
- `kubectl apply -f file.yml`

our screen. Stop sharing Hide

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  labels:
    app: myapp
    type: front-end
spec:
  containers:
  - name: container-1
    image: nginx
```

labels are custom things, but the others are kubernetes specific

```
kubectl describe pod mypod
```

```

apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-ha
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp
      labels:
        app: myapp
        type: frontend
    spec:
      containers:
        - name: container-1
          image: redis
  replicas: 3
  selector:
    matchLabels:
      type: frontend

```

Meeting | Microsoft T... | My Drive - Google Drive | Kubernetes - Goog

docs.google.com/document/d/1A0ih73Hv0Y6dwsEv_9kCV... | Kube...

File Edit View Insert

100% Normal text

```

spec:
  containers:
    - name: container-1
      image: redis
  replicas: 3
  selector:
    matchLabels:
      type: frontend

```

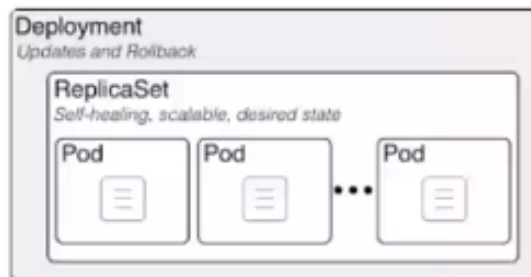
Kubectl create -f file.yml
 Kubectl explain replicaset
 Kubectl get replicaset
 Kubectl get pods
 Kubectl delete pod NAME
 Kubectl get pods
 Kubectl describe replicaset myapp-ha

```

kubectl get replica
kubectl describe replicaset.apps myapp-ha
kubectl delete replicaset.apps myapp-ha

```

Deployments



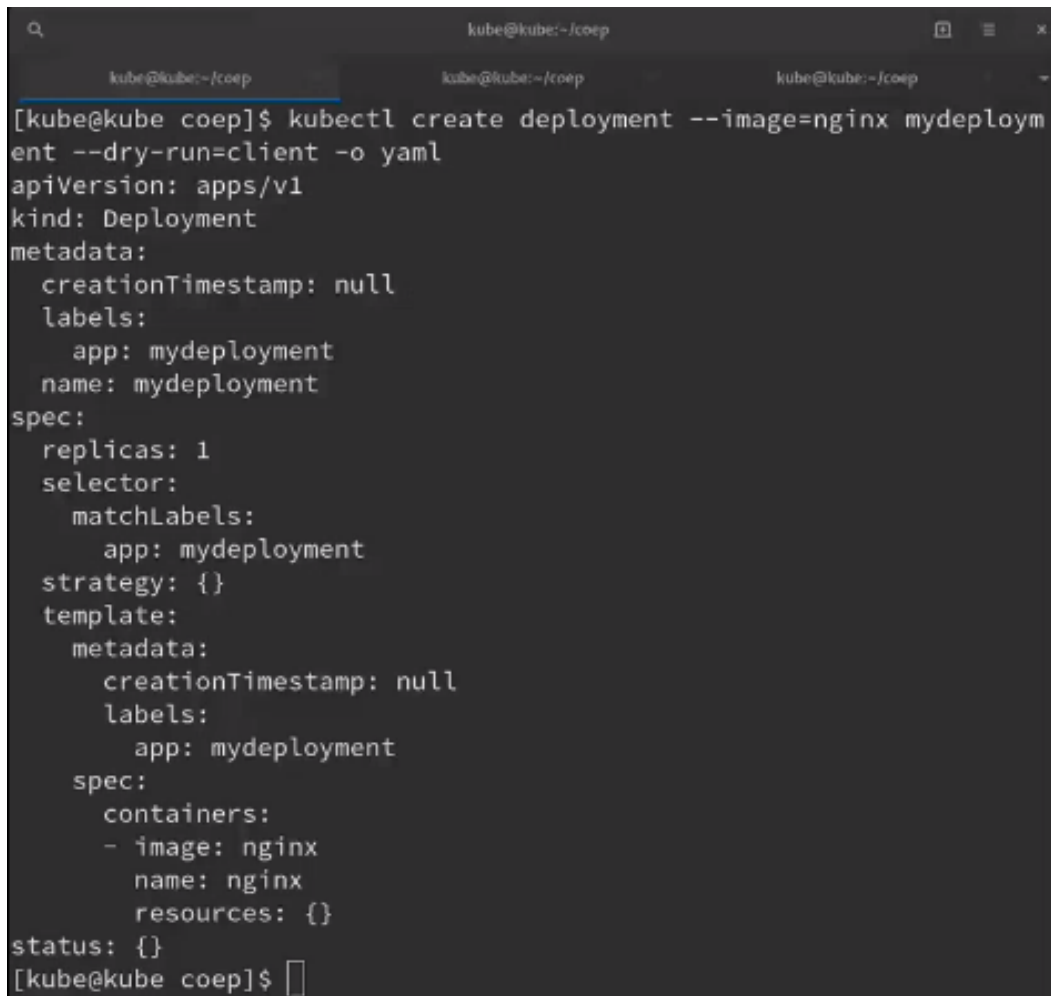
Typical requirements form any production application:

- Replications - for high availability
- Seamless upgrade
- Rolling updates - updates one after other for sear
- Rollback updates - in case of failure, updates can
- Pause - update resume capability

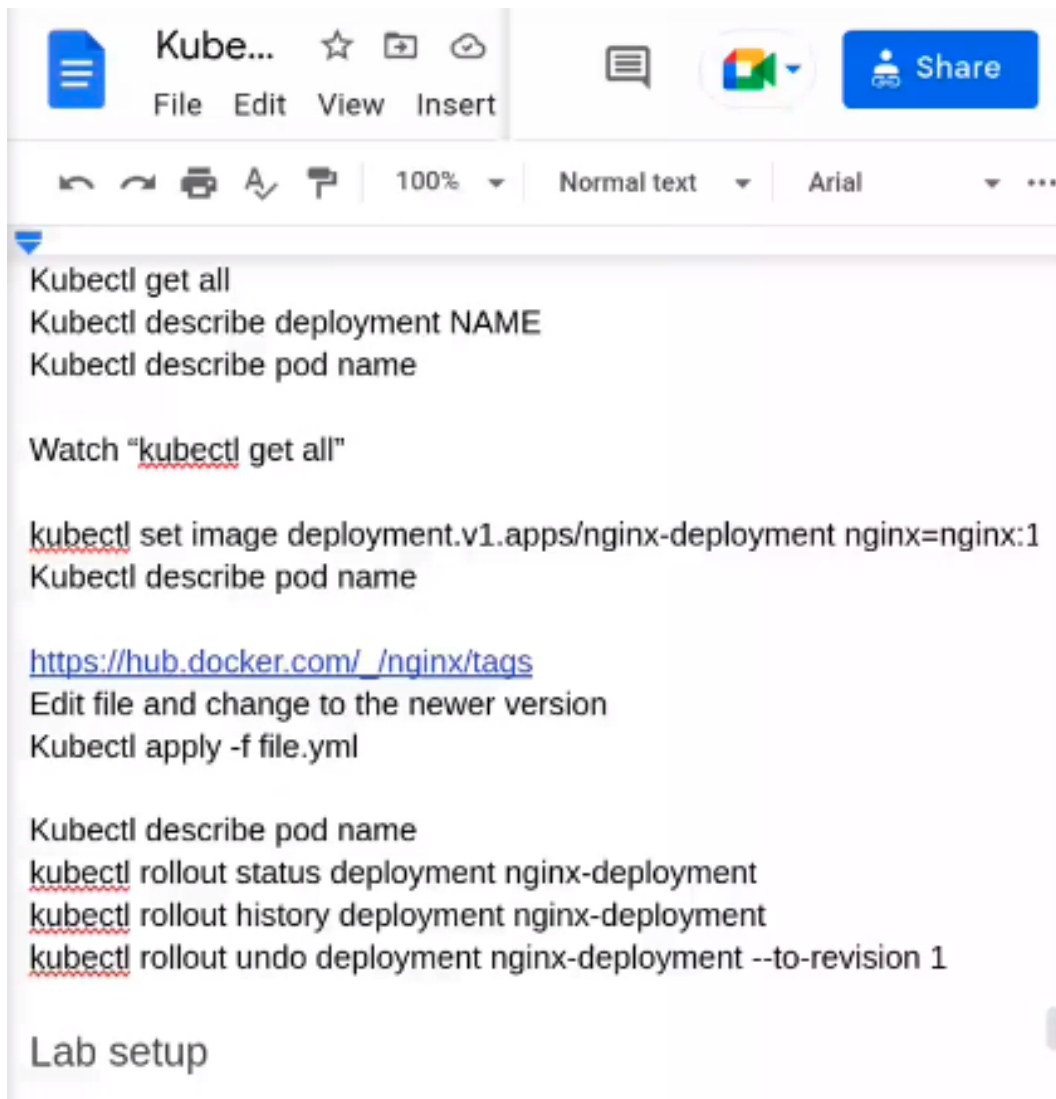
Same config as replicaset except
 kind: Deployment

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

To generate own yml file:



```
kube@kube:~/coop  
[kube@kube coop]$ kubectl create deployment --image=nginx mydeployment --dry-run=client -o yaml  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  creationTimestamp: null  
  labels:  
    app: mydeployment  
  name: mydeployment  
spec:  
  replicas: 1  
  selector:  
    matchLabels:  
      app: mydeployment  
  strategy: {}  
  template:  
    metadata:  
      creationTimestamp: null  
      labels:  
        app: mydeployment  
    spec:  
      containers:  
      - image: nginx  
        name: nginx  
        resources: {}  
status: {}  
[kube@kube coop]$
```



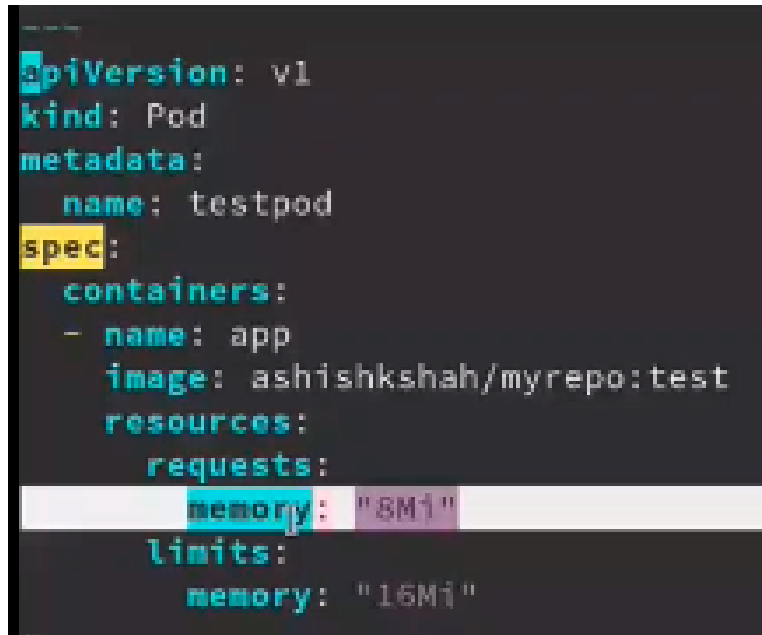
Namespaces:

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



```
apiVersion: v1
kind: Pod
metadata:
  name: testpod
spec:
  containers:
  - name: app
    image: ashishkshah/myrepo:test
    resources:
      requests:
        memory: "8Mi"
      limits:
        memory: "16Mi"
```

requests memory is the memory allotted 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

```
kube@kube:~/kubedata/coep
apiVersion: v1
kind: ResourceQuota
metadata:
  name: quota
spec:
  hard:
    cpu: "2"
    memory: "32Mi"
    pods: "4"
    replicationcontrollers: "2"
    resourcequotas: "1"
    services: "3"
```

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.

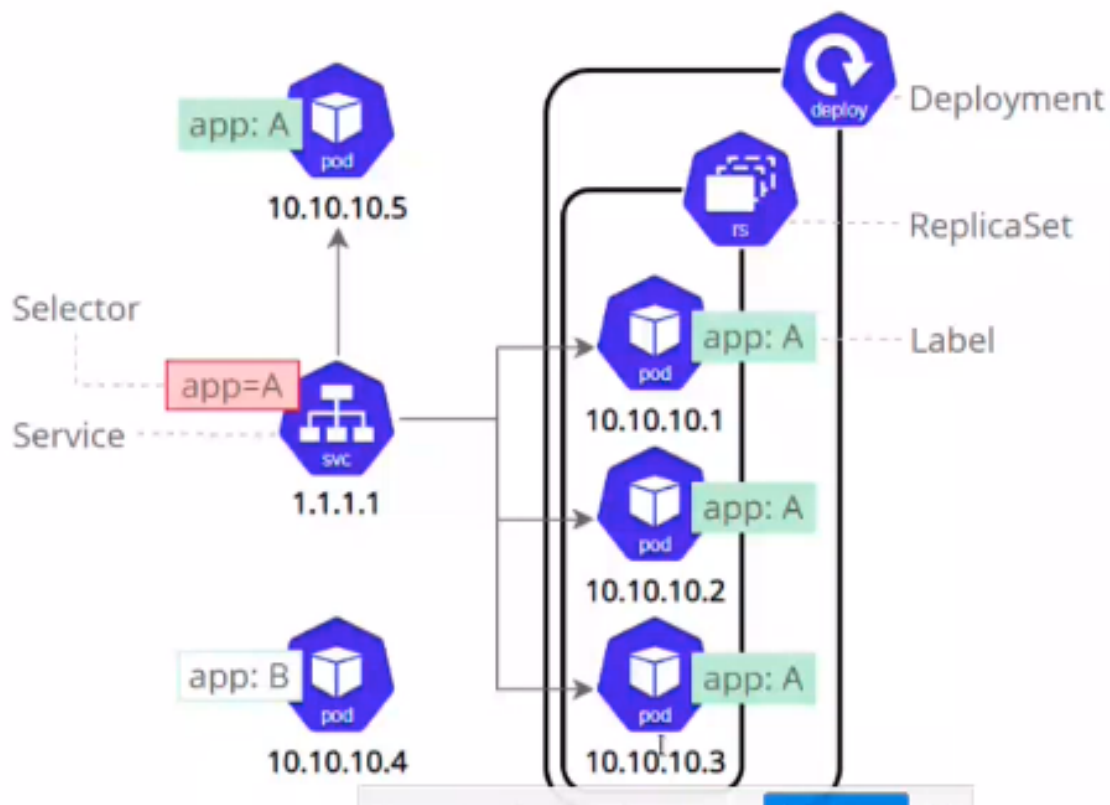
The new pod spawned will have different ip addresses and hence it will not be easy to maintain the communication among the pods with new pods running with different ips.

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 are deleted, the clusterip service will maintain its 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

NodePort

NodePort service is an extension of ClusterIP service.

It exposes the service outside of the cluster by adding a cluster-wide port on top of ClusterIP.

Limitation of clusterip service was it can not be used for communication outside the cluster.

Noteport service can be used for enabling access to the service outside of the cluster.

As the name may suggest, the NodePort service exposes the service(port) on each Node's IP.

The service can be accessed from outside the the cluster using nodeip:port

Port configured to listen on the node is mapped to the service port and it is further mapped to the port on the pod.

Node port must be in the range of 30000–32767. The ports can be allocated manually or kubernetes will take care of it if they are not manually assigned.

```
cat np-service-nginx-server.yml
```

```
apiVersion: v1
```

```
kind: Service
```

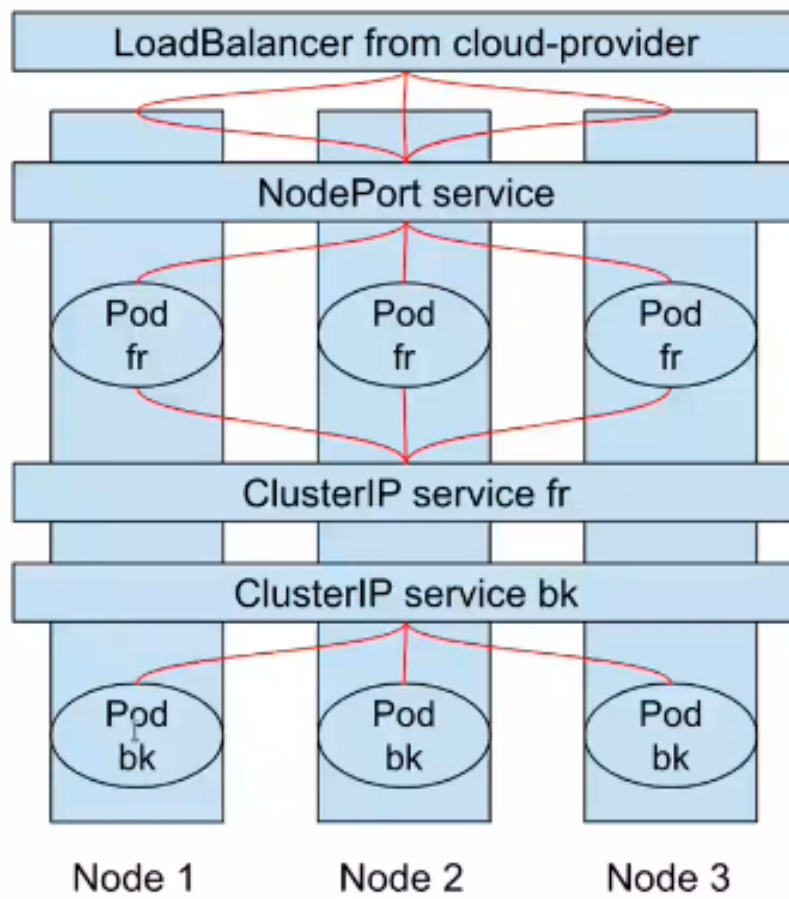
```
metadata:
```

```
  labels:
```

```
    app: np-service-server
```

```
  name: np-service-server
```

teams.microsoft.com is sharing your screen. [Stop sharing](#) [Hide](#)



```

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

```
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"
```