

KUBERNETES NOTES

KUBERNETES - GREEK WORD

HELMSMAN OR CAPTAIN

K9S - LATEST

K8S - CURRENT

CONTAINER ORCHESTRATION ENGINE

CONTAINER MANAGEMENT TOOL

DOCKER - GOOGLE

KUBERNETES - GOOGLE

GOLANG

OPENSOURCE

ADVANTAGES

Automates deployment

Scaling

Manages containerized apps

Load balancing

Storage Orchestration

- Self healing
- Rollbacks and Rollouts
- Secret and Config Management
- Batch execution
- Horizontal Scaling
- Monitoring

Container Management Tools

- Kubernetes
- Docker Swarm
- Apache Mesos - marathon

KEYWORDS

- NODE
- POD
- CONTAINER

Batch Execution

- Seq
- Parallel

Horizontal Scaling

Replication Controller ReplicaSets

Nodes can be

Physical Machine
Virtual Machine
VM on Cloud

Kubernetes capacity

No more than - 5000 Nodes, 150000 pods, 300000 containers

No more than 100 pods per node

4 components of master node

1. API Server
2. Scheduler
3. Controller manager
4. Etcd

API SERVER

Communicate with one another applications
Front-end
Exposes api for each operations
Interact with kubernetes cluster

Control Manager

Kube Controller Manager
Cloud Controller manager

Kube controller manager

1. Node Controller - notices and responds when nodes goes down
2. Replication Controller - maintain correct count of pods for every replication
3. EndPoints Controller - joins services and pods
4. Service Accounts and token controller - create default accounts and API access tokens for new namespaces.

Cloud Controller manager

1. Node Controller
2. Route Controller
3. Service controller
4. Volume controller

—cloud-provider

Worker Node components

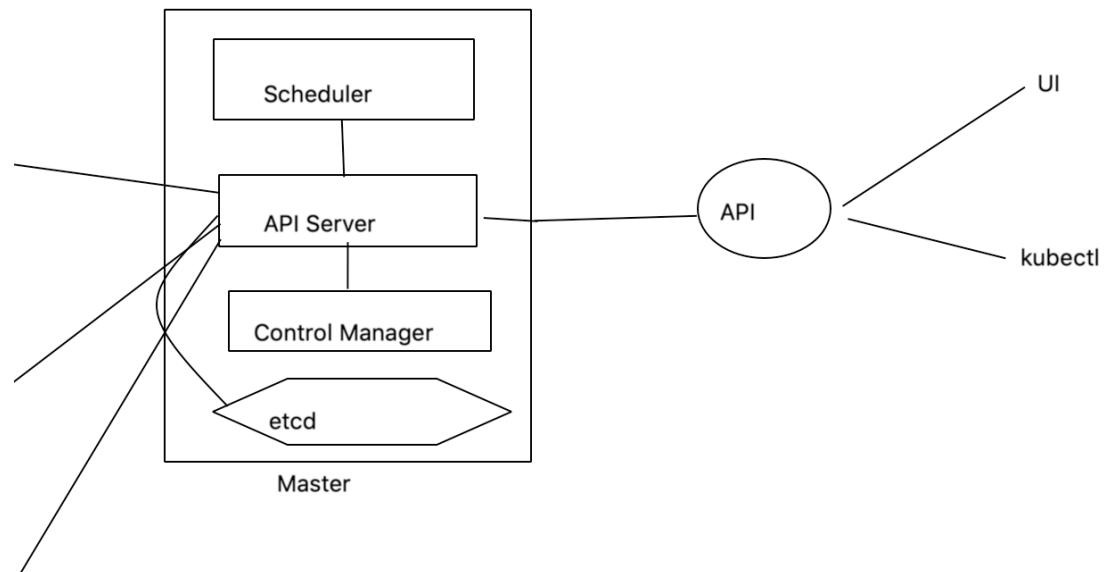
1. Kubelet
2. Kube-proxy
3. Container runtime

Kubernetes Dashboard

Monitoring

Logging

DNS



Kubernetes Installation

12GB RAM Required

4GB - base OS

VMware - 512MB

UBUNTU- 4GB

WORKER NODE - 4 GB

HDD - 40GB Free Space

Minimum 4 cores required

WE USE ONE MASTER AND 2 WORKERS ARCHITECTURE

Step 1

Create 3 images of ubuntu run on different ipaddress and network to be “bridge”

System names

Ubuntu-Master - 192.168.1.101

Ubuntu-Worker1 - 192.168.1.102

Ubuntu-Worker2 - 192.168.1.103

Sudo hostnamectl set-hostname Ubuntu-Master - in first system

Sudo hostnamectl set-hostname Ubuntu-Worker1 - in second system

Sudo hostnamectl set-hostname Ubuntu-Worker2 - in Third system

Step 2

IN ALL MACHINES

sudo apt-get update

Sudo apt-get install docker.io

docker —version

Enable the docker service

Sudo su

Systemctl enable docker

Systemctl start docker

Systemctl status docker

Step 3

IN ALL MACHINES

```
Curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add
```

Step 4

IN ALL MACHINES

Add Kubernetes repo

```
Sudo apt-add-repository "deb http://apt.kubernetes.io/  
kebernetes-cenial main"
```

Sudo apt-get install kubeadm kubelet kubectl

Sudo apt-mark hold kubeadm kubelet kubectl

Wait for 10 mins

Kubeadm version

Sudo swap off -a (**IN ALL MACHINES**)

Step 5

IN Ubuntu-Master Machine

Sudo kubeadm init —pod-network-cidr=10.244.0.0/16

It will display kubeadm join (KEEP BACKUP OF THE STATEMENT)

Mkdir -p \$HOME/.kube

Sudo cp -I /etc/kubernetes/admin.conf \$HOME/.kube/config

Sudo chown \$(id -u):\$(id -g) \$HOME/.kube/config

Sudo kubectl apply <https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>

Kubectl get pods —all-namespaces

Step 6

IN ALL WORKER MACHINES

Copy back up kubeadm join command fully with given parameter

Run in all worker machines

COME BACK TO MASTER MACHINE

Kubectl get nodes

ONLINE OPTIONS TO RUN KUBERNETES

Kubernetes Playground

Play with K8s

Play with Kuberenetes Classroom

KUBECTL COMMANDS

kubectl get nodes

Kubectl get svc

Kubectl get services

PODS

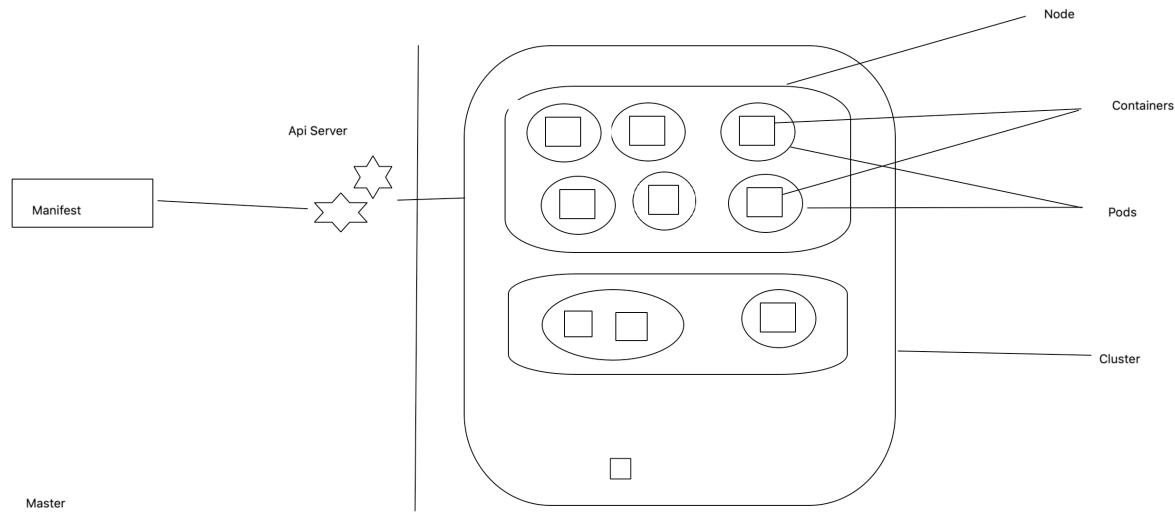
Atomic Unit of Scheduling

VIRTUALIZATION - VM

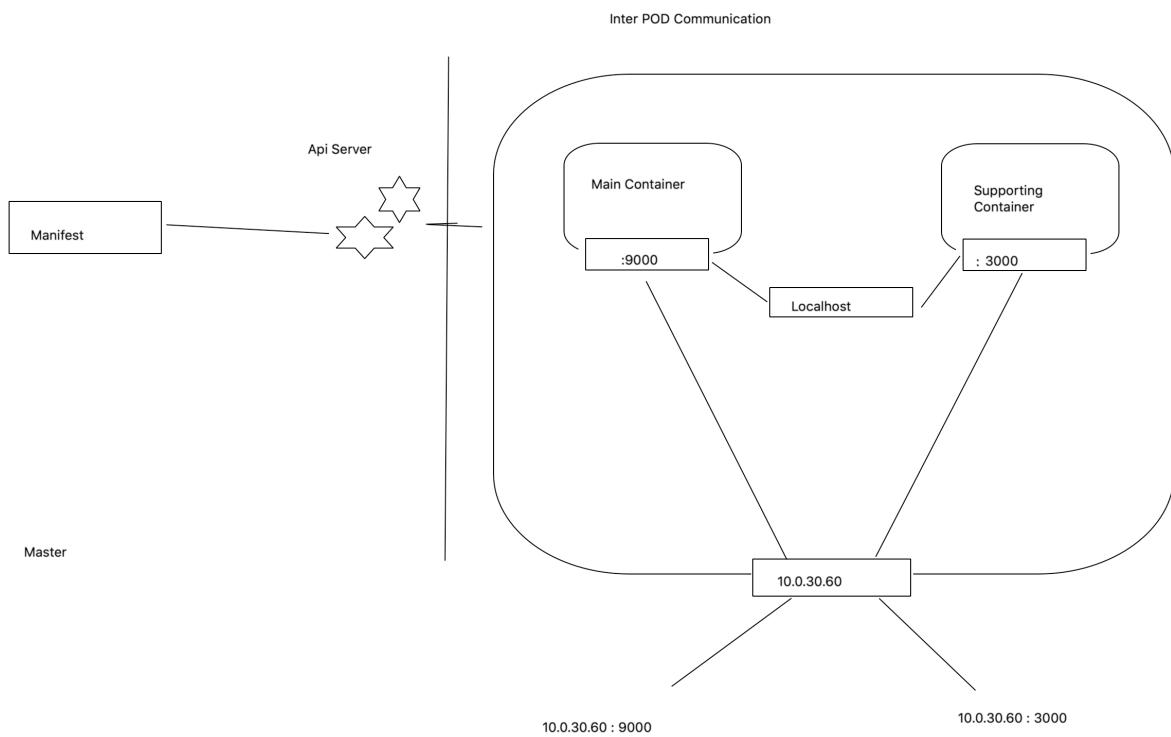
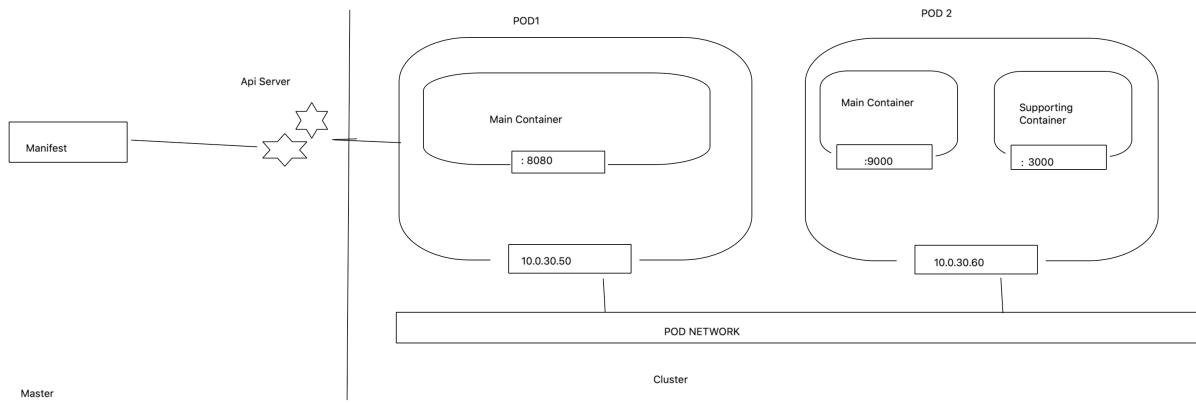
DOCKER - Container

KUBERNETES - Pod

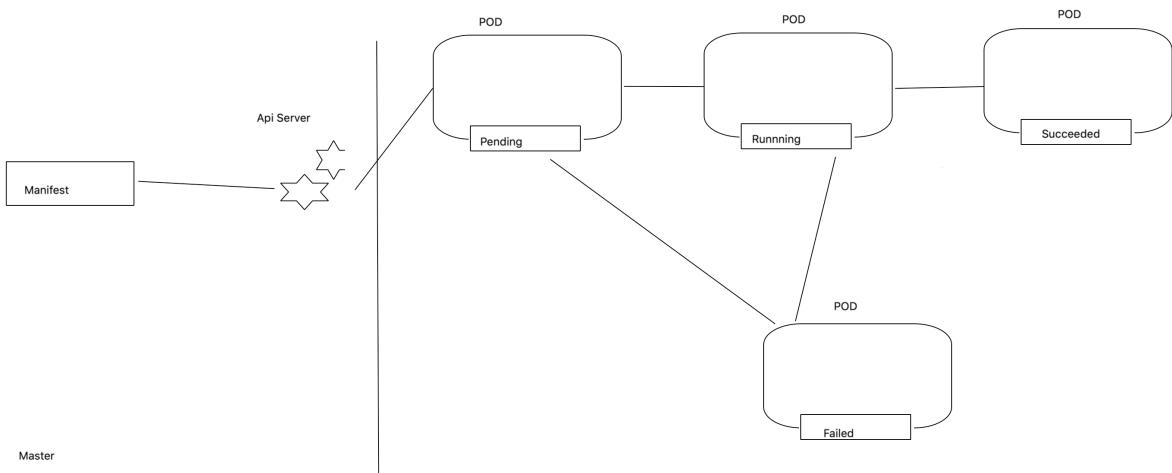
POD ARCHITECTURE



POD NETWORKING



POD LIFE CYCLE



POD DEMO

1. Create example1.yml

```
# nginx pod example
```

```
apiVersion: v1
kind : Pod
metadata:
  name : nginx-pod
  labels :
    app : nginx
    tier : dev
```

```
spec:
  containers :
```

```
- name : nginx-container  
  image : nginx
```

kubectl create -f example1.yml

kubectl get pod

SEE THE STATE

FIRST - PENDING

SECOND - RUNNING

Kubectl get pod -o wide

Kubectl get pod nginx-pod -o yaml

kubectl describe pod nginx-pod

Ping <ipaddress of the pod>

Kubectl exec -it nginx-pod - - /bin/sh

Kubectl delete pod nginx-pod

REPLICATION CONTROLLER

REPLICATION CONTROLLER is a service - Ensures that a specified number of pods are always running at any time

If there are excess pods - it kills and vice versa

New pod is created and launched if there is a failure or deleted or terminated.

RC and pods are associated with "labels"

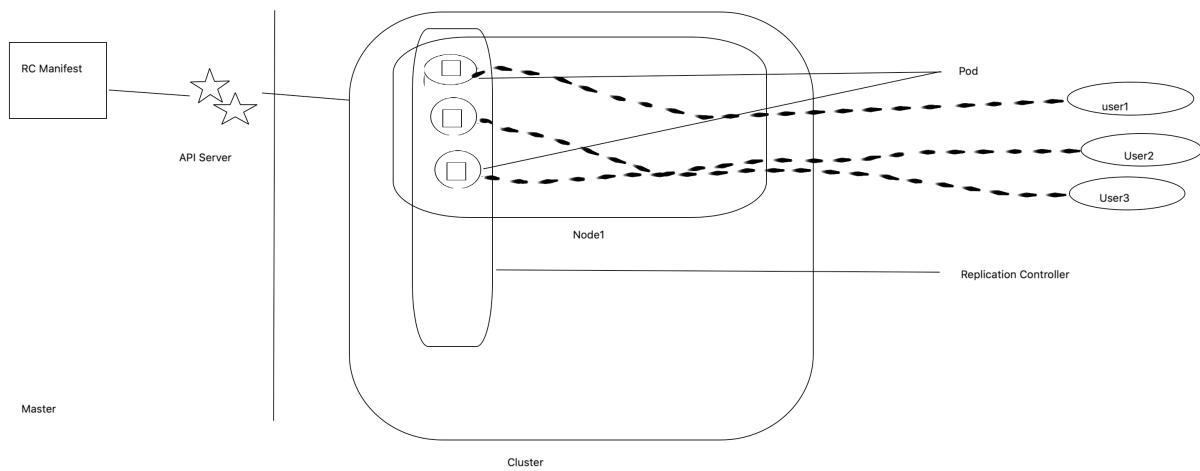
Always rc with count must be 1 then only pod is always available

Used for HA

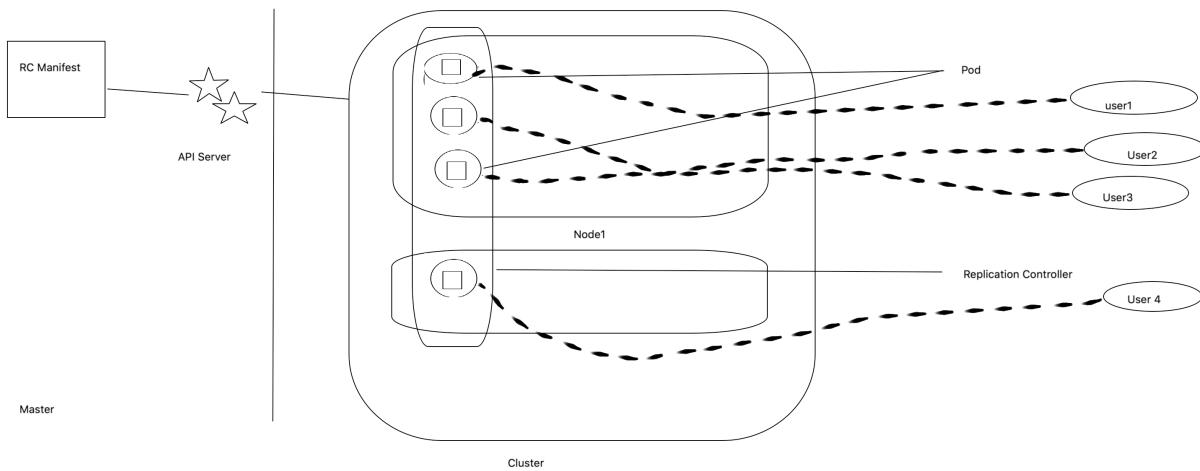
High Availability architecture

Point 1

Load Balancing



HIGH AVAILABILITY



REPLICATION CONTROLLER REPLICA SETS

WORK WISE BOTH ARE SAME

OLD TECHNIQUE

REPLICATION CONTROLLER

Uses equality based selector

NEW TECHNIQUE

REPLICA SETS

Set based selector

DEMO

```
# nginx-rc.yml
```

```
apiVersion : v1
kind : ReplicationController
metadata :
  name : nginx-rc.yml
spec :
  replicas : 3
  selector :
    app : nginx-app
  template :
    metadata :
      name : nginx-pod
      labels :
        app : nginx-app
    spec :
      containers :
        - name : nginx-container
          image : nginx
          ports :
            - containerPort : 80
```

Commands

Kubectl create -f nginx-rc.xml

Kubectl get pods

Kubectl get po

Kubectl get po -l app=nginx-app

Kubectl describe rc nginx-rc

Kubectl get po -o wide

Kubectl get nodes

Kubectl get po -o wide

Scaling UP

Kubectl scale rc nginx-rc --replicas=5

Kubectl get rc nginx-rc

Kubectl get po -o wide

Scale Down

Kubectl scale rc nginx-rc -- replicas=3

Kubectl get rc nginx-rc

Kubectl get po -o wide

Deletion

Kubectl delete -f nginx-rc.yml

Kubectl get rc

Kubectl get po -l app=nginx-app

REPLICA SETS

REPLICA SET - is a service - Ensures that a specified number of pods are always running at any time

If there are excess pods - it kills and vice versa

New pod is created and launched if there is a failure or deleted or terminated.

RS and pods are associated with “labels”

Always RS with count must be 1 then only pod is always available

Used for HA

REPLICA SET is Next gen replication controller

Replica set uses Set Based Selectors

Replication Controller uses Equality Based Selectors

Equality Based

Operators :

=, ==, !=

Examples

Environment = production

Tier !=frontend

Command line

Kubectl get pods -l environment=production

In manifest

Selector :

environment : production
tier : frontend

Supported area : services, Replication Controller

Set Based

Operators :

In, not in, exists

Examples

Environment in (production, qa, dev)
Tier not in (frontend, backend)

Command line

Kubectl get pods -l 'environment in (production)'
'tier not in [frontend, backend]'

In manifest

Selector :

```
matchExpressions :  
    -{key : environment, operator: in, values :  
[prod, qa]}  
    -{key : tier, operator: Not in, values :  
[frontend, backend]}
```

Supported area : Job, Deployment, Replica set and
Daemon set

DEMO

apiVersion : apps/v1

```
kind : ReplicaSet
metadata :
  name : nginx-rs
spec :
  replicas : 3
  selector :
    matchLabels :
      app : nginx-app
    matchExpressions :
      - {key: tier, operator: In, values : [frontend]}

template :
  metadata :
    name : nginx-pod
    labels :
      app : nginx-app
      tier : frontend
  spec :
    containers :
      - name : nginx-container
        image : nginx
        ports :
          - containerPort : 80
```

COMMANDS

Kubectl create -f nginx-rs.yml

Kubectl get pods

Kubectl get po -l tier=frontend

Kubectl get rs nginx-rs -o wide

Kubectl describe rs nginx-rs

Kubectl get po -o wide

Kubectl get nodes

Kubectl get po -o wide

Scaling UP

Kubectl scale rs nginx-rs --replicas=5

Kubectl get rc nginx-rs

Kubectl get po -o wide

Scale Down

Kubectl scale rs nginx-rs -- replicas=3

Kubectl get rs nginx-rs

Kubectl get po -o wide

Deletion

Kubectl delete -f nginx-rs.yml

Kubectl get rs

Kubectl get po -l app=nginx-app

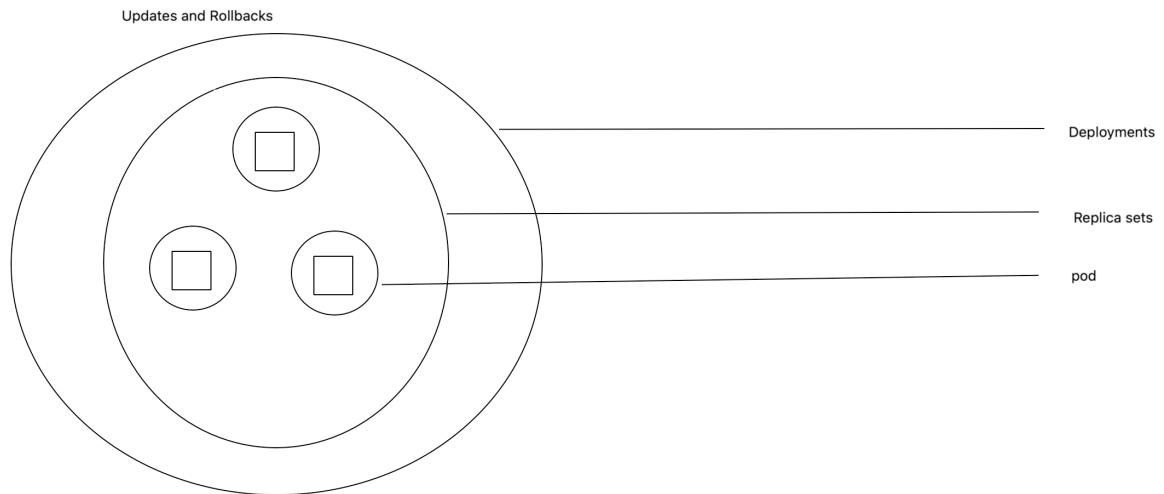
DEPLOYMENTS

SCENARIO

UPGRADE AN APPLICATION FROM V1 TO V2

Points to be addressed

1. Upgrade with zero downtime.
2. Upgrade sequentially - one after the other
3. Pause and resume upgrade process
4. Rollback upgrade to previous stable release.



Features

1. Multiple Replicas
2. Upgrade
3. Rollback
4. Scale up and down
5. Pause and resume

Deployment Algorithms

1. Recreate

2. Rolling Update (Ramped or Incremental)
3. Canary
4. Blue/Green OR Red/Black
5. A/B Deployment

DEMO

```
# Deployment
```

```
apiVersion : apps/v1
kind : Deployment
metadata :
  name : nginx-deploy
  labels :
    app : nginx-app
spec :
  replicas : 3
  selector :
    matchLabels :
      app : nginx-app
  template :
    metadata :
      labels :
        app : nginx-app
  spec :
    containers :
      - name : nginx-container
        image : nginx:1.7.9
        ports :
          - containerPort : 80
```

COMMANDS

Kubectl create -f deploy.yml

Kubectl get deploy -l app=nginx-app

Kubectl get rs -l app=nginx-app

Kubectl get po -l app=nginx-app

Kubectl describe deploy nginx-deploy

DEPLOYMENT - UPDATE

kubectl set image deploy nginx-deploy nginx-container=nginx:1.9.1

kubectl edit deploy nginx-deploy

kubectl rollout status deployment/nginx-deploy

kubectl get deploy

DEPLOYMENT - ROLLBACK

```
kubectl set image deploy nginx-deploy nginx-  
container=nginx:1.91 --record
```

```
kubectl rollout status deployment/nginx-deploy
```

```
kubectl rollout history deployment/nginx-deploy
```

```
kubectl rollout undo deployment/nginx-deploy
```

```
kubectl rollout status deployment/nginx-deploy
```

Scale up - deployment

```
kubectl scale deployment nginx-deploy --replicas=5
```

```
kubectl get deploy
```

```
kubectl get po
```

Scale Down - deployment

```
kubectl scale deployment nginx-deploy --replicas=1
```

```
kubectl get deploy
```

```
kubectl get po
```

DELETE - deployment

```
kubectl delete -f nginx-deploy.yml
```

```
kubectl get po -l app=nginx-app
```

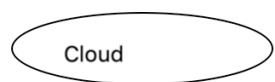
SERVICES

Requirement :

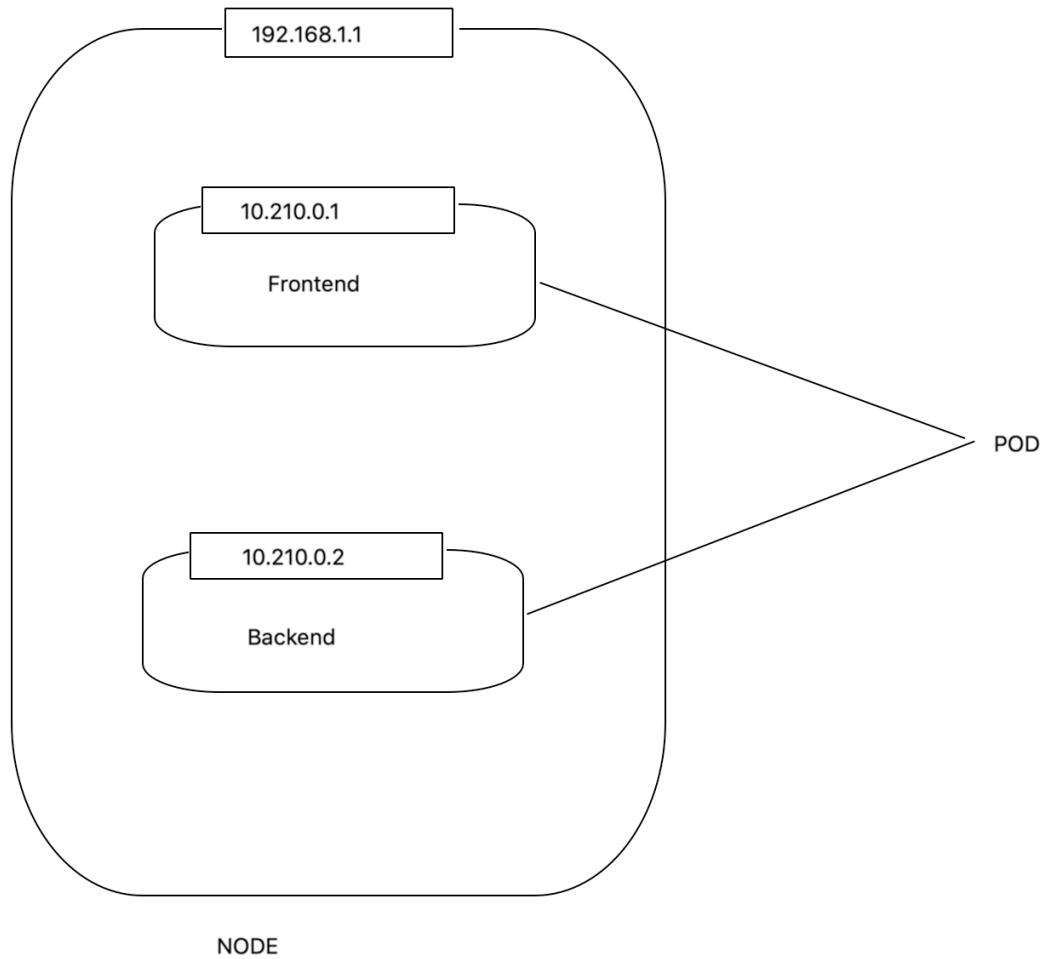
Deploy web application

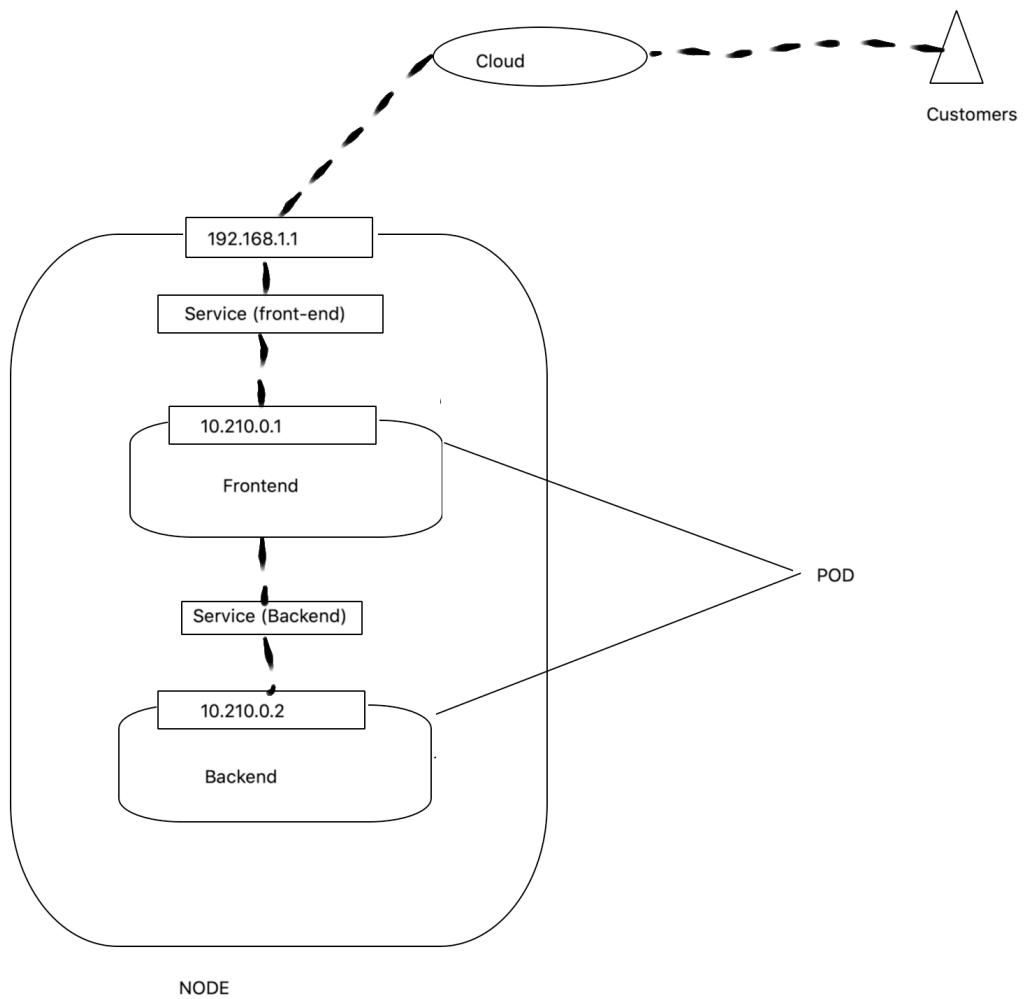
Steps :

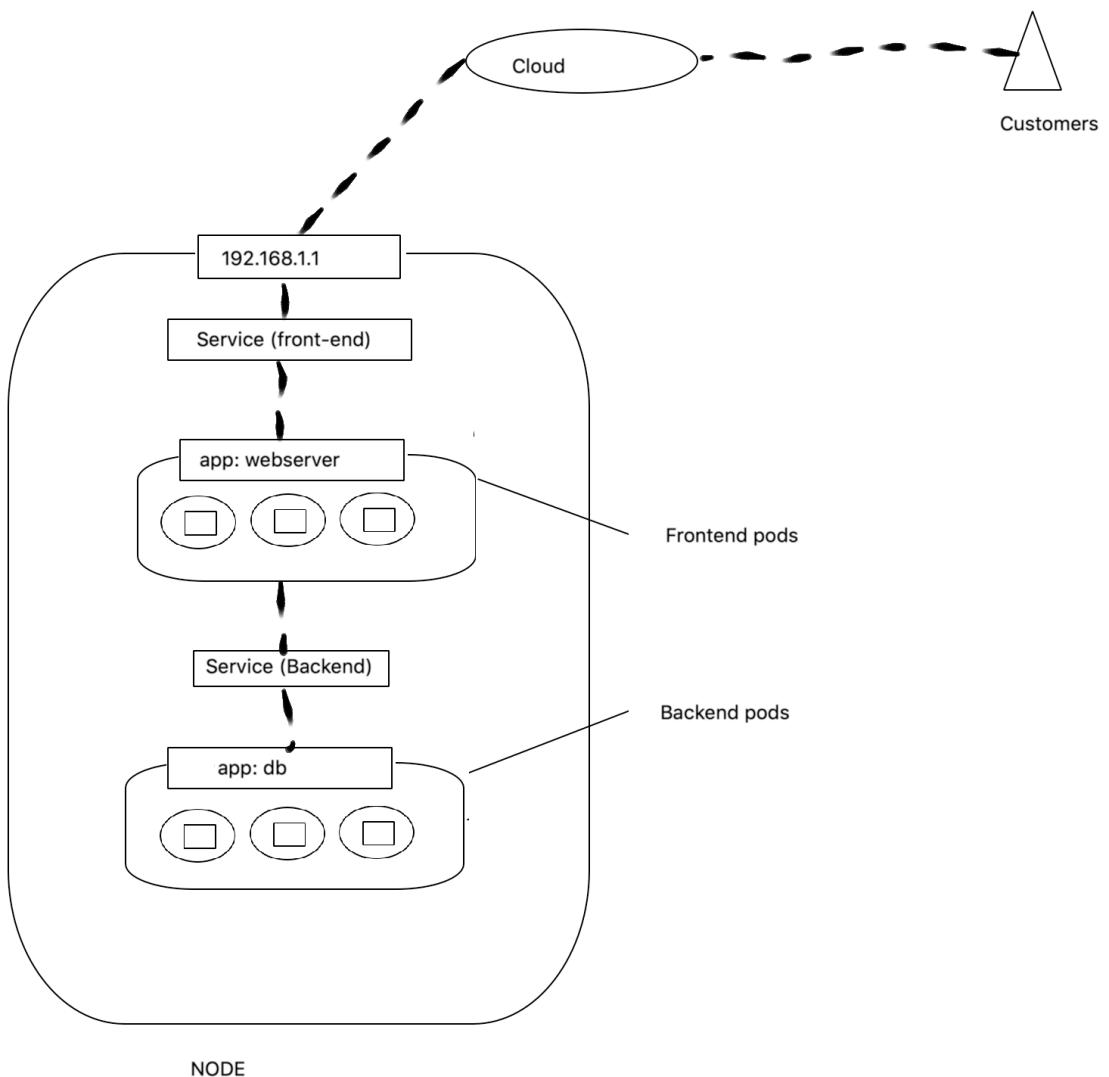
1. Front end application is a web app - how to expose to outside world ?
2. How to connect to backend service ?
3. When Pod dies How to resolve IP changes when system creates new POD?



Customers

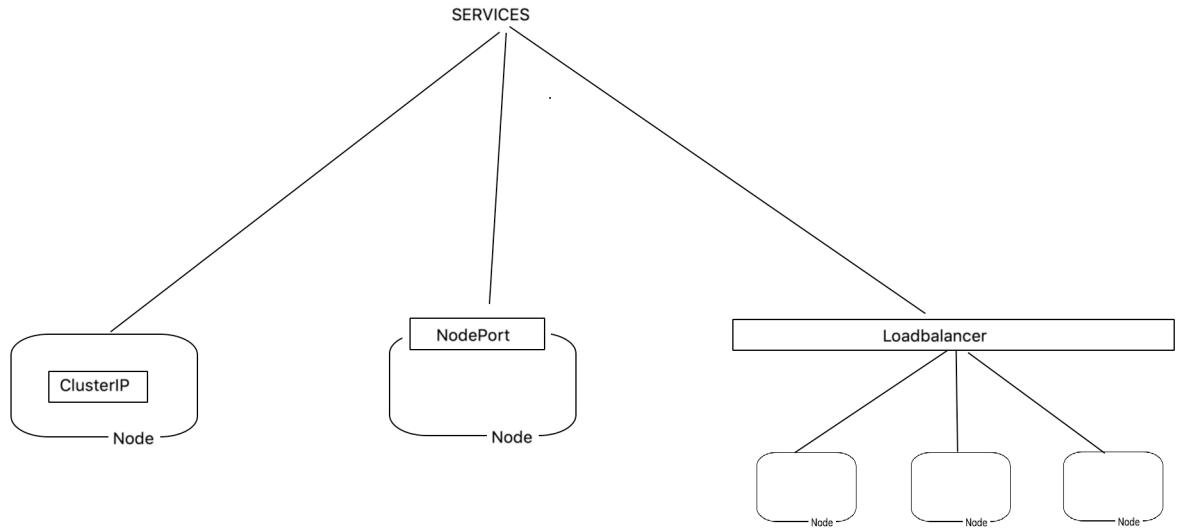




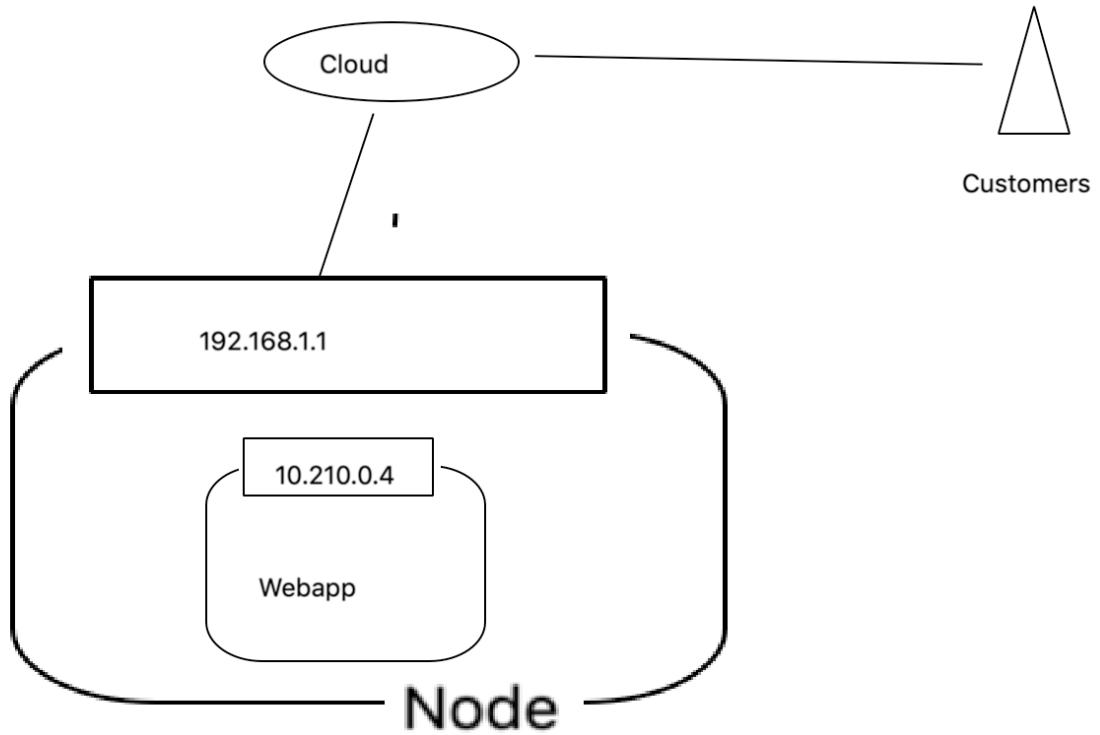


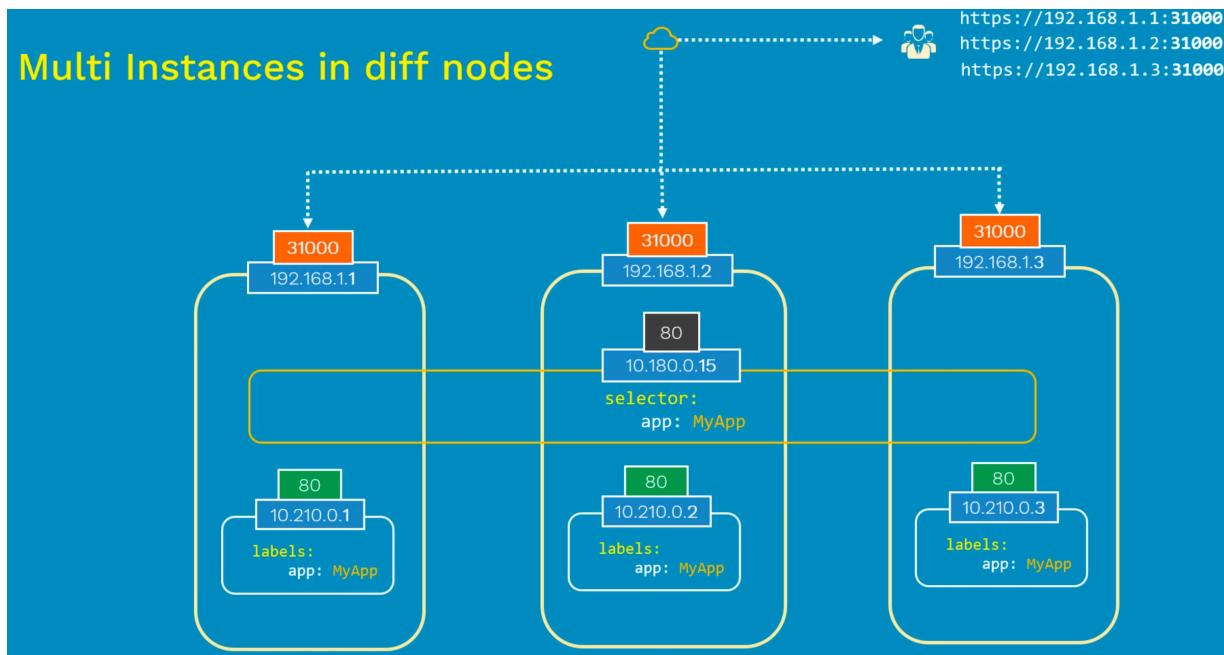
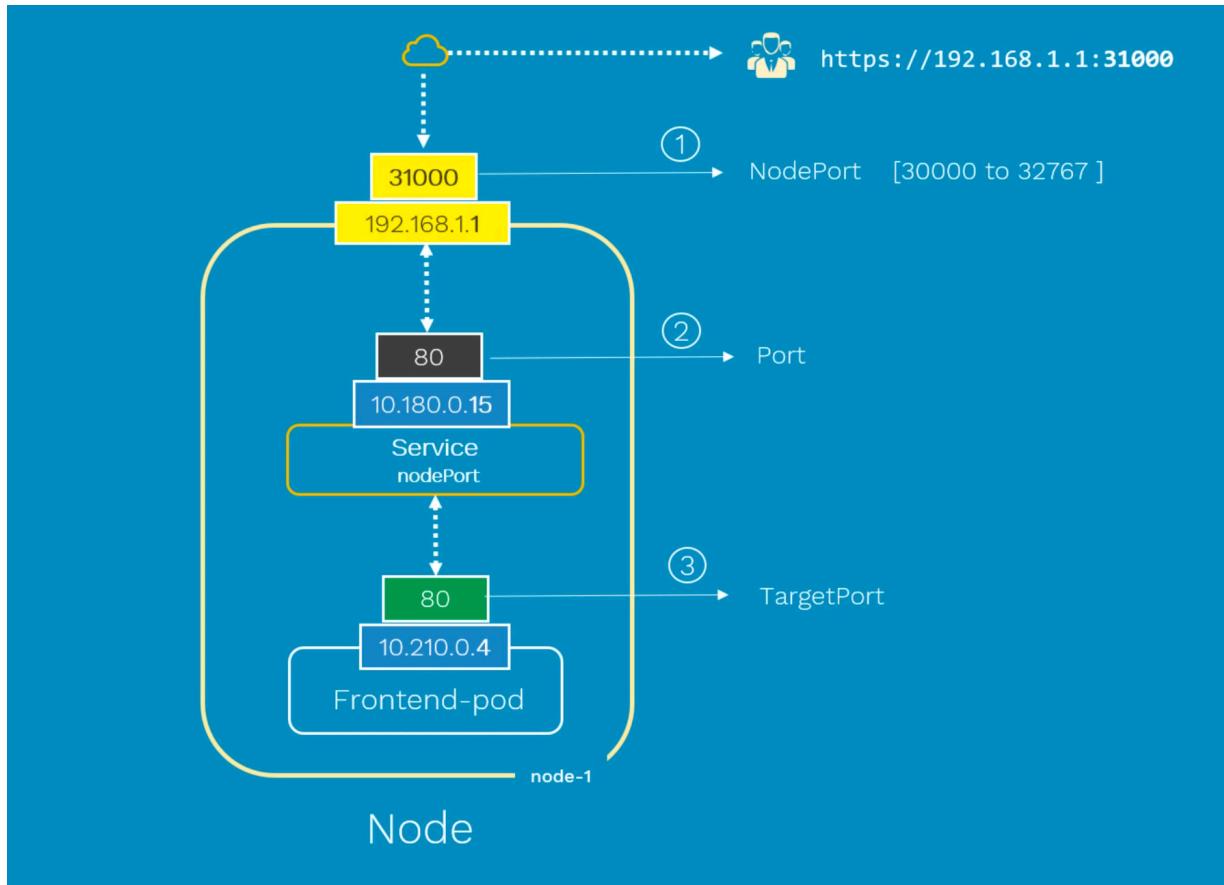
Types of Services

1. Cluster IP
2. NodePort
3. LoadBalancer



NODEPORT





DEMO

```
# Node Port Code
```

```
# nginx-deploy.yml
```

```
apiVersion : apps/v1
kind : Deployment
metadata :
  name : nginx-deployment
  labels :
    app : nginx-app
spec :
  replicas : 1
  selector :
    matchLabels :
      app : nginx-app
  template :
    metadata :
      labels :
        app : nginx-app
    spec :
      containers :
        - name : nginx-containers
          image : nginx:1.7.9
```

```
ports :  
  - containerPort : 80
```

```
# Node Port Service
```

```
# nginx-svc.yml
```

```
apiVersion : v1  
kind : Service  
metadata :  
  name : my-service  
  labels :  
    app : nginx-app  
spec :  
  selector :  
    app : nginx-app  
  type : NodePort  
  ports :  
    - nodePort : 31000  
      port : 80  
      targetPort : 80
```

Commands

```
kubectl create -f nginx-deploy.yml
```

```
kubectl create -f nginx-svc.yml
```

```
kubectl get service -l app=nginx-app
```

```
kubectl get po -o wide
```

```
kubectl describe svc my-service
```

(Only for google cloud) - gcloud compute instances list

Z

```
$. kubectl get po -o wide
```

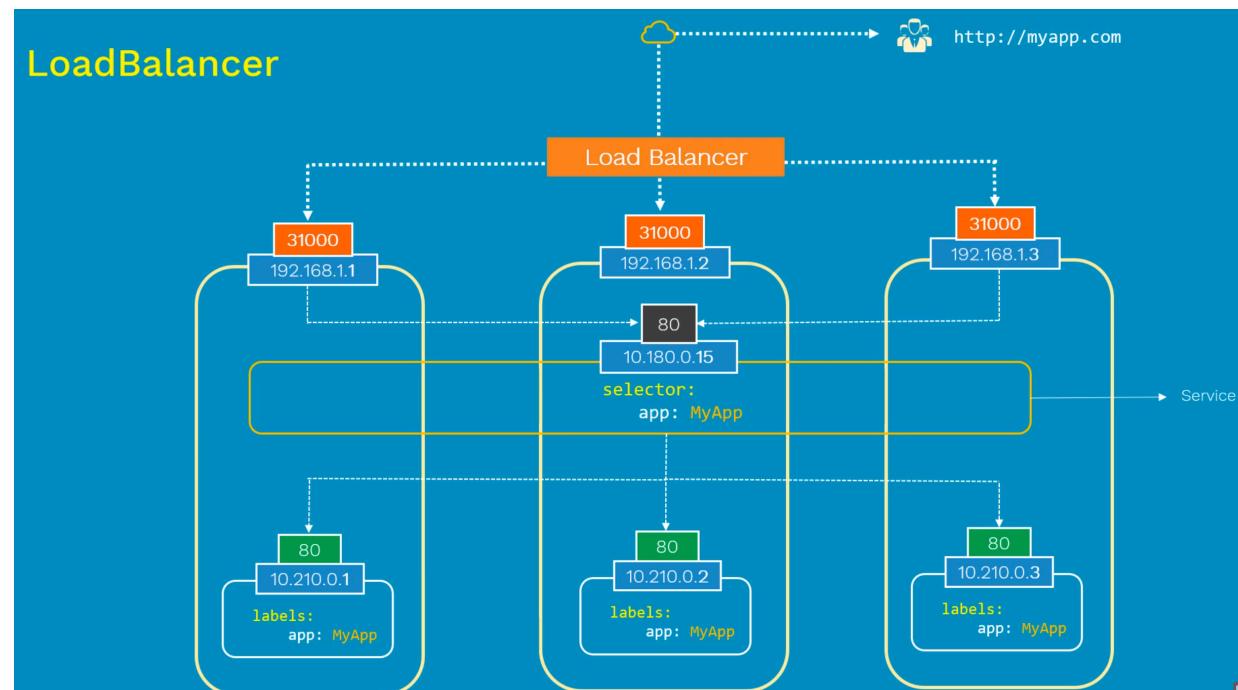
NAME	READY	STATUS	RESTARTS
AGE	IP	NODE	NOMINATED NODE
READINESS	GATES		
nginx-deployment-786888db66-xk7xx	1/1		
Running	0	2m28s	10.244.1.3
<none>	<none>		node01

```
controlplane $ curl http://10.244.1.3:80
```

Able to see the html page from container

```
kubectl delete svc my-service
```

LOADBALANCER



Demo

```
# nginx-deploy.yml
```

```
apiVersion : apps/v1
kind : Deployment
metadata :
  name : nginx-deployment
  labels :
    app : nginx-app
spec :
  replicas : 1
  selector :
    matchLabels :
      app : nginx-app
  template :
    metadata :
      labels :
        app : nginx-app
    spec :
      containers :
        - name : nginx-containers
          image : nginx:1.7.9
          ports :
            - containerPort : 80
```

You can try below **command** instead of writing service file

```
kubectl create -f nginx-deploy.yml
```

```
kubectl expose deploy nginx-deployment —name=nginx-  
service —port=80 —target-port=80 —  
type=LoadBalancer
```

```
# Load Balancer Service
```

```
apiVersion : v1  
kind : Service  
metadata :  
  name : my-service  
  labels :  
    app : nginx-app  
spec :  
  selector :  
    app : nginx-app  
  type : LoadBalancer  
  ports :  
    - nodePort : 31000  
      port : 80  
      targetPort : 80
```

COMMANDS

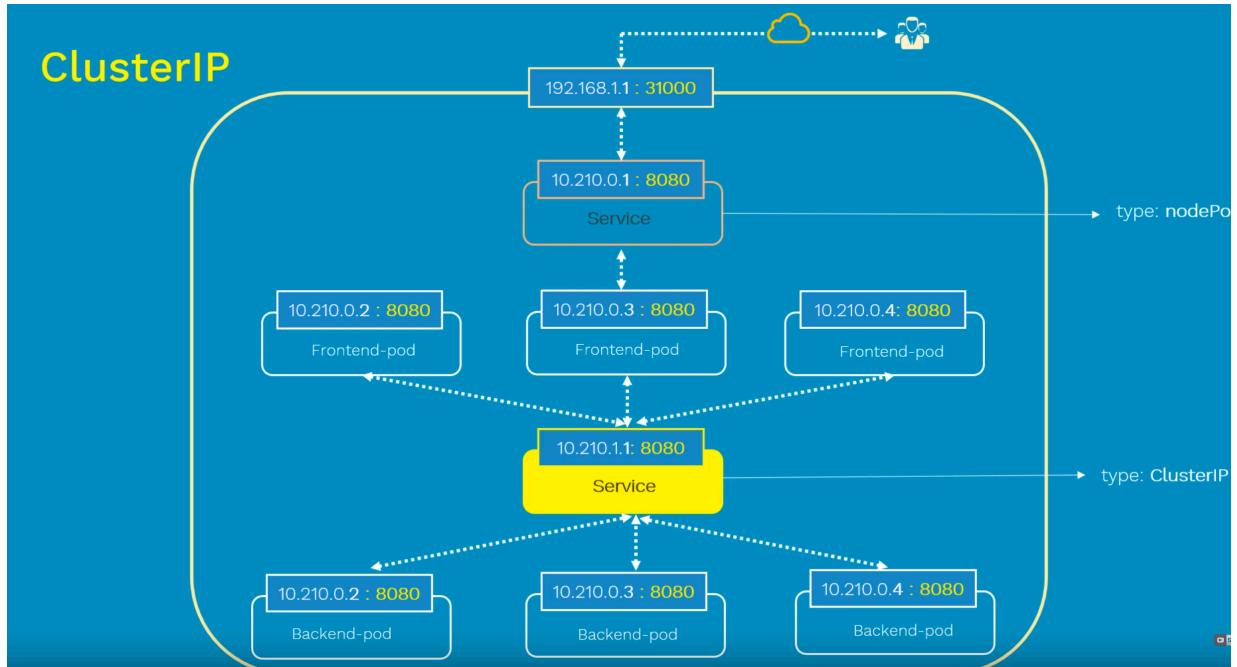
```
kubectl create -f nginx-lb-svc.yml
```

```
kubectl describe service my-service
```

```
kubectl describe service my-service | grep Load
```

```
kubectl delete svc my-service
```

CLUSTERIP



Kubernetes Volumes

Storage Types

1. Block
2. NAS
3. Object Storage
4. Google Cloud Disk
5. Cloud disks
6. AWS disks
7. Azure disk
8. FC - Fibre channel

9. RBP (ceph clock disk)
10. iSCi
11. GlusterFS
12. Azure File
13. VSphere volume
14. HostPath
15. ParkWorx Volumes
16. ScaleIO Volumes
17. Storage OS
18. Quobyte Volumes
19. Cinder (Openstack block Storage)
20. Flex Volume

For consistent storages - Kubernetes solution is Persistent Volumes (PV & PVC)

PV - Persistent Volumes

PVC - Persistent Volume Claims

Categories of Storage

1. Block Storage
2. NFS
3. Object Storage

Persistent volumes

Abstract details of how storage is provided from and how it is consumed.

PV - place of storage in cluster

PVC - request for storage

Life Cycle of PV

1. Provisioning
2. Binding
3. Using
4. Reclaiming

Provisioning

2 type

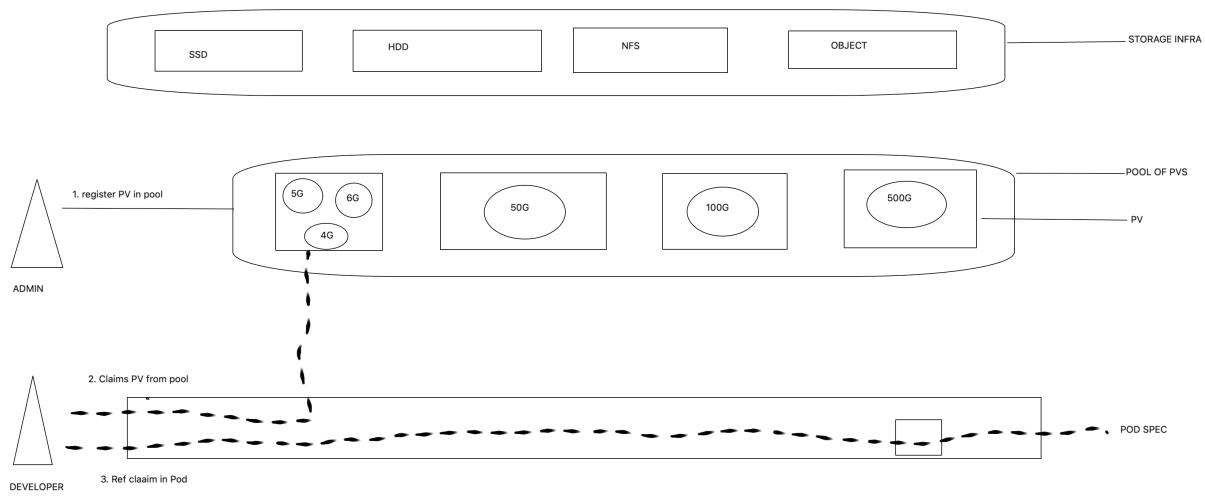
Static

Dynamic

Static - PV need to be created before PVC

Dynamic - PV is created at same time of PVC

STATIC PERSISTENT VOLUME



DYNAMIC PERSISTENT VOLUME



emptyDir

Creates empty directory, first created when a pod is assigned to a Node.

Stay as long as pod is running

Once pod is removed from a node, emptyDir is deleted forever

Example : Temp Folder

Used like Temporary space

DEMO

```
# volume example
# emptyDir demo

apiVersion : v1
kind : pod
metadata :
  name : test-ed
spec :
  containers :
    - image : nginx
      name : test-container
    volumeMounts :
      - name : cache-volume
        mountPath : /cache
  volumes :
    - name : cache-volume
      emptyDir : {}
```

Commands

Kubectl create -f test-ed.yml

Kubectl get po

kubectl exec test-ed df /cache

kubectl describe pod test-ed

Kubectl delete po test-ed

HOSTPATH

1. Mounts a file or folder from the host nodes file system in to the pod
2. Remains even after the pod is terminated
3. Similar to docker volumes
4. Use when required
5. If you have host issues then there will be problem in hostpath

DEMO

```
# volume example 2
# hostpath demo

apiVersion : v1
kind : Pod
metadata :
  name : redis-hostpath
spec :
  containers :
    - image : redis
      name : redis-container
    volumeMounts :
      - name : test-vol
        mountPath : /test-mnt
  volumes :
    - name : test-vol
      hostPath :
        path : /test-vol
```

Commands

```
kubectl create -f hostpath-volume.yml
```

```
kubectl get po
```

```
kubectl exec redis-hostpath df /test-mnt
```

```
kubectl describe pod redis-hostpath
```

AFTER SUCCESSFUL RUN

PODS ARE CREATED IN NODE 01
SO FIND FOLDER CALLED /test-vol

Mounted to .test-mnt as mentioned in manifest

STEP 2

Node 1 prompt

cd /test-vol

echo "from host" > from-host.txt

cat from-host.txt

On Control-pane (master)

kubectl exec redis-hostpath cat /test-mnt/from-host.txt

Step 3

On Control-pane (master)

```
master $ kubectl exec redis-hostpath -it -- /bin/sh  
  
# cd /test-mnt  
  
# ls  
from-host.txt  
  
# echo "from POD" > from-pod.txt  
  
# echo "from POD" > from-pod.txt  
  
# cat from-pod.txt  
from POD
```

Node 1 prompt

```
cd /test-vol
```

```
ls -l
```

Step 4

Kubectl delete po redis-hostpath

Kubectl get po

Ensure pods are deleted

(Observation)

In node 1

ls /test-vol

Folder still exists

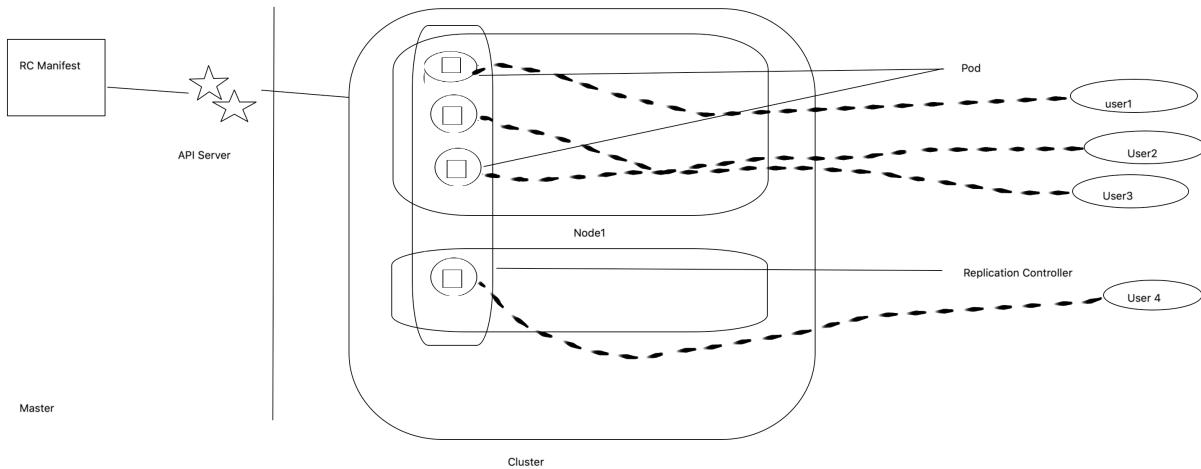
This is an advantage of host path

DaemonSets

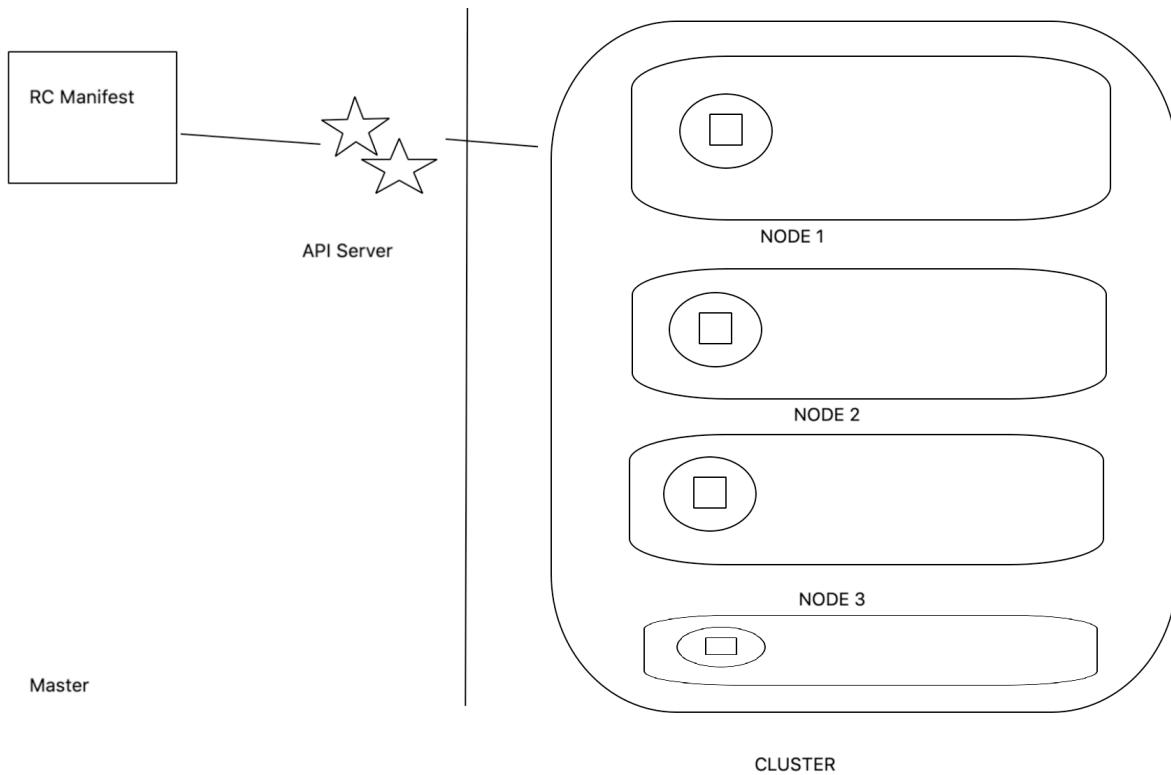
One Pod on each node

Subset of nodes inside cluster

ReplicaSet



DaemonSet



If pod to be increased then it uses or creates new nodes

If not required it will delete pod and respected node

Advantages

1. It ensures all nodes run a copy of a pod
2. As node is added to the cluster, pod is added
3. As node is removed from the cluster those pods are GC
4. Deleting daemon set will clean up the pods it created

Some Use cases of daemon

Node monitoring daemons - collects
Log collection daemons - fluentd

Kubectl log <instance name>

ELK and EFK stacks

DEMO

fluentd

```
apiVersion : apps/v1

kind : DaemonSet
metadata :
  name : fluentd-ds
spec :
  template :
    metadata :
      labels :
        name : fluentd
    spec :
      containers :
        - name : fluentd
          image : gcr.io/google-containers/fluentd-
elasticsearch:1.20
      selector :
        matchLabels :
          name : fluentd
```

Commands

kubectl get no

kubectl create -f daemonset.yml

kubectl get po -o wide

kubectl get ds

kubectl describe ds fluentd-ds

```
kubectl delete ds fluentd-ds
```

SECRETS

1. Small amount of sensitive data like passwords, tokens, key etc.
2. Reduce risk of exposing sensitive data
3. Created outside the pod
4. Stored inside etc DB
5. Size not more than 1 MB
6. Used in volumes or env variables
7. Sent only to target nodes

Can create secrets in two ways

1. Kubectl
2. Manually

DEMO

Kubectl way of creation

```
echo -n 'admin' > ./username.txt
```

```
echo -n '1f2dsfjkf82kd92f' > ./password.txt
```

```
ls
```

```
kubectl create secret generic db-user-pass --from-file=./username.txt --from-file=./password.txt
```

```
kubectl describe secrets db-user-pass
```

Manual

Step 1

```
echo -n 'admin' | base64
```

```
YWRtaW4=
```

```
echo -n 'efhejfhekjh9382938' | base64
```

```
ZWZoZWpmaGVramg5MzgyOTM4
```

Step 2

```
# secrets.yml

apiVersion : v1
kind : Secret
metadata :
  name : mysecret
type : Opaque
data :
  username : YWRtaW4=
  password : ZWZoZWpmaGVramg5MzgyOTM4
```

COMMANDS

```
kubectl create -f secrets.yml
```

```
kubectl get secrets mysecret -o yaml
```

(Find username and password)

```
echo 'YWRtaW4=' | base64 --decode
```

```
echo 'ZWZoZWpmaGVramg5MzgyOTM4' | base64 --
decode
```

STEP 3

CONSUME SECRETS IN PODS

1. Volumes
2. Env variables

```
# Use created secret in POD creation  
#mysecret-pod.yml
```

```
apiVersion : v1  
kind : Pod  
metadata :  
  name : mypod  
spec :  
  containers :  
    - name : mypod  
      image : redis  
  volumeMounts :  
    - name : foo  
      mountPath : "/etc/foo"  
      readOnly : true  
  volumes :  
    - name : foo  
      secret :
```

secretName : mysecret

Commands

kubectl create -f mysecret-pod.yml

kubectl get po

kubectl exec mypod ls /etc/foo

kubectl exec mypod cat /etc/foo/password

kubectl exec mypod cat /etc/foo/username