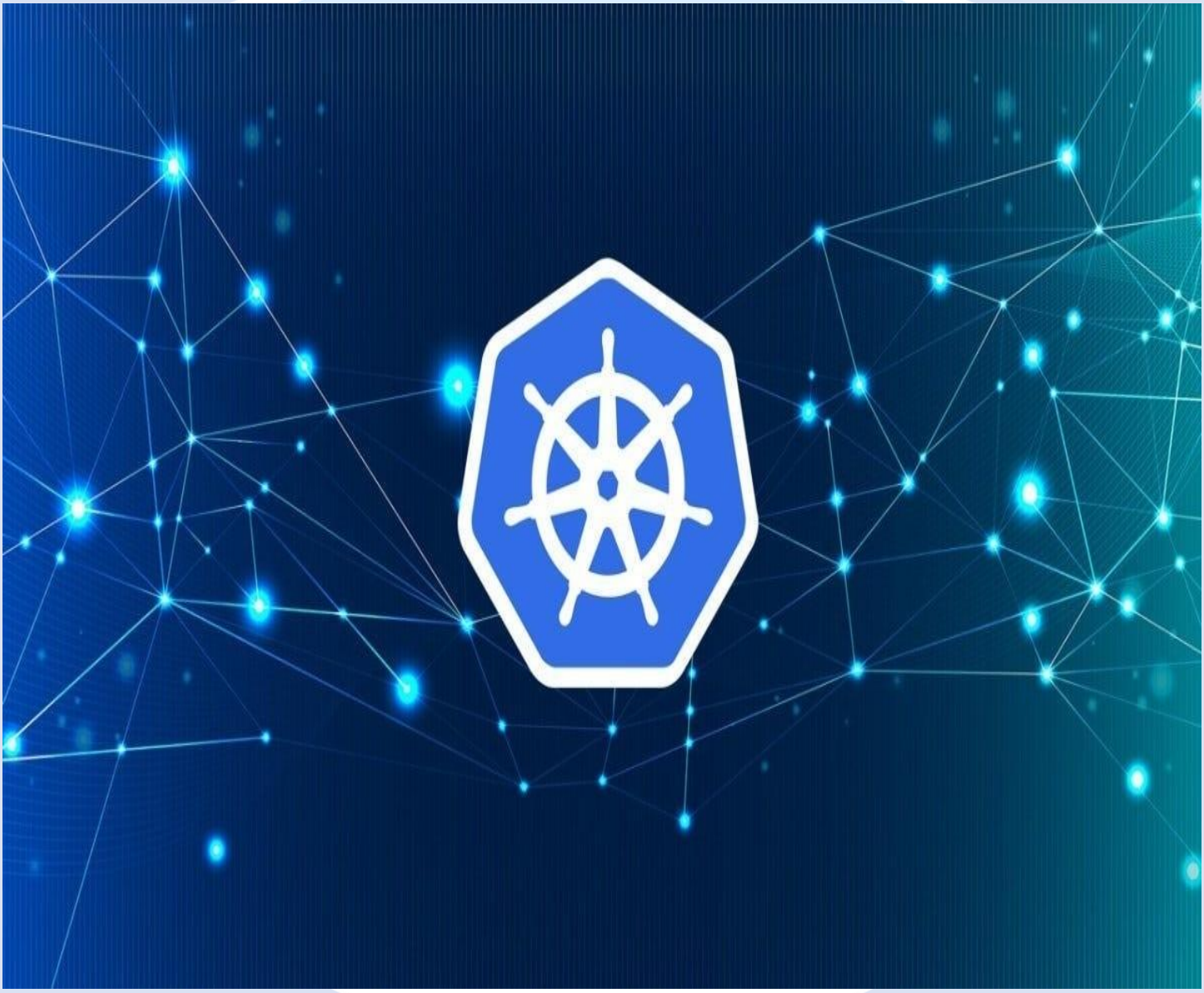
KUBERNETES



Tejas Pokale

INDEX

Introduction to Kubernetes

Kubernetes (K8s) is an open-source platform for automating the deployment, scaling, and management of containerized applications. Originally developed by Google and now maintained by the **Cloud Native Computing Foundation (CNCF)**, Kubernetes has become the industry standard for container orchestration.

Why Kubernetes?

Managing containers at scale is challenging. Kubernetes simplifies this by providing:

- **Automation**: Handles deployment and scaling.
- **Resilience**: Self-healing and failover capabilities.
- **Efficiency**: Optimizes resources for performance and cost.

Key Features of Kubernetes

1. **Automated Scheduling**: Places containers on the best-suited nodes.
2. **Self-Healing**: Restarts failed containers and reschedules them as needed.
3. **Load Balancing**: Ensures traffic is evenly distributed across containers.
4. **Storage Orchestration**: Connects containers to different storage solutions.
5. **Rolling Updates and Rollbacks**: Ensures smooth updates with minimal downtime.
6. **Declarative Configurations**: Uses YAML/JSON for easy management.

Core Components

- **Cluster**: A group of nodes, managed by Kubernetes, running your applications.
- **Pod**: The smallest deployable unit, which can contain one or more containers.
- **Node**: A worker machine that runs pods.
- **Service**: Ensures stable communication between pods and external users.
- **ConfigMap & Secret**: Stores configurations and sensitive data securely.
- **Ingress**: Manages external access to services.

Why Use Kubernetes?

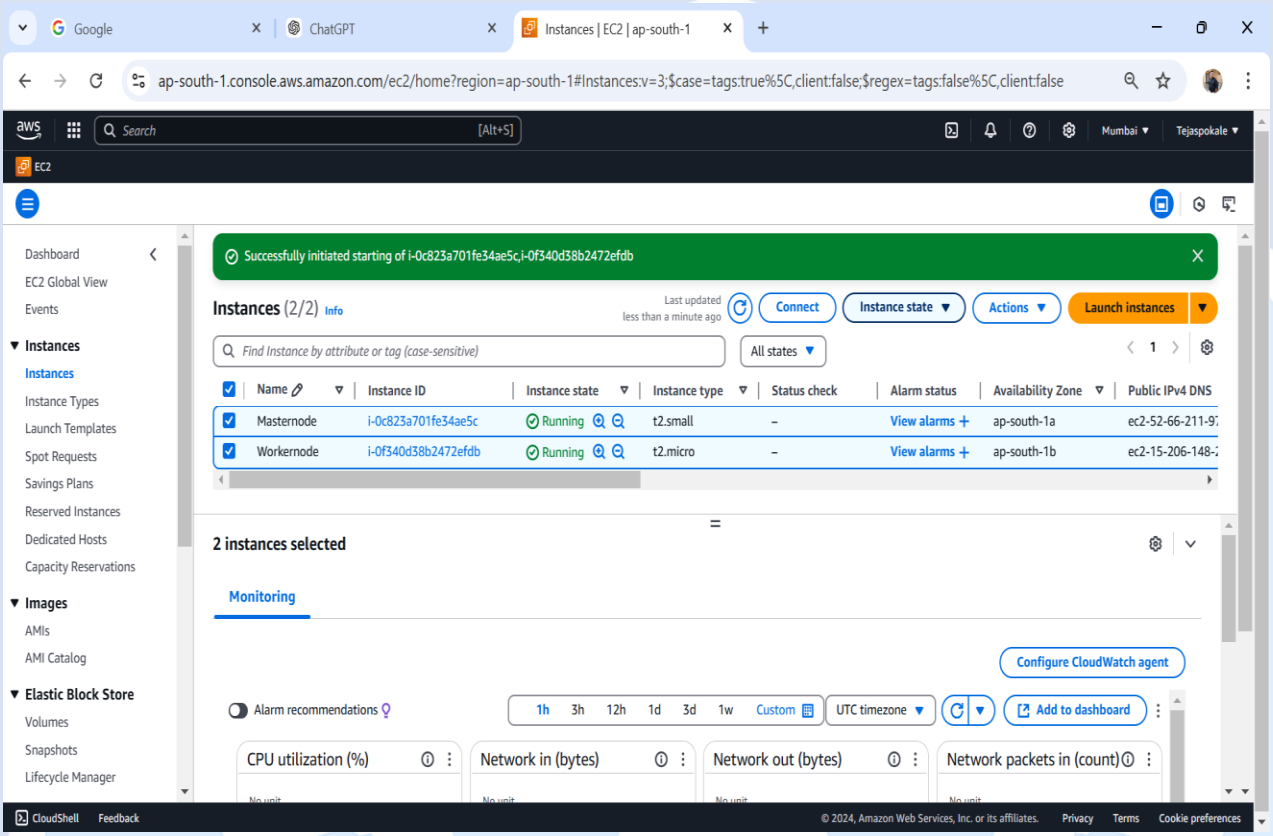
- **Portability**: Runs on any infrastructure—cloud, on-premises, or hybrid.
- **Scalability**: Easily handles growing workloads.
- **Resilience**: Provides high availability and fault tolerance.

Applications of Kubernetes

- Running **microservices** and distributed systems.
- Managing **cloud-native applications**.
- Scaling **batch processing** jobs.
- Automating **CI/CD pipelines** for faster delivery.

Kubernetes Installation on Ubuntu Linux

Step 1: Update and Install Required Dependencies



commands on **both master and worker nodes**:

sudo apt-get update -y

sudo apt-get install docker.io -y

sudo apt-get install -y apt-transport-https ca-certificates curl gnupg

sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

sudo chmod 644 /etc/apt/keyrings/kubernetes-apt-keyring.gpg

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo chmod 644 /etc/apt/sources.list.d/kubernetes.list

sudo apt-get install -y kubectl kubeadm kubelet

Step 2: Setup Master Node

Run the following commands only on the **Master Node**:

sudo kubeadm init --ignore-preflight-errors=all

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

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

kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.26.0/manifests/calico.yaml

kubeadm token create --print-join-command

- The kubeadm token create --print-join-command command generates the token and join command for worker nodes.
- Copy the output for use in the next step.

Step 3: Open Required Port

- Open **port 6443** on both master and worker nodes to ensure communication between them.

Step 4: Join Worker Nodes

Run the **kubeadm join** command generated on the master node on each **Worker Node**:

sudo kubeadm join <master_ip>:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash>

Step 5: Verify Cluster Setup

Run the following on the **Master Node** to check the status of the nodes:

kubectl get nodes

This command should display all nodes (master and workers) with their status.

```
ubuntu@ip-172-31-46-144:~$ kubectl get node
NAME               STATUS   ROLES           AGE   VERSION
ip-172-31-0-244    Ready    <none>          30h   v1.30.7
ip-172-31-46-144   Ready    control-plane   30h   v1.30.7
ubuntu@ip-172-31-46-144:~$
```

Check pods

Kubectl get pods

```
ubuntu@ip-172-31-46-144:~$ kubectl get pods
NAME       READY   STATUS    RESTARTS   AGE
nginxpod   1/1     Running   0          15m
ubuntu@ip-172-31-46-144:~$
```

To check master node pods

Kubectl get pods -n kube-system

```
ubuntu@ip-172-31-46-144:~$ kubectl get pods -n kube-system
NAME                                       READY   STATUS             RESTARTS         AGE
calico-kube-controllers-6cdb97b867-4rvc2   1/1     Running            1 (23m ago)      30h
calico-node-grzcj                          0/1     CrashLoopBackOff   14 (53s ago)     30h
calico-node-vzr9d                          0/1     Running            10 (9m6s ago)    30h
coredns-55cb58b774-6x772                   1/1     Running            1 (23m ago)      30h
coredns-55cb58b774-vbt6s                   1/1     Running            1 (24m ago)      30h
etcd-ip-172-31-46-144                      1/1     Running            1 (23m ago)      30h
kube-apiserver-ip-172-31-46-144            1/1     Running            1 (23m ago)      30h
kube-controller-manager-ip-172-31-46-144   1/1     Running            1 (23m ago)      30h
kube-proxy-k5fw5                           0/1     CrashLoopBackOff   11 (2m21s ago)   30h
kube-proxy-ppmk4                           0/1     CrashLoopBackOff   14 (64s ago)     30h
kube-scheduler-ip-172-31-46-144            1/1     Running            1 (23m ago)      30h
ubuntu@ip-172-31-46-144:~$
```

Create a directory

mkdir nginxpod

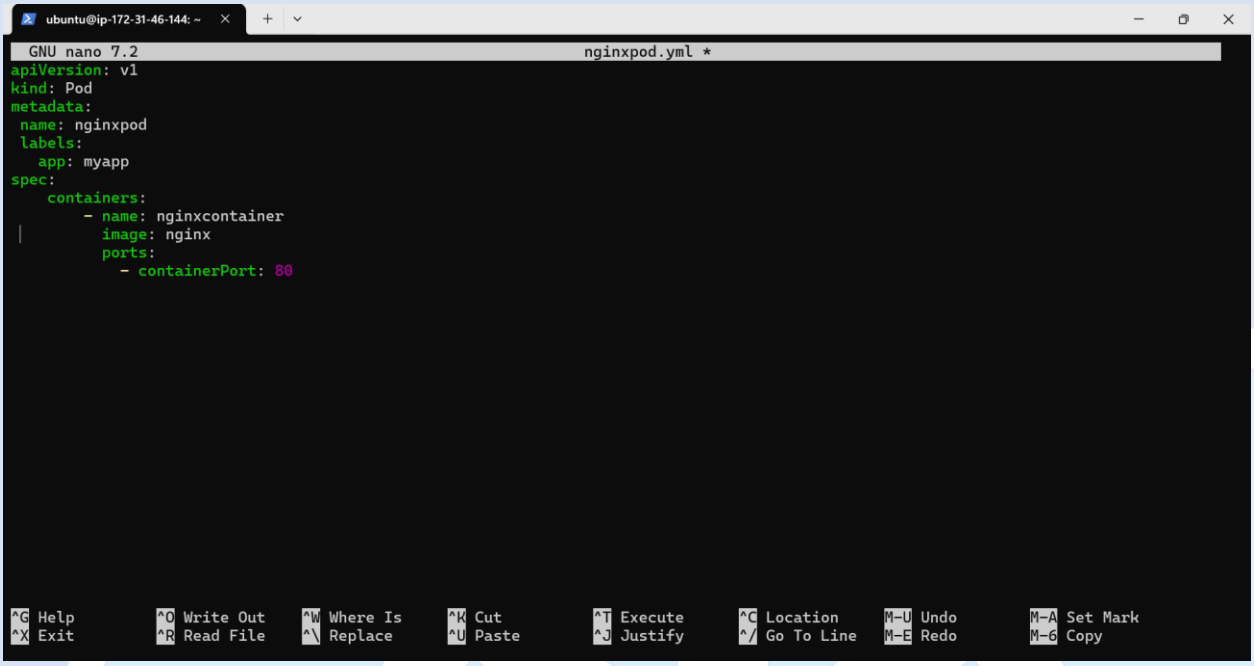
```
ubuntu@ip-172-31-46-144:~$ mkdir nginxpod
ubuntu@ip-172-31-46-144:~$ ls
nginxpod
ubuntu@ip-172-31-46-144:~$
```

Create .yml file

Nano nginxpod.yml

```
ubuntu@ip-172-31-46-144:~$ cd nginxpod/
ubuntu@ip-172-31-46-144:~/nginxpod$ nano nginxpod.yml
ubuntu@ip-172-31-46-144:~/nginxpod$
```

Nginxpod.yml file



Kubectl appy -f nginxpod.yml

```
ubuntu@ip-172-31-46-144:~$ cd nginxpod/
ubuntu@ip-172-31-46-144:~/nginxpod$ nano nginxpod.yml
ubuntu@ip-172-31-46-144:~/nginxpod$ kubectl apply -f nginxpod.yml
pod/nginxpod unchanged
ubuntu@ip-172-31-46-144:~/nginxpod$
```

Sudo crictl ps

```
ubuntu@ip-172-31-46-144:~/nginxpod$ sudo crictl ps
WARN[0000] runtime connect using default endpoints: [unix:///run/containerd/containerd.sock unix:///run/crio/crio.sock unix:///var/ru
n/cri-dockerd.sock]. As the default settings are now deprecated, you should set the endpoint instead.
WARN[0000] image connect using default endpoints: [unix:///run/containerd/containerd.sock unix:///run/crio/crio.sock unix:///var/run/
cri-dockerd.sock]. As the default settings are now deprecated, you should set the endpoint instead.
CONTAINER           IMAGE               CREATED             STATE               NAME                      ATTEMPT             POD ID
          POD
57c4430c71bb5       44f52c09decec       3 minutes ago       Running             calico-node               16                  ba66574
e1c0da       calico-node-grzcj
5655e80c6dfe7       45ae357729e3a       37 minutes ago      Running             calico-kube-controllers   1                   fd4bc86
ccea48       calico-kube-controllers-6cdb97b867-4rvc2
dd12849018199       c69fa2e9cbf5f       37 minutes ago      Running             coredns                   1                   e41953f
4a2118       coredns-55cb58b774-6x772
c2c793fd75638       2e96e5913fc06       37 minutes ago      Running             etcd                      1                   f21610f
b01746       etcd-ip-172-31-46-144
4ad3b7518002e       4511f99369d0f       37 minutes ago      Running             kube-scheduler            1                   a10e5ec
4c9148       kube-scheduler-ip-172-31-46-144
6de03f8b1a7e7       453fa29415db2       37 minutes ago      Running             kube-apiserver            1                   915e109
7684bd       kube-apiserver-ip-172-31-46-144
d6d561d7d5199       c0af7defe3a28       37 minutes ago      Running             kube-controller-manager   1                   d03e209
894fe9       kube-controller-manager-ip-172-31-46-144
ubuntu@ip-172-31-46-144:~/nginxpod$
```

Kubectl pet pod -o wide

```
ubuntu@ip-172-31-46-144:~/nginxpod$ kubectl get pod -o wide
NAME       READY   STATUS    RESTARTS   AGE   IP              NODE              NOMINATED NODE   READINESS GATES
nginxpod   1/1     Running   0          31m   192.168.193.131   ip-172-31-0-244   <none>           <none>
ubuntu@ip-172-31-46-144:~/nginxpod$
```

Kubectl describe pod (pod name)

```
ubuntu@ip-172-31-46-144: ~/
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-p2qj5 (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-p2qj5:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age                  From               Message
  ----    ------     ----                 ----               -------
  Normal  Scheduled  32m                  default-scheduler  Successfully assigned default/nginxpod to ip-172-31-0-244
  Normal  Pulling    32m                  kubelet            Pulling image "nginx"
  Normal  Pulled     32m                  kubelet            Successfully pulled image "nginx" in 8.064s (8.064s including waiting).
 Image size: 72099501 bytes.
  Normal  Created    32m                  kubelet            Created container nginxcontainer
  Normal  Started    32m                  kubelet            Started container nginxcontainer
  Normal  Killing    6m33s (x10 over 26m)  kubelet           Stopping container nginxcontainer
ubuntu@ip-172-31-46-144:~/nginxpod$
```

Kubectl logs nginxpod(pod name)

```
ubuntu@ip-172-31-46-144:~/nginxpod$ kubectl logs nginxpod
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/12/11 10:42:10 [notice] 1#1: using the "epoll" event method
2024/12/11 10:42:10 [notice] 1#1: nginx/1.27.3
2024/12/11 10:42:10 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/12/11 10:42:10 [notice] 1#1: OS: Linux 6.8.0-1018-aws
2024/12/11 10:42:10 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/12/11 10:42:10 [notice] 1#1: start worker processes
2024/12/11 10:42:10 [notice] 1#1: start worker process 28
ubuntu@ip-172-31-46-144:~/nginxpod$
```

Kubectl exec -it nginxpod /bin/bash

```
ubuntu@ip-172-31-46-144:~/nginxpod$ kubectl exec -it nginxpod /bin/bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
root@nginxpod:/# ls
bin   dev                docker-entrypoint.sh  home  lib64  mnt  proc  run   srv  tmp  var
boot  docker-entrypoint.d  etc                   lib   media  opt  root  sbin  sys  usr
root@nginxpod:/#
```

Kubernetes Services

A **Service** in Kubernetes allows stable communication between applications (Pods) and external users or systems by abstracting the underlying Pods' dynamic nature.

Types of Kubernetes Services

1. **ClusterIP (Default)**:
    - Accessible only within the cluster.
    - Used for internal communication between Pods and other cluster components.
2. **NodePort**:
    - Exposes the Service on a specific port of each cluster node.
    - Allows external access using <NodeIP>:<NodePort>.
    - Suitable for basic external connectivity.
3. **LoadBalancer**:
    - Integrates with cloud providers to create an external load balancer.
    - Automatically allocates an external IP for access.
    - Ideal for scaling and production-grade applications.
4. **ExternalName**:
    - Maps the Service to an external DNS name.
    - No traffic routing; directly resolves to the external resource.

Key Use Cases

- **ClusterIP**: Internal microservices communication.
- **NodePort**: Testing and exposing services during development.
- **LoadBalancer**: Production-level external traffic handling.
- **ExternalName**: Accessing external databases or APIs.

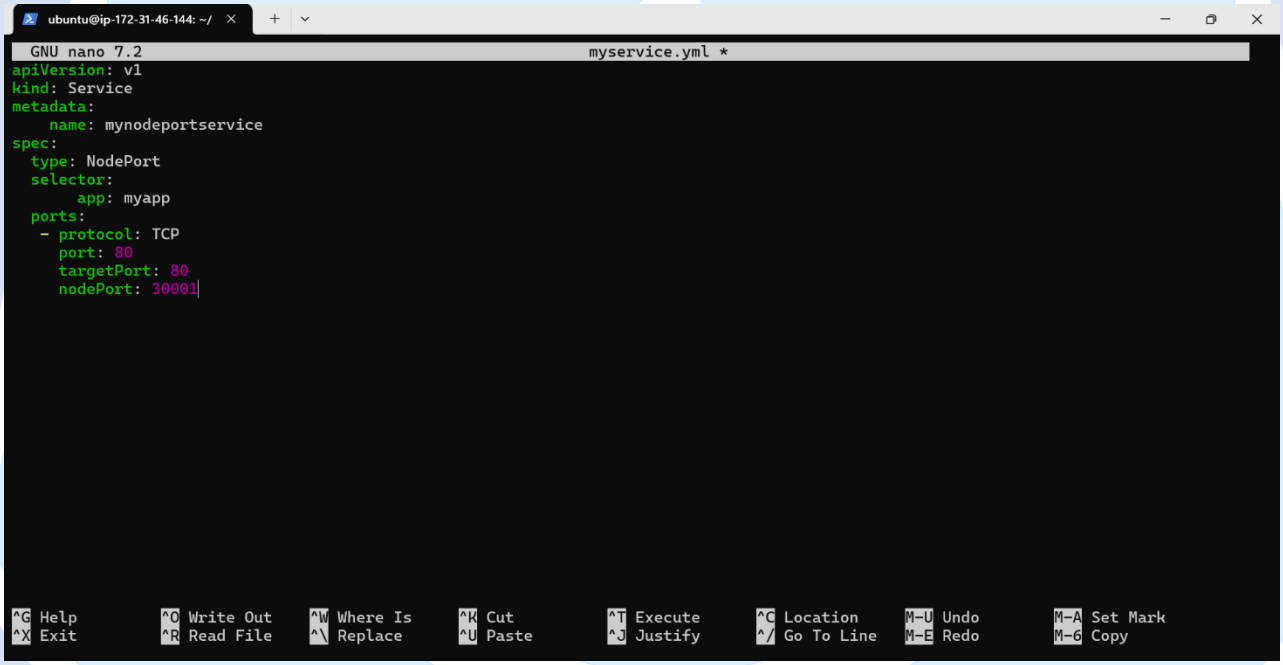
NodePort: for debugging and testing

create .yml file

nano myservice.yml

```
ubuntu@ip-172-31-46-144:~$ kubectl get node
NAME                STATUS   ROLES           AGE     VERSION
ip-172-31-0-244     Ready    <none>          2d23h   v1.30.7
ip-172-31-46-144    Ready    control-plane   2d23h   v1.30.7
ubuntu@ip-172-31-46-144:~$ ls
nginxpod
ubuntu@ip-172-31-46-144:~$ cd nginxpod/
ubuntu@ip-172-31-46-144:~/nginxpod$ ls
nginxpod.yml
ubuntu@ip-172-31-46-144:~/nginxpod$ nano myservice.yml
```

myservice.yml



Kubectl appy -f myservice.yml

```
ubuntu@ip-172-31-46-144:~/nginxpod$ kubectl apply -f myservice.yml
service/mynodeportservice created
ubuntu@ip-172-31-46-144:~/nginxpod$
```

Kubectl get services

```
ubuntu@ip-172-31-46-144:~/nginxpod$ kubectl apply -f myservice.yml
service/mynodeportservice created
ubuntu@ip-172-31-46-144:~/nginxpod$ kubectl get services
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes          ClusterIP   10.96.0.1        <none>        443/TCP        3d
mynodeportservice   NodePort    10.111.225.200   <none>        80:30001/TCP   37s
ubuntu@ip-172-31-46-144:~/nginxpod$
```

Kubectl get pod

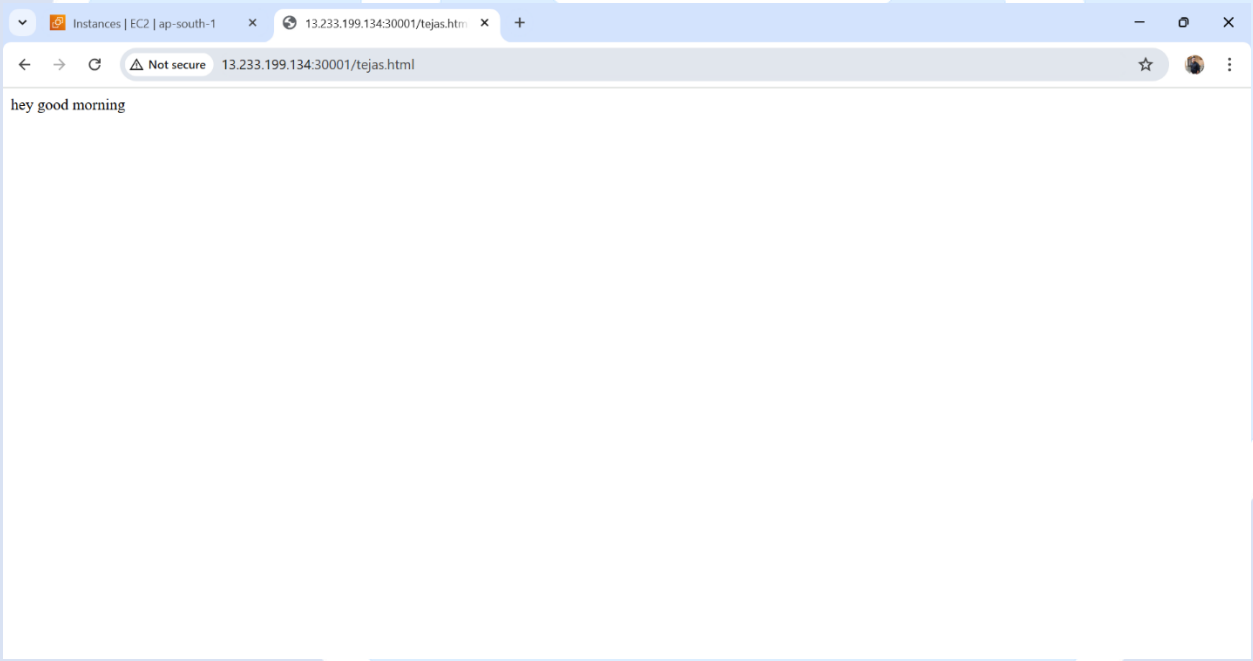
Kubectl get services -o wide

```
ubuntu@ip-172-31-46-144:~/nginxpod$ kubectl apply -f myservice.yml
service/mynodeportservice unchanged
ubuntu@ip-172-31-46-144:~/nginxpod$ kubectl get pod
NAME       READY   STATUS    RESTARTS   AGE
nginxpod   1/1     Running   0          45s
ubuntu@ip-172-31-46-144:~/nginxpod$ kubectl get services -o wide
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE     SELECTOR
kubernetes          ClusterIP   10.96.0.1        <none>        443/TCP        3d      <none>
mynodeportservice   NodePort    10.111.225.200   <none>        80:30001/TCP   7m41s   app=myapp
ubuntu@ip-172-31-46-144:~/nginxpod$
```

Create html page

```
root@nginxpod:/usr/share/nginx/html# touch tejas.html
root@nginxpod:/usr/share/nginx/html# echo "hey good morning">tejas.html
root@nginxpod:/usr/share/nginx/html# echo "hey good morning" > tejas.html
root@nginxpod:/usr/share/nginx/html#
```

Check on browse



LOAD BALANCER

Copy existing myservice.yml file to loadwalaservice.yml

```
ubuntu@ip-172-31-46-144:~/nginxpod$ ls
myservice.yml  nginxpod.yml
ubuntu@ip-172-31-46-144:~/nginxpod$ cp myservice.yml  loadwalaservice.yml
ubuntu@ip-172-31-46-144:~/nginxpod$ ls
loadwalaservice.yml  myservice.yml  nginxpod.yml
ubuntu@ip-172-31-46-144:~/nginxpod$
```

nano loadwalaservice.yml

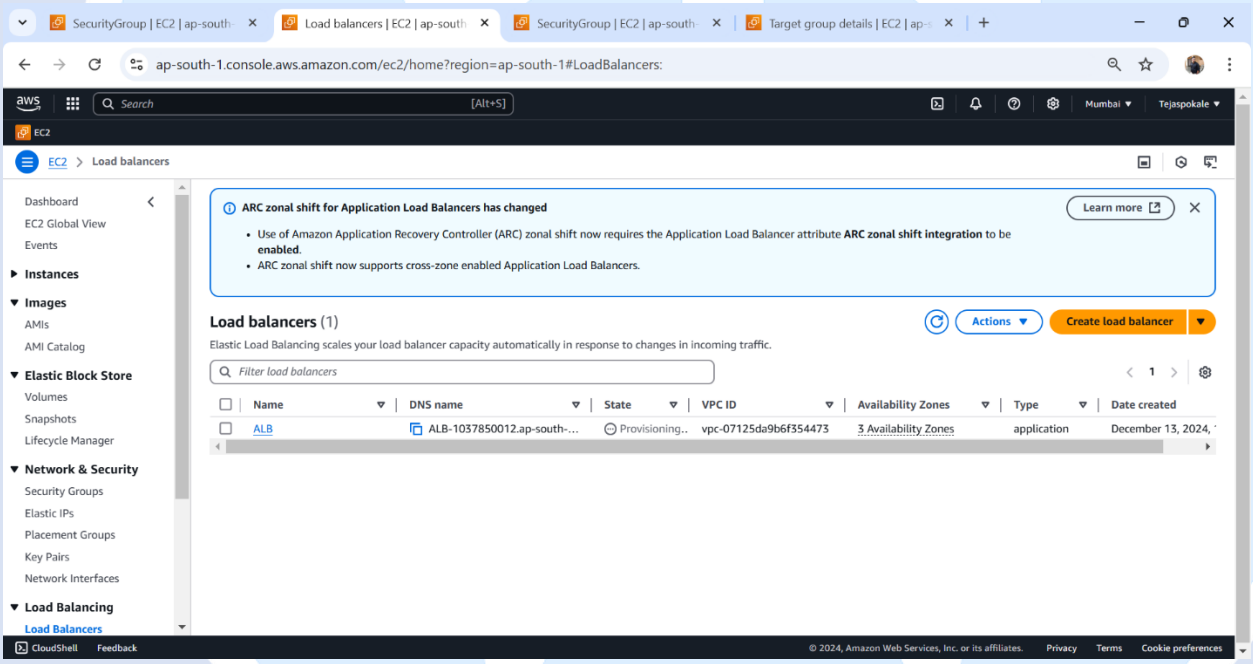
```
ubuntu@ip-172-31-46-144: ~/
 GNU nano 7.2                                  loadwalaservice.yml
apiVersion: v1
kind: Service
metadata:
    name: loadwalaservice
spec:
  type: LoadBalancer
  selector:
      app: myapp
  ports:
   - protocol: TCP
     port: 80
     targetPort: 80

                                [ Read 13 lines ]
^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location  M-U Undo    M-A Set Mark
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo    M-6 Copy
```

Kubectl apply -f loadwalaservice.yml

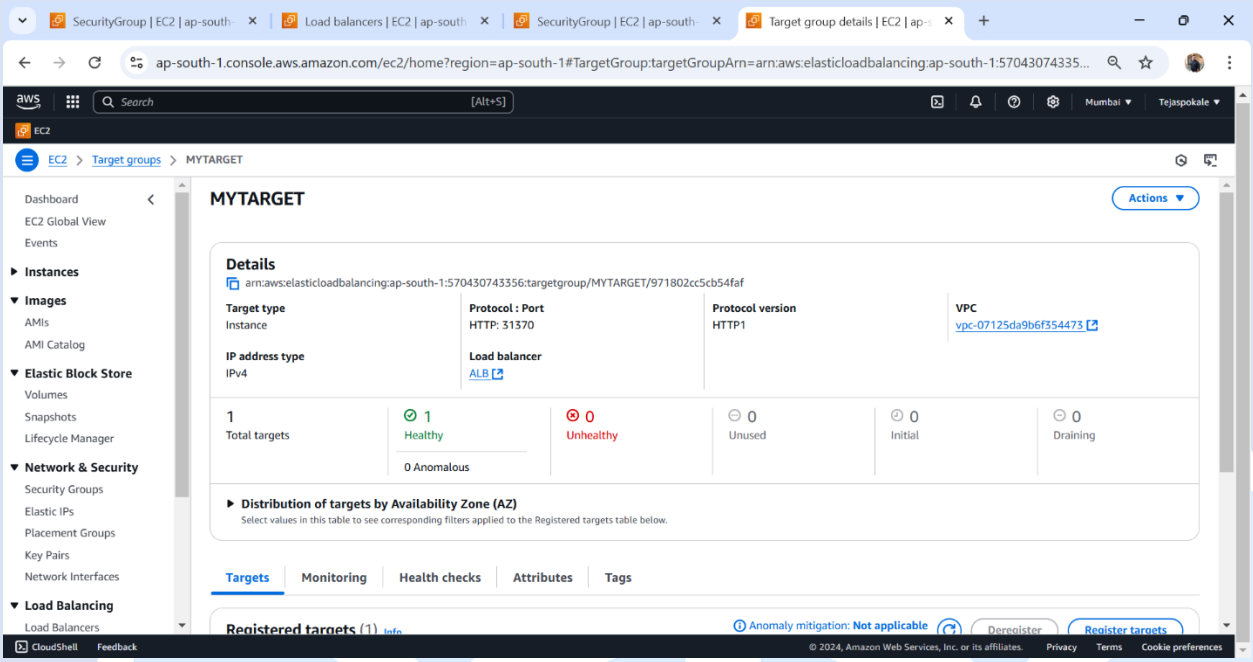
Kubectl get services

```
ubuntu@ip-172-31-46-144:~/nginxpod$ nano loadwalaservice.yml
ubuntu@ip-172-31-46-144:~/nginxpod$ kubectl apply -f loadwalaservice.yml
service/loadwalaservice unchanged
ubuntu@ip-172-31-46-144:~/nginxpod$ kubectl get services
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes          ClusterIP      10.96.0.1       <none>        443/TCP        3d
loadwalaservice     LoadBalancer   10.100.54.128   <pending>     80:31370/TCP   66s
mynodeportservice   NodePort       10.111.225.200  <none>        80:30001/TCP   33m
ubuntu@ip-172-31-46-144:~/nginxpod$
```

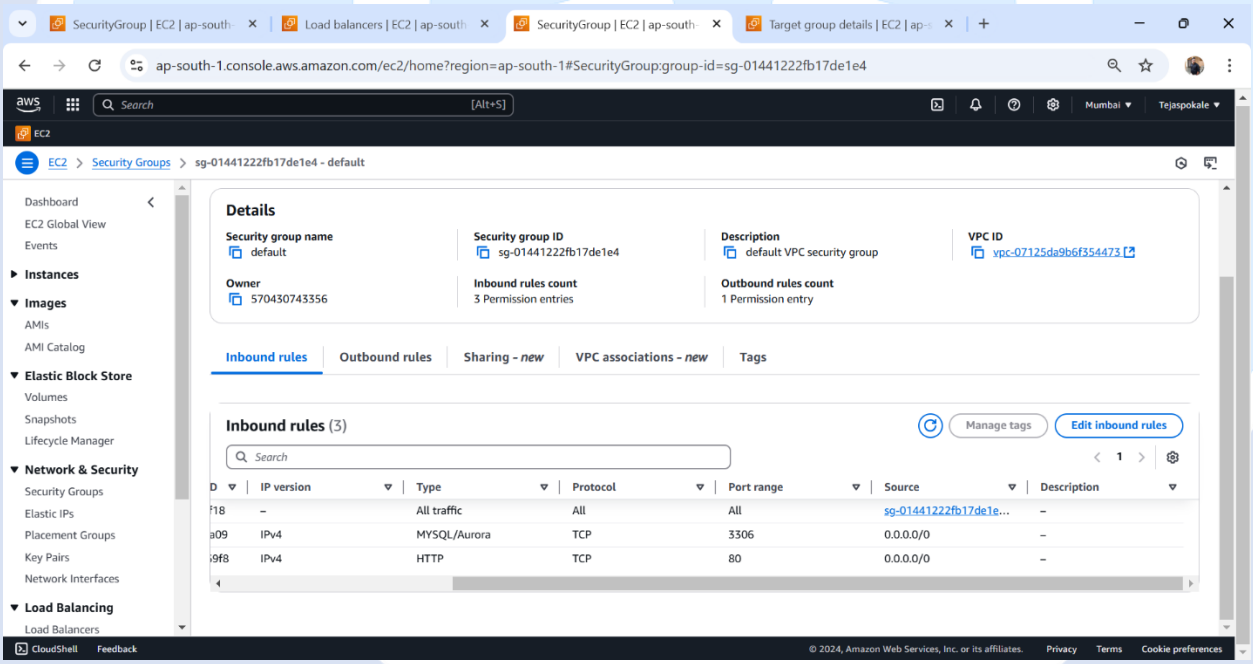
Launch the load balancer manually



Create the target group



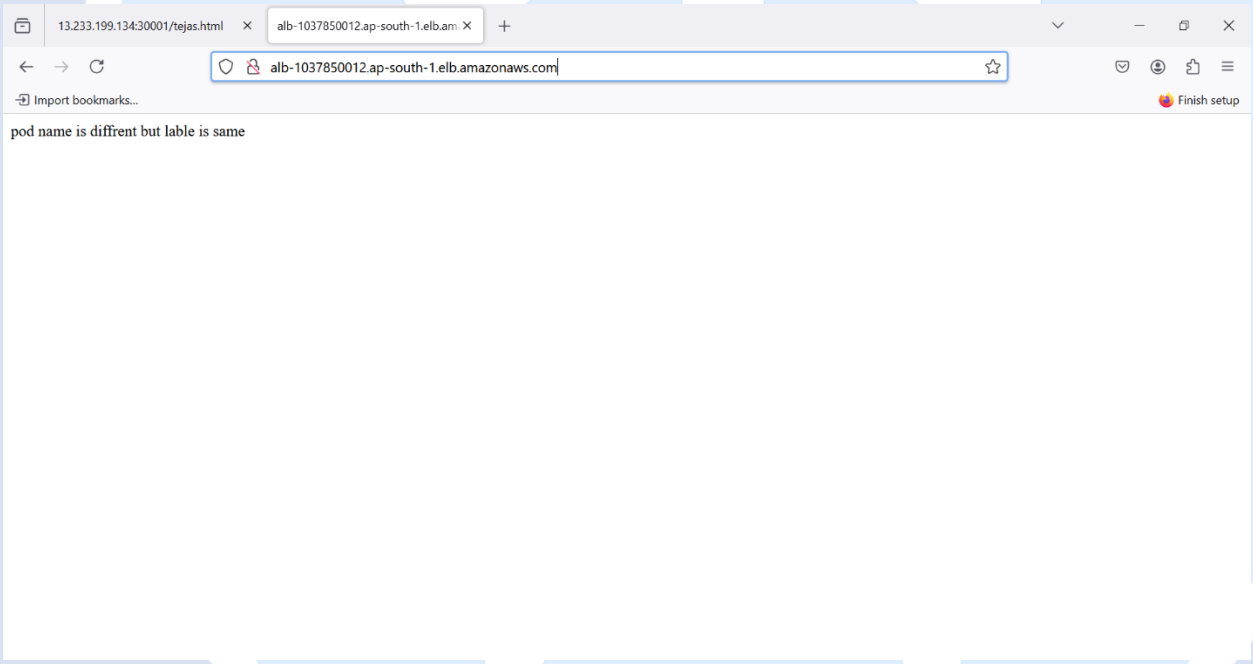
Edit security group



Create new index.htm page and show loadbalancing. pod name is different but labale is same

```
ubuntu@ip-172-31-46-144:~/nginxpod$ kubectl exec -it newngixpod -- /bin/bash
Error from server (NotFound): pods "newngixpod" not found
ubuntu@ip-172-31-46-144:~/nginxpod$ kubectl exec -it newnginxpod -- /bin/bash
root@newnginxpod:/# cd /usr/share/nginx/html/
root@newnginxpod:/usr/share/nginx/html# rm index.html
root@newnginxpod:/usr/share/nginx/html# touch index.html
root@newnginxpod:/usr/share/nginx/html# echo "naya he wha dikha kya rey" > index.html
root@newnginxpod:/usr/share/nginx/html#
```

Copy ip and paste it to browser



Replication Controller, ReplicaSet, and Scaling in Kubernetes

- **Replication Controller**:
  - Ensures a specified number of pod replicas are running.
  - **Deprecated** in favor of ReplicaSet.
- **ReplicaSet**:

- Manages pod replicas with more flexible selectors.
- Works with **Deployments** for rolling updates.

- **Scaling**:
  - **Manual Scaling**: Adjust the number of replicas manually.
  - **Autoscaling**: Pods are scaled automatically based on metrics like CPU usage (via **Horizontal Pod Autoscaler**)

Replication Controller

Mkdir replicationcontrollerwala

Cd replicationcontrollerwala

```
ubuntu@ip-172-31-46-144:~$ #ReplicationController file
ubuntu@ip-172-31-46-144:~$ mkdir replicationcontrollerwala
ubuntu@ip-172-31-46-144:~$ cd replicationcontrollerwala/
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ nano myreplica.yml
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$
```

Create .yml file

nano myreplica.yml

```
ubuntu@ip-172-31-46-144: ~/
 GNU nano 7.2                                    myreplica.yml *
apiVersion: v1
kind: ReplicationController
metadata:
   name: myrc
spec:
  replicas: 3
  selector:
     env: dev
  template:
       metadata:
          labels:
              env: dev
       spec:
            containers:
                - name: nginxcontaine
                  image: nginx
                  ports:
                    - containerPort: 80

^G Help      ^O Write Out  ^W Where Is  ^K Cut       ^T Execute   ^C Location  M-U Undo    M-A Set Mark
^X Exit      ^R Read File  ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line M-E Redo    M-6 Copy
```

Kubectl apply -f myreplica.yml

Kubectl get rc

Kubectl get pod

```
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl apply -f myreplica.yml
replicationcontroller/myrc unchanged
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get rc
NAME   DESIRED   CURRENT   READY   AGE
myrc   3         3         3       39s
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get pod
NAME          READY   STATUS    RESTARTS      AGE
myrc-78t28    1/1     Running   0             60s
myrc-tfpcm    1/1     Running   0             60s
myrc-tvld9    1/1     Running   0             60s
newnginxpod   1/1     Running   1 (11m ago)   23h
nginxpod      1/1     Running   1 (11m ago)   24h
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$
```

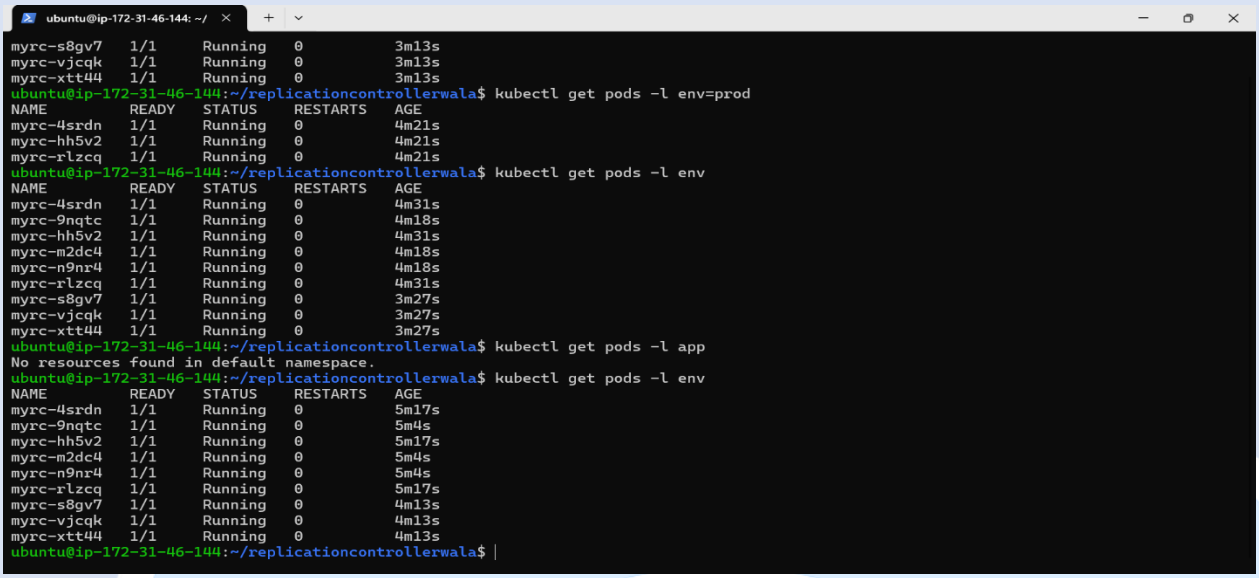
To check self healing

```
ubuntu@ip-172-31-46-144: ~/
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ #TO CHECK SELF HEALING
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get pod
NAME          READY   STATUS    RESTARTS      AGE
myrc-78t28    1/1     Running   0             2m25s
myrc-tfpcm    1/1     Running   0             2m25s
myrc-tvld9    1/1     Running   0             2m25s
newnginxpod   1/1     Running   1 (12m ago)   23h
nginxpod      1/1     Running   1 (12m ago)   24h
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl delete pod myrc-78t28 --force
Warning: Immediate deletion does not wait for confirmation that the running resource has been terminated. The resource may continue t
o run on the cluster indefinitely.
pod "myrc-78t28" force deleted
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get rc
NAME   DESIRED   CURRENT   READY   AGE
myrc   3         3         3       3m32s
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get pod
NAME          READY   STATUS    RESTARTS      AGE
myrc-5qwkr    1/1     Running   0             34s
myrc-tfpcm    1/1     Running   0             3m37s
myrc-tvld9    1/1     Running   0             3m37s
newnginxpod   1/1     Running   1 (13m ago)   23h
nginxpod      1/1     Running   1 (13m ago)   24h
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$
```

Kubectl get pods -l env=prod

Kubectl get pods -l env

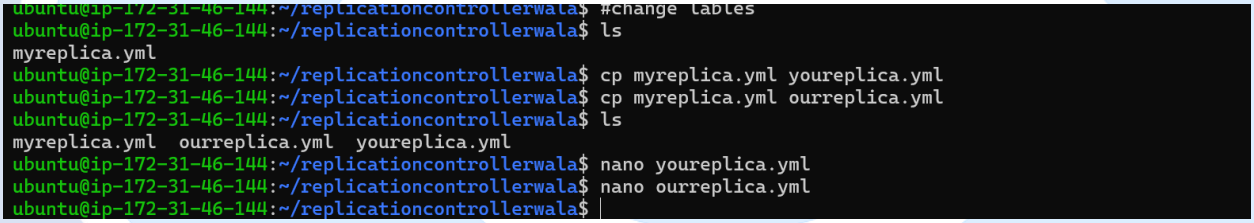
kubectl get pods -l app



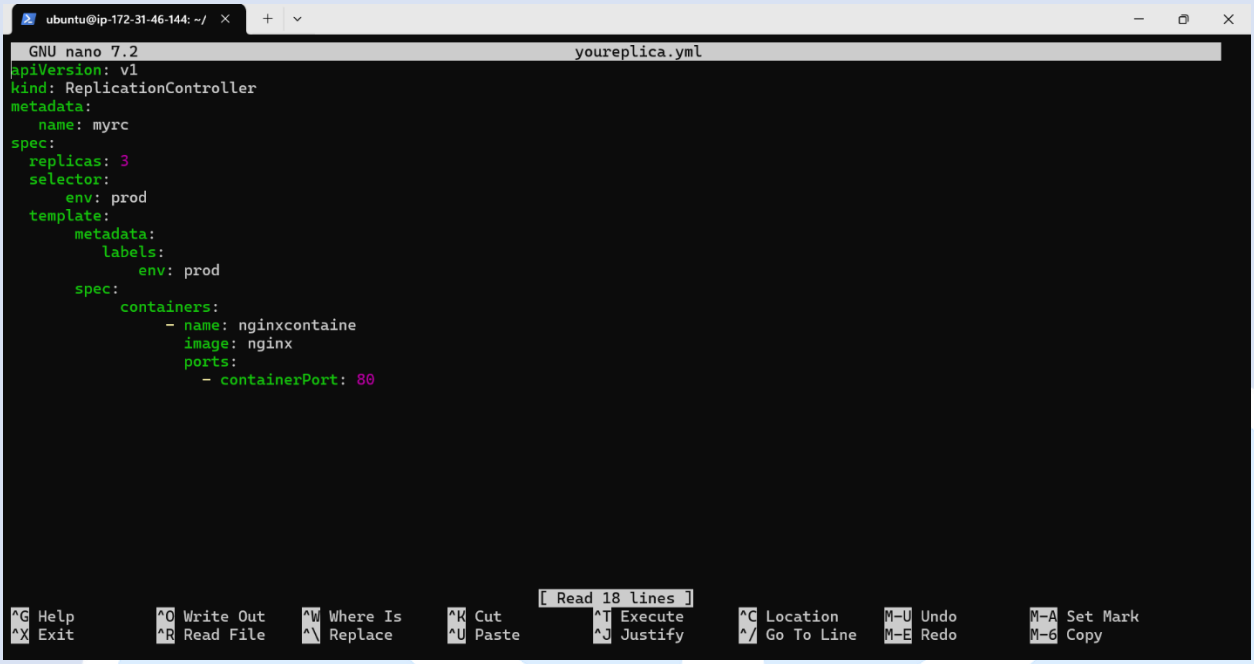
Change lable

Cp myreplica.yml youreplica.yml

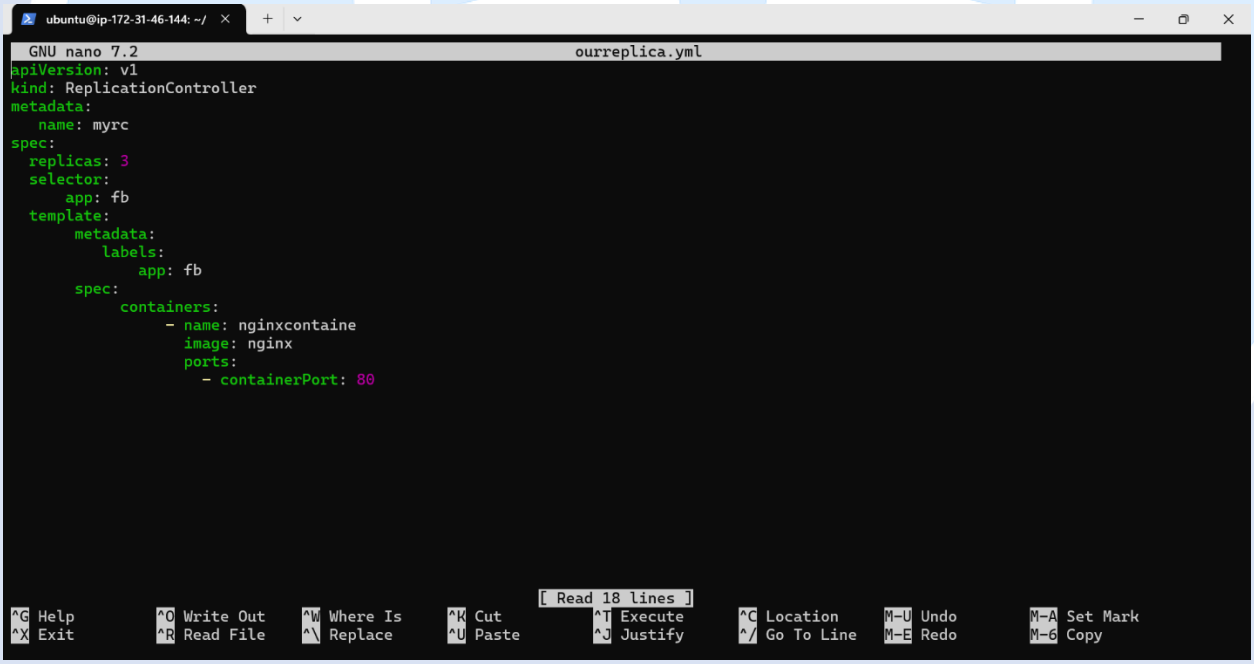
Cp myreplica.yml ourreplica.yml



Nano yourreplica.yml



Nano ourreplica.yml



Kubectl apply -f youreplica.yml

Kubectl apply -f oureplica.yml

Kubectl get rc-o wide

```
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ ls
myreplica.yml  ourreplica.yml  youreplica.yml
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl apply -f youreplica.yml
replicationcontroller/myrc configured
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl apply -f ourreplica.yml
replicationcontroller/myrc configured
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get rc -o wide
NAME    DESIRED   CURRENT   READY   AGE   CONTAINERS      IMAGES   SELECTOR
myrc    3         3         3       29m   nginxcontaine   nginx    app=fb
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$
```

```
ubuntu@ip-172-31-46-144: ~/
myrc-s8gv7   1/1     Running   0          3m13s
myrc-vjcqk   1/1     Running   0          3m13s
myrc-xtt44   1/1     Running   0          3m13s
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get pods -l env=prod
NAME         READY   STATUS    RESTARTS   AGE
myrc-4srdn   1/1     Running   0          4m21s
myrc-hh5v2   1/1     Running   0          4m21s
myrc-rlzcq   1/1     Running   0          4m21s
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get pods -l env
NAME         READY   STATUS    RESTARTS   AGE
myrc-4srdn   1/1     Running   0          4m31s
myrc-9nqtc   1/1     Running   0          4m18s
myrc-hh5v2   1/1     Running   0          4m31s
myrc-m2dc4   1/1     Running   0          4m18s
myrc-n9nr4   1/1     Running   0          4m18s
myrc-rlzcq   1/1     Running   0          4m31s
myrc-s8gv7   1/1     Running   0          3m27s
myrc-vjcqk   1/1     Running   0          3m27s
myrc-xtt44   1/1     Running   0          3m27s
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get pods -l app
No resources found in default namespace.
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get pods -l env
NAME         READY   STATUS    RESTARTS   AGE
myrc-4srdn   1/1     Running   0          5m17s
myrc-9nqtc   1/1     Running   0          5m4s
myrc-hh5v2   1/1     Running   0          5m17s
myrc-m2dc4   1/1     Running   0          5m4s
myrc-n9nr4   1/1     Running   0          5m4s
myrc-rlzcq   1/1     Running   0          5m17s
myrc-s8gv7   1/1     Running   0          4m13s
myrc-vjcqk   1/1     Running   0          4m13s
myrc-xtt44   1/1     Running   0          4m13s
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$
```

Kubectl get rc -o wide

Kubectl get rs -o wide

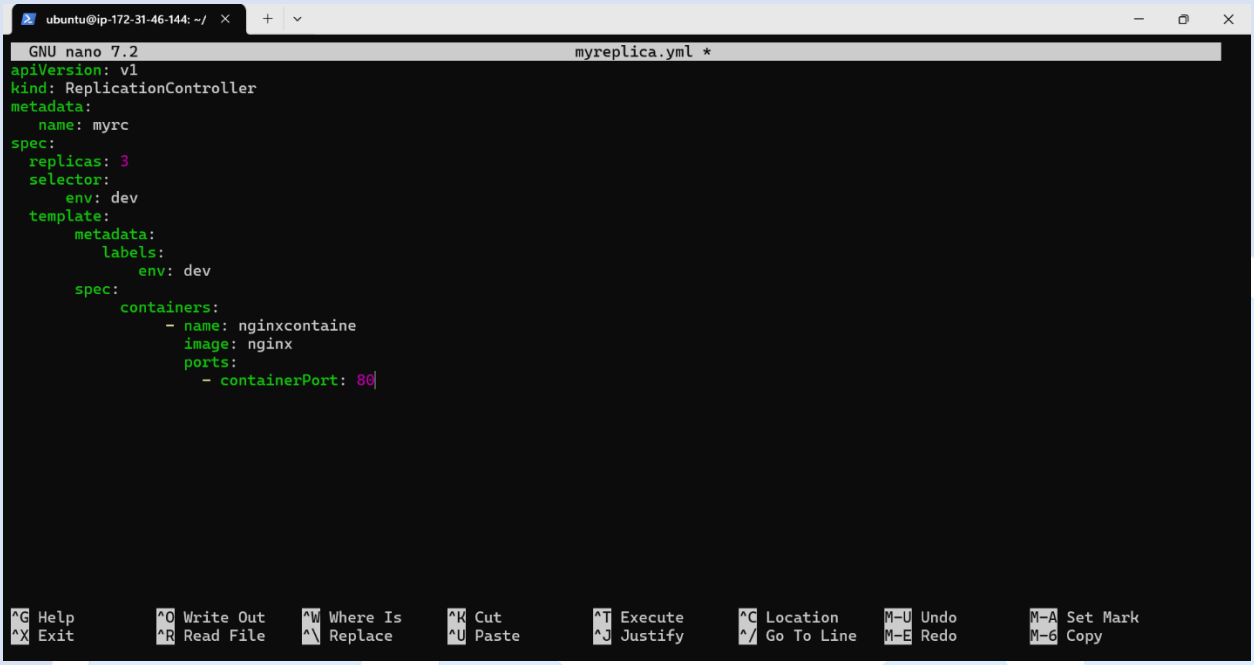
Scaling:

Kubectl scale rc myrc –replicase

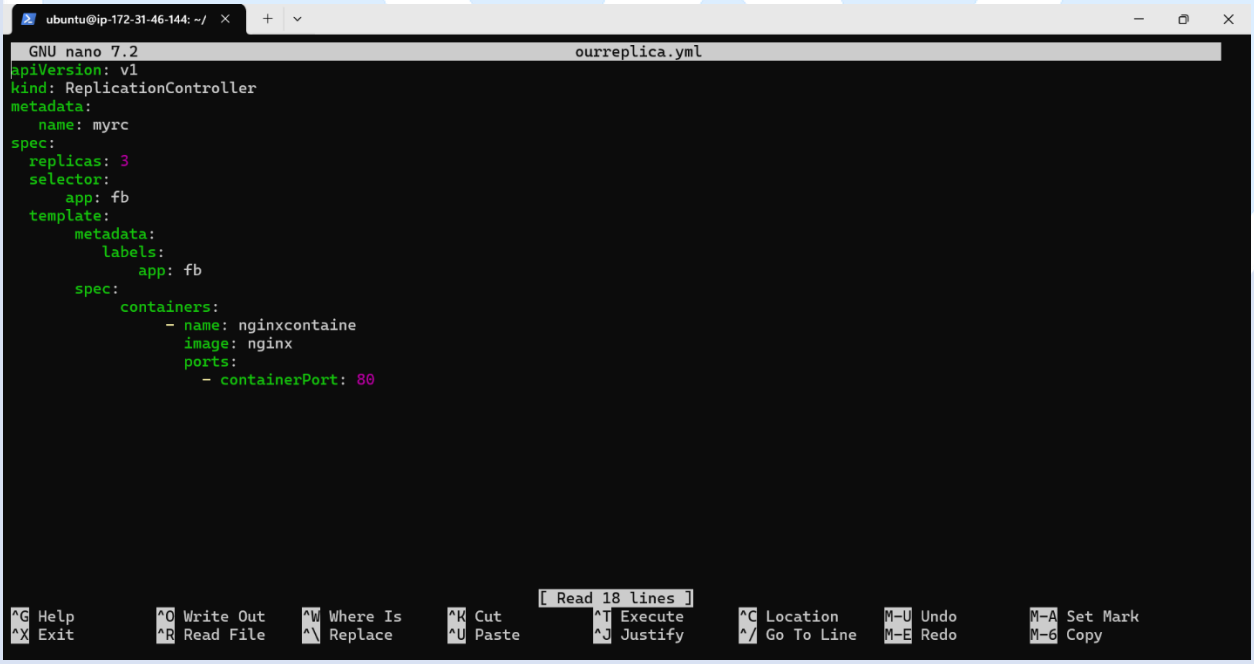
Kubectl get pod

```
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ #SCALING
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl scale rc myrc --replicas=5
replicationcontroller/myrc scaled
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get pod
NAME          READY   STATUS    RESTARTS      AGE
myrc-5qwkr    1/1     Running   0             4m40s
myrc-7kpfv    1/1     Running   0             56s
myrc-hl7g9    1/1     Running   0             56s
myrc-tfpcm    1/1     Running   0             7m43s
myrc-tvld9    1/1     Running   0             7m43s
newnginxpod   1/1     Running   1 (18m ago)   23h
nginxpod      1/1     Running   1 (18m ago)   24h
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get rc
NAME    DESIRED   CURRENT   READY   AGE
myrc    5         5         5       8m31s
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$
```

nano myreplica.yml



nano ourreplica.yml



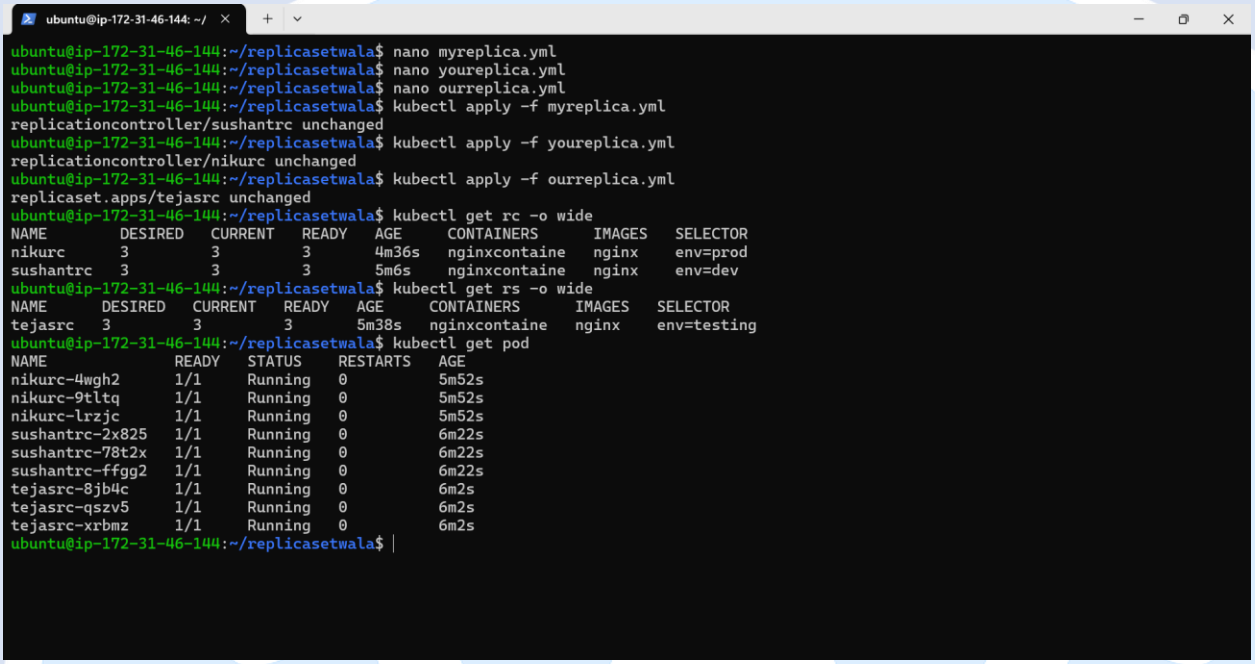
Kubectl apply -f myreplica.yml

Kubectl apply -f youreplica.yml

Kubectl apply -f ourreplica.yml

Kubectl get rc -o wide

Kubectl get rs -o wide



```
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get rc
NAME    DESIRED   CURRENT   READY   AGE
myrc    5         5         5       8m31s
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ ls
myreplica.yml
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl apply -f myreplica.yml
replicationcontroller/myrc configured
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get rc
NAME    DESIRED   CURRENT   READY   AGE
myrc    3         3         3       11m
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl describe rc myrc
Name:         myrc
Namespace:    default
Selector:     env=dev
Labels:       env=dev
Annotations:  <none>
Replicas:     3 current / 3 desired
Pods Status:  5 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  env=dev
  Containers:
   nginxcontaine:
    Image:         nginx
    Port:          80/TCP
    Host Port:     0/TCP
    Environment:   <none>
    Mounts:        <none>
  Volumes:         <none>
  Node-Selectors:  <none>
  Tolerations:     <none>
Events:
  Type    Reason            Age    From                    Message
  ----    ------            ----   ----                    -------
```

To show selectors

Kubectl get replicationcontroller -o wide

Kubectl get pods -l env=dev

```
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ #TO SHOW THE SELECTORS
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$  kubectl get replicationcontroller -o wide
NAME    DESIRED   CURRENT   READY   AGE   CONTAINERS      IMAGES   SELECTOR
myrc    3         3         3       15m   nginxcontaine   nginx    env=dev
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get pods -l env=dev
NAME         READY   STATUS        RESTARTS   AGE
myrc-5qwkr   1/1     Running       0          14m
myrc-7kpfv   1/1     Terminating   0          10m
myrc-hl7g9   1/1     Terminating   0          10m
myrc-tfpcm   1/1     Running       0          17m
myrc-tvld9   1/1     Running       0          17m
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$
```

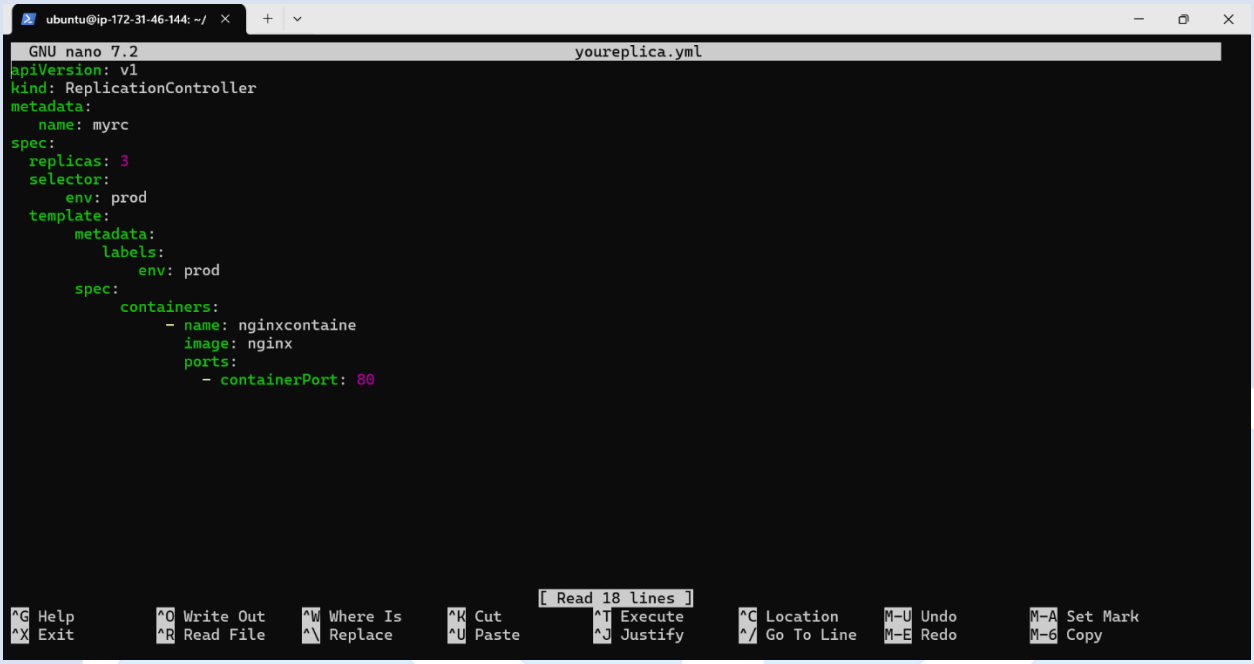
Change lables

cp myreplica.yml yourreplica.yml

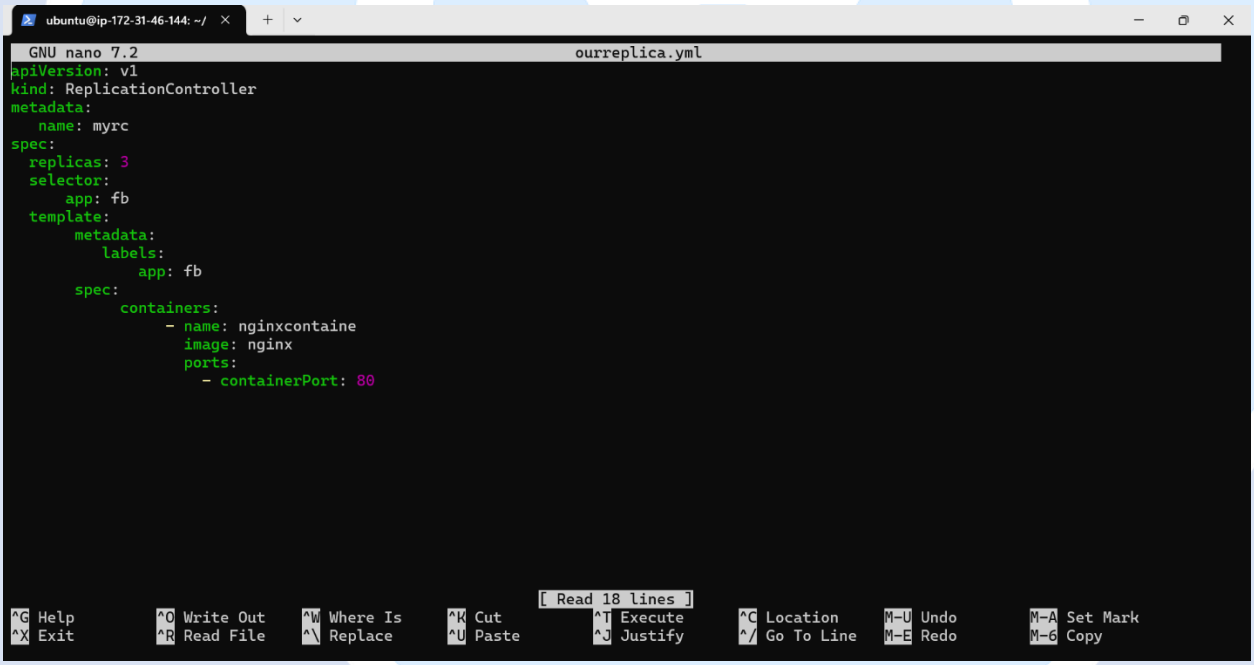
cp myreplica.yml ourreplical.yml

```
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ #change lables
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ ls
myreplica.yml
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ cp myreplica.yml youreplica.yml
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ cp myreplica.yml ourreplica.yml
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ ls
myreplica.yml  ourreplica.yml  youreplica.yml
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ nano youreplica.yml
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ nano ourreplica.yml
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$
```

nano yourreplica.yml



nano ourreplica.yml



kubectl get pods -l env=prod

kubectl get pods -l env

kubectl get pod -l env

```
ubuntu@ip-172-31-46-144: ~/
myrc-s8gv7   1/1     Running   0          3m13s
myrc-vjcqk   1/1     Running   0          3m13s
myrc-xtt44   1/1     Running   0          3m13s
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get pods -l env=prod
NAME         READY   STATUS    RESTARTS   AGE
myrc-4srdn   1/1     Running   0          4m21s
myrc-hh5v2   1/1     Running   0          4m21s
myrc-rlzcq   1/1     Running   0          4m21s
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get pods -l env
NAME         READY   STATUS    RESTARTS   AGE
myrc-4srdn   1/1     Running   0          4m31s
myrc-9nqtc   1/1     Running   0          4m18s
myrc-hh5v2   1/1     Running   0          4m31s
myrc-m2dc4   1/1     Running   0          4m18s
myrc-n9nr4   1/1     Running   0          4m18s
myrc-rlzcq   1/1     Running   0          4m31s
myrc-s8gv7   1/1     Running   0          3m27s
myrc-vjcqk   1/1     Running   0          3m27s
myrc-xtt44   1/1     Running   0          3m27s
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get pods -l app
No resources found in default namespace.
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$ kubectl get pods -l env
NAME         READY   STATUS    RESTARTS   AGE
myrc-4srdn   1/1     Running   0          5m17s
myrc-9nqtc   1/1     Running   0          5m4s
myrc-hh5v2   1/1     Running   0          5m17s
myrc-m2dc4   1/1     Running   0          5m4s
myrc-n9nr4   1/1     Running   0          5m4s
myrc-rlzcq   1/1     Running   0          5m17s
myrc-s8gv7   1/1     Running   0          4m13s
myrc-vjcqk   1/1     Running   0          4m13s
myrc-xtt44   1/1     Running   0          4m13s
ubuntu@ip-172-31-46-144:~/replicationcontrollerwala$
```

ReplicaSet

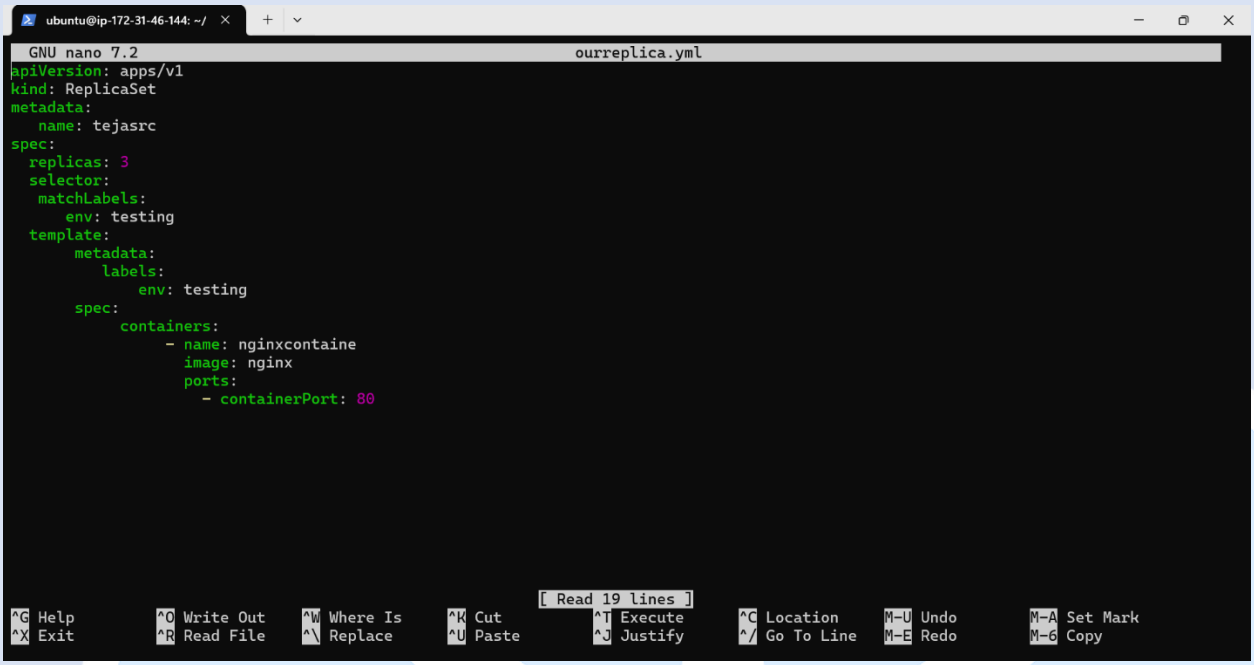
Cd replicasetwala

```
ubuntu@ip-172-31-46-144:~$ ls
nginxpod  replicasetwala  replicationcontrollerwala
ubuntu@ip-172-31-46-144:~$ cd replicasetwala/
ubuntu@ip-172-31-46-144:~/replicasetwala$ ls
myreplica.yml  ourreplica.yml  youreplica.yml
ubuntu@ip-172-31-46-144:~/replicasetwala$
```

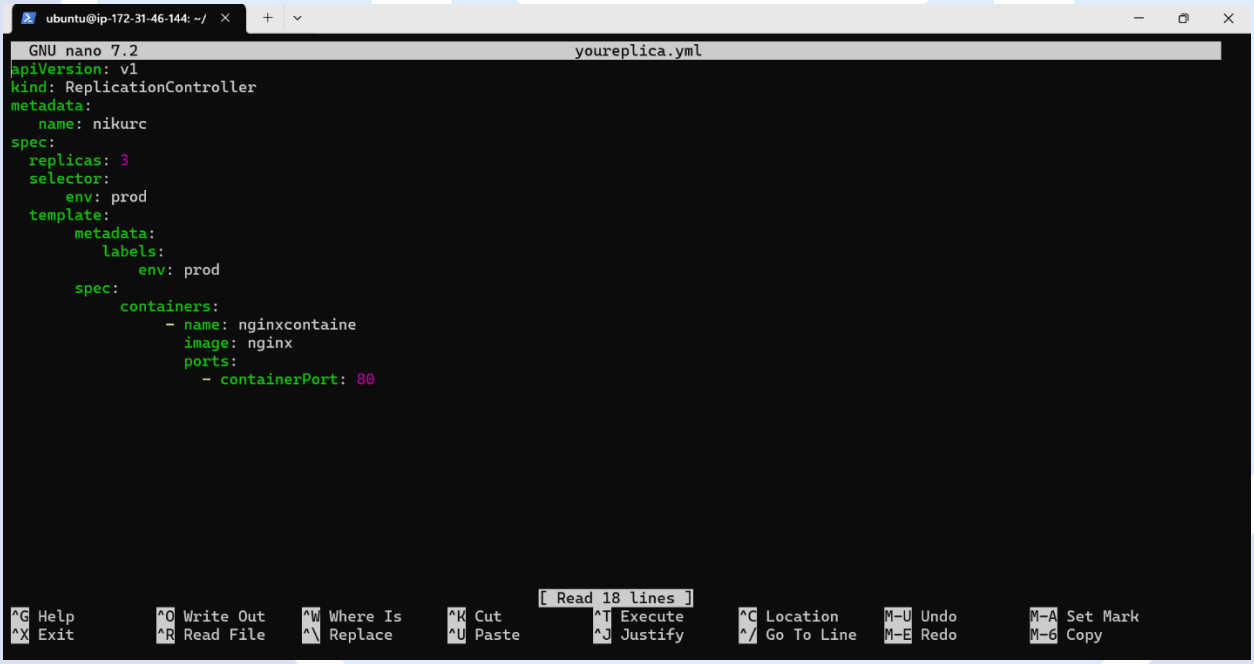
Nano replica.yml

```
ubuntu@ip-172-31-46-144: ~/
 GNU nano 7.2                                    myreplica.yml
apiVersion: v1
kind: ReplicationController
metadata:
   name: sushantrc
spec:
  replicas: 3
  selector:
     env: dev
  template:
      metadata:
         labels:
            env: dev
      spec:
          containers:
              - name: nginxcontaine
                image: nginx
                ports:
                  - containerPort: 80

                                [ Wrote 18 lines ]
^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo      M-6 Copy
```

Nanoourreplica.yml



Nano yourreplica.yml



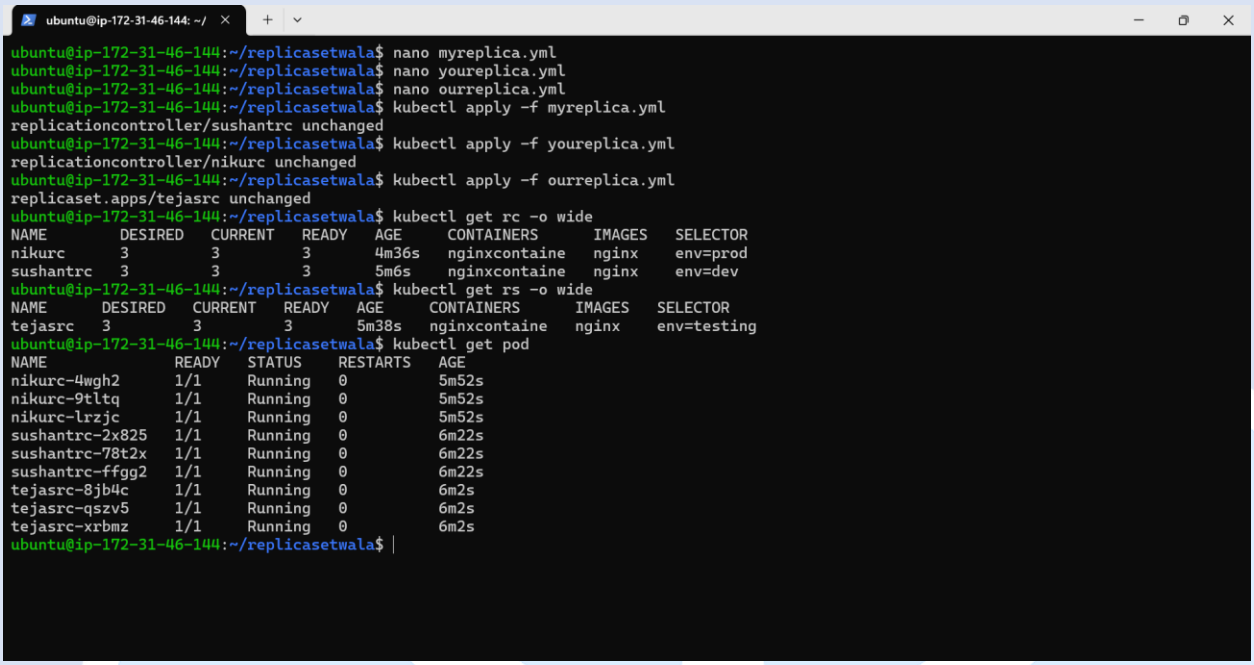
kubectl apply -f myreplica.yml

kubectl apply -f youreplica.yml

kubectl apply -f ourreplica.yml

kubectl rc grt -o wide

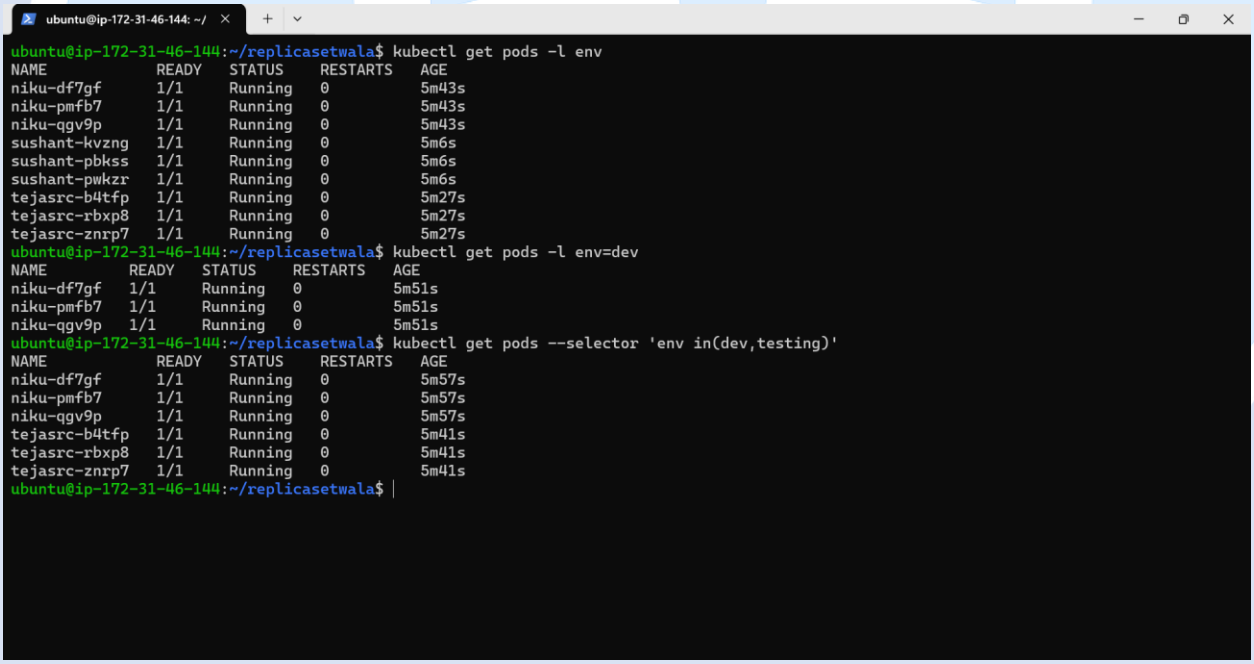
kubectl rs grt -o wide



Kubectl get pods -l env

Kubectl get pod -l env=dev

Kubectl get pod --selector 'env in {dev,testing}'

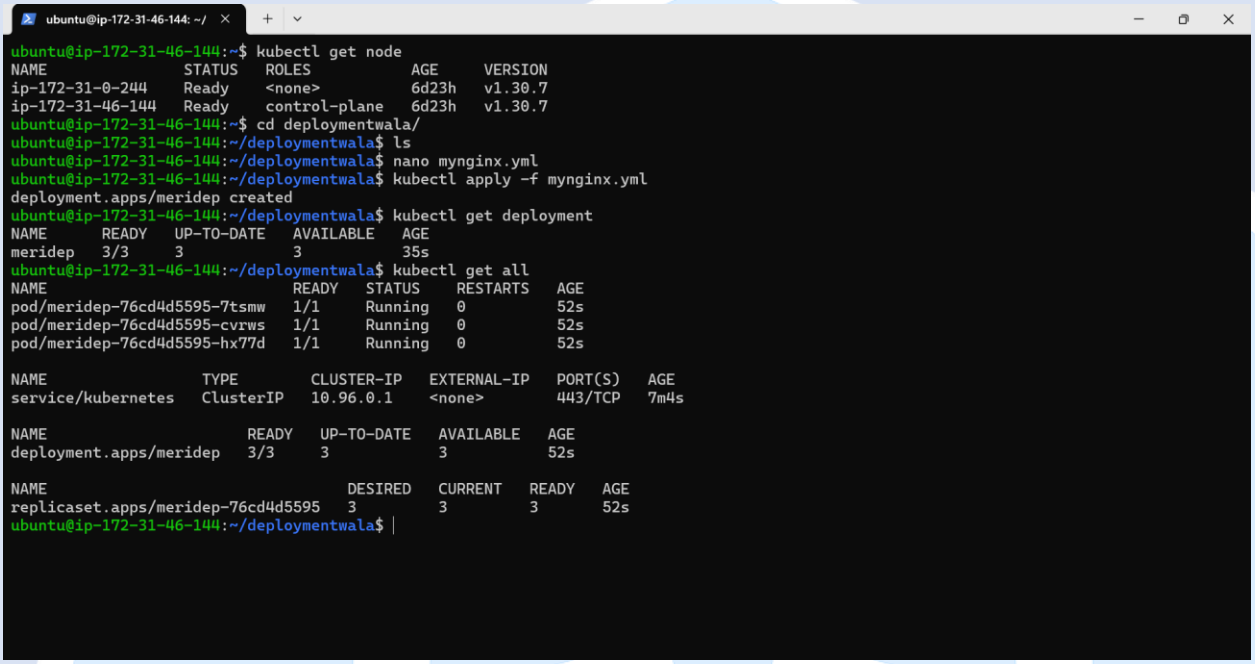


DEPLOYMENT

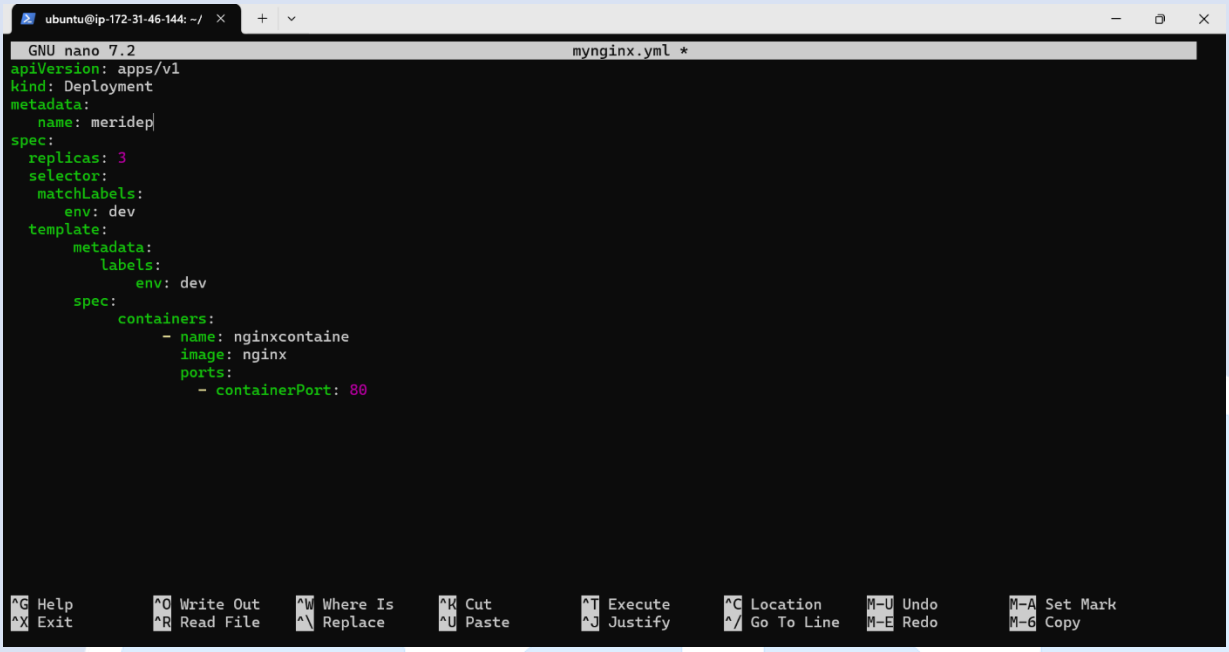
Mkdir deploymentwala

Cd mydeploymentwala

nano mynginx.yml



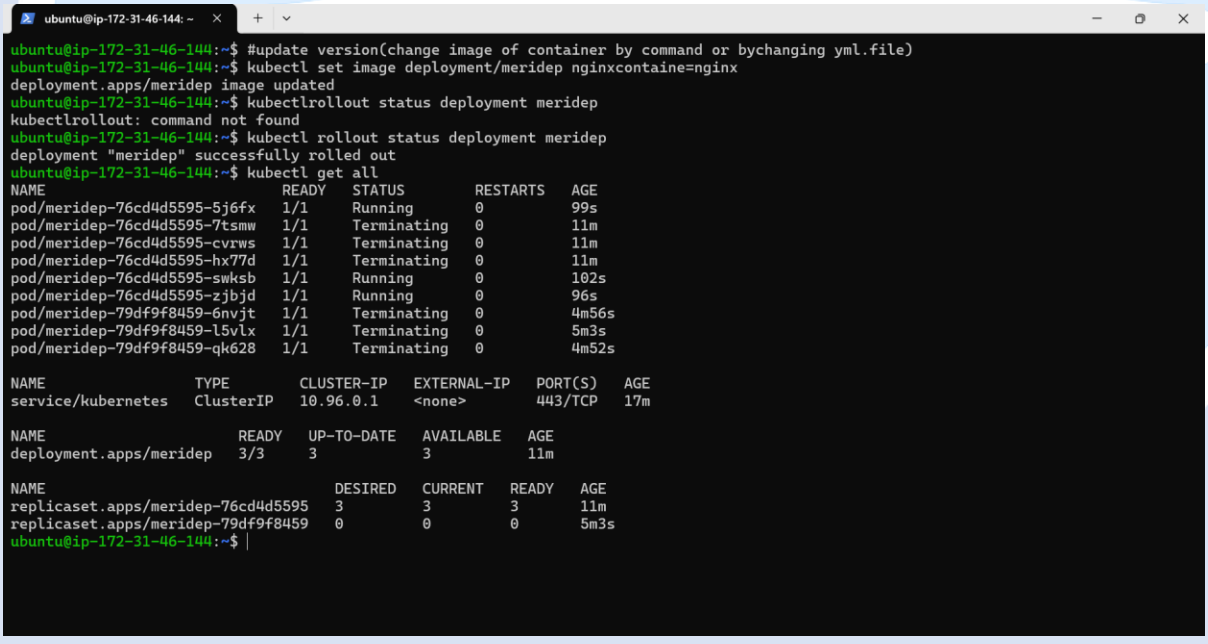
Mynginx.yml



kubectl apply -f mynginx.yml

kubectl get all

Version change(by command)

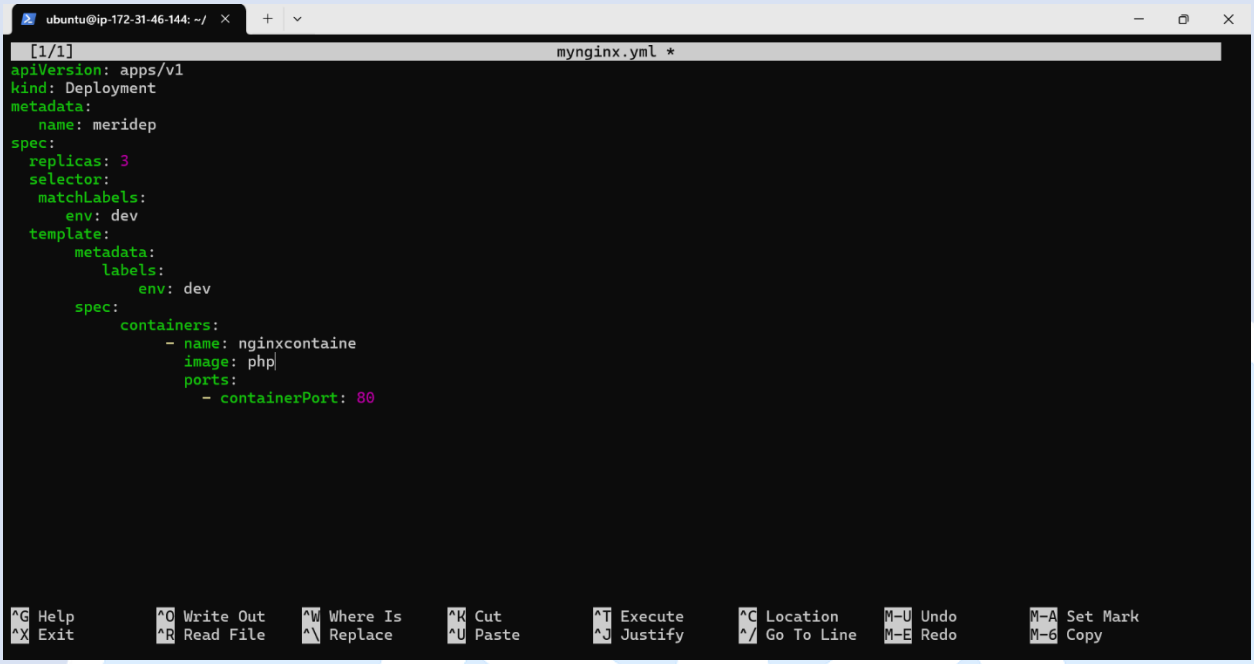


Kubctl set image deployment/meridep ngintconaner

Kubectl rollout status deployment meridep

Version (change by .yml file)

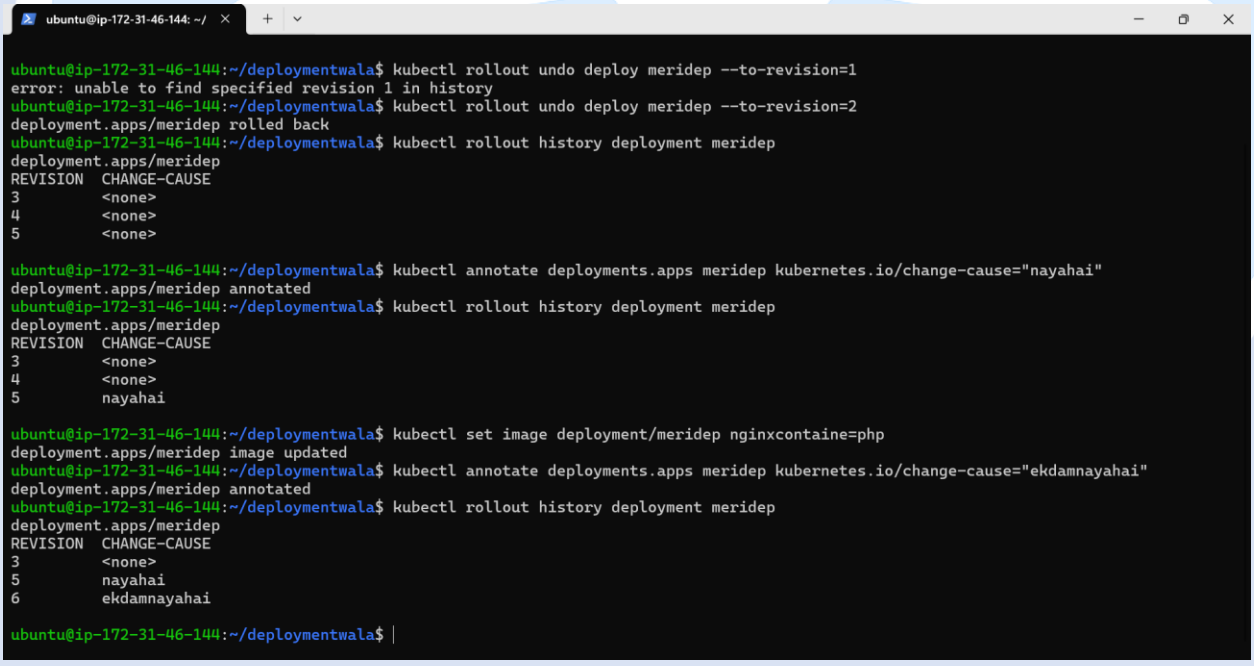
Nano mynginx.yml



kubectl rollout history deployment meridep

kubectl rollout undo deploy meridep -- to—revision=2

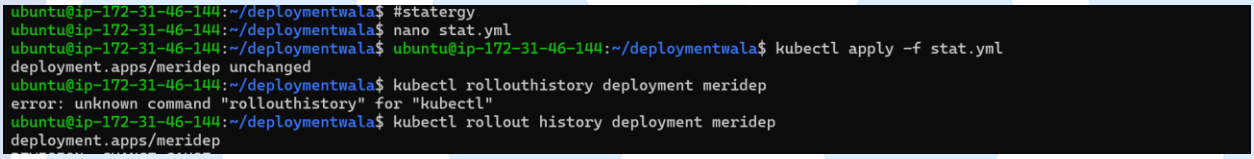
klubectl annotate deployments.apps meridep kubernetes.io/change-cause”nayahai”



Statergy

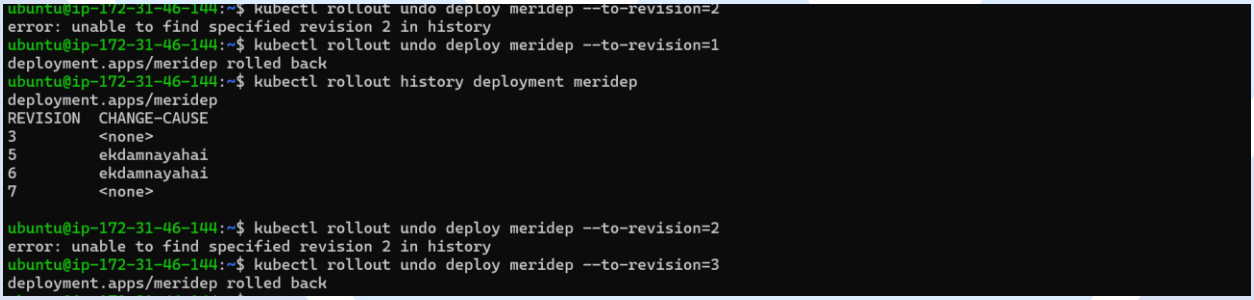
Nano sat.yml

Kubectl apply -f stat.yml



kubectl rollout undo deploy meridep -- to—revision=2

kubectl rollout history deployment meridep



HEALTHPROBE

In Kubernetes, a **health probe** is a mechanism used by the Kubernetes system to monitor the health of applications running in a container. Kubernetes uses these probes to determine if a container is healthy or unhealthy. There are two main types of health probes:

1. **Liveness Probe**
    - Checks whether the application inside a container is still running.
    - If the liveness probe fails, Kubernetes kills the container and restarts it (based on the pod's restart policy).
    - Use Case: To detect and remedy situations where the application is in a deadlock or not responding.
2. **Readiness Probe**
    - Checks whether the application inside a container is ready to serve requests.
    - If the readiness probe fails, Kubernetes removes the pod from the Service's endpoints, so it doesn't receive traffic.
    - Use Case: To ensure that the application has completed its startup process and is ready to handle requests.

3. **Startup Probe**

- It ensures that the application is fully initialized before Kubernetes considers it "live."
- During the startup phase, **Liveness Probes** are disabled to avoid restarting the container prematurely.
- Once the Startup Probe succeeds, Kubernetes switches to using the Liveness Probe (if configured) for ongoing health checks.

Liveness Probe

```
ubuntu@ip-172-31-46-144: ~/
 GNU nano 7.2                                        health.yml *
apiVersion: apps/v1
kind: Deployment
metadata:
  name: meridep
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 2
      maxUnavailable: 0
  selector:                          # Added the selector field
    matchLabels:
      app: meridep
      env: dev
  template:
    metadata:
      labels:
        app: meridep
        env: dev
    spec:
      containers:
        - name: nginxcontainer
          image: nginx
          ports:
            - containerPort: 80


Save modified buffer?
 Y Yes
 N No           ^C Cancel
```

Kubectl apply -f health.yml

Kubectl get pod

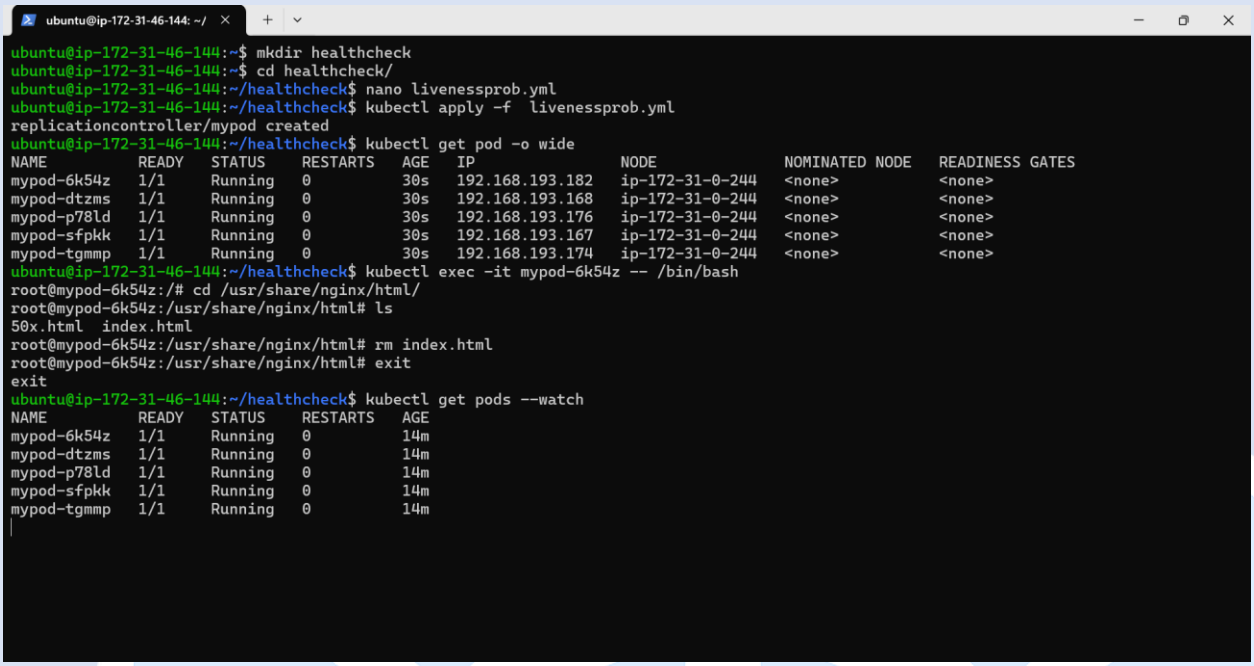
Kubectl delete pod (pod name)

```
ubuntu@ip-172-31-46-144:~/deploymentwala$ #HEALTHPROBE(HEALTH CHECK)
ubuntu@ip-172-31-46-144:~/deploymentwala$ #LIVENESS PROBE
ubuntu@ip-172-31-46-144:~/deploymentwala$ nano health.yml
ubuntu@ip-172-31-46-144:~/deploymentwala$ kubectl apply -f health.yml
deployment.apps/meridep created
ubuntu@ip-172-31-46-144:~/deploymentwala$  kubectl get pod
NAME                      READY   STATUS    RESTARTS   AGE
meridep-64888bb4d-2f8n4   1/1     Running   0          16s
meridep-64888bb4d-2nk6k   1/1     Running   0          16s
meridep-64888bb4d-8znkn   1/1     Running   0          16s
ubuntu@ip-172-31-46-144:~/deploymentwala$ kubectl delete pod meridep-64888bb4d-2f8n4
pod "meridep-64888bb4d-2f8n4" deleted
```

new pod is created

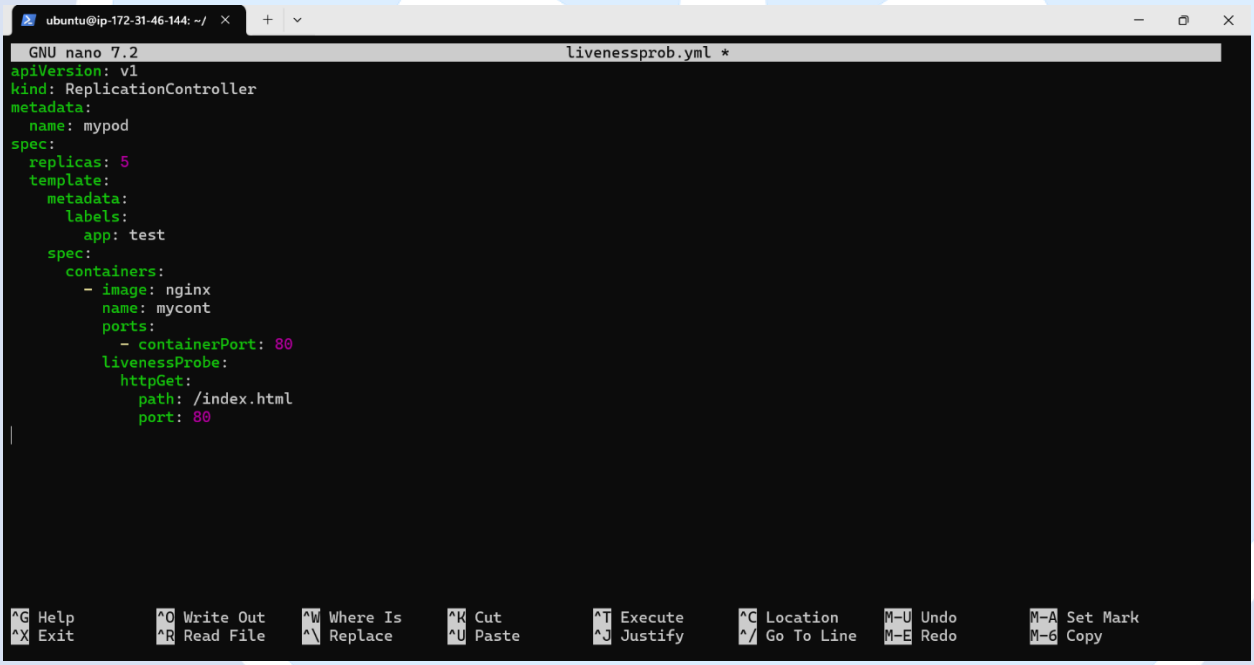
```
ubuntu@ip-172-31-46-144:~$ kubectl get pod
NAME                      READY   STATUS        RESTARTS   AGE
meridep-64888bb4d-2f8n4   1/1     Terminating   0          3m27s
meridep-64888bb4d-2nk6k   1/1     Running       0          3m27s
meridep-64888bb4d-8znkn   1/1     Running       0          3m27s
meridep-64888bb4d-jvl22   1/1     Running       0          118s
ubuntu@ip-172-31-46-144:~$
```

Liveness Probe



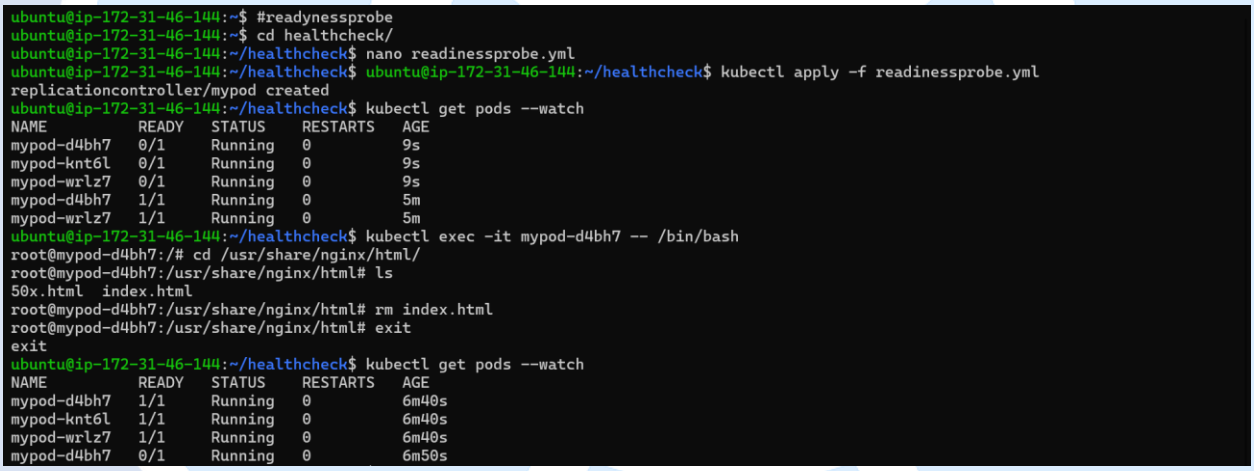
```
ubuntu@ip-172-31-46-144:~$ mkdir healthcheck
ubuntu@ip-172-31-46-144:~$ cd healthcheck/
ubuntu@ip-172-31-46-144:~/healthcheck$ nano livenessprob.yml
ubuntu@ip-172-31-46-144:~/healthcheck$ kubectl apply -f  livenessprob.yml
replicationcontroller/mypod created
ubuntu@ip-172-31-46-144:~/healthcheck$ kubectl get pod -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP               NODE              NOMINATED NODE   READINESS GATES
mypod-6k54z   1/1     Running   0          30s   192.168.193.182   ip-172-31-0-244   <none>           <none>
mypod-dtzms   1/1     Running   0          30s   192.168.193.168   ip-172-31-0-244   <none>           <none>
mypod-p78ld   1/1     Running   0          30s   192.168.193.176   ip-172-31-0-244   <none>           <none>
mypod-sfpkk   1/1     Running   0          30s   192.168.193.167   ip-172-31-0-244   <none>           <none>
mypod-tgmmp   1/1     Running   0          30s   192.168.193.174   ip-172-31-0-244   <none>           <none>
ubuntu@ip-172-31-46-144:~/healthcheck$ kubectl exec -it mypod-6k54z -- /bin/bash
root@mypod-6k54z:/# cd /usr/share/nginx/html/
root@mypod-6k54z:/usr/share/nginx/html# ls
50x.html  index.html
root@mypod-6k54z:/usr/share/nginx/html# rm index.html
root@mypod-6k54z:/usr/share/nginx/html# exit
exit
ubuntu@ip-172-31-46-144:~/healthcheck$ kubectl get pods --watch
NAME          READY   STATUS    RESTARTS   AGE
mypod-6k54z   1/1     Running   0          14m
mypod-dtzms   1/1     Running   0          14m
mypod-p78ld   1/1     Running   0          14m
mypod-sfpkk   1/1     Running   0          14m
mypod-tgmmp   1/1     Running   0          14m
```

Livenessprobe.yml



```
  GNU nano 7.2                                  livenessprob.yml *
apiVersion: v1
kind: ReplicationController
metadata:
  name: mypod
spec:
  replicas: 5
  template:
    metadata:
      labels:
        app: test
    spec:
      containers:
        - image: nginx
          name: mycont
          ports:
            - containerPort: 80
          livenessProbe:
            httpGet:
              path: /index.html
              port: 80

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy
```

Readiness Probe



Startup Probe:

```
ubuntu@ip-172-31-46-144: ~/
 GNU nano 7.2                                    startupprobr.yml *
apiVersion: v1
kind: ReplicationController
metadata:
  name: mypod
spec:
  replicas: 5
  template:
    metadata:
      labels:
        app: test
    spec:
      containers:
        - image: nginx
          name: mycont
          ports:
            - containerPort: 80
          livenessProbe:
            httpGet:
              path: /index.html
              port: 80
          readinessProbe:
            httpGet:
              path: /index.html
              port: 80
          startupProbe:
            httpGet:
              path: /index.html
              port: 80
            initialDelaySeconds: 5

^G Help          ^O Write Out     ^W Where Is      ^K Cut           ^T Execute       ^C Location      M-U Undo         M-A Set Mark
^X Exit          ^R Read File     ^\ Replace       ^U Paste         ^J Justify       ^/ Go To Line    M-E Redo         M-6 Copy
```

```
ubuntu@ip-172-31-46-144:~/healthcheck$ #startuprobe
ubuntu@ip-172-31-46-144:~/healthcheck$ nano startupprobr.yml
ubuntu@ip-172-31-46-144:~/healthcheck$ kubectl apply -f startupprobr.yml
replicationcontroller/mypod created
ubuntu@ip-172-31-46-144:~/healthcheck$ kubectl get pods --watch
NAME          READY   STATUS              RESTARTS   AGE
mypod-8x87z   0/1     ContainerCreating   0          17s
mypod-g452h   0/1     ContainerCreating   0          17s
mypod-rb86q   0/1     ContainerCreating   0          17s
mypod-rm6s5   0/1     ContainerCreating   0          17s
mypod-txwhz   0/1     ContainerCreating   0          17s
^Cubuntu@ip-172-31-46-144:~/healthcheck$ kubectl get pods
NAME          READY   STATUS              RESTARTS   AGE
mypod-8x87z   0/1     ContainerCreating   0          116s
mypod-g452h   0/1     ContainerCreating   0          116s
mypod-rb86q   0/1     ContainerCreating   0          116s
mypod-rm6s5   0/1     ContainerCreating   0          116s
mypod-txwhz   0/1     ContainerCreating   0          116s
ubuntu@ip-172-31-46-144:~/healthcheck$
```

Namespaces in Kubernetes

A namespace in Kubernetes is a virtual cluster that provides logical isolation for resources within the same physical cluster. It is primarily used for organizing and managing workloads, enabling multi-tenancy, and implementing access controls in Kubernetes environments

Purpose of Namespaces

1. **Resource Isolation**: Separate workloads and resources to avoid conflicts.
2. **Access Management**: Control user and team access to specific resources.
3. **Resource Organization**: Structure resources based on projects, teams, or environments.
4. **Multi-Tenancy**: Enable multiple users or teams to share the same Kubernetes cluster securely.

Default Namespaces

1. **default**: The namespace used when no specific namespace is assigned.
2. **kube-system**: Contains Kubernetes system components (e.g., API server, scheduler).
3. **kube-public**: Accessible by all users and used for public resources.
4. **kube-node-lease**: Stores node heartbeat leases to enhance performance of node health monitoring.

User-Defined Namespaces

Users can create custom namespaces to organize resources for specific purposes, such as:

- Projects or teams (e.g., team-a, team-b).
- Environments (e.g., dev, staging, prod).

Key Features

1. **Logical Isolation**: Workloads in one namespace are independent of those in another.
2. **Scoped Resources**: Namespaces group resources such as pods, services, and deployments.
3. **Role-Based Access Control (RBAC)**: Assign granular access permissions per namespace.
4. **Resource Quotas**: Limit CPU, memory, and other resource usage within a namespace.

How Namespaces Work

- Resources are scoped to a namespace unless they are cluster-wide (e.g., nodes, storage classes).
- Users can specify the namespace in commands or set a default namespace for ease of use.

Managing Namespaces

```
ubuntu@ip-172-31-46-144:~$ #NAMESPACE
ubuntu@ip-172-31-46-144:~$ kubectl get namespace
NAME              STATUS   AGE
default           Active   16d
kube-node-lease   Active   16d
kube-public       Active   16d
kube-system       Active   16d
ubuntu@ip-172-31-46-144:~$
```

```
ubuntu@ip-172-31-46-144:~$ kubectl get pods -n kube-system
NAME                                       READY   STATUS             RESTARTS         AGE
calico-kube-controllers-6cdb97b867-4rvc2   1/1     Running            5 (105m ago)     16d
calico-node-grzcj                          0/1     Running            87 (6m59s ago)   16d
calico-node-vzr9d                          0/1     CrashLoopBackOff   85 (3m12s ago)   16d
coredns-55cb58b774-6x772                   1/1     Running            5 (105m ago)     16d
coredns-55cb58b774-vbt6s                   1/1     Running            5 (105m ago)     16d
etcd-ip-172-31-46-144                      1/1     Running            6 (105m ago)     16d
kube-apiserver-ip-172-31-46-144            1/1     Running            6 (105m ago)     16d
kube-controller-manager-ip-172-31-46-144   1/1     Running            6 (105m ago)     16d
kube-proxy-k5fw5                           0/1     CrashLoopBackOff   85 (68s ago)     16d
kube-proxy-ppmk4                           1/1     Running            91 (5m13s ago)   16d
kube-scheduler-ip-172-31-46-144            1/1     Running            6 (105m ago)     16d
ubuntu@ip-172-31-46-144:~$
```

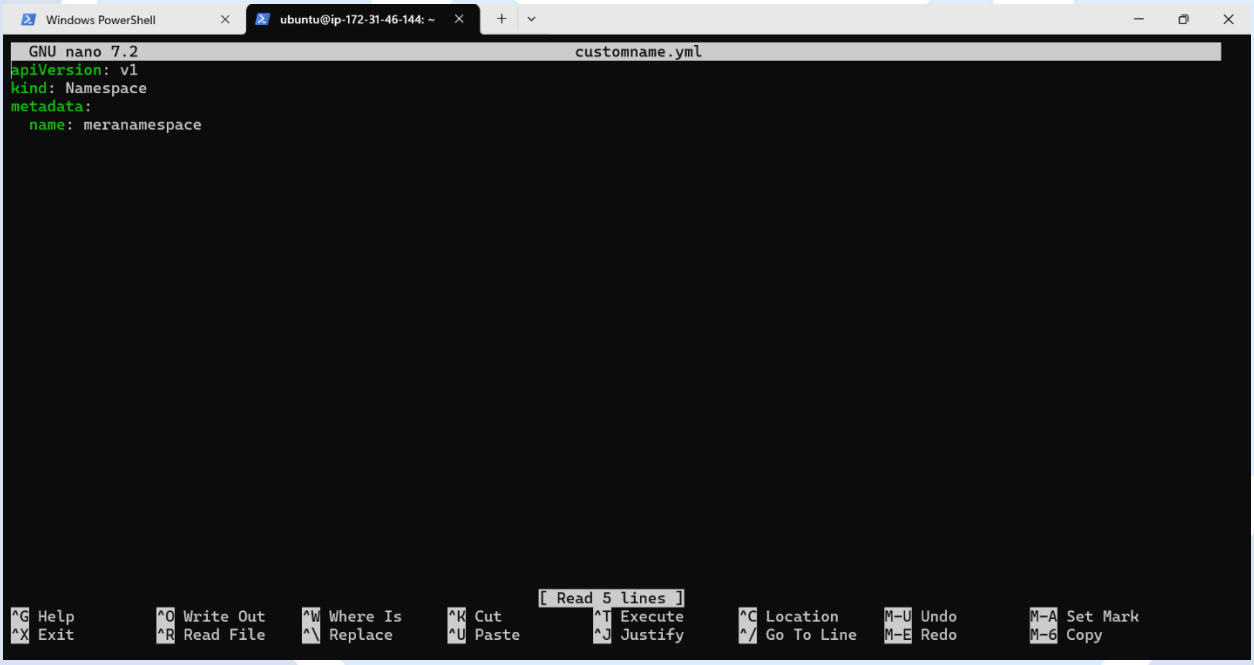
```
ubuntu@ip-172-31-46-144:~$ kubectl create namespace  mynamespace
namespace/mynamespace created
ubuntu@ip-172-31-46-144:~$  kubectl get namespace
NAME              STATUS   AGE
default           Active   16d
kube-node-lease   Active   16d
kube-public       Active   16d
kube-system       Active   16d
mynamespace       Active   22s
ubuntu@ip-172-31-46-144:~$
```

```
ubuntu@ip-172-31-46-144:~$ kubectl describe namespace mynamespace
Name:         mynamespace
Labels:       kubernetes.io/metadata.name=mynamespace
Annotations:  <none>
Status:       Active

No resource quota.

No LimitRange resource.
ubuntu@ip-172-31-46-144:~$
```

```
ubuntu@ip-172-31-46-144:~$ kubectl apply -f deploymentwala/mynginx.yml  -n mynamespace
deployment.apps/spacet unchanged
ubuntu@ip-172-31-46-144:~$ kubectl get pods -n mynamespace
NAME                      READY   STATUS             RESTARTS      AGE
spacet-745f666bf5-2xthq   0/1     CrashLoopBackOff   6 (76s ago)   7m32s
spacet-745f666bf5-q2ct2   0/1     CrashLoopBackOff   6 (89s ago)   7m32s
spacet-745f666bf5-tbwhm   0/1     CrashLoopBackOff   6 (77s ago)   7m32s
ubuntu@ip-172-31-46-144:~$ #DEAFULT NAMESPACE CONTEXT IS SET TO MYNAMESPACE
ubuntu@ip-172-31-46-144:~$ kubectl config set-context --current --namespace=mynamespace
Context "kubernetes-admin@kubernetes" modified.
ubuntu@ip-172-31-46-144:~$ kubectl get pods
NAME                      READY   STATUS             RESTARTS       AGE
spacet-745f666bf5-2xthq   0/1     CrashLoopBackOff   6 (110s ago)   8m6s
spacet-745f666bf5-q2ct2   0/1     CrashLoopBackOff   6 (2m3s ago)   8m6s
spacet-745f666bf5-tbwhm   0/1     CrashLoopBackOff   6 (111s ago)   8m6s
ubuntu@ip-172-31-46-144:~$
```

Namespace by using.ymlfile

```
ubuntu@ip-172-31-46-144:~$ #create namespace by .yml
ubuntu@ip-172-31-46-144:~$ nano customname.yml
ubuntu@ip-172-31-46-144:~$ kubectl apply -f customname.yml
namespace/meranamespace unchanged
ubuntu@ip-172-31-46-144:~$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   16d
kube-node-lease   Active   16d
kube-public       Active   16d
kube-system       Active   16d
meranamespace     Active   93s
mynamespace       Active   24m
ubuntu@ip-172-31-46-144:~$
```



Volumes in Kubernetes

A volume in Kubernetes is a directory that can be accessed by containers in a pod. It enables data to persist across container restarts and allows containers within the same pod to share data. Volumes address the ephemeral nature of container storage, making it suitable for applications requiring stateful data or configuration.

Purpose of Volumes

1. **Persistent Storage**: Retain data even if a container restarts.
2. **Data Sharing**: Enable data sharing between containers in the same pod.
3. **Flexible Configuration**: Pass configuration or secrets securely to containers.

Types of Volumes in Kubernetes

1. **EmptyDir**
    - **Definition**: A temporary volume that is created when a pod starts and deleted when the pod stops.
    - **Use Case**: Suitable for temporary data such as caches, scratch space, or intermediate files.
    - **Key Characteristics**:
        - Data is lost when the pod is deleted.
        - All containers in the pod can access the data.

2. **HostPath**
    - **Definition**: A volume that maps a file or directory from the node's filesystem into the pod.
    - **Use Case**: Useful for accessing specific files or directories on the host machine, such as logs or system-level resources.
    - **Key Characteristics**:
        - Tightly coupled with the host node, making pods less portable.
        - Requires careful configuration to avoid security risks.
3. **PersistentVolumeClaim (PVC)**

- **Definition**: A mechanism to request and claim persistent storage resources. PVCs bind to PersistentVolumes, which represent actual storage, such as network storage or cloud-based volumes.
- **Use Case**: Suitable for applications requiring persistent storage that survives pod restarts or re-creation.
- **Key Characteristics**:
  - Decouples storage from pods, making it reusable and portable.
  - Supports dynamic provisioning with storage classes.

4. **ConfigMap**
   - **Definition**: A volume that injects configuration data into containers as files or environment variables.
   - **Use Case**: Ideal for storing non-sensitive configuration data such as environment settings, configuration files, or command-line arguments.
   - **Key Characteristics**:
     - Centralizes configuration management.
     - Enables updates to configuration without restarting containers.
5. **Secret**
   - **Definition**: A volume specifically designed to securely pass sensitive information like passwords, tokens, or API keys to containers.
   - **Use Case**: Essential for securely managing sensitive data in applications.
   - **Key Characteristics**:
     - Encrypted at rest within the Kubernetes cluster.
     - Accessible as environment variables or mounted files in containers.
     - Ensures sensitive data is not hard-coded into container images or application code.

EmptyDir

```
ubuntu@ip-172-31-46-144:~/volumeswala$ #VOLUMES
ubuntu@ip-172-31-46-144:~/volumeswala$ #EMPTY DIRECTORY
ubuntu@ip-172-31-46-144:~/volumeswala$ nano twocontainer.yml
ubuntu@ip-172-31-46-144:~/volumeswala$ kubectl apply -f twocontainer.yml
deployment.apps/mydeployment unchanged
ubuntu@ip-172-31-46-144:~/volumeswala$ kubectl get pod
NAME                            READY   STATUS    RESTARTS   AGE
mydeployment-847bbc4c97-j4tfj   1/1     Running   0          48s
mydeployment-847bbc4c97-q75sg   1/1     Running   0          48s
mydeployment-847bbc4c97-slvvt   1/1     Running   0          48s
ubuntu@ip-172-31-46-144:~/volumeswala$
```


```
ubuntu@ip-172-31-46-144:~/volumeswala$ #VOLUMES
ubuntu@ip-172-31-46-144:~/volumeswala$ #EMPTY DIRECTORY
ubuntu@ip-172-31-46-144:~/volumeswala$ nano twocontainer.yml
ubuntu@ip-172-31-46-144:~/volumeswala$ kubectl apply -f twocontainer.yml
deployment.apps/mydeployment unchanged
ubuntu@ip-172-31-46-144:~/volumeswala$ kubectl get pod
NAME                            READY   STATUS    RESTARTS   AGE
mydeployment-847bbc4c97-j4tfj   1/1     Running   0          48s
mydeployment-847bbc4c97-q75sg   1/1     Running   0          48s
mydeployment-847bbc4c97-slvvt   1/1     Running   0          48s
ubuntu@ip-172-31-46-144:~/volumeswala$
```


HostPath

```
ubuntu@ip-172-31-46-144:~/volumeswala$ #HOSTPATH
ubuntu@ip-172-31-46-144:~/volumeswala$ nano hostnginx.yml
ubuntu@ip-172-31-46-144:~/volumeswala$ kubectl apply -f hostnginx.yml
deployment.apps/mydeployment created
ubuntu@ip-172-31-46-144:~/volumeswala$ kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
mydeployment-7f4d64bc6b-jhbnl   1/1     Running   0          13s
mydeployment-7f4d64bc6b-lhnnd   1/1     Running   0          13s
mydeployment-7f4d64bc6b-ltf5f   1/1     Running   0          13s
ubuntu@ip-172-31-46-144:~/volumeswala$ ls
```

```
ubuntu@ip-172-31-46-144: ~/
  GNU nano 7.2                                  hostnginx.yml *
apiVersion: apps/v1
kind: Deployment
metadata:
   name: mydeployment
spec:
  replicas: 3
  strategy:
     type: RollingUpdate
     rollingUpdate:
         maxSurge: 2
         maxUnavailable: 0
  selector:
   matchLabels:
      env: dev
  template:
       metadata:
          labels:
             env: dev
       spec:
            containers:
                - name: nginxcontainer
                  image: nginx
                  ports:
                    - containerPort: 80
                  volumeMounts:
                   - mountPath: "/usr/share/nginx/html"
                     name: myvolume
            volumes:
               - name: myvolume

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy
```

Nfs

```
ubuntu@ip-172-31-46-144: ~/
ubuntu@ip-172-31-46-144:~/efswala$ ls
test
ubuntu@ip-172-31-46-144:~/efswala$ cd
ubuntu@ip-172-31-46-144:~$ cd volumeswala/
ubuntu@ip-172-31-46-144:~/volumeswala$ sudo nano nfswala.yml
ubuntu@ip-172-31-46-144:~/volumeswala$ kubectl apply -f nfswala.yml
pod/nginx-php-pod created
ubuntu@ip-172-31-46-144:~/volumeswala$ kubectl get pod
NAME            READY   STATUS              RESTARTS   AGE
nginx-php-pod   0/2     ContainerCreating   0          11s
ubuntu@ip-172-31-46-144:~/volumeswala$
```

```
ubuntu@ip-172-31-46-144: ~/
  GNU nano 7.2                                    nfswala.yml
apiVersion: v1                                    nfswala.yml
kind: Pod
metadata:
  name: nginx-php-pod
  labels:
    env: dev
spec:
  containers:
    - name: nginx-container
      image: nginx
      ports:
        - containerPort: 80
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: myvolume

    - name: php-container
      image: php
      ports:
        - containerPort: 90
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: myvolume

  volumes:
    - name: myvolume
      nfs:
        server: fs-0bb29a1ffc4726caa.efs.ap-south-1.amazonaws.com
        path: /test
                                [ Read 29 lines ]
^G Help          ^O Write Out    ^W Where Is     ^K Cut          ^T Execute      ^C Location     M-U Undo        M-A Set Mark
^X Exit          ^R Read File    ^\ Replace      ^U Paste        ^J Justify      ^/ Go To Line   M-E Redo        M-6 Copy
```

Config map

```
ubuntu@ip-172-31-46-144:~$ mkdir configwala
ubuntu@ip-172-31-46-144:~$ cd configwala/
ubuntu@ip-172-31-46-144:~/configwala$ nano mysql.yml
ubuntu@ip-172-31-46-144:~/configwala$ kubectl create configmap mysql-config --from-literal=password=password@123
configmap/mysql-config created
ubuntu@ip-172-31-46-144:~/configwala$ kubectl get configmap
NAME               DATA   AGE
kube-root-ca.crt   1      4h19m
mysql-config       1      32s
ubuntu@ip-172-31-46-144:~/configwala$ kubectl describe configmap mysql-config
Name:         mysql-config
Namespace:    mynamespace
Labels:       <none>
Annotations:  <none>

Data
====
password:
----
password@123

BinaryData
====

Events:  <none>
ubuntu@ip-172-31-46-144:~/configwala$
```

```
  GNU nano 7.2                                    mysql.yml *
apiVersion: v1
kind: Pod
metadata:
  name: mysqlwala
spec:
imagePullSecrets:
    - name: mydockersecret
  containers:
    - name: mysqlwalacontainer
      image: mysql:8.0
      env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
              configMapKeyRef:
                 name: mysql-config
                 key: mysql-password

^G Help          ^O Write Out     ^W Where Is      ^K Cut           ^T Execute       ^C Location      M-U Undo         M-A Set Mark
^X Exit          ^R Read File     ^\ Replace       ^U Paste         ^J Justify       ^/ Go To Line    M-E Redo         M-6 Copy
```

```
ubuntu@ip-172-31-46-144:~/configwala$ nano mysql.yml
ubuntu@ip-172-31-46-144:~/configwala$ kubectl apply -f mysql.yml
pod/test-mysql unchanged
ubuntu@ip-172-31-46-144:~/configwala$ kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
test-mysql   1/1     Running   0          51s
ubuntu@ip-172-31-46-144:~/configwala$
```

By using another imge nginx

```
ubuntu@ip-172-31-46-144:~/configwala$ nano nginxwala.yml
ubuntu@ip-172-31-46-144:~/configwala$ kubectl apply -f nginxwala.yml
pod/nginxwala created
ubuntu@ip-172-31-46-144:~/configwala$ kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
nginxwala    0/1     Pending   0          4s
test-mysql   1/1     Running   0          4m26s
ubuntu@ip-172-31-46-144:~/configwala$
```

Config.file

```
location / {
    root   /usr/share/nginx/html;
    index  index.html index.htm;
}

#error_page  404              /404.html;

# redirect server error pages to the static page /50x.html
#
error_page   500 502 503 504  /50x.html;
location = /50x.html {
    root   /usr/share/nginx/html;
}

 # proxy the PHP scripts to Apache listening on 127.0.0.1:80
#
#location ~ \.php$ {
#    proxy_pass   http://127.0.0.1;
#}

# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
#
```

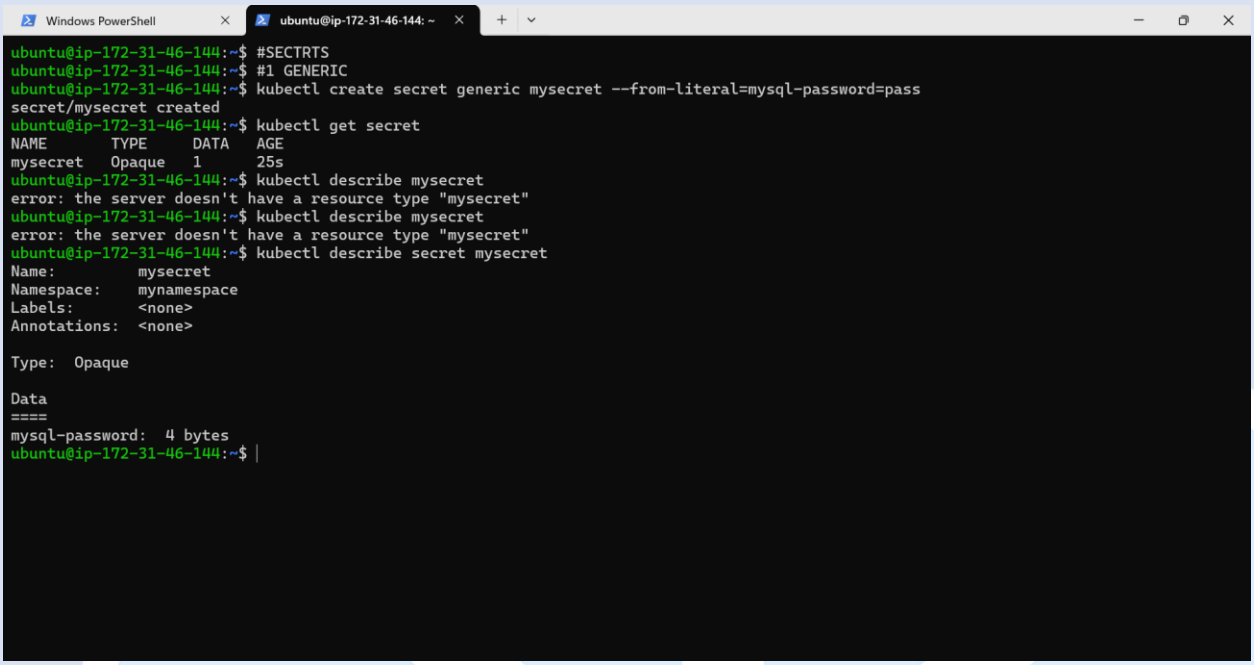
```
ubuntu@ip-172-31-8-103:~/configwala$ nano default.conf
ubuntu@ip-172-31-8-103:~/configwala$ ls
default.conf  mysqlwala.yml  nginxwala.yml
ubuntu@ip-172-31-8-103:~/configwala$ kubectl create configmap nginx-conf --from-file default.conf
configmap/nginx-conf created
ubuntu@ip-172-31-8-103:~/configwala$ kubectl get configmap
NAME               DATA   AGE
kube-root-ca.crt   1      13d
mysql-config       1      24m
nginx-conf         1      8s
ubuntu@ip-172-31-8-103:~/configwala$ kubectl describe configmap nginx-config
```

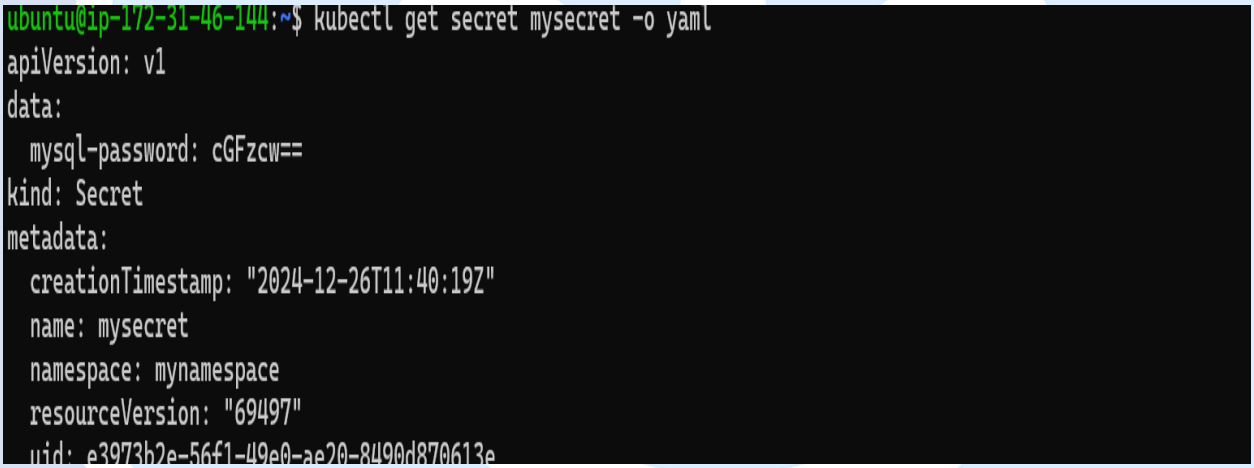
```
 GNU nano 7.2                                  nginxwala.yml *
apiVersion: v1
kind: Pod
metadata:
  name: nginxwala
spec:
  containers:
    - name: nginxwalacontainer
      image: nginx
      volumeMounts:
        - name: nginxconfigvolume
          mountPath: /etc/nginx/conf.d/default.conf
  volumes:
    - name: nginxconfigvolume
      configMap:
        name: nginx-conf
```

```
ubuntu@ip-172-31-8-103:~$ cd configwala/
ubuntu@ip-172-31-8-103:~/configwala$ kubectl apply -f nginxwala.yml
pod/nginxwala created
ubuntu@ip-172-31-8-103:~/configwala$ kubectl get pods
NAME        READY   STATUS              RESTARTS   AGE
nginxwala   0/1     ContainerCreating   0          4s
ubuntu@ip-172-31-8-103:~/configwala$ kubectl describe pod nginxwala
```

```
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   13m
ubuntu@ip-172-31-8-103:~/configwala$ kubectl get configmap
NAME               DATA   AGE
kube-root-ca.crt   1      14d
mysql-config       1      51m
nginx-conf         1      26m
```

Secret

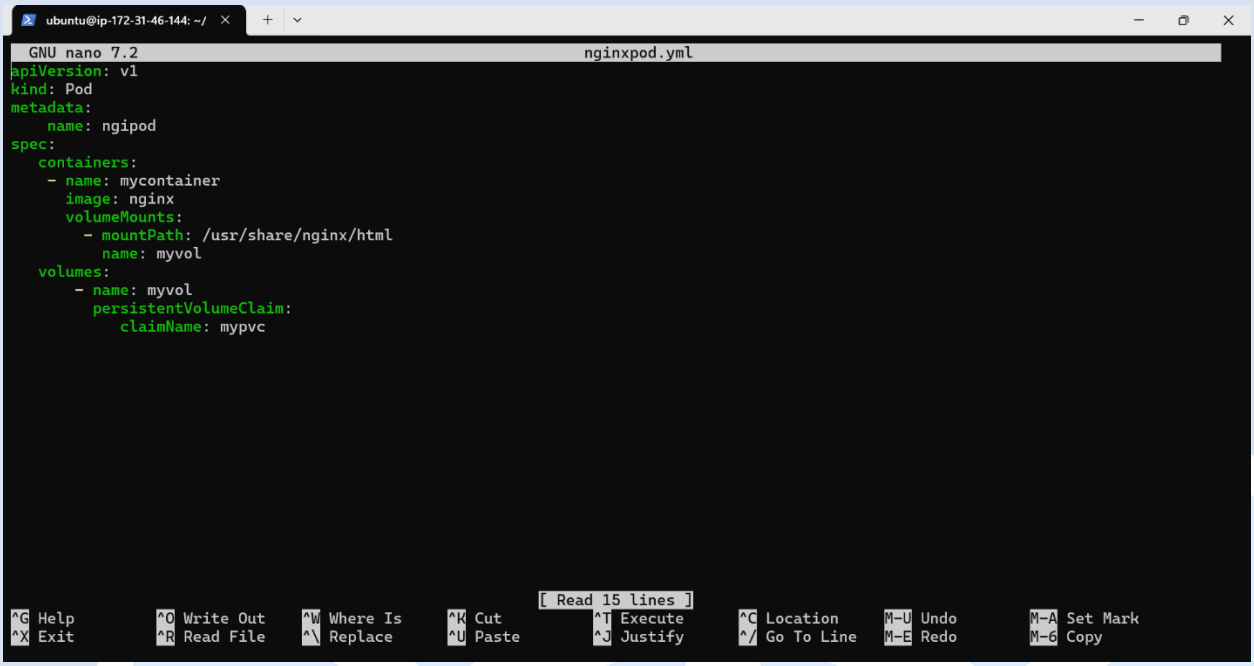

Windows PowerShell
ubuntu@ip-172-31-46-144: ~
ubuntu@ip-172-31-46-144:~$ #SECTRTS
ubuntu@ip-172-31-46-144:~$ #1 GENERIC
ubuntu@ip-172-31-46-144:~$ kubectl create secret generic mysecret --from-literal=mysql-password=pass
secret/mysecret created
ubuntu@ip-172-31-46-144:~$ kubectl get secret
NAME TYPE DATA AGE
mysecret Opaque 1 25s
ubuntu@ip-172-31-46-144:~$ kubectl describe mysecret
error: the server doesn't have a resource type "mysecret"
ubuntu@ip-172-31-46-144:~$ kubectl describe mysecret
error: the server doesn't have a resource type "mysecret"
ubuntu@ip-172-31-46-144:~$ kubectl describe secret mysecret
Name: mysecret
Namespace: mynamespace
Labels: <none>
Annotations: <none>
Type: Opaque
Data
====
mysql-password: 4 bytes
ubuntu@ip-172-31-46-144:~$

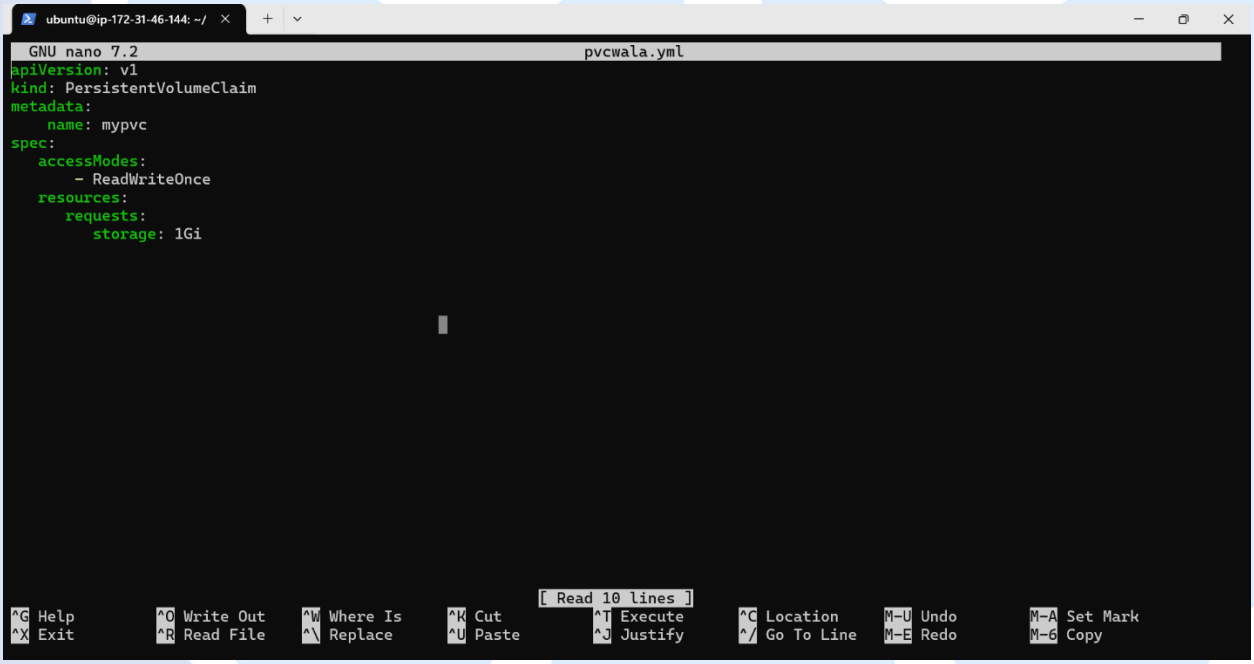

ubuntu@ip-172-31-46-144:~$ kubectl get secret mysecret -o yaml
apiVersion: v1
data:
mysql-password: cGFzcw==
kind: Secret
metadata:
creationTimestamp: "2024-12-26T11:40:19Z"
name: mysecret
namespace: mynamespace
resourceVersion: "69497"

PersistentVolumeClaim (PVC)

```
ubuntu@ip-172-31-46-144:~/pvwala$ #PERSISTENT VOLUME
ubuntu@ip-172-31-46-144:~/pvwala$ nano pvwala.yml
ubuntu@ip-172-31-46-144:~/pvwala$ nano pvcwala.yml
ubuntu@ip-172-31-46-144:~/pvwala$ nano nginxpod.yml
ubuntu@ip-172-31-46-144:~/pvwala$ kubectl apply -f pvwala.yml
persistentvolume/mypv unchanged
ubuntu@ip-172-31-46-144:~/pvwala$ kubectl apply -f pvcwala.yml
persistentvolumeclaim/mypvc unchanged
ubuntu@ip-172-31-46-144:~/pvwala$ kubectl apply -f nginxpod.yml
pod/ngipod configured
ubuntu@ip-172-31-46-144:~/pvwala$ kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM           STORAGECLASS   VOLUMEAT
mypv   1Gi        RWO            Retain           Bound    default/mypvc                  <unset>
ubuntu@ip-172-31-46-144:~/pvwala$ kubectl get pvc
NAME    STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
mypvc   Bound    mypv     1Gi        RWO                           <unset>                 5m12s
ubuntu@ip-172-31-46-144:~/pvwala$ kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM           STORAGECLASS   VOLUMEAT
mypv   1Gi        RWO            Retain           Bound    default/mypvc                  <unset>
ubuntu@ip-172-31-46-144:~/pvwala$ kubectl get pod
NAME     READY   STATUS    RESTARTS   AGE
ngipod   1/1     Running   0          6m23s
ubuntu@ip-172-31-46-144:~/pvwala$
```

```
ubuntu@ip-172-31-46-144: ~/
 GNU nano 7.2                                  pvwala.yml *
apiVersion: v1
kind: PersistentVolume
metadata:
    name: mypv
spec:
   capacity:
      storage: 1Gi
   accessModes:
      - ReadWriteOnce
   persistentVolumeReclaimPolicy: Retain
   hostPpath:
       path: /mydata

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy
```


ubuntu@ip-172-31-46-144: ~/
GNU nano 7.2 nginxpod.yml
apiVersion: v1
kind: Pod
metadata:
name: ngipod
spec:
containers:
- name: mycontainer
image: nginx
volumeMounts:
- mountPath: /usr/share/nginx/html
name: myvol
volumes:
- name: myvol
persistentVolumeClaim:
claimName: mypvc
[Read 15 lines]
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-6 Copy


ubuntu@ip-172-31-46-144: ~/
GNU nano 7.2 pvcwala.yml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: mypvc
spec:
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 1Gi
[Read 10 lines]
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-6 Copy

DaemonSet in Kubernetes

A **DaemonSet** in Kubernetes is a controller that ensures a copy of a specific pod runs on all (or selected) nodes in the cluster. It is typically used to deploy background system-level services or applications that need to run on every node.

Purpose of DaemonSet

1. **System Monitoring**: Deploying monitoring agents (e.g., Prometheus Node Exporter, Fluentd).
2. **Log Aggregation**: Running log collection agents on every node.
3. **Networking**: Setting up network components such as DNS services or proxies (e.g., CoreDNS, kube-proxy).
4. **Custom Node Tasks**: Running tasks specific to each node, such as node health checks or backups.

Key Features of DaemonSet

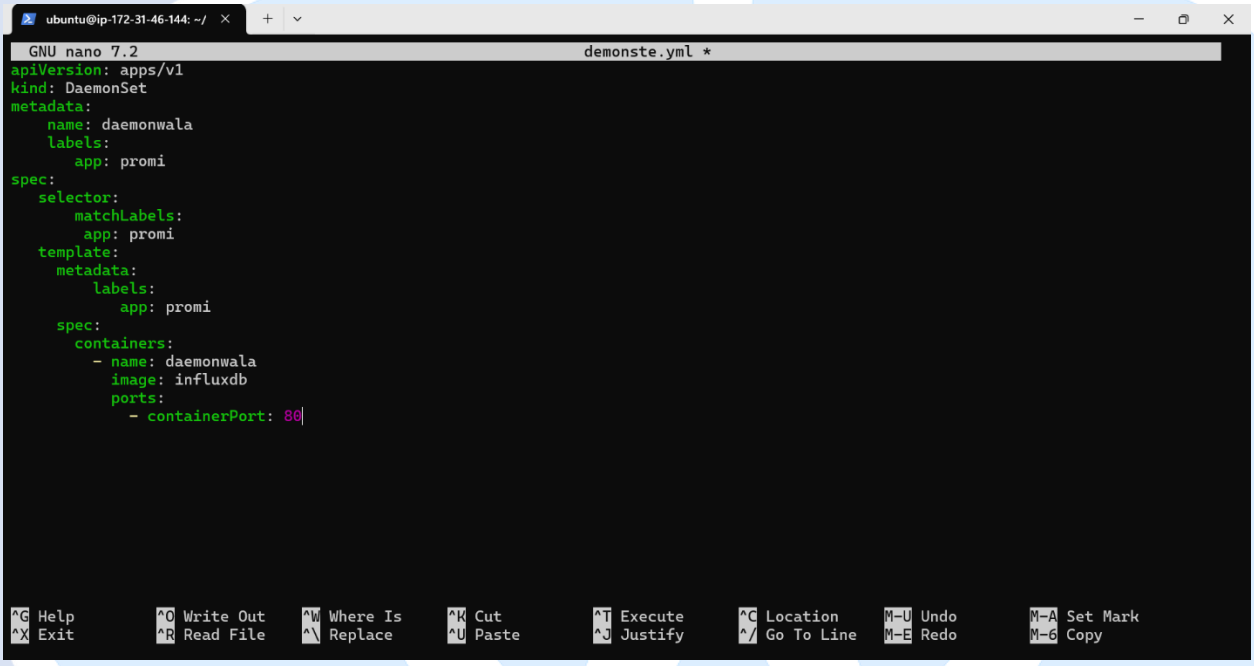
1. **One Pod per Node**: Ensures exactly one pod is running per eligible node.
2. **Node Selection**: Can restrict pods to specific nodes using labels and node selectors.
3. **Automatic Updates**: New nodes automatically get the DaemonSet pods upon joining the cluster.
4. **Controlled Deletion**: When deleted, the associated pods are removed from the nodes.

How DaemonSet Works

1. **Deployment**:
    - DaemonSets deploy pods based on a defined configuration, and Kubernetes ensures that one pod is scheduled per node.
2. **Targeting Nodes**:
    - **Node Selector**: Specifies which nodes the DaemonSet should target.
    - **Taints and Tolerations**: Configures pods to tolerate specific taints on nodes.
    - **Affinity Rules**: Fine-tunes node selection using affinity and anti-affinity rules.
3. **Updates**:
    - Rolling updates can be performed to update the pods managed by a DaemonSet.

Creating a DaemonSet

```
ubuntu@ip-172-31-46-144:~$ #DEMONSET
ubuntu@ip-172-31-46-144:~$ mkdir demonset
ubuntu@ip-172-31-46-144:~$ cd demonset
ubuntu@ip-172-31-46-144:~/demonset$ nano demonste.yml
ubuntu@ip-172-31-46-144:~/demonset$ kubectl apply -f demonste.yml
daemonset.apps/daemonwala created
ubuntu@ip-172-31-46-144:~/demonset$ kubectl get demonset
error: the server doesn't have a resource type "demonset"
ubuntu@ip-172-31-46-144:~/demonset$ kubectl get daemonset
NAME         DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
daemonwala   1         1         1       1            1           <none>          110s
ubuntu@ip-172-31-46-144:~/demonset$ kubectl get ds
NAME         DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
daemonwala   1         1         1       1            1           <none>          3m8s
ubuntu@ip-172-31-46-144:~/demonset$
```



Jobs in Kubernetes

A **Job** in Kubernetes is a controller that ensures a specified number of pods successfully complete a task. Unlike Deployments, which maintain a set number of running pods, Jobs are used for batch processing and tasks that have a definite completion point. Once the task is completed, the pods are terminated.

Purpose of Jobs

1. **Batch Processing**: Running tasks that need to be completed once (e.g., data processing, backups).
2. **One-Time Operations**: Executing one-off tasks that don't need to be continuously running.
3. **Data Migration**: Performing tasks like database migration or initialization.
4. **ETL Tasks**: Running Extract, Transform, Load (ETL) jobs that are periodic or triggered.

Key Features of Jobs

1. **Task Completion**: Jobs ensure that a task is successfully completed, retrying pods in case of failure.
2. **Pod Management**: Creates one or more pods and tracks their completion.
3. **Parallel Execution**: Supports running multiple pods in parallel to complete the task faster.
4. **Pod Retry Mechanism**: Automatically retries failed pods until the desired number of completions is achieved.
5. **Graceful Termination**: When the task completes, the associated pods are terminated.

How Jobs Work

1. **Specifying Desired Completions**:
    - A Job will continue running pods until the specified number of completions is reached.
    - This number can be defined using the completions field in the Job's specification.
2. **Pod Management**:
    - The Job controller creates pods to run the specified task and monitors their completion.

Once the task is completed successfully, the pod terminates.

3. **Retry Mechanism**:
    - If a pod fails to complete the task, the Job controller will retry the pod. The number of retries can be controlled via the backoffLimit field.

4. **Parallelism:**

   - **Jobs can be configured to run multiple pods simultaneously using the parallelism field. This can speed up completion when the task is divisible**

```
ubuntu@ip-172-31-46-144:~$ #JoB
ubuntu@ip-172-31-46-144:~$ mkdir jobwala
ubuntu@ip-172-31-46-144:~$ cd jobwala/
ubuntu@ip-172-31-46-144:~/jobwala$ nano job.yml
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl apply -f job.yml
job.batch/hellojob created
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl get job
NAME       STATUS     COMPLETIONS   DURATION   AGE
hellojob   Complete   1/1           8s         30s
```

```
ubuntu@ip-172-31-46-144: ~/
 GNU nano 7.2                                job.yml *
apiVersion: batch/v1
kind: Job
metadata:
   name: hellojob
spec:
   template:
      metadata:
         labels:
           app: hello
      spec:
        containers:
          - name: hello
            image: busybox
            command: ["echo","hello,k8s"]
        restartPolicy: Never
   backoffLimit: 4

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy
```

```
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl logs hellojob-k7krb
hello,k8s
ubuntu@ip-172-31-46-144:~/jobwala$
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl logs hellojob-k7krb
hello,k8s
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl describe pod hellojob-k7krb
Name:             hellojob-k7krb
Namespace:        default
Priority:         0
Service Account:  default
Node:             ip-172-31-0-244/172.31.0.244
Start Time:       Thu, 26 Dec 2024 17:01:08 +0000
Labels:           app=hello
```

```
ubuntu@ip-172-31-46-144:~$ #JoB
ubuntu@ip-172-31-46-144:~$ mkdir jobwala
ubuntu@ip-172-31-46-144:~$ cd jobwala/
ubuntu@ip-172-31-46-144:~/jobwala$ nano job.yml
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl apply -f job.yml
job.batch/hellojob created
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl get job
NAME       STATUS     COMPLETIONS   DURATION   AGE
hellojob   Complete   1/1           8s         30s
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl log hellojob
error: unknown command "log" for "kubectl"

Did you mean this?
        top
        logs
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl logs hellojob
error: error from server (NotFound): pods "hellojob" not found in namespace "default"
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl get pod
NAME              READY   STATUS      RESTARTS   AGE
hellojob-k7krb    0/1     Completed   0          108s
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl logs hellojob-k7krb
hello,k8s
ubuntu@ip-172-31-46-144:~/jobwala$
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl logs hellojob-k7krb
hello,k8s
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl describe pod hellojob-k7krb
Name:             hellojob-k7krb
Namespace:        default
Priority:         0
Service Account:  default
Node:             ip-172-31-0-244/172.31.0.244
Start Time:       Thu, 26 Dec 2024 17:01:08 +0000
Labels:           app=hello
```

```
 GNU nano 7.2                                  jobwala.yml *
apiVersion: batch/v1
kind: Job
metadata:
   name: hellojob
spec:
   completions: 5
   parallelism: 2
   template:
      metadata:
         labels:
           app: hello
      spec:
        containers:
          - name: hello
            image: busybox
            command: ["echo","hello,k8s"]
        restartPolicy: Never
   backoffLimit: 4
```

```
ubuntu@ip-172-31-8-103:~$ kubectl apply -f daemonwala/jobwala.yml
job.batch/hellojob created
ubuntu@ip-172-31-8-103:~$ kubectl get pods
NAME              READY   STATUS              RESTARTS   AGE
hellojob-8xjp2    0/1     ContainerCreating   0          1s
hellojob-98781    0/1     Completed           0          5s
hellojob-rmfkd    0/1     ContainerCreating   0          1s
hellojob-zv6tz    0/1     Completed           0          5s
ubuntu@ip-172-31-8-103:~$ kubectl get pods
NAME              READY   STATUS      RESTARTS   AGE
hellojob-8xjp2    0/1     Completed   0          29s
hellojob-98781    0/1     Completed   0          33s
hellojob-rmfkd    0/1     Completed   0          29s
hellojob-xx9t7    0/1     Completed   0          25s
hellojob-zv6tz    0/1     Completed   0          33s
ubuntu@ip-172-31-8-103:~$ kubectl logs hellojob-zv6tz
hello,k8s
```

CronJob in Kubernetes

A **CronJob** in Kubernetes is a resource used to run jobs on a scheduled basis, much like cron jobs in Linux. It allows you to run batch jobs or tasks at specific times or intervals, providing powerful scheduling capabilities in Kubernetes.

Purpose of CronJob

1. **Scheduled Tasks**: Run tasks at regular intervals, such as backups, database cleanup, and periodic reporting.
2. **Periodic Jobs**: Automate recurring tasks that need to be executed on a schedule.
3. **Task Automation**: Eliminate the need for manual intervention for repetitive tasks.

How CronJob Works

1. **Cron-like Scheduling**:
    - CronJobs are based on a cron expression (a string that defines time intervals), which specifies when the job should run.
    - The cron format is: * * * * * (minute, hour, day of month, month, day of week).
2. **Job Creation**:
    - CronJobs automatically create Jobs according to the schedule defined by the cron expression.
    - The job runs at the specified time and completes its task.
3. **Job Execution**:
    - The CronJob creates a new Job and ensures that the associated task (specified in the Job template) is executed as a pod.
    - After the pod completes successfully, the job is considered finished.
4. **Concurrency Policy**:
    - CronJobs allow you to control how multiple jobs should be handled when the scheduled time occurs:
        - **Allow**: Allows multiple jobs to run concurrently.
        - **Forbid**: Prevents a new job from starting if the previous one hasn't completed.
        - **Replace**: Terminates any currently running job and starts a new one.

5. **Time Zone**:
    - Kubernetes CronJobs use UTC time by default for scheduling.
    - To handle time zone-specific schedules, you need to manage it outside of Kubernetes or use external tools.

Creating a CronJob

*** * * * ***

| | | | |

| | | | +---- Day of week (0 - 7) (Sunday = 0 or 7)

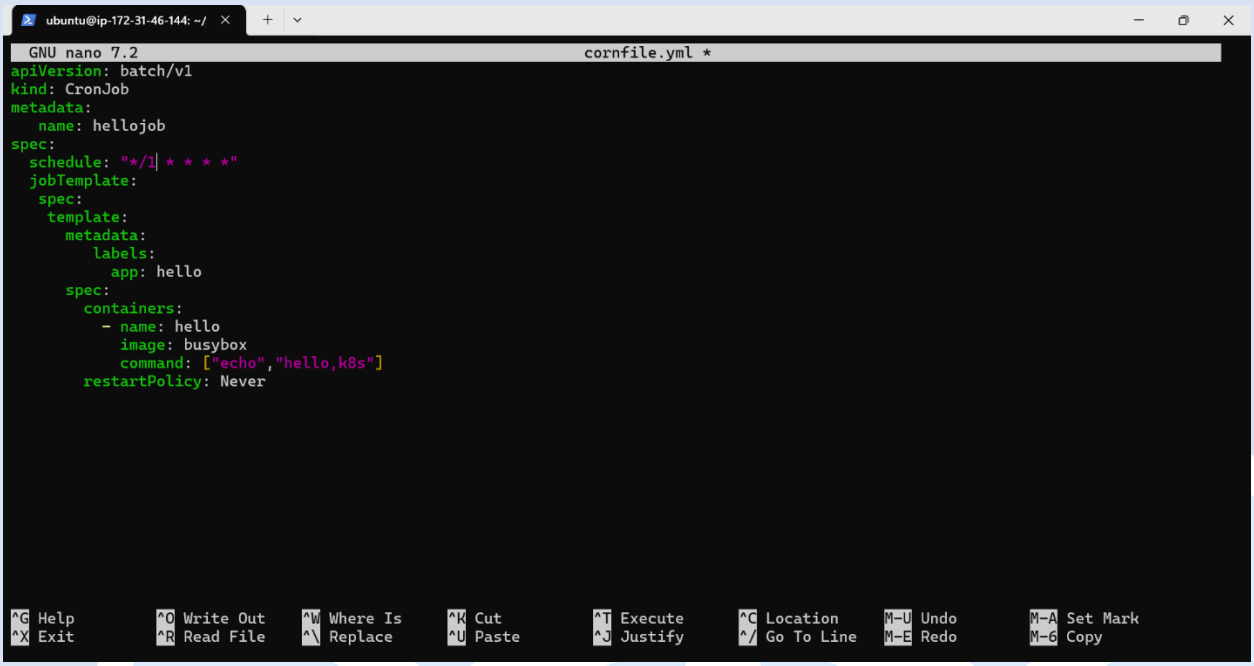
| | | +------ Month (1 - 12)

| | +-------- Day of month (1 - 31)

| +---------- Hour (0 - 23)

+------------ Minute (0 - 59)

```
ubuntu@ip-172-31-46-144:~/jobwala$ #cornjob
ubuntu@ip-172-31-46-144:~/jobwala$ nano cornfile.yml
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl apply -f cornfile.yml
cronjob.batch/hellojob unchanged
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl get pod
NAME                        READY   STATUS      RESTARTS   AGE
hellojob-28920591-mkhfz     0/1     Completed   0          97s
hellojob-28920592-6rks9     0/1     Completed   0          37s
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl get cornjob
error: the server doesn't have a resource type "cornjob"
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl get job
NAME               STATUS     COMPLETIONS   DURATION   AGE
hellojob-28920591   Complete   1/1           5s         2m19s
hellojob-28920592   Complete   1/1           4s         79s
hellojob-28920593   Complete   1/1           4s         19s
ubuntu@ip-172-31-46-144:~/jobwala$ kubectl get pods
NAME                        READY   STATUS      RESTARTS   AGE
hellojob-28920591-mkhfz     0/1     Completed   0          2m36s
hellojob-28920592-6rks9     0/1     Completed   0          96s
hellojob-28920593-86tbl     0/1     Completed   0          36s
ubuntu@ip-172-31-46-144:~/jobwala$
```



StatefulSet in Kubernetes

A **StatefulSet** in Kubernetes is used to manage **stateful applications** that need stable identities and persistent storage. This is different from a regular deployment where pods are stateless. StatefulSets are ideal for applications like databases that require data to be saved and remain consistent even if the pod is restarted or moved.

Why Use StatefulSet?

1. **Stable Identity**: Each pod in a StatefulSet gets a unique name (like mysql-0, mysql-1, etc.) that doesn't change, even if the pod restarts.
2. **Persistent Storage**: Each pod gets its own dedicated storage that is preserved even if the pod is rescheduled to another node.
3. **Order of Deployment**: Pods are created, updated, and deleted in a specific order. This is helpful for apps that need to start in a particular sequence (e.g., databases).
4. **Scaling**: Pods in a StatefulSet are scaled in order (first pod-0, then pod-1, etc.), and scaling down happens in reverse order.

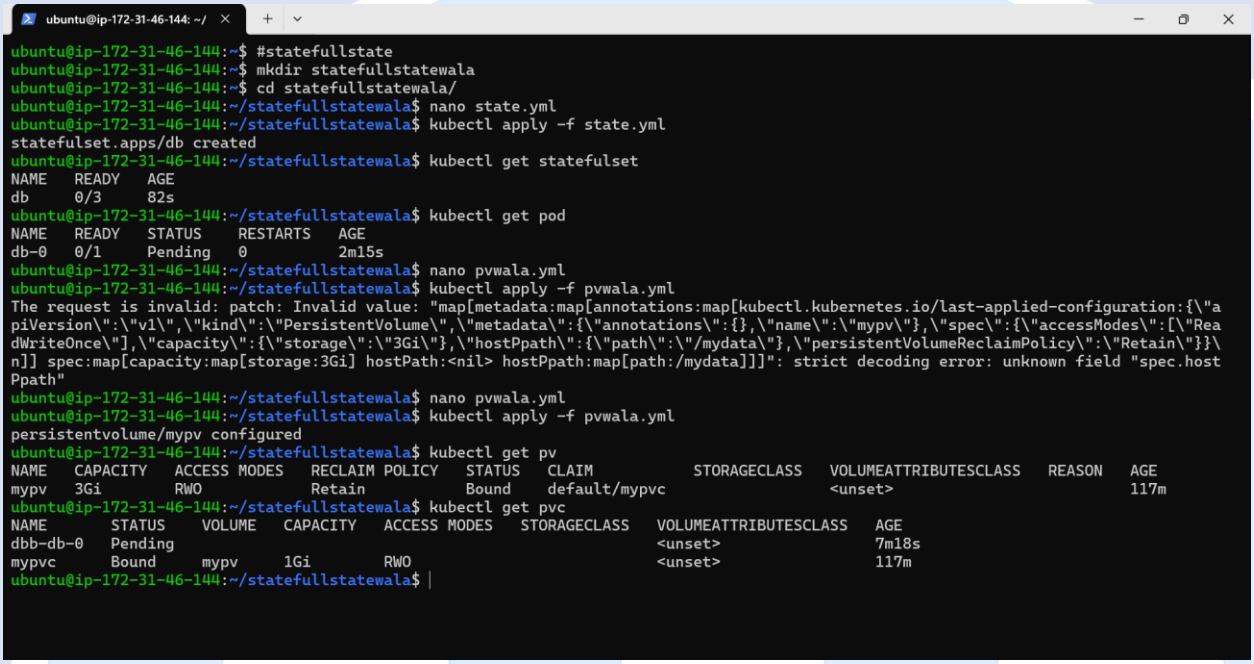
How Does StatefulSet Work?

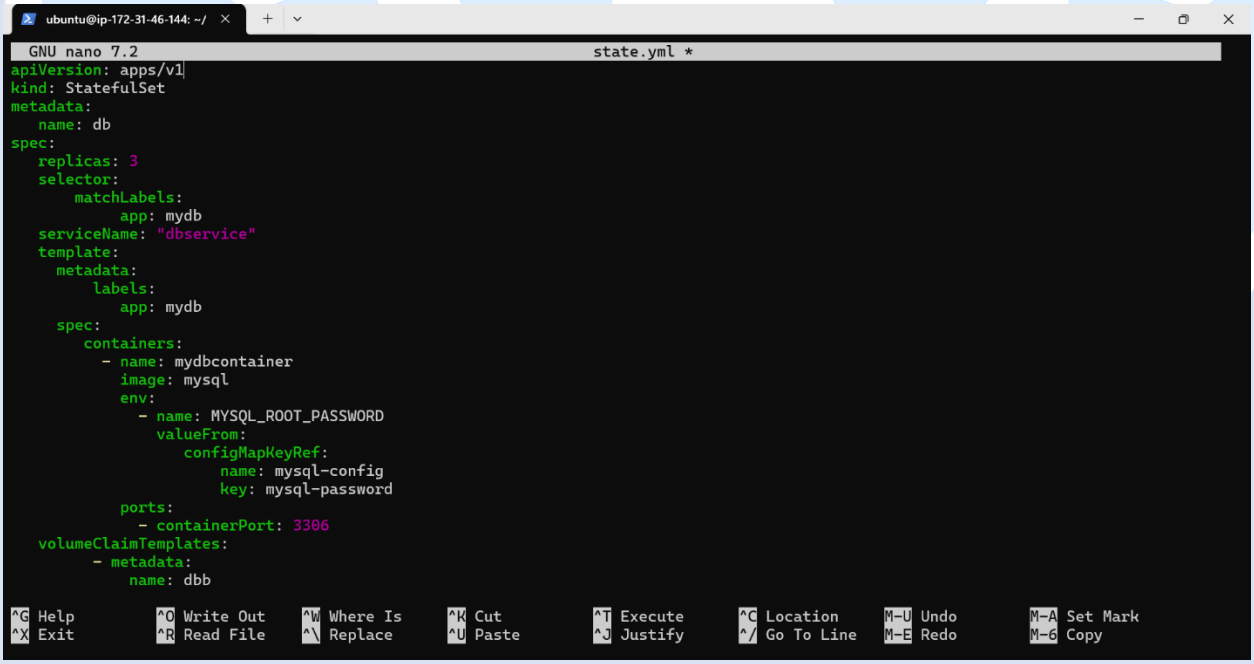
1. **Stable Pod Names**: Pods in a StatefulSet get stable, unique names that help other services identify them easily.

2. **Storage Persistence**: Each pod has a **PersistentVolume** associated with it, which ensures the data is retained even when pods are deleted or moved.

3. **Ordered Deployment and Shutdown**: Pods are created and deleted in a specific order, which is important for some applications (like databases) where certain pods need to start before others.

4. **Rolling Updates**: When updating a StatefulSet, Kubernetes updates the pods one at a time, ensuring minimal disruption.





Horizontal Pod Autoscaler

Horizontal Pod Autoscaler (HPA) in Kubernetes automatically adjusts the number of pods in a deployment based on resource usage (like CPU, memory) or custom metrics. It ensures efficient resource utilization and handles varying workloads effectively.

Key Points:

- Scales pods **up** or **down** based on demand.
- Uses **Metrics Server** or custom metrics for scaling decisions.
- Configurable via YAML or CL

- **Scales Pods**: Automatically increases or decreases the number of pods based on application demand.
- **Metrics-Based**: Relies on **Metrics Server** or custom metrics (e.g., CPU, memory, or external metrics).
- **Easy Configuration**: Managed using YAML files or Kubernetes CLI commands for flexibility.

```
ubuntu@ip-172-31-46-144:~$ #horizantal pod auto scaler
ubuntu@ip-172-31-46-144:~$ #install the metric server
ubuntu@ip-172-31-46-144:~$ nano components.yml
ubuntu@ip-172-31-46-144:~$ kubectl apply -f components.yml
serviceaccount/metrics-server created
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
clusterrole.rbac.authorization.k8s.io/system:metrics-server created
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
service/metrics-server created
deployment.apps/metrics-server created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
```

```
ubuntu@ip-172-31-46-144:~$ kubectl get pod -n kube-system
NAME                                       READY   STATUS             RESTARTS          AGE
calico-kube-controllers-6cdb97b867-4rvc2   1/1     Running            9 (12m ago)       19d
calico-node-grzcj                          0/1     CrashLoopBackOff   167 (34s ago)     19d
calico-node-vzr9d                          0/1     Running            156 (3m51s ago)   19d
coredns-55cb58b774-6x772                   1/1     Running            9 (12m ago)       19d
coredns-55cb58b774-qlvx6                   1/1     Running            2 (12m ago)       3d15h
etcd-ip-172-31-46-144                      1/1     Running            10 (12m ago)      19d
kube-apiserver-ip-172-31-46-144            1/1     Running            10 (12m ago)      19d
kube-controller-manager-ip-172-31-46-144   1/1     Running            10 (12m ago)      19d
kube-proxy-k5fw5                           0/1     CrashLoopBackOff   159 (19s ago)     19d
kube-proxy-ppmk4                           0/1     CrashLoopBackOff   171 (58s ago)     19d
kube-scheduler-ip-172-31-46-144            1/1     Running            10 (12m ago)      19d
metrics-server-554cf459c5-496vn            1/1     Running            0                 2m34s
ubuntu@ip-172-31-46-144:~$
```

ubuntu@ip-172-31-0-244: ~ | ubuntu@ip-172-31-46-144: ~/

```
 GNU nano 7.2                                        mynginx.yml *
apiVersion: apps/v1
kind: Deployment
metadata:
   name: mydeployment
spec:
  replicas: 1
  strategy:
     type: RollingUpdate
     rollingUpdate:
         maxSurge: 2
         maxUnavailable: 0
  selector:
   matchLabels:
      env: dev
  template:
       metadata:
          labels:
              env: dev
       spec:
            containers:
                - name: nginxcontaine
                  image: nginx
                  ports:
                    - containerPort: 80
                  resources:
                        requests:
                          cpu: "100m"
                        limits:
                          cpu: "500m"

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy
```

```
ubuntu@ip-172-31-46-144:~$ #deploy and application
ubuntu@ip-172-31-46-144:~$ mkdir autoscalingdeploymentwala
ubuntu@ip-172-31-46-144:~$ cd autoscalingdeploymentwala/
ubuntu@ip-172-31-46-144:~/autoscalingdeploymentwala$ nano mynginx.yml
ubuntu@ip-172-31-46-144:~/autoscalingdeploymentwala$ ubuntu@ip-172-31-46-144:~/autoscalingdeploymentwala$ kubectl apply -f mynginx.ym
l
deployment.apps/mydeployment created
ubuntu@ip-172-31-46-144:~/autoscalingdeploymentwala$ kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
mydeployment-545fc4bd99-2fxtw   1/1     Running   0          10s
ubuntu@ip-172-31-46-144:~/autoscalingdeploymentwala$
```

```
ubuntu@ip-172-31-46-144:~/autoscalingdeploymentwala$ # to incresed the traffic
ubuntu@ip-172-31-46-144:~/autoscalingdeploymentwala$ kubectl run trafficpod --image=busybox -- /bin/sh -c "while true; do wget -q -O
- http://172.31.0.244; sleep 5; done"
pod/trafficpod created
ubuntu@ip-172-31-46-144:~/autoscalingdeploymentwala$ kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
mydeployment-545fc4bd99-2fxtw   1/1     Running   0          13m
mydeployment-545fc4bd99-bjgnp   1/1     Running   0          8m31s
trafficpod                    1/1     Running   0          19s
ubuntu@ip-172-31-46-144:~/autoscalingdeploymentwala$
```

An **Ingress** in Kubernetes is an API object that manages external access to services within a cluster, typically HTTP and HTTPS traffic. It acts as a smart router, allowing you to define rules for directing traffic to the appropriate backend services.

Key Features of Ingress:

- **HTTP/HTTPS Routing**: Routes requests to services based on hostnames, paths, or both.
- **Load Balancing**: Distributes traffic across multiple pods.
- **TLS Termination**: Supports SSL/TLS for secure communication.
- **Custom Rules**: Define rules for complex traffic management.

This example routes traffic based on the hostname and path:

```
ubuntu@ip-172-31-46-144:~/ingresswala$ #Ingress
ubuntu@ip-172-31-46-144:~/ingresswala$ nano myingtrss.yml
ubuntu@ip-172-31-46-144:~/ingresswala$ kubectl apply -f myingtrss.yml
ingress.networking.k8s.io/myingress unchanged
ubuntu@ip-172-31-46-144:~/ingresswala$
```

Windows PowerShell | ubuntu@ip-172-31-46-144: ~/

```
 GNU nano 7.2                                    myingtrss.yml *
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myingress
spec:
  rules:
   - host: tujanena.com
     http:
      paths:
       - path: /
         pathType: Prefix
         backend:
          service:
            name: myservice1
            port:
              number: 80
   - host: example.com
     http:
      paths:
       - path: /
         pathType: Prefix
         backend:
          service:
            name: myservice2
            port:
              number: 80

^G Help          ^O Write Out     ^W Where Is      ^K Cut           ^T Execute       ^C Location      M-U Undo         M-A Set Mark
^X Exit          ^R Read File     ^\ Replace       ^U Paste         ^J Justify       ^/ Go To Line    M-E Redo         M-6 Copy
```

THE END