



kubernetes



1. Overview Docker and Kubernates

2. Kubernates Architecture

3. Kubernates Cluster types

4. Kubernates types of Services

5. minikube and yml Setup

★ (Not recommended)

6. EX 1: Create pod using.yml

7. EX 2: Create pod using nodeport using.yml

8. namespace

9. we can craete namespace by using.yml file

10. we can craete namespace + pod by using.yml file

11. k8S Resources - 🤖 ReplicationController (RC)

12. k8S Resources - 🤖 ReplicaSet

13. k8S Resources - 🤖 Deployment in Kubernetes

14. EKS in AWS



kubernetes

Docker & Kubernetes Overview

What is Docker?

Docker is a free and open-source containerization software that allows you to package applications along with their dependencies into a single unit called a Docker Image. This image can run on any system that has Docker installed, making deployment easy and consistent across different environments.

Why use Docker?

- Portability – Run the same application on any machine, regardless of OS configuration.
- Dependency Management – Ensures that all required software (e.g., libraries, databases, and runtimes) is included within the image.
- Fast Deployment – No need to manually install dependencies every time you set up a new environment.
- Resource Efficiency – Uses fewer resources compared to traditional virtual machines.

How Docker Works?

Create a Docker Image – Package the app code + dependencies into a lightweight container image.

Run the Docker Container – Deploy this image as a container using the docker run command.

Execute Anywhere – The same image runs on any machine with Docker installed.

Once the image is built, it can run on any machine without requiring additional software setup.

Kubernetes (Container Orchestration Software)

What is Kubernetes?

Kubernetes (K8s) is a free and open-source orchestration tool developed by Google to manage containerized applications.

 **Orchestration** = Managing multiple containers efficiently

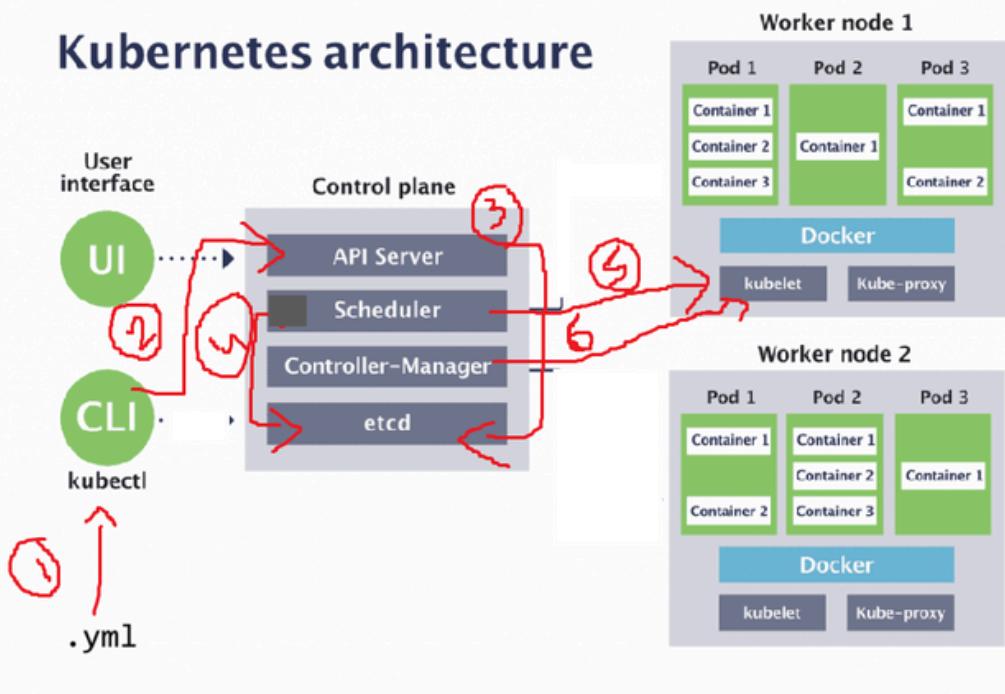
Kubernetes automates key tasks like:

1. Creating, starting, and stopping containers.
2. Scaling up/down based on demand.
3. Handling failures automatically.

Why use Kubernetes?

- Orchestration** – Efficiently manages multiple containers across a cluster of machines.
- Self-Healing** – If a container crashes, Kubernetes automatically replaces it.
- Load Balancing** – Distributes traffic across multiple containers to avoid overloading.
- Auto Scaling** – Increases or decreases the number of running containers based on traffic load.
- Automated Deployments** – Supports CI/CD for rolling updates and version control.

Kubernetes architecture



1. `kubectl read .yml/command file`
2. Then `kubectl` sends `.yml` details to `API Server`
3. Then `API Server` stores `.yml` details in `etcd`
4. Then `Scheduler` reads `etcd`
5. `Scheduler& Controller-Manager` it acts input `kublet`
6. `Controller-Manager` which will help us to do all pods and containers created or not it will check

Kubelet After input of 5 and 6 The Kubelet ensures the worker node is running the assigned workload

Kube Proxy The Kube Proxy manages networking for seamless cluster communication.

As we know The **Controller Manager** continuously monitors the cluster to ensure tasks run correctly.

Kubernetes Cluster Types

A Kubernetes Cluster is a group of servers working together to run containerized applications. It can be set up in two main ways:

A Kubernetes Cluster = Control Plane + Worker Nodes + Pods + Resources + Networking + Storage

1) Self-Managed Cluster

In this setup, we manually install and manage Kubernetes on our own infrastructure.

a) MiniKube (Single Node)

- > Runs a single-node cluster on a local machine.
- > Best for learning and practicing Kubernetes concepts.
- > Not suitable for production as it lacks high availability and scalability.

b) Kubeadm (Multi-Node)

- > A tool for setting up a multi-node cluster manually.
- > Requires configuring the control plane, worker nodes, and networking.
- > Gives full control over the cluster but requires deep Kubernetes expertise.
- > Used for on-premise or customized Kubernetes deployments.

2) Cloud Provider-Managed Cluster (Pre-configured, ready-to-use)

Cloud providers offer fully managed Kubernetes services, where they handle cluster maintenance, updates, and availability.

a) AWS EKS (Elastic Kubernetes Service)

- > A managed Kubernetes service on Amazon Web Services.

b) Azure AKS (Azure Kubernetes Service)

- > Microsoft Azure's managed Kubernetes offering.

c) GCP GKE (Google Kubernetes Engine)

- > Google Cloud's fully managed Kubernetes solution.

Kubernetes Services

-> A Kubernetes Service is used to expose a group of Pods so that they can be accessed reliably. Since Pods can be created and destroyed at any time (with changing IPs), a Service gives them a stable network identity.

Why Do We Use Services?

- > Pods are short-lived and can crash or restart.
 - > Each time a Pod is created, it gets a new IP address.
 - > Directly accessing Pods via IP is not reliable.
 - > A Service gives a static IP to a group of Pods.
-

Types of Kubernetes Services

Kubernetes offers different types of services depending on how and where you want to expose your Pods:

- ◆ 1. ClusterIP (Default)
- ◆ 2. NodePort
- ◆ 3. LoadBalancer

ClusterIP Service (Internal Access Only)

Key Points:

- > Pods are short-lived objects; if one crashes, Kubernetes replaces it with a new Pod.
 - > Every new Pod gets a different IP address.
- Note:  Never rely on Pod IPs to access an application.
- > A ClusterIP Service groups multiple Pods and assigns them a single static IP.
 - > This static IP allows other components inside the cluster to access the group of Pods reliably, even when individual Pods change.

Access Scope:

- > Only accessible within the Kubernetes cluster.
- > Not reachable from the outside world (internet or external clients).

Use Case:

- > Internal services such as databases, backend APIs, authentication services, etc.
- Example: You don't want to expose a database Pod to the internet, so you use a ClusterIP service to allow access only from other internal Pods.

What is a NodePort Service in Kubernetes?

- > A NodePort service is a type of Kubernetes Service that exposes your Pods outside the cluster using a port on each worker node (called a "NodePort").

Why Use NodePort?

By default, Pods and ClusterIP services are only accessible within the cluster.

NodePort makes them accessible externally by opening a static port (from 30000 to 32767) on each worker node.

It allows you to access your application using:

`http://<NodeIP>:<NodePort>`

Note: Here all traffic is routed to one worker node. Means load balancing cannot happen here.

What is a LoadBalancer Service in Kubernetes?

- > It not only provides external access to your app but also handles automatic traffic distribution across the backend Pods running on different worker nodes.

Step-1 : Setup Linux VM

Login into AWS Cloud account

Create Linux VM with Ubuntu AMI (t2.medium or t3.medium)

Select Storage as 50 GB (Default is 8 GB only for Linux)

Create Linux VM and connect to it using SSH Client

The screenshot shows the AWS EC2 Instances Launch an Instance page. In the 'Amazon Machine Image (AMI)' section, 'Ubuntu Server 24.04 LTS (HVM), SSD Volume Type' is selected. Below it, the 'Free tier eligible' status is shown. In the 'Instance type' section, 't2.medium' is selected. Other configuration options like Firewall (security group), Storage (volumes), and a note about the free tier are visible. A 'Launch instance' button is at the bottom right.

This screenshot shows the same EC2 launch page with more detailed configurations. Under 'Instance type', 't2.medium' is selected. The 'Key pair (login)' section shows 'Ssh'. The 'Network settings' section includes 'Network' (Info) and 'Subnet' (Info). A note about the free tier is present. The 'Launch instance' button is visible.

The final screenshot shows the EC2 launch page with storage configuration. It specifies a 50 GB gp3 root volume. Advanced options like 'Configure storage' and 'Advanced details' are visible. A note about the free tier is present. The 'Launch instance' button is highlighted in red at the bottom right.

click create/launch instance

Step-2 : Install Docker In Ubuntu VM

```
sudo apt update  
curl -fsSL get.docker.com | /bin/bash  
sudo usermod -aG docker ubuntu  
exit
```

```
ubuntu@ip-172-31-47-144:~$ sudo apt update  
curl -fsSL get.docker.com | /bin/bash  
sudo usermod -aG docker ubuntu  
exit
```

Step-3 : Updating system packages and installing Minikube dependencies

```
sudo apt update  
sudo apt install -y curl wget apt-transport-https
```

```
ubuntu@ip-172-31-47-144:~$ sudo apt update  
sudo apt install -y curl wget apt-transport-https  
Hit:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble InRelease  
Hit:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease  
Hit:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease  
Hit:4 https://download.docker.com/linux/ubuntu noble InRelease  
0% [Connecting to security.ubuntu.com (185.125.190.82)]
```

Step-4 : Installing Minikube

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
```

```
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

```
minikube version
```

```
ubuntu@ip-172-31-47-144:~$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
ubuntu@ip-172-31-47-144:~$ sudo install minikube-linux-amd64 /usr/local/bin/minikube
minikube version
```

```
Type here to search 7:56 PM
S 4/14/2025
```

S

Step-5 : Install Kubectl (Kubernetes Client)

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
kubectl version -o yaml
```

```
ubuntu@ip-172-31-47-144:~$ curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
ubuntu@ip-172-31-47-144:~$ chmod +x kubectl
ubuntu@ip-172-31-47-144:~$ sudo mv kubectl /usr/local/bin/
ubuntu@ip-172-31-47-144:~$ kubectl version -o yaml
```

```
classplusapp.com is sharing your screen.
```

Step-6 : Start MiniKube Server

```
minikube start – driver=docker
```

```
ubuntu@ip-172-31-47-144:~$ minikube start – driver=docker
* minikube v1.35.0 on Ubuntu 24.04 (xen/amd64)
* Automatically selected the docker driver. Other choices: none, ssh
* Using Docker driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.46 ...
```

Step-7 : Check MiniKube status

```
minikube status
```

```
ubuntu@ip-172-31-47-144:~$ minikube status
minikube
  type: Control Plane
  host: Running
  kubelet: Running
  apiserver: Running
  kubeconfig: Configured
```

```
ubuntu@ip-172-31-47-144:~$ |
```

Step-8 : Access K8S Cluster

kubectl cluster-info

```
ubuntu@ip-172-31-47-144:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

ubuntu@ip-172-31-47-144:~$ kubectl cluster-info
Kubernetes control plane is running at https://192.168.49.2:8443
CoreDNS is running at https://192.168.49.2:8443/api/v1/namespaces/kube-system/services/
/kube-dns:dns/proxy
```

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

ubuntu@ip-172-31-47-144:~\$

classplusapp.com is sharing your screen.

Step-9 : Access K8S Nodes

kubectl get nodes

```
ubuntu@ip-172-31-47-144:~$ kubectl get nodes
NAME      STATUS    ROLES      AGE     VERSION
minikube  Ready     control-plane   83s    v1.32.0
ubuntu@ip-172-31-47-144:~$ |
```

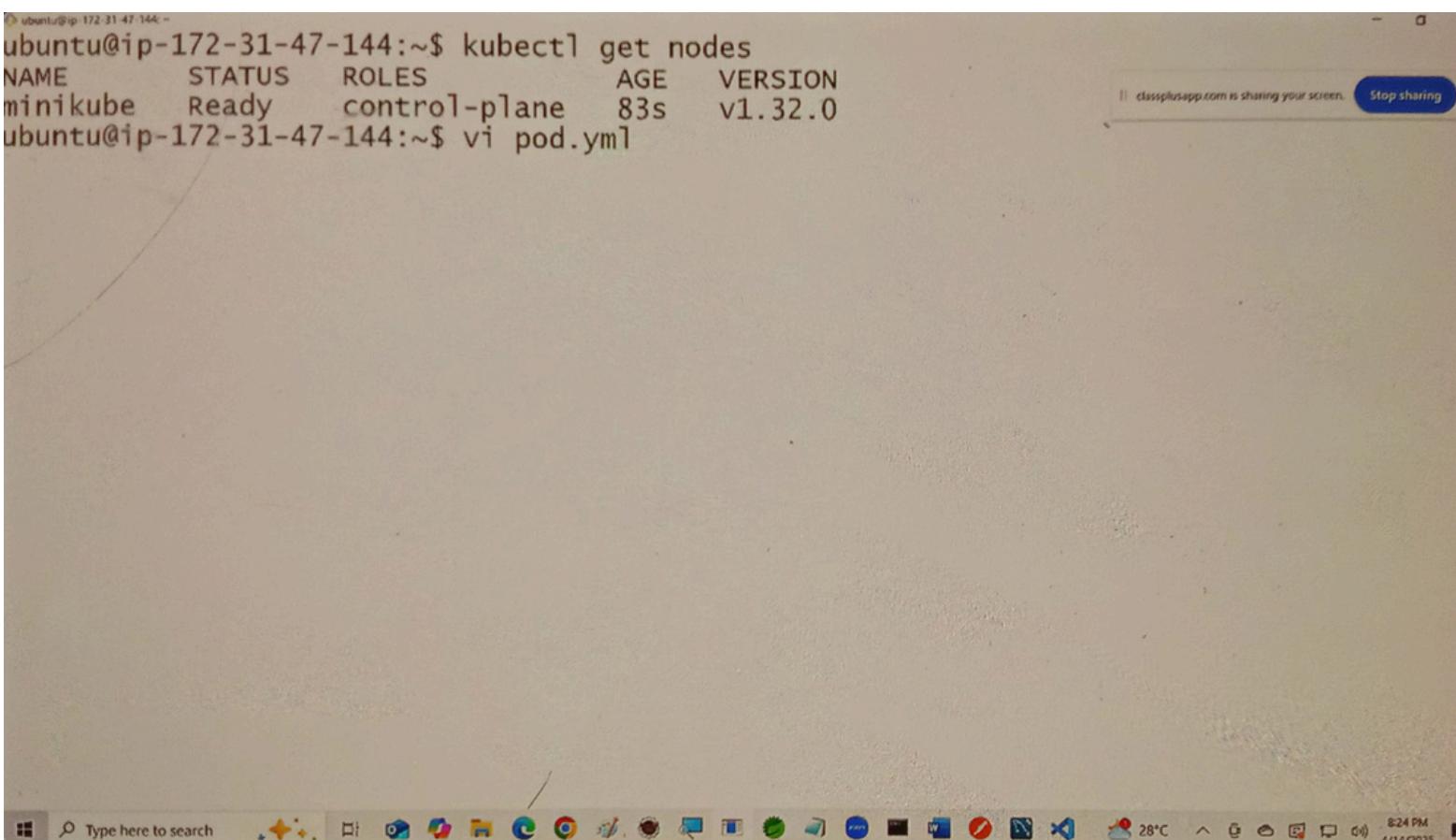
classplusapp.com is sharing your screen.

classplusapp.com is sharing your screen.

EX 1: Create pod using yml

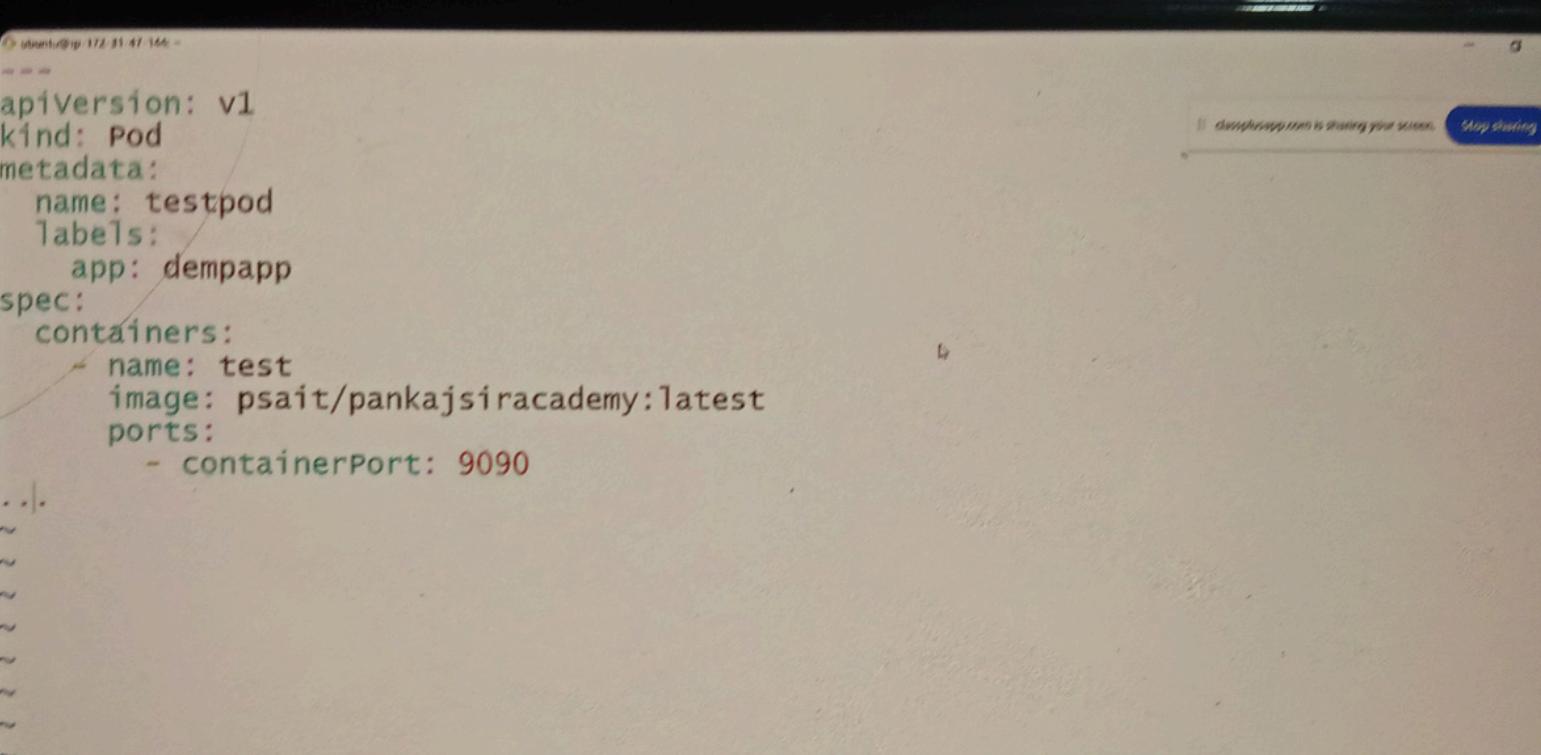
Step 1: vi pod.yml

```
ubuntu@ip-172-31-47-144:~$ kubectl get nodes
NAME      STATUS    ROLES      AGE   VERSION
minikube  Ready     control-plane   83s  v1.32.0
ubuntu@ip-172-31-47-144:~$ vi pod.yml
```



Step 2: write pod configuration in pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: testpod
  labels:
    app: demapp
spec:
  containers:
    - name: test
      image: psait/pankajsiracademy:latest
      ports:
        - containerPort: 9090
```



step 3: kubectl apply -f pod.yml

step 4: to check pod is created or not

kubectl get pods

```
ubuntu@ip-172-31-47-144:~$ kubectl get nodes
NAME      STATUS    ROLES      AGE     VERSION
minikube  Ready     control-plane  83s    v1.32.0
ubuntu@ip-172-31-47-144:~$ vi pod.yml
ubuntu@ip-172-31-47-144:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-47-144:~$ kubectl apply -f pod.yml
error: the path "pod.yml" does not exist
ubuntu@ip-172-31-47-144:~$ kubectl apply -f pod.yaml
pod/testpod created
ubuntu@ip-172-31-47-144:~$ kubectl get pods
NAME      READY    STATUS    RESTARTS   AGE
testpod  1/1      Running   0          16s
ubuntu@ip-172-31-47-144:~$
```

It will checks logs of pods which you created earlier

kubectl logs testpod

```
ubuntu@ip-172-31-47-144:~$ kubectl logs testpod
```

ubuntu@ip-172-31-47-144: ~
=====
:: Spring Boot :: (v3.4.4)

```
2025-04-14T14:55:39.517Z INFO 1 --- [demo] [main] com.demo_dockers.DemoApplication : Starting DemoApplication v0.0.1-SNAPSHOT using Java 17.0.2 with PID 1 (/usr/app/demo-app.jar started by root in /usr/app)
2025-04-14T14:55:39.519Z INFO 1 --- [demo] [main] com.demo_dockers.DemoApplication : No active profile set, falling back to 1 default profile: "default"
2025-04-14T14:55:40.580Z INFO 1 --- [demo] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 9090 (http)
2025-04-14T14:55:40.595Z INFO 1 --- [demo] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-04-14T14:55:40.595Z INFO 1 --- [demo] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.39]
2025-04-14T14:55:40.721Z INFO 1 --- [demo] [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-04-14T14:55:40.723Z INFO 1 --- [demo] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1145 ms
2025-04-14T14:55:41.386Z INFO 1 --- [demo] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 9090 (http) with context path '/'
2025-04-14T14:55:41.413Z INFO 1 --- [demo] [main] com.demo_dockers.DemoApplication : Started DemoApplication in 2.51 seconds (process running for 3.185)
ubuntu@ip-172-31-47-144:~$
```

EX 2: Create pod using yml

```
ubuntu@ip-172-31-47-144:~$ vi service.yml|
```

```
ubuntu@ip-172-31-47-144:~$
```

```
apiVersion: v1
kind: Service
metadata:
  name: testpod-service
spec:
  type: NodePort
  selector:
    app: dempapp          # This must match the Pod's label
  ports:
    - port: 80            # Exposed port for external access
      targetPort: 9090    # Port on which the app is running inside the container
      nodePort: 30080     # External port exposed on each node
```

classplusapp.com is sharing your screen. Stop sharing

-- INSERT --

14,1

All



kubectl apply -f service.yml

Check which service is running

kubectl get svc

```
ubuntu@ip-172-31-47-144:~$ vi service.yml
ubuntu@ip-172-31-47-144:~$ kubectl apply -f service.yml
service/testpod-service created
ubuntu@ip-172-31-47-144:~$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
kubernetes     ClusterIP  10.96.0.1    <none>       443/TCP       35m
testpod-service NodePort   10.106.250.75 <none>       80:30080/TCP  38s
ubuntu@ip-172-31-47-144:~$ minikube ip
192.168.49.2
ubuntu@ip-172-31-47-144:~$ curl http://192.168.49.2:30080
Hello world from psaubuntu@ip-172-31-47-144:~$ |
```

check ip address of minikube
minikube ip

run your application by using curl
curl IP+port_number

you can delete the service also like below

kubectl delete svc testpod-service

```
ubuntu@ip-172-31-47-144:~$ kubectl get pods
NAME      READY  STATUS   RESTARTS  AGE
testpod  1/1    Running  0          13m
ubuntu@ip-172-31-47-144:~$ kubectl delete pod testpod
pod "testpod" deleted
ubuntu@ip-172-31-47-144:~$ kubectl get pods
No resources found in default namespace.
ubuntu@ip-172-31-47-144:~$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
kubernetes     ClusterIP  10.96.0.1    <none>       443/TCP       39m
testpod-service NodePort   10.106.250.75 <none>       80:30080/TCP  3m56s
ubuntu@ip-172-31-47-144:~$ kubectl delete svc testpod-service
service "testpod-service" deleted
ubuntu@ip-172-31-47-144:~$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
kubernetes     ClusterIP  10.96.0.1    <none>       443/TCP       39m
ubuntu@ip-172-31-47-144:~$ |
```

namespace

start minikube

```
buntu@ip-172-31-47-144:~$ minikube start --driver=docker  
minikube v1.35.0 on Ubuntu 24.04 (xen/amd64)
```

Because we earlier created minikube and we stopped minikube we now just starting minikube

```
ubuntu@ip-172-31-47-144:~$ minikube start --driver=docker  
* minikube v1.35.0 on Ubuntu 24.04 (xen/amd64)  
* using the docker driver based on existing profile  
* Starting "minikube" primary control-plane node in "minikube" cluster  
* Pulling base image v0.0.46 ...  
* Restarting existing docker container for "minikube" ...  
* Preparing Kubernetes v1.32.0 on Docker 27.4.1 ...  
* Verifying Kubernetes components...  
- Using image gcr.io/k8s-minikube/storage-provisioner:v5  
* Enabled addons: default-storageclass, storage-provisioner  
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default  
ubuntu@ip-172-31-47-144:~$ minikube status  
minikube  
type: Control Plane  
host: Running  
kubelet: Running  
apiserver: Running  
kubeconfig: Configured
```

minikube status

running

before create namespace how many namespace is present

```
buntu@ip-172-31-47-144:~$ kubectl get ns  
NAME STATUS AGE  
default Active 23h → default namespace  
kube-node-lease Active 23h  
kube-public Active 23h  
kube-system Active 23h  
buntu@ip-172-31-47-144:~$
```

namespace

A Namespace in Kubernetes is like a folder or directory in a computer's file system. It helps organize and group resources, like Pods, Services, Deployments, etc.

kubectl create namespace name

```
ubuntu@ip-172-31-47-144:~$ kubectl get ns
NAME      STATUS   AGE
default   Active   23h
kube-node-lease   Active   23h
kube-public   Active   23h
kube-system   Active   23h
ubuntu@ip-172-31-47-144:~$ kubectl create namespace backend-ns-1
namespace/backend-ns-1 created
ubuntu@ip-172-31-47-144:~$ kubectl get ns
NAME      STATUS   AGE
backend-ns-1   Active   3s
default   Active   23h
kube-node-lease   Active   23h
kube-public   Active   23h
kube-system   Active   23h
ubuntu@ip-172-31-47-144:~$
```

backend-ns-1 namespace created
to check how many namespace is present

ex 1

we can craete namespace by using yml file like below

kubectl create namespace name

```
ubuntu@ip-172-31-47-144:~$ vi namespace.yml
```

paste below code in .yml

```
apiVersion: v1
kind: Namespace
metadata:
  name: backend-ns2
```

note : before paste yml code check syntax

```
...
---  
apiVersion: v1  
kind: Namespace  
metadata:  
  name: backend-ns-2  
...
```

after creating yml file now run as shown below

```
ubuntu@ip-172-31-47-144:~$ vi namespace.yml
ubuntu@ip-172-31-47-144:~$ kubectl apply -f namespace.yml
namespace/backend-ns-2 created
ubuntu@ip-172-31-47-144:~$ kubectl get ns
NAME      STATUS   AGE
backend-ns-1  Active  89s
backend-ns-2  Active  8s
default     Active  23h
kube-node-lease  Active  23h
kube-public   Active  23h
kube-system   Active  23h
ubuntu@ip-172-31-47-144:~$
```

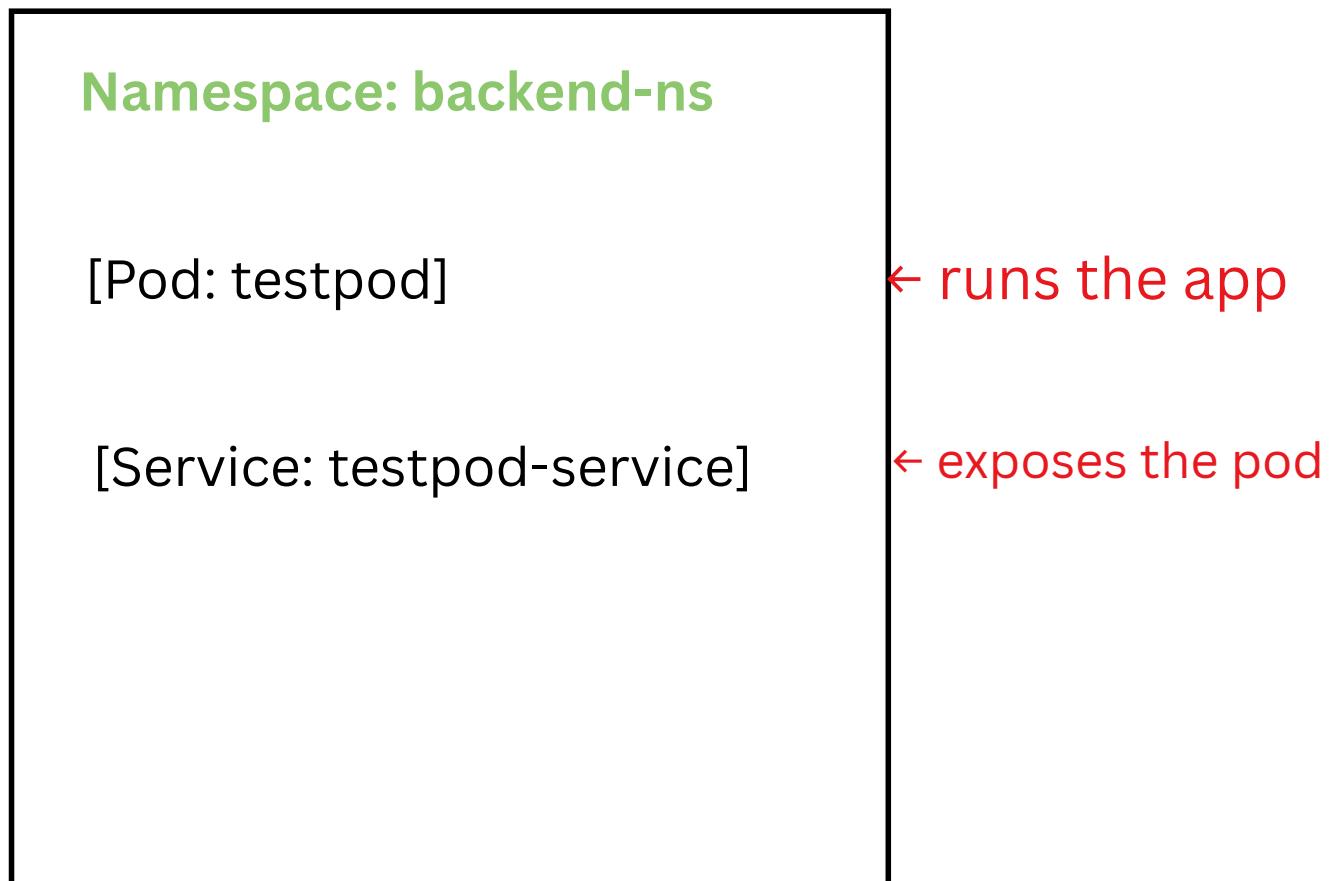
name space craeted

see backend-ns-2 namespace is created

ex 2

we can craete namespace + pod by using yml file like below

kubectl create namespace name



```
ubuntu@ip-172-31-47-144:~$ vi ns-example.yml
```

paste below code in .yml

Namespace with POD with Service creation yml file

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: backend-ns
---
apiVersion: v1
kind: Pod
metadata:
  name: testpod
  namespace: backend-ns
  labels:
    app: dempapp
spec:
  containers:
    - name: test
      image: psait/pankajsiracademy:latest
      ports:
        - containerPort: 9090
---
apiVersion: v1
kind: Service
metadata:
  name: testpod-service
  namespace: backend-ns
spec:
  type: NodePort
  selector:
    app: dempapp      # This must match the Pod's label
  ports:
    - port: 80      # Exposed port for external access
      targetPort: 9090  # Port on which the app is running inside the container
      nodePort: 30080  # External port exposed on each node
...

```

note : before paste yml code check syntax

```
apiVersion: v1
kind: Pod
metadata:
  name: testpod
  namespace: backend-ns-3
  labels:
    app: dempapp
spec:
  containers:
    - name: test
      image: psait/pankajsiracademy:latest
      ports:
        - containerPort: 9090
---
apiVersion: v1
kind: Service
metadata:
  name: testpod-service
  namespace: backend-ns-3
spec:
  type: NodePort
  selector:
    app: dempapp      # This must match the Pod's label
  ports:
    - port: 80      # Exposed port for external access
      targetPort: 9090  # Port on which the app is running inside the container
      nodePort: 30080  # External port exposed on each node
```

after creating yml file now run as shown below

```
ubuntu@ip-172-31-47-144:~$ vi ns-example.yaml
ubuntu@ip-172-31-47-144:~$ kubectl apply -f ns-example.yaml
namespace/backend-ns-3 created
pod/testpod created
service/testpod-service created
ubuntu@ip-172-31-47-144:~$ kubectl get ns
NAME      STATUS   AGE
backend-ns-1  Active  7m
backend-ns-2  Active  5m39s
backend-ns-3  Active  18s
default     Active  24h
kube-node-lease  Active  24h
kube-public    Active  24h
kube-system    Active  24h
```

namespace is created

backend-ns-3 created

```
ubuntu@ip-172-31-47-144:~$ kubectl get pods -n backend-ns-3
NAME      READY   STATUS    RESTARTS   AGE
testpod   1/1     Running   0          70s
```

check your pod is running or not

to check ip address minikube ip

to run application use curl +IP address

```
ubuntu@ip-172-31-47-144:~$ minijube ip
Command 'minijube' not found, did you mean:
  command 'minitube' from deb minitube (3.9.3-2)
Try: sudo apt install <deb name>
ubuntu@ip-172-31-47-144:~$ minikube ip
192.168.49.2
```

run curl command to see your application

see here your application running

do delete namespace shown below

note: 🔥 If you delete a Namespace, all the resources inside it (like Pods, Services, Deployments, ConfigMaps, etc.) are also deleted automatically.

```
ubuntu@ip-172-31-47-144:~$ kubectl delete ns backend-ns-3
namespace "backend-ns-3" deleted → see here deleted
ubuntu@ip-172-31-47-144:~$ kubectl get pods -n backend-ns-3
No resources found in backend-ns-3 namespace.
ubuntu@ip-172-31-47-144:~$ k
↓
see here not found
```

To check how many pods running shown below

```
ubuntu@ip-172-31-47-144:~$ kubectl delete ns backend-ns-3
namespace "backend-ns-3" deleted
ubuntu@ip-172-31-47-144:~$ kubectl get pods -n backend-ns-3
No resources found in backend-ns-3 namespace.
ubuntu@ip-172-31-47-144:~$ kubectl get pods
  READY   STATUS    RESTARTS   AGE
pp-cn7h2  1/1     Running   1 (5h3m ago)  5h4m
pp-1fbkb  1/1     Running   1 (5h3m ago)  5h7m
pp-nw2vv  1/1     Running   1 (5h3m ago)  5h10m
pp-q2pp4  1/1     Running   1 (5h3m ago)  5h10m
pp-q9vfk  1/1     Running   1 (5h3m ago)  5h4m
ubuntu@ip-172-31-47-144:~$ → see here all running pods
```

k8S Resources

- > When you create a Pod directly using kind: Pod, Kubernetes does not manage its lifecycle – if it crashes or is deleted, it's gone forever unless recreated manually.
- > K8S resources manages POD lifecycle
- > To let Kubernetes manage, restart, and scale Pods, we use higher-level controllers like the ones you listed.

- ⌚ 1) ReplicationController (RC)
- ⌚ 2) ReplicaSet (RS)
- 🚀 3) Deployment
- ❖ 4) DaemonSet
- 📅 5) StatefulSet

⌚ ReplicationController (RC)

(RC is old, replaced mostly by ReplicaSet and Deployment, but still good to understand!)

A ReplicationController ensures a specific number of Pods are always running.

It monitors the Pods and if one crashes or is deleted, it creates a new one automatically.

It's basically a Pod manager.

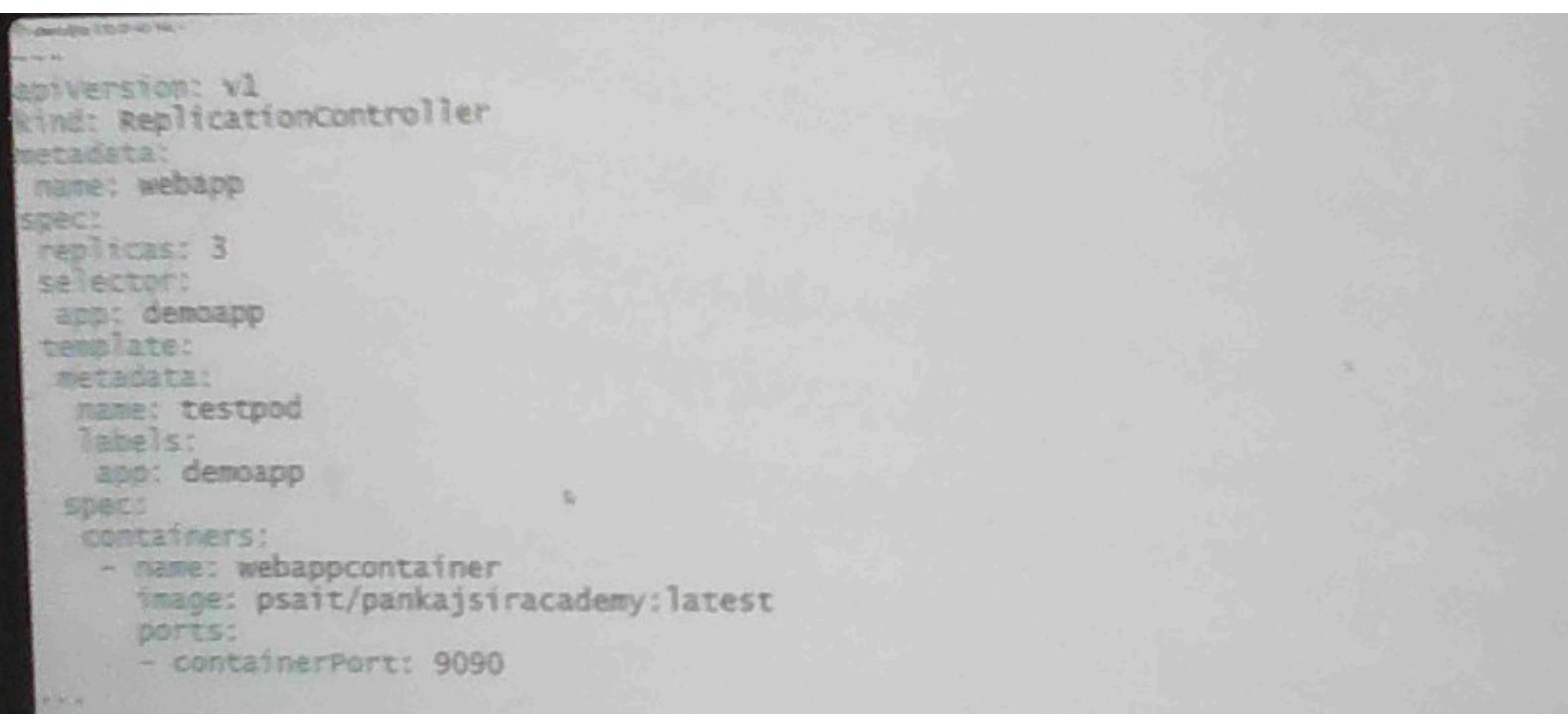
we here replace pod with ReplicationController(RC) in yml

step 1 vi name.yml

```
ubuntu@ip-172-31-47-144:~$ vi repcontroller.yml
```

step 2 write code in yml

```
---  
apiVersion: v1  
kind: ReplicationController  
metadata:  
  name: webapp  
spec:  
  replicas: 3  
  selector:  
    app: dempapp  
  template:  
    metadata:  
      name: testpod  
      labels:  
        app: dempapp  
    spec:  
      containers:  
        - name: webappcontainer  
          image: psait/pankajsiracademy:latest  
          ports:  
            - containerPort: 9090  
...  
  
note : before paste yml code check syntax
```



A screenshot of a terminal window displaying a portion of a YAML configuration file. The file defines a ReplicationController named 'webapp' with three replicas. It uses a template with a pod named 'testpod' and a label 'app: dempapp'. The container within the pod is named 'webappcontainer', runs the image 'psait/pankajsiracademy:latest', and exposes port 9090. The terminal window has a light gray background and a dark gray header bar.

```
apiVersion: v1  
kind: ReplicationController  
metadata:  
  name: webapp  
spec:  
  replicas: 3  
  selector:  
    app: demoapp  
  template:  
    metadata:  
      name: testpod  
      labels:  
        app: demoapp  
    spec:  
      containers:  
        - name: webappcontainer  
          image: psait/pankajsiracademy:latest  
          ports:  
            - containerPort: 9090
```

step 3

after creating yml file now run as shown below

kubectl apply -f name.yml

```
ubuntu@ip-172-31-47-144:~$ vi repcontroller.yml
ubuntu@ip-172-31-47-144:~$ kubectl apply -f recontroller.yml
error: the path "recontroller.yml" does not exist
ubuntu@ip-172-31-47-144:~$ kubectl apply -f repcontroller.yml
replicationcontroller/webapp configured
ubuntu@ip-172-31-47-144:~$
```

step 4

to check pod is created or not

kubectl get pods

```
ubuntu@ip-172-31-47-144:~$ vi repcontroller.yml
ubuntu@ip-172-31-47-144:~$ kubectl apply -f recontroller.yml
error: the path "recontroller.yml" does not exist
ubuntu@ip-172-31-47-144:~$ kubectl apply -f repcontroller.yml
replicationcontroller/webapp configured
ubuntu@ip-172-31-47-144:~$ kubectl get pods
NAME      READY  STATUS    RESTARTS   AGE
webapp-8868g  1/1   Running   0          14s
webapp-c9qdr  1/1   Running   0          14s
webapp-cnfh2  1/1   Running   1 (5h12m ago) 5h13m
webapp-1fbikb 1/1   Running   1 (5h12m ago) 5h17m
webapp-nx2vv  1/1   Running   1 (5h12m ago) 5h19m
webapp-q2pp4  1/1   Running   1 (5h12m ago) 5h19m
webapp-o9vfk  1/1   Running   1 (5h12m ago) 5h13m
webapp-w2fbf  1/1   Running   0          14s
ubuntu@ip-172-31-47-144:~$
```

NOTE:

As per above yml we used ReplicaController(RC) it uses replicas 3 it means it created 3 pods even tho if you delete any pods

now you can delete any pod like below shown **but** it will regenerate pods automatically because of RC

```
ubuntu@ip-172-31-47-144:~$ kubectl apply -f repcontroller.yml
replicationcontroller/webapp configured
ubuntu@ip-172-31-47-144:~$ kubectl get pods
NAME      READY  STATUS   RESTARTS   AGE
webapp-8868g 1/1    Running  0          14s
webapp-c9gdr 1/1    Running  0          14s
webapp-cn1h2 1/1    Running  1 (5h12m ago) 5h13m
webapp-lfbkb 1/1    Running  1 (5h12m ago) 5h17m
webapp-nw2vv 1/1    Running  1 (5h12m ago) 5h19m
webapp-q2pp4 1/1    Running  1 (5h12m ago) 5h19m
webapp-q9vfk 1/1    Running  1 (5h12m ago) 5h13m
webapp-w2fbf 1/1    Running  0          14s
ubuntu@ip-172-31-47-144:~$ kubectl delete pod webapp-8868g
pod "webapp-8868g" deleted
ubuntu@ip-172-31-47-144:~$ kubectl get pods
NAME      READY  STATUS   RESTARTS   AGE
webapp-c9gdr 1/1    Running  0          65s
webapp-cn1h2 1/1    Running  1 (5h13m ago) 5h14m
webapp-lfbkb 1/1    Running  1 (5h13m ago) 5h18m
webapp-lpf91 1/1    Running  0          4s
webapp-nw2vv 1/1    Running  1 (5h13m ago) 5h20m
webapp-q2pp4 1/1    Running  1 (5h13m ago) 5h20m
webapp-q9vfk 1/1    Running  1 (5h13m ago) 5h14m
webapp-w2fbf 1/1    Running  0          65s
ubuntu@ip-172-31-47-144:~$
```

You can increase replicas it means increasing number of pods like below

```
ubuntu@ip-172-31-47-144:~$ kubectl get pods
NAME      READY  STATUS   RESTARTS   AGE
webapp-c9gdr 1/1    Running  0          2m49s
webapp-cn1h2 1/1    Running  1 (Sh15m ago) 5h16m
webapp-lfbkb 1/1    Running  1 (Sh15m ago) 5h19m
webapp-lpf91 1/1    Running  0          108s
webapp-nw2vv 1/1    Running  1 (Sh15m ago) 5h22m
webapp-q2pp4 1/1    Running  1 (Sh15m ago) 5h22m
webapp-q9vfk 1/1    Running  1 (Sh15m ago) 5h16m
webapp-w2fbf 1/1    Running  0          2m49s
ubuntu@ip-172-31-47-144:~$ kubectl scale rc webapp --replicas=5 ✓
replicationcontroller/webapp scaled
ubuntu@ip-172-31-47-144:~$ kubectl get pods
NAME      READY  STATUS   RESTARTS   AGE
webapp-9bbjd 1/1    Running  0          13s
webapp-c9gdr 1/1    Running  0          3m41s
webapp-cn1h2 1/1    Running  1 (Sh16m ago) 5h17m
webapp-lfbkb 1/1    Running  1 (Sh16m ago) 5h20m
webapp-lpf91 1/1    Running  0          2m40s
webapp-ngmkh 1/1    Running  0          13s
webapp-nw2vv 1/1    Running  1 (Sh16m ago) 5h23m
webapp-q2pp4 1/1    Running  1 (Sh16m ago) 5h23m
webapp-q9vfk 1/1    Running  1 (Sh16m ago) 5h17m
webapp-w2fbf 1/1    Running  0          3m41s
ubuntu@ip-172-31-47-144:~$
```

ReplicaSet in Kubernetes

◆ What is a ReplicaSet?

A ReplicaSet (RS) is a Kubernetes resource that ensures a specified number of identical Pods are running at any given time.

💡 Key Features:

Self-healing: If a Pod crashes or is manually deleted, the RS will automatically create a new Pod to maintain the desired number.

Scaling: You can increase or decrease the number of replicas (Pods) easily.

Selector-based matching: RS manages only those Pods that match its label selector.

step 1 create yml

```
ubuntu@ip-172-31-47-144:~$ vi replica-set.yml'
^AC
ubuntu@ip-172-31-47-144:~$ vi replica-set.yml|
```

step 2 write yml code

```
---  
apiVersion: apps/v1  
kind: ReplicaSet  
metadata:  
  name: webapp  
spec:  
  replicas: 3  
  selector:  
    matchLabels:  
      app: dempapp      # Must match pod template labels  
  template:  
    metadata:  
      labels:  
        app: dempapp  
    spec:  
      containers:  
        - name: webappcontainer  
          image: psait/pankajsiracademy:latest  
          ports:  
            - containerPort: 9090
```

→ Resource type

note : before paste yml code check syntax

- **matchLabels** is a shorthand for equality-based match (e.g., app=web).
- **matchExpressions** gives you more flexibility (In, NotIn, Exists, DoesNotExist).
- You can use both together in a single ReplicaSet selector if needed.

```
---  
apiVersion: v1  
kind: Service  
metadata:  
  name: webappservice  
spec:  
  type: NodePort  
  selector:  
    app: dempapp      # Must match pod labels  
  ports:  
    - port: 80  
      targetPort: 9090  
      nodePort: 30096
```

→ service type

```
template:  
  metadata:  
    labels:  
      app: dempapp  
spec:  
  containers:  
    - name: webappcontainer  
      image: psait/pankajsiracademy:latest  
      ports:  
        - containerPort: 9090  
  
apiVersion: v1  
kind: Service  
metadata:  
  name: webappservice  
spec:  
  type: NodePort  
  selector:  
    app: dempapp      # Must match pod labels  
  ports:  
    - port: 80  
      targetPort: 9090  
      nodePort: 30096  
-- INSERT --
```

step 3 : kubectl apply -f name.yml

```
ubuntu@ip-172-31-47-144:~$ kubectl apply -f replica-set.yml
replicaset.apps/webapp unchanged
service/webappservice configured
ubuntu@ip-172-31-47-144:~$
```

see here created

step 4: get -> minikube ip

step 5: run curl like below -> curl + minikube ip

```
ubuntu@ip-172-31-47-144:~$ minikube ip
192.168.49.2
ubuntu@ip-172-31-47-144:~$ curl http://192.168.49.2:30096
Hello world from psabuntu@ip-172-31-47-144:~$
```

step 6: check how many pods running -> kubectl get pods

```
Hello world from psabuntu@ip-172-31-47-144:~$ kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
webapp-5ps5k 1/1     Running   0          10h
webapp-9bbjd  1/1     Running   0          23h
webapp-c9gdr  1/1     Running   0          23h
webapp-cn1h2  1/1     Running   1 (28h ago) 28h
webapp-d85n7  1/1     Running   0          10h
webapp-1pf9l  1/1     Running   0          23h
webapp-ngmkh  1/1     Running   0          23h
webapp-nw2vv  1/1     Running   1 (28h ago) 28h
webapp-q2pp4  1/1     Running   1 (28h ago) 28h
webapp-w2fbf  1/1     Running   0          23h
webapp-zxn4c  1/1     Running   0          10h
ubuntu@ip-172-31-47-144:~$
```

you can delete the specific pod also -> kubectl delete pod_name

Note : it will regenerate pod because of replicas

```
ubuntu@ip-172-31-47-144:~$ kubectl delete pod webapp-5ps5k
pod "webapp-5ps5k" deleted
ubuntu@ip-172-31-47-144:~$ kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
webapp-9bbjd 1/1     Running   0          23h
webapp-c9gdr  1/1     Running   0          23h
webapp-cn1h2  1/1     Running   1 (28h ago) 28h
webapp-d85n7  1/1     Running   0          10h
webapp-1pf91  1/1     Running   0          23h
webapp-mvvnn  1/1     Running   0          5s
webapp-ngmkh  1/1     Running   0          23h
webapp-nw2vv  1/1     Running   1 (28h ago) 28h
webapp-q2pp4  1/1     Running   1 (28h ago) 28h
webapp-w2fbf  1/1     Running   0          23h
webapp-zxnc4  1/1     Running   0          10h
ubuntu@ip-172-31-47-144:~$
```

In Companies (Especially on AWS / EKS):

Concept	What's Used in Practice
Pods	✗ Not created directly
ReplicaSets	⚙️ Used indirectly via Deployments
ReplicationControllers	✗ Deprecated / not used
Deployments	✓ Used to manage Pods & ReplicaSets
EKS (Elastic Kubernetes Service)	✓ Managed Kubernetes Control Plane
Helm Charts	✓ Used for templating & versioning apps
CI/CD Tools (e.g., ArgoCD, GitLab, Jenkins, CodePipeline)	✓ For automated deployment pipelines

Deployment in Kubernetes

Feature	ReplicaSet	Deployment
Manages Pods	✓ Yes	✓ Yes (via ReplicaSet)
Rolling Updates	✗ No	✓ Yes
Rollbacks	✗ No	✓ Yes
YAML Kind	ReplicaSet	Deployment
Use in Real World	Rare	Very Common

A Deployment in Kubernetes is one of the most recommended and used resources for managing Pod lifecycles. It ensures reliable application deployment with features like zero downtime, auto-scaling, rolling updates, and rollback capabilities.

⌚ Key Advantages of Using Deployments

Zero Downtime: Deployments ensure high availability by using strategies like rolling updates. Even when pods are being updated, the service remains available to users.

Auto Scaling: With Kubernetes Horizontal Pod Autoscaler, you can automatically scale your Pods based on CPU or memory usage (or other custom metrics).

Rolling Update & Rollback: Kubernetes allows rolling updates, which means it will gradually update Pods one by one, ensuring the application remains available throughout the process. If something goes wrong during the update, you can rollback to a previous stable version.

💻 When to Choose Which Strategy?

Use **RollingUpdate:**

For most production workloads where high availability and zero downtime are required.

When you are gradually releasing new versions of your application and want to avoid service disruption.

Use **Recreate:**

For non-critical applications or during maintenance windows where it's okay to have a temporary outage.

When you need to clear everything and redeploy fresh Pods (e.g., clearing persistent state or major upgrades).

1

choose AMI ubuntu or based on requirement

The screenshot shows the AWS EC2 'Launch an instance' wizard. Step 1: Choose AMI. The 'Amazon Machine Image (AMI)' section is selected. It shows the 'Ubuntu Server 24.04 LTS (HVM), SSD Volume Type' AMI (ami-0e35ddab05955cf57). The instance is 'Free tier eligible'. Other AMIs listed include macOS, Ubuntu, Windows, Red Hat, and SUSE. A summary panel on the right shows the instance type as t2.micro, security group as 'New security group', and storage as 'EBS General Purpose (SSD) Volume Type'. Buttons for 'Launch instance' and 'Preview code' are at the bottom.

2

choose instance type t2.medium or higher

The screenshot shows the AWS EC2 'Launch an instance' wizard. Step 2: Choose instance type. The 'Instance type' section is selected, showing 't2.medium' as the chosen option. It lists family details: t2 (2 vCPU, 4 GiB Memory, Current generation: true), On-Demand Linux base pricing (0.0496 USD per Hour), On-Demand Ubuntu Pro base pricing (0.0531 USD per Hour), On-Demand Windows base pricing (0.0676 USD per Hour), On-Demand RHEL base pricing (0.0784 USD per Hour), and On-Demand SUSE base pricing (0.1496 USD per Hour). A note states 'Additional costs apply for AMIs with pre-installed software'. A summary panel on the right shows the instance type as t2.medium, security group as 'New security group', and storage as 'EBS General Purpose (SSD) Volume Type'. Buttons for 'Launch instance' and 'Preview code' are at the bottom.

3 select storage minimum 50 gb

The screenshot shows the AWS EC2 'Launch an instance' wizard. In the 'Configure storage' section, a root volume of 50 GiB is selected as gp3, described as 'Root volume, 3000 IOPS, Not encrypted'. A note indicates that free-tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. Below this, it states that the selected AMI contains more instance store volumes than the instance allows, and only the first 0 instance store volumes from the AMI will be accessible from the instance. There is a note to click refresh to view backup information. The 'Storage (volumes)' section is collapsed. On the right, the 'Summary' section shows 1 instance, the software image (AMI) as Canonical, Ubuntu, 24.04, amd64, and the virtual server type (instance type) as t2.medium. It also shows a new security group and storage volumes. At the bottom, there are 'Cancel', 'Launch instance', and 'Preview code' buttons.

4

launch instance

The screenshot shows the AWS EC2 'Launch an instance' wizard. The configuration is identical to the previous step: 50 GiB gp3 root volume, Canonical, Ubuntu 24.04 AMI, t2.medium instance type, and a new security group. The 'Storage (volumes)' section is collapsed. The 'Launch instance' button is highlighted with a blue oval. The status bar at the bottom indicates 'classplusapp.com is sharing your screen.'

5 Install Docker In Ubuntu VM

```
sudo apt update  
curl -fsSL get.docker.com | /bin/bash  
sudo usermod -aG docker ubuntu  
exit
```

A screenshot of a terminal window titled "ubuntu@ip-172-31-47-100:~\$". The window contains the following text:

```
ubuntu@ip-172-31-47-100:~$ sudo apt update  
curl -fsSL get.docker.com | /bin/bash  
sudo usermod -aG docker ubuntu  
exit
```

6 Updating system packages and installing Minikube dependencies

```
sudo apt update  
sudo apt install -y curl wget apt-transport-https
```

A screenshot of a terminal window titled "ubuntu@ip-172-31-47-100:~\$". The window contains the following text:

```
ubuntu@ip-172-31-47-100:~$ sudo apt update  
sudo apt install -y curl wget apt-transport-https
```

7 Installing Minikube

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
```

```
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

```
minikube version
```

A screenshot of a terminal window titled "ubuntu@ip-172-31-47-100:~\$". The window contains the following text:

```
ubuntu@ip-172-31-47-100:~$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64  
sudo install minikube-linux-amd64 /usr/local/bin/minikube  
minikube version
```

8

Install Kubectl (Kubernetes Client)

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl  
chmod +x kubectl  
sudo mv kubectl /usr/local/bin/  
kubectl version -o yaml
```

```
ubuntu@ip-172-31-47-100:~$ curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl  
chmod +x kubectl  
sudo mv kubectl /usr/local/bin/  
kubectl version -o yaml
```

9

Start MiniKube Server

```
ubuntu@ip-172-31-47-100:~$ minikube start --driver=docker
```

10

create vi name.yml

```
ubuntu@ip-172-31-47-100:~$ vi deployment-nodeport.yml
```



write code in yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: dempapp      # Must match pod template labels
  template:
    metadata:
      labels:
        app: dempapp
    spec:
      containers:
        - name: webappcontainer
          image: psait/pankajsiracademy:latest
          ports:
            - containerPort: 9090
```

→ **Resource type**

note : check yml syntax

```
apiVersion: v1
kind: Service
metadata:
  name: webappservice
spec:
  type: NodePort
  selector:
    app: dempapp      # Must match pod labels
  ports:
    - port: 80
      targetPort: 9090
      nodePort: 30095
```

→ **service type**

```
template:
  metadata:
    labels:
      app: dempapp
  spec:
    containers:
      - name: webappcontainer
        image: psait/pankajsiracademy:latest
        ports:
          - containerPort: 9090

apiVersion: v1
kind: Service
metadata:
  name: webappservice
spec:
  type: NodePort
  selector:
    app: dempapp      # Must match pod labels
  ports:
    - port: 80
      targetPort: 9090
      nodePort: 30095
```

12

- to configure → `kubectl apply -f name.yml`
- to check pod--> `kubectl get pods`
- to check service → `kubectl get svc`

```
ubuntu@ip-172-31-47-100:~$ kubectl apply -f deployment-nodeport.yml
deployment.apps/webapp created
service/webappservice created
ubuntu@ip-172-31-47-100:~$ kubectl get pods
NAME           READY   STATUS            RESTARTS   AGE
webapp-67994f66cc-8wzcb  0/1    ContainerCreating  0          13s
webapp-67994f66cc-gvpfsp 0/1    ContainerCreating  0          13s
webapp-67994f66cc-m56fx  0/1    ContainerCreating  0          13s
ubuntu@ip-172-31-47-100:~$ kubectl get svc
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
kubernetes  ClusterIP  10.96.0.1    <none>        443/TCP      80s
webappservice  NodePort  10.101.83.8  <none>        80:30095/TCP  25s
ubuntu@ip-172-31-47-100:~$
```

13

- check ip address of minikube--> `minikube ip`
- to hit your application use curl--> `curl+ip-Address`

```
ubuntu@ip-172-31-47-100:~$ minikube ip
192.168.49.2
ubuntu@ip-172-31-47-100:~$ curl http://192.168.49.2:30095
Hello world from psaubuntu@ip-172-31-47-100:~$
```

14

- to check service → kubectl get svc
- to delete service (note service is deleting not source it means accessing application not possible now)
→ kubectl delete svc service_name
- see here pod is not detected check --> kubectl get pods

```
ubuntu@ip-172-31-47-100:~$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes     ClusterIP  10.96.0.1    <none>        443/TCP         4m1s
webappservice   NodePort   10.101.83.8  <none>        80:30095/TCP   3m6s
ubuntu@ip-172-31-47-100:~$ kubectl delete svc webappservice
service "webappservice" deleted
ubuntu@ip-172-31-47-100:~$ kubectl get pods
NAME            READY   STATUS    RESTARTS   AGE
webapp-67994f66cc-8wzcb  1/1     Running   0          3m31s
webapp-67994f66cc-gvpfsp 1/1     Running   0          3m31s
webapp-67994f66cc-m56fx  1/1     Running   0          3m31s
ubuntu@ip-172-31-47-100:~$
```

- you can delete pod here (note but it will regenerate because of replicas)

15

- kubectl delete pod pod_name
- see here pod again generated → kubectl get pods

```
ubuntu@ip-172-31-47-100:~$ kubectl delete pod webapp-67994f66cc-8wzcb
pod "webapp-67994f66cc-8wzcb" deleted
ubuntu@ip-172-31-47-100:~$ kubectl get pods
NAME            READY   STATUS    RESTARTS   AGE
webapp-67994f66cc-gvpfsp 1/1     Running   0          4m11s
webapp-67994f66cc-m56fx  1/1     Running   0          4m11s
webapp-67994f66cc-wkvwc  1/1     Running   0          3s
```



then how to delete pods ?

use source metadata name to delete pod permanently
source metadata name is present in yml

--> kubectl delete resource_type metadata_name

NOTE : OR You can make replicas=0

--> kubectl scale --replicas=0 <resource_type> <metadata_name>

```
untu@ip-172-31-47-100:~$ kubectl delete deployment webapp
deployment.apps "webapp" deleted
untu@ip-172-31-47-100:~$ kubectl get pods
No resources found in default namespace. → see here pod is not found
untu@ip-172-31-47-100:~$
```

Till now we used **service type nodeport** it is publically accessible but not load balance works so now use **service type is LoadBalancer**

1

create vi name.yml

```
ubuntu@ip-172-31-47-100:~$ vi deployment-loadbalancer.yml
```

write code in yml

```
---  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: webapp  
spec:  
  replicas: 3  
  selector:  
    matchLabels:  
      app: dempapp    # Must match pod template labels  
  template:  
    metadata:  
      labels:  
        app: dempapp  
    spec:  
      containers:  
        - name: webappcontainer  
          image: psait/pankajsiracademy:latest  
          ports:  
            - containerPort: 9090
```

```
---  
apiVersion: v1  
kind: Service  
metadata:  
  name: webappservice  
spec:  
  type: LoadBalancer  #  Changed from NodePort to LoadBalancer  
  selector:  
    app: dempapp  
  ports:  
    - port: 80  
      targetPort: 9090
```

```
ubuntu@ip-172-31-47-100: ~  
template:  
  metadata:  
    labels:  
      app: dempapp  
  spec:  
    containers:  
      - name: webappcontainer  
        image: psait/pankajsiracademy:latest  
        ports:  
          - containerPort: 9090  
  
---  
apiVersion: v1  
kind: Service  
metadata:  
  name: webappservice  
spec:  
  type: LoadBalancer      #  Changed from NodePort to LoadBalancer  
  selector:  
    app: dempapp  
  ports:  
    - port: 80  
      targetPort: 9090  
      nodePort: 30099  
-- INSERT (paste) --
```

- to configure → kubectl apply -f name.yml
- check ip address of minikube--> minikube ip
- to hit your application use curl--> curl+ip-Address

```
ubuntu@ip-172-31-47-100:~$ kubectl apply -f deployment-loadbalancer.yml
deployment.apps/webapp unchanged
service/webappservice configured
ubuntu@ip-172-31-47-100:~$ curl http://192.168.49.2:30099
Hello world from psaubuntu@ip-172-31-47-100:~$
```



