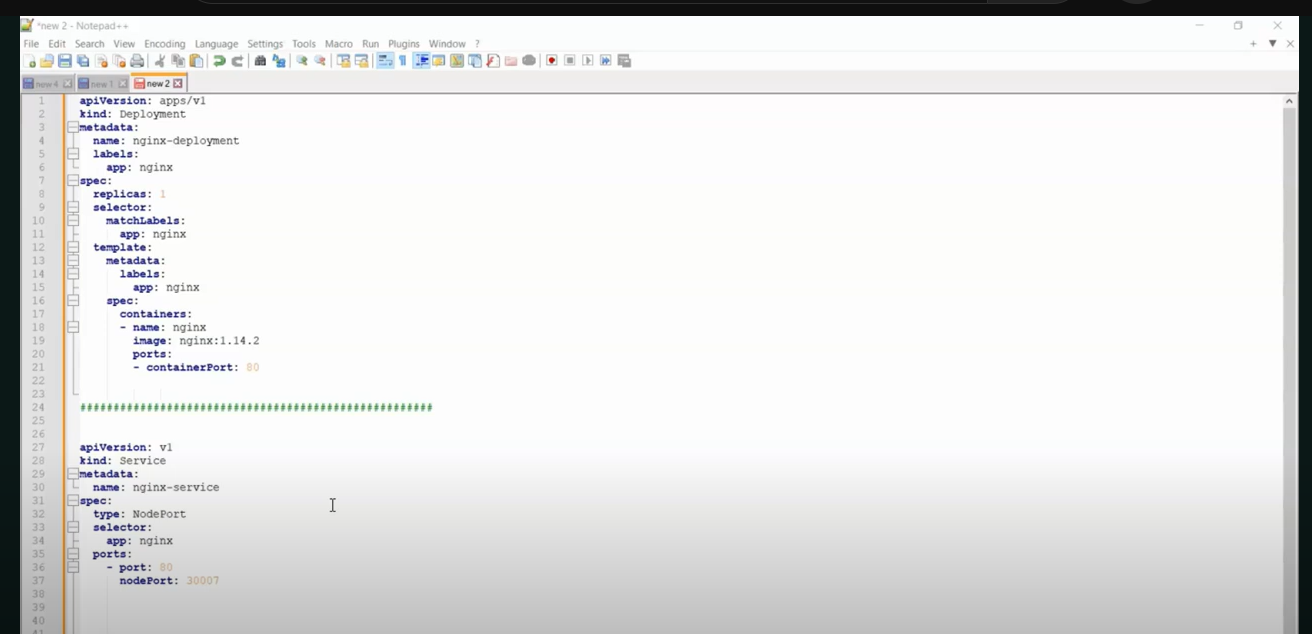
Deployment and service yaml file



apiVersion: apps/v1

kind: Deployment

metadata:

name: my-app

spec:

replicas: 3

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-app-container

image: my-app-image:latest

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: my-app-service

spec:

type: LoadBalancer

selector:

app: my-app

ports:

- protocol: TCP

port: 80

targetPort: 80

**Configmap.yaml file and how to use it in pods**

apiVersion: v1

kind: ConfigMap

metadata:

name: my-app-config

data:

DATABASE\_URL: "postgres://user:password@localhost:5432/mydb"

LOG\_LEVEL: "info"

APP\_MODE: "production"

**How to use this in POD?**

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-app-container

image: my-app-image:latest

env:

- name: DATABASE\_URL

valueFrom:

configMapKeyRef:

name: my-app-config

key: DATABASE\_URL

- name: LOG\_LEVEL

valueFrom:

configMapKeyRef:

name: my-app-config

key: LOG\_LEVEL

- name: APP\_MODE

valueFrom:

configMapKeyRef:

name: my-app-config

key: APP\_MODE

**Secret.yaml file and how to use it in pods**

apiVersion: v1

kind: Secret

metadata:

name: my-app-secret

type: Opaque

data:

DATABASE\_PASSWORD: cGFzc3dvcmQ= # base64 encoded value for "password"

API\_KEY: c29tZWFwaV9rZXk= # base64 encoded value for "someapi\_key"

**How to use this in POD?**

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-app-container

image: my-app-image:latest

env:

- name: DATABASE\_PASSWORD

valueFrom:

secretKeyRef:

name: my-app-secret

key: DATABASE\_PASSWORD

- name: API\_KEY

valueFrom:

secretKeyRef:

name: my-app-secret

key: API\_KEY

**Persistent volume(PV) and Persistent volume Claim(PVC) in kuberenets**

apiVersion: v1

kind: PersistentVolume

metadata:

name: my-aws-pv

spec:

capacity:

storage: 10Gi

accessModes:

- ReadWriteOnce

persistentVolumeReclaimPolicy: Retain

awsElasticBlockStore:

volumeID: aws://us-west-2a/vol-0abcd1234efgh5678 # Replace with your EBS volume ID

fsType: ext4

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: **my-pvc**

spec:

accessModes:

- ReadWriteOnce # Must match the PV access mode

resources:

requests:

storage: 10Gi # Must match or be less than the PV size

**How to use this in POD?**

template:

metadata:

labels:

app: example-app

spec:

containers:

- name: example-container

image: nginx

volumeMounts:

- mountPath: "/usr/share/nginx/html"

name: example-storage

volumes:

- name: example-storage

persistentVolumeClaim:

claimName: **my-pvc**

**StatefulSet :**

StatefulSets are used for applications where:

* **Each pod needs a stable network identity** (e.g., DNS hostname) that does not change over its lifetime.
* **Data persistence** is required across pod restarts (e.g., databases like MySQL, MongoDB, or applications like Apache Kafka and ZooKeeper).
* **Ordered deployment, scaling, and updates** are important to maintain application consistency or data integrity.

Eg:

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: example-statefulset

spec:

serviceName: "example-service"

replicas: 3

selector:

matchLabels:

app: example-app

template:

metadata:

labels:

app: example-app

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

volumeMounts:

- name: www

mountPath: /usr/share/nginx/html

volumeClaimTemplates:

- metadata:

name: www

spec:

accessModes: [ "ReadWriteOnce" ]

resources:

requests:

storage: 1Gi

**How to enable autoscaling in Kubernetes?**

 Horizontal Pod Autoscaler (HPA): Automatically scales the number of pod replicas up or down based on CPU utilization or other custom metrics.

 Vertical Pod Autoscaler (VPA): Adjusts the CPU and memory requests/limits of containers within pods based on their usage.

 Cluster Autoscaler: Automatically adjusts the number of nodes in a cluster based on the resource requests of the pods. If the cluster is underutilized, it scales down nodes, and if it lacks resources, it scales up.

Prerequisites

1. Metrics Server: Ensure the Metrics Server is installed in your cluster. The Metrics Server collects resource metrics (e.g., CPU and memory usage) from nodes and pods, which the HPA uses to make scaling decisions.
2. Resource Requests and Limits: Make sure your pods define CPU and/or memory requests and limits. HPA uses these values to determine whether to scale up or down based on utilization percentages.

Deployment file

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-app

spec:

replicas: 2

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-app-container

image: my-app-image

resources:

requests:

cpu: "100m"

limits:

cpu: "500m"

**HPA Configuration**:

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

metadata:

name: my-app-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: my-app

minReplicas: 2

maxReplicas: 10

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 50

**kubectl command for enabling autoscaling:**

* kubectl autoscale deployment <deployment-name> --cpu-percent=<target-percentage> --min=<min-replicas> --max=<max-replicas>
* eg: kubectl autoscale deployment my-app --cpu-percent=50 --min=2 --max=10

**Rolling update and bluegreen deployment and how to enable that on deployment files?**

A **Rolling Update** gradually replaces the old version of your application with a new one. It ensures that your application remains available during the update by incrementally updating pods one (or a few) at a time.

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-app

spec:

replicas: 5

strategy:

type: RollingUpdate

rollingUpdate:

maxSurge: 1

maxUnavailable: 1

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-app-container

image: my-app:1.1 # New version of the app

ports:

- containerPort: 80

**ingress service in kuberenets?**

1) Simple Ingress:

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: simple-ingress

spec:

rules:

- host: example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: my-service

port:

number: 80

2)path based:

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: path-based-ingress

spec:

rules:

- host: example.com

http:

paths:

- path: /app1

pathType: Prefix

backend:

service:

name: app1-service

port:

number: 80

- path: /app2

pathType: Prefix

backend:

service:

name: app2-service

port:

number: 80

3)host-based(Name-Based Virtual Hosting):

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: host-based-ingress

spec:

rules:

- host: app1.example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: app1-service

port:

number: 80

- host: app2.example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: app2-service

port:

number: 80

4)TLS/HTTPS Ingress:

This type of ingress secures traffic using HTTPS with TLS certificates. It requires a TLS secret that contains the certificate and private key.

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: tls-ingress

spec:

tls:

- hosts:

- secure.example.com

secretName: tls-secret

rules:

- host: secure.example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: secure-service

port:

number: 443