1. **What are different types of services?**

A service is an abstraction that defines a logical set of pods and a policy by which to access them. Kubernetes provides following set of services

* 1. **ClusterIP (Default)**

Description:

* The ClusterIP service exposes the service on a cluster-internal IP. This type of service makes the service accessible only within the cluster.
* It is the default type if no type is specified when creating a service.

Use Case:

* Used for communication between services within the cluster.

Example:

apiVersion: v1

kind: Service

metadata:

name: my-clusterip-service

spec:

selector:

app: MyApp

ports:

- protocol: TCP

port: 80

targetPort: 9376

type: ClusterIP

* 1. **NodePort**

Description:

* The NodePort service exposes the service on each node's IP at a static port (the NodePort). A ClusterIP service, to which the NodePort service routes, is automatically created.
* This makes the service accessible from outside the cluster using <NodeIP>:<NodePort>.

Use Case:

* Used for exposing a service to external traffic (outside the cluster) on a static port.

Example:

apiVersion: v1

kind: Service

metadata:

name: my-nodeport-service

spec:

selector:

app: MyApp

ports:

- protocol: TCP

port: 80

targetPort: 9376

nodePort: 30007

type: NodePort

* 1. **LoadBalancer**

Description:

* The **LoadBalancer** service exposes the service externally using a cloud provider's load balancer. The cloud provider will automatically provision a load balancer, assign it a public IP address, and forward traffic to the **NodePort** service.
* This type of service combines a **NodePort** and **ClusterIP** service, with a load balancer from the cloud provider.

Use Case:

* Used for exposing a service to external traffic with a cloud provider's load balancer.

Example:

apiVersion: v1

kind: Service

metadata:

name: my-loadbalancer-service

spec:

selector:

app: MyApp

ports:

- protocol: TCP

port: 80

targetPort: 9376

type: LoadBalancer

* 1. **ExternalName**

Description:

* The **ExternalName** service maps a service to the contents of the **externalName** field (e.g., **foo.bar.example.com**). Unlike other service types, it does not define any ports or selectors and does not create any proxying rules.
* It returns a CNAME record with the name provided in the **externalName** field.

Use Case:

* Used for creating a service that acts as an alias for an external service outside the Kubernetes cluster.

Example:

apiVersion: v1

kind: Service

metadata:

name: my-externalname-service

spec:

type: ExternalName

externalName: my.database.example.com

1. **What is a pod?**

Pod is the basic deployable unit in Kubernetes that can contain one or more containers. Pods provide an environment for containers to share resources and communicate with each other efficiently. They are managed by higher-level controllers to ensure scalability, resilience, and automated management within a Kubernetes cluster.

1. **How to access the custom application with a specific port?**

**Steps to Expose a Custom Application:**

1. **Deploy the Application:**

First, create a deployment for the custom application if you haven't already. Assuming there is a custom application in a Docker image called my-custom-app that listens on port 8080.

Code:

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-custom-app

spec:

replicas: 1

selector:

matchLabels:

app: my-custom-app

template:

metadata:

labels:

app: my-custom-app

spec:

containers:

- name: my-custom-app

image: my-custom-app:latest

ports:

- containerPort: 8080

1. **Create a Service:**

Create a service to expose the application. I have created a NodePort service for external access.

Code:

apiVersion: v1

kind: Service

metadata:

name: my-custom-app-service

spec:

selector:

app: my-custom-app

ports:

- protocol: TCP

port: 80 # The port that the service will expose

targetPort: 8080 # The port on the pod where the application is running

nodePort: 30001 # The port on each node to expose the service

type: NodePort

1. **Apply the Configuration:**

Use kubectl command to apply the configuration

Code:

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

1. **Access the Application:**

Once the service is created, you can access the application using the node’s IP address and the nodePort.

For example, if your node’s IP address is 192.168.1.100 and the nodePort is 30001, you can access the application at http://192.168.1.100:30001.