**🎯 Assignment Title: Create a Kubernetes Cluster Using Minikube**

**✅ Objective:**

To install and configure a local Kubernetes (K8s) cluster using Minikube and deploy a test application.

**🛠️ Tools & Technologies Required:**

* OS: Windows/Linux/macOS
* Minikube
* Docker (or other supported drivers)
* kubectl (Kubernetes CLI)

**📦 Step 1: Prerequisites**

**✅ Install Tools**

**1. Install Virtualization Support**

**For Windows:**

* Enable virtualization in BIOS
* Install [Hyper-V](https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/enable-hyper-v) or [VirtualBox](https://www.virtualbox.org/)

**For Linux/macOS:**

* Use Docker or KVM2

**2. Install kubectl**

bash

CopyEdit

# For Ubuntu/Debian

sudo apt update

sudo apt install -y curl

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

chmod +x kubectl

sudo mv kubectl /usr/local/bin/

kubectl version --client

**3. Install Minikube**

bash

CopyEdit

# For Ubuntu/Debian

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

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

Or download from: https://minikube.sigs.k8s.io/docs/start/

**🚀 Step 2: Start Minikube Cluster**

**✅ Start Minikube (Using Docker)**

bash

CopyEdit

minikube start --driver=docker

Alternate drivers: --driver=virtualbox, --driver=hyperv

**✅ Verify Status**

bash

CopyEdit

minikube status

**✅ Check Nodes**

bash

CopyEdit

kubectl get nodes

Output should be:

pgsql

CopyEdit

NAME STATUS ROLES AGE VERSION

minikube Ready control-plane 1m v1.30.x

**📦 Step 3: Create and Deploy a Test App**

**✅ Create Deployment**

bash

CopyEdit

kubectl create deployment hello-minikube --image=kicbase/echo-server:1.0

**✅ Expose the Deployment**

bash

CopyEdit

kubectl expose deployment hello-minikube --type=NodePort --port=8080

**✅ Check Services**

bash

CopyEdit

kubectl get services

Example Output:

pgsql

CopyEdit

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

hello-minikube NodePort 10.96.109.123 <none> 8080:30123/TCP 1m

**✅ Access Application**

bash

CopyEdit

minikube service hello-minikube

This opens the service in your default browser.

**🧹 Step 4: Cleanup Resources**

**✅ Delete Deployment and Services**

bash

CopyEdit

kubectl delete service hello-minikube

kubectl delete deployment hello-minikube

**🧯 Step 5: Stop and Delete Minikube**

bash

CopyEdit

minikube stop

minikube delete

**📸 Optional: Screenshots for Report**

Take screenshots for the following:

* Minikube start
* kubectl get nodes
* kubectl get services
* Web app running
* Cleanup confirmation

**📝 Sample Report Format (for Submission)**

You can copy-paste below as your report body.

**Assignment: Create a Kubernetes Cluster Using Minikube**

**✅ Objective:**

To demonstrate the creation of a local Kubernetes cluster using Minikube and deploy a test application.

**✅ Steps Performed:**

1. Installed kubectl and Minikube.
2. Started Kubernetes cluster using minikube start.
3. Deployed hello-minikube using kubectl create deployment.
4. Exposed service using kubectl expose deployment.
5. Verified application via browser using minikube service.
6. Cleaned up all resources and stopped Minikube.

**✅ Output Screenshots:**

* Cluster running with kubectl get nodes
* Exposed service
* Browser window with app
* Cleanup done

**✅ Conclusion:**

Minikube allows quick local Kubernetes development. Successfully deployed and accessed a test service, gaining hands-on experience with cluster creation and service deployment.

## ****Assignment Title****: Create a Kubernetes Cluster Using kubeadm

### 🎯 ****Objective****:

To create a multi-node Kubernetes cluster manually using kubeadm, including control plane and worker nodes setup.

## 🛠️ ****Tools Required****:

* 2+ Linux Machines (or VMs) – Ubuntu 20.04+ recommended
  + **1 Master Node (Control Plane)**
  + **1+ Worker Nodes**
* Internet Access
* SSH Access
* Kubernetes Packages (kubeadm, kubelet, kubectl)
* Docker or containerd runtime

## ⚙️ Step 1: Prepare All Nodes

### 🔧 A. Set Hostnames

bash

CopyEdit

# On Master Node

sudo hostnamectl set-hostname master-node

# On Worker Node(s)

sudo hostnamectl set-hostname worker-node

### 🔧 B. Update and Disable Swap

bash

CopyEdit

sudo apt update && sudo apt upgrade -y

sudo swapoff -a

sudo sed -i '/ swap / s/^/#/' /etc/fstab

### 🔧 C. Add Required Kernel Modules

bash

CopyEdit

sudo modprobe overlay

sudo modprobe br\_netfilter

sudo tee /etc/modules-load.d/k8s.conf <<EOF

overlay

br\_netfilter

EOF

sudo tee /etc/sysctl.d/k8s.conf <<EOF

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

net.ipv4.ip\_forward = 1

EOF

sudo sysctl --system

## 🐳 Step 2: Install Container Runtime (containerd)

bash

CopyEdit

sudo apt install -y containerd

sudo mkdir -p /etc/containerd

containerd config default | sudo tee /etc/containerd/config.toml

sudo systemctl restart containerd

sudo systemctl enable containerd

## 📦 Step 3: Install kubeadm, kubelet, kubectl

bash

CopyEdit

sudo apt update

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

sudo curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

sudo tee /etc/apt/sources.list.d/kubernetes.list <<EOF

deb https://apt.kubernetes.io/ kubernetes-xenial main

EOF

sudo apt update

sudo apt install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl

## 🚀 Step 4: Initialize Kubernetes Control Plane

**On the Master Node only:**

sudo kubeadm init --pod-network-cidr=192.168.0.0/16

After success, it will output a kubeadm join command – **copy it!**

### 🔑 Set up kubectl access for regular user

bash

CopyEdit

mkdir -p $HOME/.kube

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

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

## 🌐 Step 5: Install Pod Network (Weave or Calico)

Example: **Install Calico**

bash

CopyEdit

kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml

## 🔗 Step 6: Join Worker Nodes

On each **worker node**, run the kubeadm join command from Step 4.

Example:

bash

CopyEdit

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

## ✅ Step 7: Verify the Cluster

On **master node**:

bash

CopyEdit

kubectl get nodes

Expected Output:

pgsql

CopyEdit

NAME STATUS ROLES AGE VERSION

master-node Ready control-plane 10m v1.30.x

worker-node Ready <none> 5m v1.30.x

## 🧹 Step 8: (Optional) Reset or Clean Up

### 🧼 Reset Node

bash

sudo kubeadm reset

sudo rm -rf ~/.kube

## Sample Report Format

### ****Assignment: Create a Kubernetes Cluster Using kubeadm****

#### Objective:

To manually create a Kubernetes cluster using kubeadm, with one master and at least one worker node.

#### Steps:

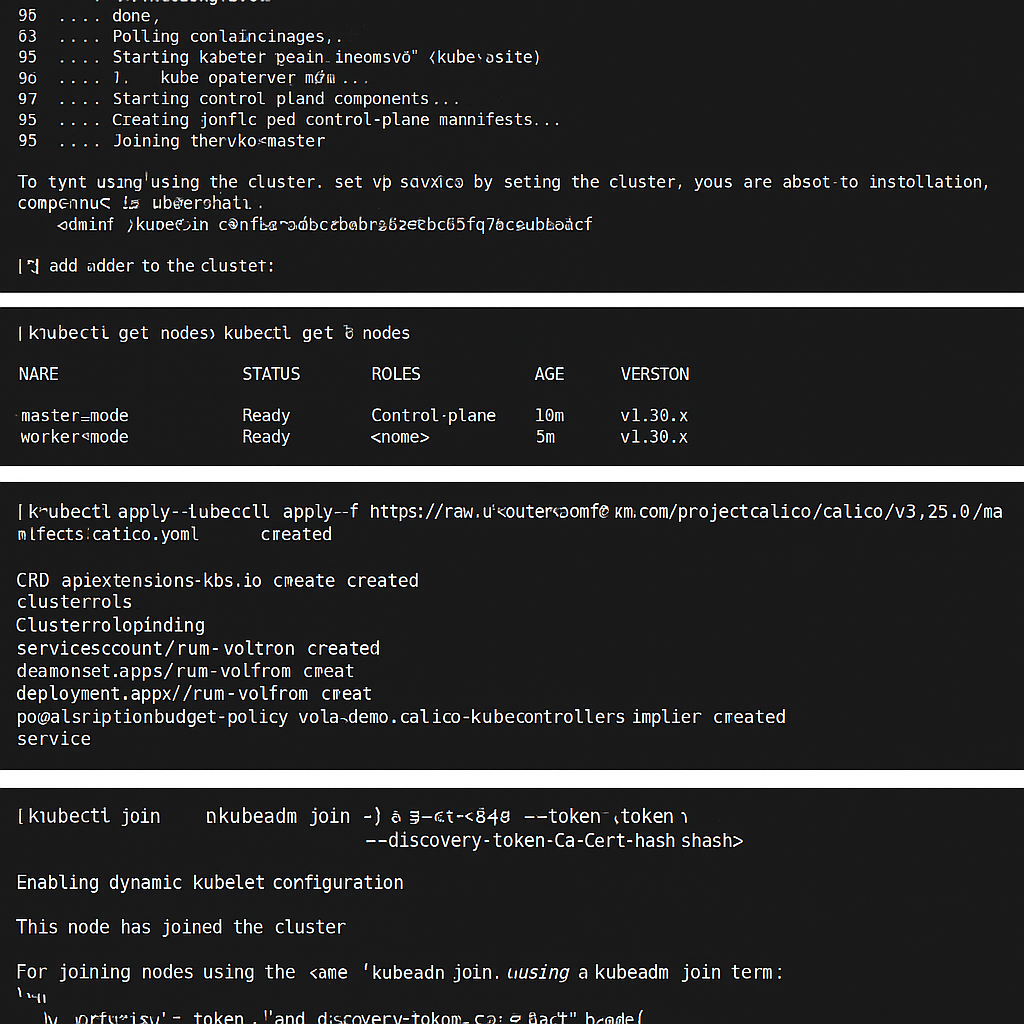
* Configured hostnames, disabled swap, and set kernel parameters.
* Installed containerd runtime and Kubernetes components.
* Initialized cluster using kubeadm init.
* Installed Calico for pod networking.
* Joined worker nodes using the generated token.
* Verified setup using kubectl get nodes.

#### Output Screenshots:

* Master init success
* Worker join success
* kubectl get nodes with "Ready" status
* Network plugin deployed

#### Conclusion:

Successfully set up a Kubernetes cluster using kubeadm, understanding each manual step of bootstrapping and networking configuration.



## Assignment Title:

**Deploy an AKS Cluster Using the Azure Portal and Configure User Access**

## 🎯 Objective:

To deploy a Kubernetes cluster using Azure Kubernetes Service (AKS) via Azure Portal, access the Kubernetes dashboard, and configure RBAC (Role-Based Access Control) for multiple users.

## 📦 Tools Required:

* Azure Subscription
* Azure Portal Access
* Azure CLI (optional, for validation)
* Azure AD user accounts

## 🚀 Step 1: Deploy AKS Using Azure Portal

### ✅ A. Login and Create Resource

1. Go to <https://portal.azure.com>
2. Click **“Create a resource”** → **“Kubernetes Service”**

### ✅ B. Configure Basics

* **Subscription**: Select your subscription.
* **Resource Group**: Create new or use an existing one (e.g., AKS-Group)
* **Cluster Name**: Example - myAKSCluster
* **Region**: Choose nearby (e.g., Central India)
* **Kubernetes Version**: Select default/stable version

### ✅ C. Node Pools

* **Node size**: Standard\_DS2\_v2
* **Node count**: Minimum 1

### ✅ D. Authentication

* Enable **Azure AD Integration**
* Enable **Managed Identity**

### ✅ E. Networking

* Choose **Basic** for beginner setup
* Leave default VNet settings

### ✅ F. Monitoring

* Enable **Container Insights**

### ✅ G. Review + Create

* Click **“Review + Create”**
* After validation → Click **“Create”**

⏱️ Deployment takes 5–10 minutes

## 🧭 Step 2: Access AKS Dashboard

Azure disables the Kubernetes Dashboard by default. To access it:

### ✅ A. Open Cloud Shell or Local Terminal

#### Option 1: Using Azure Cloud Shell

bash

CopyEdit

az login

az aks get-credentials --resource-group AKS-Group --name myAKSCluster

#### Option 2: Use Azure CLI locally

bash

CopyEdit

az login

az aks get-credentials --resource-group AKS-Group --name myAKSCluster

### ✅ B. Deploy Dashboard

bash

CopyEdit

kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml

### ✅ C. Create Access Token for Admin

bash

CopyEdit

kubectl create serviceaccount dashboard-admin-sa

kubectl create clusterrolebinding dashboard-admin-sa \

--clusterrole=cluster-admin \

--serviceaccount=default:dashboard-admin-sa

### ✅ D. Get Token

bash

CopyEdit

kubectl get secret $(kubectl get serviceaccount dashboard-admin-sa \

-o jsonpath="{.secrets[0].name}") -o go-template="{{.data.token | base64decode}}"

### ✅ E. Access Dashboard

Start proxy:

bash

CopyEdit

kubectl proxy

Visit:  
👉 http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/

Login using the token you copied.

## 👥 Step 3: Create Roles for Multiple Users (RBAC)

### ✅ A. Create a New Namespace (optional)

bash

CopyEdit

kubectl create namespace team-a

### ✅ B. Create Role YAML

yaml

CopyEdit

# role.yaml

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

namespace: team-a

name: team-a-reader

rules:

- apiGroups: [""]

resources: ["pods", "services"]

verbs: ["get", "list", "watch"]

### ✅ C. Create RoleBinding YAML

yaml

CopyEdit

# rolebinding.yaml

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: team-a-read-access

namespace: team-a

subjects:

- kind: User

name: user@example.com # Azure AD email

apiGroup: rbac.authorization.k8s.io

roleRef:

kind: Role

name: team-a-reader

apiGroup: rbac.authorization.k8s.io

### ✅ D. Apply the YAMLs

bash

CopyEdit

kubectl apply -f role.yaml

kubectl apply -f rolebinding.yaml

You can repeat this for other users and roles (like admin, editor, etc.).

## ✅ Step 4: Verification

### A. Check Nodes

bash

CopyEdit

kubectl get nodes

### B. Check Role Binding

bash

CopyEdit

kubectl get rolebindings -n team-a

### C. Test User Access

Have the user login using:

bash

CopyEdit

az login --username user@example.com

az aks get-credentials --resource-group AKS-Group --name myAKSCluster

kubectl get pods -n team-a

## 📸 Screenshots to Include in Report:

1. AKS creation page
2. Node pool and Kubernetes version settings
3. kubectl get nodes output
4. Dashboard access with token
5. Role and RoleBinding YAML applied
6. User access confirmation

## 📝 Sample Report Format

### ****Assignment: Deploy an AKS Cluster Using Azure Portal****

#### Objective:

To deploy a Kubernetes cluster using Azure Portal, enable dashboard access, and assign different RBAC roles to Azure AD users.

#### Key Steps:

* Created AKS via Azure Portal with basic setup.
* Accessed AKS using Azure CLI.
* Installed Kubernetes dashboard and generated access token.
* Created custom roles and role bindings for specific users.
* Verified user access based on assigned roles.

#### Output:

* Dashboard successfully accessed via proxy.
* Users with read-only access verified for specific namespaces.
* AKS nodes and services visible through dashboard.

#### Conclusion:

AKS simplifies Kubernetes management with full Azure integration. Role-based access enhances security and controlled visibility in team environments.

**Deploy a Microservice Application on AKS Cluster and Expose It to Public Internet**

## 🎯 Objective:

To deploy a multi-tier microservice application (e.g., a simple frontend-backend app) on AKS, expose it via a LoadBalancer service, and access it using a public IP address.

## 🛠 Tools & Requirements:

* Azure Subscription
* AKS Cluster (already deployed or new)
* Azure CLI or Cloud Shell
* kubectl
* Sample Microservice App (e.g., Azure Sample Voting App)

## 🚀 Step 1: Create AKS Cluster (If Not Created Already)

If not already deployed, follow this quick setup (can be done via Portal or CLI):

bash

CopyEdit

az aks create \

--resource-group AKS-Group \

--name MicroserviceAKS \

--node-count 2 \

--enable-addons monitoring \

--generate-ssh-keys

Get credentials:

bash

CopyEdit

az aks get-credentials --resource-group AKS-Group --name MicroserviceAKS

## 📦 Step 2: Clone a Microservice App (Voting App)

Use the Azure Sample Voting App – a microservice with:

* Frontend: Python Flask app
* Backend: Redis

bash

CopyEdit

git clone https://github.com/Azure-Samples/azure-voting-app-redis.git

cd azure-voting-app-redis

## 🏗 Step 3: Deploy Microservice to AKS

### ✅ A. Create Kubernetes Resources

Open the azure-vote.yaml file (or use this):

yaml

CopyEdit

apiVersion: v1

kind: Service

metadata:

name: azure-vote-back

spec:

ports:

- port: 6379

selector:

app: azure-vote-back

clusterIP: None

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: azure-vote-back

spec:

replicas: 1

selector:

matchLabels:

app: azure-vote-back

template:

metadata:

labels:

app: azure-vote-back

spec:

containers:

- name: azure-vote-back

image: redis

resources:

requests:

cpu: 100m

memory: 128Mi

---

apiVersion: v1

kind: Service

metadata:

name: azure-vote-front

spec:

type: LoadBalancer

ports:

- port: 80

selector:

app: azure-vote-front

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: azure-vote-front

spec:

replicas: 1

selector:

matchLabels:

app: azure-vote-front

template:

metadata:

labels:

app: azure-vote-front

spec:

containers:

- name: azure-vote-front

image: mcr.microsoft.com/azuredocs/azure-vote-front:v1

resources:

requests:

cpu: 250m

memory: 64Mi

env:

- name: REDIS

value: "azure-vote-back"

### ✅ B. Apply the YAML

bash

CopyEdit

kubectl apply -f azure-vote.yaml

Check the resources:

bash

CopyEdit

kubectl get all

## 🌐 Step 4: Expose App to Public Internet

Since azure-vote-front is using **type: LoadBalancer**, AKS will automatically provision a **public IP**.

### ✅ Get Public IP:

bash

CopyEdit

kubectl get service azure-vote-front

Output:

pgsql

CopyEdit

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

azure-vote-front LoadBalancer 10.0.67.41 52.183.65.108 80:30620/TCP 1m

📎 Copy the EXTERNAL-IP and open in your browser:

cpp

CopyEdit

http://<EXTERNAL-IP>

You should see a voting app interface live on the public internet!

## 🔐 Step 5: Optional Enhancements

### ✅ Use Ingress Controller (Advanced)

To use DNS name + TLS, install NGINX Ingress Controller and configure an Ingress resource.

### ✅ Add Autoscaling

bash

CopyEdit

kubectl autoscale deployment azure-vote-front --cpu-percent=50 --min=1 --max=5

## 🧹 Step 6: Clean Up Resources

bash

CopyEdit

kubectl delete -f azure-vote.yaml

Or delete entire cluster:

bash

CopyEdit

az aks delete --name MicroserviceAKS --resource-group AKS-Group --yes --no-wait

## 📸 Screenshots to Include in Report

1. AKS cluster details (node pool, name)
2. YAML file contents (azure-vote.yaml)
3. kubectl get all output
4. Public IP from kubectl get service
5. Voting App running in browser

## 📝 Sample Report Format

### Assignment: Deploy Microservice App on AKS & Access Publicly

#### Objective:

To deploy a Python-based voting microservice on an AKS cluster and access it using a public IP address.

#### Steps:

* Created AKS cluster with 2 nodes using Azure CLI
* Cloned Azure Voting App microservice project
* Defined Redis backend and Flask frontend in azure-vote.yaml
* Applied deployment and exposed frontend using LoadBalancer service
* Accessed the app through public IP

#### Output:

* AKS cluster created and configured successfully
* Microservice deployed and functional
* Public IP exposed the app to the internet
* Verified app accessibility via browser

#### Conclusion:

Successfully deployed and exposed a microservice app on AKS using LoadBalancer. Learned key skills like service exposure, YAML deployment, and AKS resource management.

## Assignment Title:

**Expose Services in Kubernetes Using NodePort, ClusterIP, and LoadBalancer**

## 🎯 Objective:

To understand and implement different Kubernetes service types (ClusterIP, NodePort, LoadBalancer) and how they expose applications inside or outside the cluster.

## 📦 Prerequisites:

* Kubernetes cluster (Minikube, kubeadm, or AKS)
* kubectl configured
* Sample deployment (Nginx used in this assignment)

## 🏗 Step 1: Create a Deployment (Nginx)

bash

CopyEdit

kubectl create deployment nginx --image=nginx

Verify:

bash

CopyEdit

kubectl get deployments

kubectl get pods

## 🌐 Step 2: Expose as ClusterIP (Default)

### 📘 Explanation:

* Internal-only access
* Used for communication between services inside cluster

### 🔧 Command:

bash

CopyEdit

kubectl expose deployment nginx --port=80 --target-port=80 --name=nginx-clusterip --type=ClusterIP

### ✅ Verify:

bash

CopyEdit

kubectl get service nginx-clusterip

Example Output:

pgsql

CopyEdit

NAME TYPE CLUSTER-IP PORT(S) AGE

nginx-clusterip ClusterIP 10.96.125.12 80/TCP 1m

🔒 This service is **not accessible externally**. You can use it within other pods using:

bash

CopyEdit

curl http://nginx-clusterip

## 🌐 Step 3: Expose as NodePort

### 📘 Explanation:

* Exposes service on a static port on each node (range: 30000–32767)
* Accessible from outside via <NodeIP>:NodePort

### 🔧 Command:

bash

CopyEdit

kubectl expose deployment nginx --port=80 --target-port=80 --name=nginx-nodeport --type=NodePort

### ✅ Verify:

bash

CopyEdit

kubectl get service nginx-nodeport

Example Output:

pgsql

CopyEdit

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

nginx-nodeport NodePort 10.96.55.88 <none> 80:32123/TCP 1m

### 🌐 Access:

* Use this on your browser or curl:

bash

CopyEdit

http://<NodeIP>:32123

If using Minikube:

bash

CopyEdit

minikube service nginx-nodeport --url

## 🌐 Step 4: Expose as LoadBalancer (AKS or Cloud Setup)

### 📘 Explanation:

* Creates a **cloud provider load balancer**
* Automatically assigns public IP
* External access via public DNS or IP

### 🔧 Command:

bash

CopyEdit

kubectl expose deployment nginx --port=80 --target-port=80 --name=nginx-lb --type=LoadBalancer

### ✅ Verify:

bash

CopyEdit

kubectl get service nginx-lb

Example Output:

pgsql

CopyEdit

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

nginx-lb LoadBalancer 10.96.53.23 20.44.124.10 80:32456/TCP 2m

🟢 Wait until EXTERNAL-IP is assigned (may take a few mins in AKS)

### 🌐 Access:

bash

CopyEdit

http://<EXTERNAL-IP>

## 🧹 Step 5: Clean Up

bash

CopyEdit

kubectl delete deployment nginx

kubectl delete service nginx-clusterip nginx-nodeport nginx-lb

## 🧪 Summary Table

| **Service Type** | **Accessible From** | **Use Case** | **Example URL** |
| --- | --- | --- | --- |
| ClusterIP | Inside cluster only | Internal microservice comms | http://nginx-clusterip |
| NodePort | Node IP + port | Expose on self-managed clusters | http://<NodeIP>:32123 |
| LoadBalancer | Public Internet | Expose on cloud (AKS, GKE, EKS) | http://<EXTERNAL-IP> |

## 📸 Suggested Screenshots

1. kubectl expose command for each type
2. Output of kubectl get service
3. Browser showing NodePort and LoadBalancer access
4. Service YAML (optional)

## 📄 Optional: YAML Definitions

### ✅ ClusterIP YAML:

yaml

CopyEdit

apiVersion: v1

kind: Service

metadata:

name: nginx-clusterip

spec:

selector:

app: nginx

ports:

- protocol: TCP

port: 80

targetPort: 80

type: ClusterIP

### ✅ NodePort YAML:

yaml

CopyEdit

apiVersion: v1

kind: Service

metadata:

name: nginx-nodeport

spec:

selector:

app: nginx

ports:

- protocol: TCP

port: 80

targetPort: 80

nodePort: 32123

type: NodePort

### ✅ LoadBalancer YAML:

yaml

CopyEdit

apiVersion: v1

kind: Service

metadata:

name: nginx-lb

spec:

selector:

app: nginx

ports:

- protocol: TCP

port: 80

targetPort: 80

type: LoadBalancer

Apply using:

bash

CopyEdit

kubectl apply -f <file>.yaml

## ✅ Sample Report Summary

### Assignment: Expose Kubernetes Services using NodePort, ClusterIP, LoadBalancer

* **ClusterIP** used for internal-only communication
* **NodePort** exposed the app on host IP and static port
* **LoadBalancer** used Azure cloud to expose public IP
* Verified all services using kubectl get svc and browser

📌 Conclusion: Learned different service exposure methods and their appropriate use cases.