

Program 3: Container Orchestration with Kubernetes:

- **Tool:** Kubernetes
- **Program:**
 - Set up a Kubernetes cluster (use Minikube or a cloud provider).
 - Deploy a sample application using a Deployment and Service.
 - Scale the application using `kubectl scale`.

1. Set up a Kubernetes cluster using Minikube:

Prerequisites:

- Docker Desktop installed and running.
- Windows 10/11 (PowerShell or CMD is fine)

Manual Install

- Download Minikube for Windows:
<https://github.com/kubernetes/minikube/releases/latest>
- Download `minikube-windows-amd64.exe`
Rename it to `minikube.exe`
- Add it to a folder in your system's PATH
(e.g., `C:\tools\minikube\` and add that to environment variables > PATH)

Start Minikube Using Docker:

Once installed, start it using Docker as the driver:

Open PowerShell as Administrator. Run the following command in powershell

```
❏ minikube start --driver=docker
```

See it download the base image and initialize the cluster.

Verify its working: Check the status:

```
❏ minikube status
```

Check cluster:

```
❏ kubectl get nodes
```

You should see a node named minikube in the Ready state.

What is Kubernetes?

Kubernetes ("K8s") is an open-source platform that helps you:

- Run, Manage, Scale, Update your containerized applications automatically.

Why do people use Kubernetes?

Python web app in a Docker container.

Without Kubernetes:

- manually start containers
- monitor them yourself
- If they crash, you restart them manually
- figure out how to load balance traffic
- handle deployments by hand

With Kubernetes:

- It runs multiple copies (pods) of your app
- It restarts them if they crash
- It scales up/down based on traffic
- It load balances requests
- It updates apps with zero downtime (rolling updates)
- It manages configs & secrets securely

Key Concepts

Term	What It Is
Pod	The smallest unit – runs one or more containers
Deployment	Defines how many pods to run and how to manage them
Service	A stable IP or name to access your app (load balancing)
ConfigMap	Stores non-sensitive config (env vars)
Secret	Stores sensitive data (passwords, API keys)
Node	A worker machine (VM or physical) that runs pods
Cluster	A group of nodes controlled by Kubernetes

Deploy a sample application using a Deployment and Service.

In terminal (PowerShell or CMD): type

```
❏ minikube start
```

Verify with:

```
❏ minikube status
```

Create a simple Pod YAML

Let's make a pod that runs a basic NGINX container.

Save this as pod.yaml:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

Apply the pod YAML

```
❏ Run:    kubectl apply -f pod.yaml
```

Check if it's running:

```
❏ kubectl get pods
```

You should see:

NAME	READY	STATUS	RESTARTS	AGE
my-nginx	1/1	Running	0	<time>

Access the **pod (for web apps)**. You can access it inside the cluster:

- ❑ **kubectl get pods -o wide** (it displays complete information about the each running pods)

```
PS C:\Users\Admin\Desktop\p4\app1> kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP            NODE       NOMINATED NODE   RE
ADINESS GATES
hw-deployment-9dcf4b4d6-4zwdb       1/1     Running   0           2m7s  10.244.0.88   minikube   <none>           <n
one>
hw-deployment-9dcf4b4d6-cr14m       1/1     Running   0           36m   10.244.0.86   minikube   <none>           <n
one>
hw-deployment-9dcf4b4d6-tpwkj       1/1     Running   0           36m   10.244.0.87   minikube   <none>           <n
one>
nginx                                1/1     Running   0           59m   10.244.0.81   minikube   <none>           <n
one>
```

- ❑ **minikube ssh** -❑ it will login into the minikube cluster

Then use:

- ❑ **curl <nginx ip-address>** to See the NGINX welcome page inside the cluster.

Create a Kubernetes Deployment and Service for a simple Python web application (like Flask) running in Minikube.

Sample Python App (Flask)

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello from App 1!! Kubernetes, also known as K8s, is an open source system for automating deployment, scaling, and management of containerized applications"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

requirement.txt

```
flask==3.0.0
```

Dockerfile:

```
FROM python:3.12-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py .
EXPOSE 5000
CMD ["python", "app.py"]
```

Build and Push Docker Image

Make sure Docker is running. Then build:

```
❏ docker build -t chethanaravi/app1-k8s:latest .
```

```
❏ docker push chethanaravi/app1-k8s:latest
```

Now the image is locally available inside Minikube.

Kubernetes Deployment (deployment.yaml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hw-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: hello-world
  template:
    metadata:
      labels:
```

```
  app: hello-world
spec:
  containers:
  - name: hw-container
    image: chethanaravi/app1-k8s:latest
  ports:
  - containerPort: 5000
```

Kubernetes Service (service.yaml)

```
apiVersion: v1
kind: Service
metadata:
  name: hello-world
spec:
  type: NodePort
  selector:
    app: hello-world
  ports:
  - port: 5000
    targetPort: 5000
```

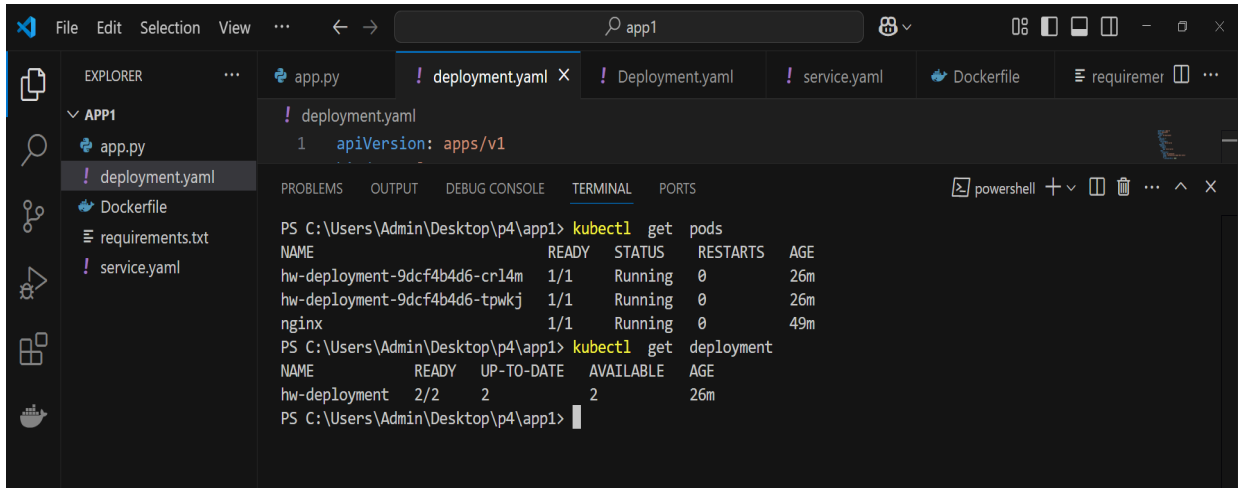
This makes your app accessible via NodePort on port 30005. Apply the Manifests

❏ **kubectl apply -f deployment.yaml**

❏ **kubectl apply -f service.yaml**

Verify:

- **kubectl get pods**
- **kubectl get svc**



The screenshot shows a Visual Studio Code editor with a terminal window open. The terminal is running PowerShell commands to interact with a Kubernetes cluster. The first command is `kubectl get pods`, which returns a table of pods. The second command is `kubectl get deployment`, which returns a table of deployment information.

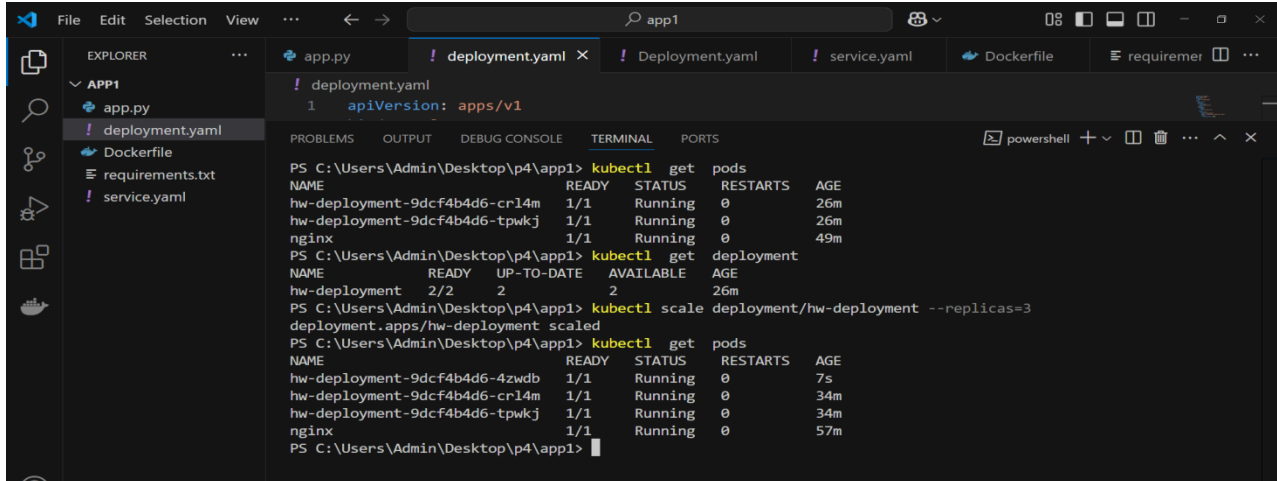
```
PS C:\Users\Admin\Desktop\p4\app1> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hw-deployment-9dcf4b4d6-cr14m       1/1     Running   0           26m
hw-deployment-9dcf4b4d6-tpwkj       1/1     Running   0           26m
nginx                                1/1     Running   0           49m
PS C:\Users\Admin\Desktop\p4\app1> kubectl get deployment
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
hw-deployment       2/2     2             2           26m
PS C:\Users\Admin\Desktop\p4\app1>
```

Replicating pods in Kubernetes is easy using **Deployments**. This is to tell Kubernetes how many **replicas** (copies) of your pod you want.

Syntax:

❏ `kubectl scale deployment <deployment-name> --replicas=<number>`

❏ **Example:** `kubectl scale deployment/hw-deployment --replicas=3`



The screenshot shows a Visual Studio Code editor with a terminal window open. The terminal is running PowerShell commands to interact with a Kubernetes cluster. The first command is `kubectl get pods`, which returns a table of pods. The second command is `kubectl get deployment`, which returns a table of deployment information. The third command is `kubectl scale deployment/hw-deployment --replicas=3`, which scales the deployment. The fourth command is `kubectl get pods`, which returns a table of pods showing three replicas.

```
PS C:\Users\Admin\Desktop\p4\app1> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hw-deployment-9dcf4b4d6-cr14m       1/1     Running   0           26m
hw-deployment-9dcf4b4d6-tpwkj       1/1     Running   0           26m
nginx                                1/1     Running   0           49m
PS C:\Users\Admin\Desktop\p4\app1> kubectl get deployment
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
hw-deployment       2/2     2             2           26m
PS C:\Users\Admin\Desktop\p4\app1> kubectl scale deployment/hw-deployment --replicas=3
deployment.apps/hw-deployment scaled
PS C:\Users\Admin\Desktop\p4\app1> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hw-deployment-9dcf4b4d6-4zwdb       1/1     Running   0           7s
hw-deployment-9dcf4b4d6-cr14m       1/1     Running   0           34m
hw-deployment-9dcf4b4d6-tpwkj       1/1     Running   0           34m
nginx                                1/1     Running   0           57m
PS C:\Users\Admin\Desktop\p4\app1>
```

To see how many replicas are running:

❏ `kubectl get deployment`

❏ `kubectl get pods`

Output:

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
hw-deployment	3/3	3	3	5m

```

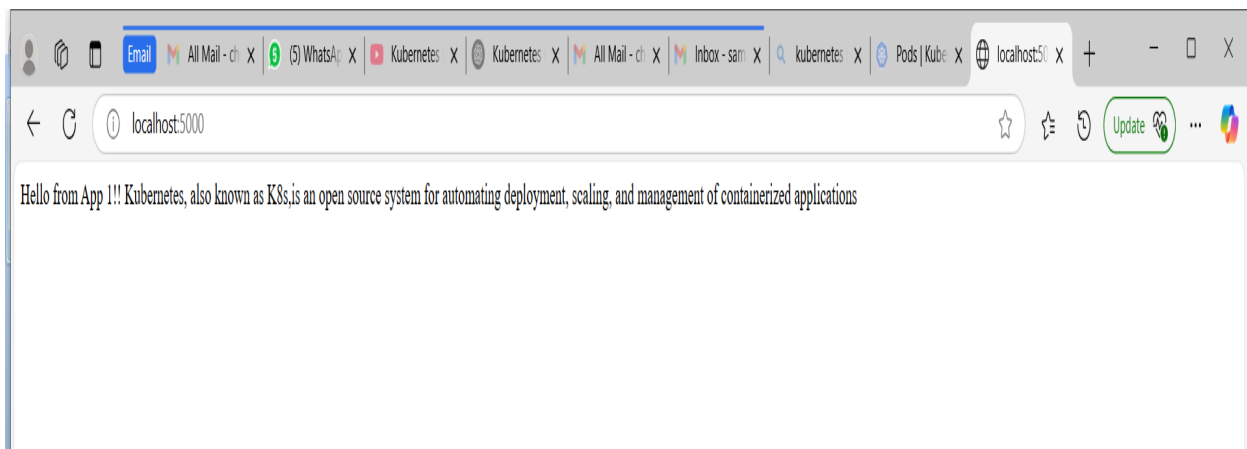
PS C:\Users\Admin\Desktop\p4\app1> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hw-deployment-9dcf4b4d6-4zwdb       1/1     Running   0           7m25s
hw-deployment-9dcf4b4d6-cr14m       1/1     Running   0           41m
hw-deployment-9dcf4b4d6-tpwkj       1/1     Running   0           41m
nginx                                1/1     Running   0           64m
PS C:\Users\Admin\Desktop\p4\app1> kubectl delete pod hw-deployment-9dcf4b4d6-4zwdb
pod "hw-deployment-9dcf4b4d6-4zwdb" deleted
PS C:\Users\Admin\Desktop\p4\app1> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hw-deployment-9dcf4b4d6-cr14m       1/1     Running   0           42m
hw-deployment-9dcf4b4d6-f6qz7       1/1     Running   0           37s
hw-deployment-9dcf4b4d6-tpwkj       1/1     Running   0           42m
nginx                                1/1     Running   0           65m
PS C:\Users\Admin\Desktop\p4\app1>

```

Forwards container port 5000 to host port 5000

❑ kubectl port-forward svc/hello-world 5000:5000

❑ Goto browser and type <http://localhost:5000>



Simple Python application in Kubernetes using ConfigMap and Secret.

- Python App (app.py)

```
from flask import Flask
import os

app = Flask(__name__)

@app.route('/')
def index():
    app_env = os.getenv("APP_ENV", "not set")
    db_password = os.getenv("DB_PASSWORD", "not set")
    return f"APP_ENV: {app_env} <br> DB_PASSWORD: {db_password}"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

- Dockerfile

```
FROM python:3.9-slim

WORKDIR /app
COPY app1.py .
RUN pip install flask
CMD ["python", "app1.py"]
```

- Kubernetes Deployment (deployment.yaml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: python-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: python-app
  template:
    metadata:
      labels:
        app: python-app
    spec:
      containers:
        - name: app-container
          image: chethanaravi/python-app:latest
          ports:
            - containerPort: 5000
          env:
            - name: APP_ENV
              valueFrom:
                configMapKeyRef:
                  name: my-config
                  key: APP_ENV
            - name: DB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: my-secret
                  key: DB_PASSWORD
```

- Service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: python-service
spec:
  type: NodePort
  selector:
    app: python-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
      nodePort: 30005
```

- ConfigMap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
data:
  APP_ENV: production
```

- Secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
stringData:
  DB_PASSWORD: mypassword123
```

Build and Push Docker Image

❑ **docker built -t python-app .**

❑ **docker built -t chethanaravi/python-app:latest**

Apply Everything

- **kubectl apply -f configmap.yaml**
- **kubectl apply -f secret.yaml**
- **kubectl apply -f deployment.yaml**
- **kubectl apply -f service.yaml**

Check the Pod and Service Status

❑ **kubectl get pods**

❑ **kubectl get svc**

The screenshot shows the Visual Studio Code interface with a project named 'APP3'. The Explorer pane on the left lists files: app1.py, ConfigMap.yaml, Deployment.yaml, Dockerfile, Secret.yaml, and Service.yaml. The main editor displays the Service.yaml file with the configuration: `apiVersion: v1`. The integrated terminal shows the following commands and output:

```
PS C:\Users\Admin\Desktop\p4\app3> kubectl apply -f deployment.yaml
deployment.apps/python-app created
PS C:\Users\Admin\Desktop\p4\app3> kubectl apply -f service.yaml
service/python-service created
PS C:\Users\Admin\Desktop\p4\app3> kubectl apply -f ConfigMap.yaml
configmap/my-config unchanged
PS C:\Users\Admin\Desktop\p4\app3> kubectl apply -f Service.yaml
service/python-service unchanged
PS C:\Users\Admin\Desktop\p4\app3> kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
hw-deployment-9dcf4b4d6-cr14m	1/1	Running	0	130m
hw-deployment-9dcf4b4d6-f6qz7	1/1	Running	0	89m
hw-deployment-9dcf4b4d6-tpwkj	1/1	Running	0	130m
nginx	1/1	Running	0	153m
python-app-5984c86dc-55f4f	1/1	Running	0	116s

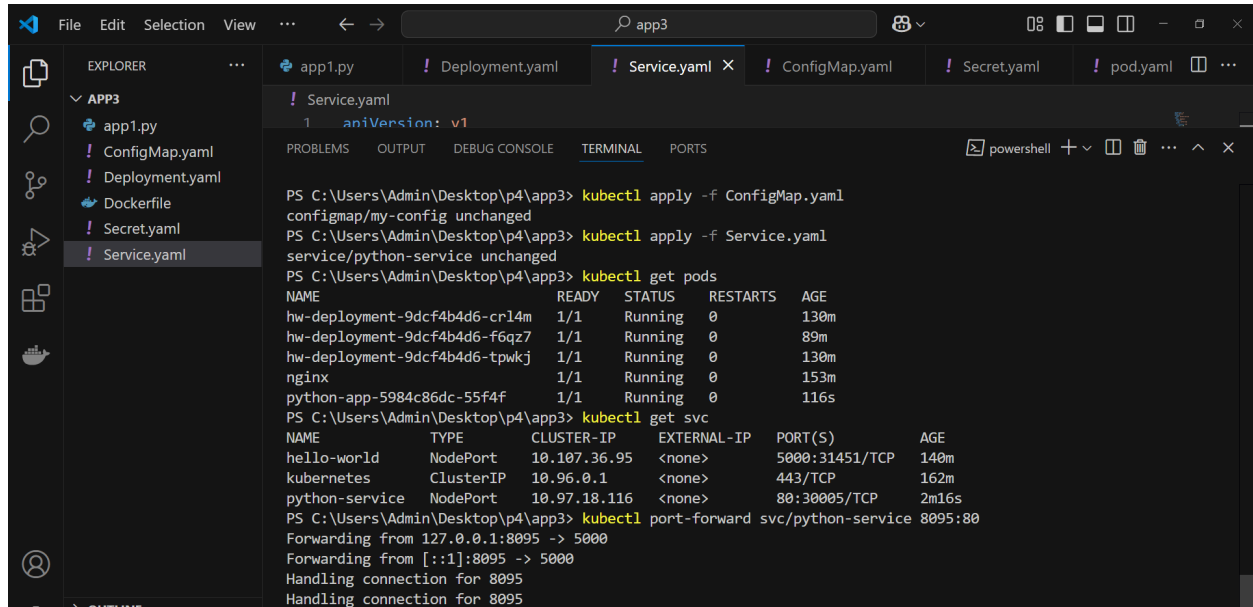
```
PS C:\Users\Admin\Desktop\p4\app3> kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
hello-world	NodePort	10.107.36.95	<none>	5000:31451/TCP	140m
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	162m
python-service	NodePort	10.97.18.116	<none>	80:30005/TCP	2m16s

The terminal ends with the prompt `PS C:\Users\Admin\Desktop\p4\app3>`. An 'Activate Windows' watermark is visible in the bottom right corner.

Port Forwarding

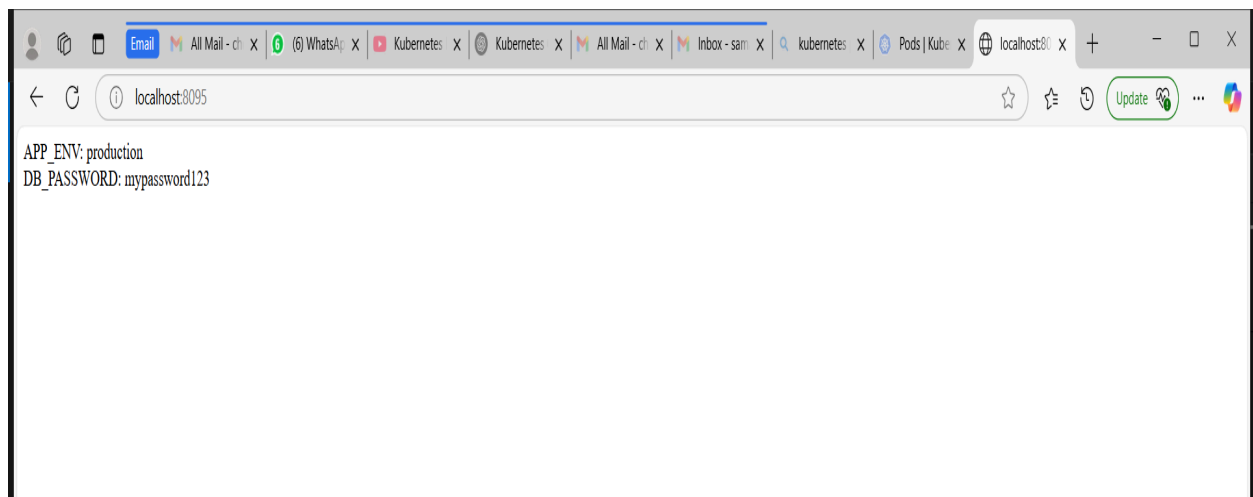
- **kubectl port-forward svc/python-service 8095:80**
- <http://localhost:8095>



The screenshot shows a VS Code terminal window with the following commands and output:

```
PS C:\Users\Admin\Desktop\p4\app3> kubectl apply -f ConfigMap.yaml
configmap/my-config unchanged
PS C:\Users\Admin\Desktop\p4\app3> kubectl apply -f Service.yaml
service/python-service unchanged
PS C:\Users\Admin\Desktop\p4\app3> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hw-deployment-9dcf4b4d6-cr14m       1/1     Running   0           130m
hw-deployment-9dcf4b4d6-f6qz7       1/1     Running   0           89m
hw-deployment-9dcf4b4d6-tpwkj       1/1     Running   0           130m
nginx                                1/1     Running   0           153m
python-app-5984c86dc-55f4f          1/1     Running   0           116s
PS C:\Users\Admin\Desktop\p4\app3> kubectl get svc
NAME             TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
hello-world      NodePort    10.107.36.95 <none>        5000:31451/TCP   140m
kubernetes        ClusterIP   10.96.0.1    <none>        443/TCP          162m
python-service    NodePort    10.97.18.116 <none>        80:30005/TCP     2m16s
PS C:\Users\Admin\Desktop\p4\app3> kubectl port-forward svc/python-service 8095:80
Forwarding from 127.0.0.1:8095 -> 5000
Forwarding from [::1]:8095 -> 5000
Handling connection for 8095
Handling connection for 8095
```

This bypasses NodePort and goes directly to the service inside the cluster.



Note:

- Delete All Pods in the Current Namespace (usually default):

kubectrl delete pods --all

- Delete Everything (Pods, Deployments, Services, etc.)

kubectrl delete all --all

- Prevent Pods from Coming Back

kubectrl delete deployment <deployment-name> / kubectrl delete deployments --all