First of all we need python application required files from developes:-

1. app.py
2. requirements.txt ……………….we can create this by running :- pip freeze > requirements.txt(If error got occurred while installing many time then we can adjust add or delete only required files in it )

After files has been taken from developers:-

We can check if application is running or not by running :- python app.py

**Step 1: Prepare Your Python Application**

Ensure your Python application has a requirements.txt file and a main entry script (e.g., app.py for a Flask or FastAPI application).

**Example Python Application (app.py)**

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route("/")

def home():

return "Hello, Kubernetes with Docker!"

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host="0.0.0.0", port=5000)

**requirements.txt**

flask

**Step 2: Create a Dockerfile**

This file defines how your application should be containerized.

**Dockerfile**

# Use an official Python runtime as a parent image

FROM python:3.9

# Set the working directory

WORKDIR /app

# Copy the current directory contents into the container

COPY . /app

# Install dependencies

RUN pip install --no-cache-dir -r requirements.txt

# Expose port 5000 for Flask

EXPOSE 5000

# Define environment variable

ENV FLASK\_APP=app.py

# Command to run the application

CMD ["python", "app.py"]

**Step 3: Build and Run the Docker Image**

Run the following commands in your terminal:

# Build the Docker image

docker build -t my-python-app .

# Run the container locally to test

docker run -p 5000:5000 my-python-app

Visit http://localhost:5000 in your browser to check if it's working.

**Step 4: Push the Docker Image to Docker Hub**

First, log in to Docker Hub:

docker login

Tag the image:

docker tag my-python-app your-dockerhub-username/my-python-app:v1

Push the image:

docker push your-dockerhub-username/my-python-app:v1

**Step 5: Set Up Kubernetes Deployment**

Now, create Kubernetes YAML files to deploy the application.

**deployment.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: python-app

spec:

replicas: 2

selector:

matchLabels:

app: python-app

template:

metadata:

labels:

app: python-app

spec:

containers:

- name: python-app

image: your-dockerhub-username/my-python-app:v1

ports:

- containerPort: 5000

**service.yaml**

apiVersion: v1

kind: Service

metadata:

name: python-service

spec:

selector:

app: python-app

ports:

- protocol: TCP

port: 80

targetPort: 5000

type: LoadBalancer

**Step 6: Deploy to Kubernetes**

**1️⃣ Start Minikube (if using locally)**

minikube start

**2️⃣ Apply Kubernetes Configuration**

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

**3️⃣ Verify Deployment**

kubectl get pods

kubectl get services

**4️⃣ Access the Application**

minikube service python-service

**Step 7: Scale and Update Deployment**

**Scale the app**

kubectl scale deployment python-app --replicas=4

**Update the image**

kubectl set image deployment/python-app python-app=your-dockerhub-username/my-python-app:v2

**Now to find ip address on which application is running :-**

kubectl describe service python-service

**Your service python-service is exposed as a LoadBalancer with:**

* **Cluster IP**: 10.99.150.207 (only accessible within the cluster)
* **External IP**: localhost
* **Port**: 80 (forwarded from NodePort 30241)

**✅ Find the IP Where Your Application is Running**

**🔹 Method 1: Use Minikube**

If you're using Minikube, run:

minikube service python-service --url

This will return the exact URL.

**🔹 Method 2: Get NodePort and Access via Localhost**

Since the external IP is localhost, you can access the service using:

http://localhost:80

or

http://localhost:30241

if 80 is not working.

**🔹 Method 3: Describe the Service**

Run:

kubectl describe service python-service

Look for the **"Ingress"** or **"External IP"** field.

**🔹 Method 4: Get Pod IPs**

To check where the actual app is running:

kubectl get pods -o wide

This will show the internal pod IP addresses.

**🔹 How to Access the Application?**

**1️⃣ Using LoadBalancer IP (Preferred)**

Since LoadBalancer Ingress is **localhost**, try opening in your browser or via curl:

http://localhost:80

or

curl http://localhost:80

**2️⃣ Using NodePort**

Your service is mapped to NodePort: 30241, so you can also try:

http://localhost:30241

**3️⃣ Using Pod IP (Inside Cluster Only)**

The actual pods are running on:

* 10.1.0.7:5000
* 10.1.0.6:5000

These are internal cluster IPs, meaning you **cannot access them directly** from outside unless you use a kubectl port-forward or expose them.