**PROJECT 1: DEPLOY A MULTI-TIER WEB APPLICATION ON KUBERNETES**

🛠 **Prerequisites**

- Kubernetes Cluster (minikube/kubeadm)

- Docker

- kubectl CLI

**Folder Structure & File Usages**

k8s-project/

│── mysql/ # MySQL Database Configuration

│ ├── mysql-pv.yaml # Persistent Volume for MySQL Data Storage

│ ├── mysql-secret.yaml # Stores MySQL Root Password Securely

│ ├── mysql-deployment.yaml # Deploys MySQL Database as a StatefulSet

│── flask/ # Flask Backend Configuration

│ ├── app.py # Flask API Code to Handle Requests

│ ├── Dockerfile # Flask App Containerization Instructions

│ ├── requirements.txt # Dependencies for Flask

│ ├── flask-deployment.yaml # Deploys Flask Application

│ ├── flask-service.yaml # Exposes Flask App as a Cluster Service

│── nginx/ # Nginx Configuration

│ ├── nginx-configmap.yaml # Reverse Proxy Configuration for Flask

│ ├── nginx-deployment.yaml # Deploys Nginx

│ ├── nginx-service.yaml # Exposes Nginx via NodePort

📄 **Explanation of Each File**

**Persistent Volume (`mysql-pv.yaml`)**

- Kubernetes Persistent Volume (PV) is used to store MySQL data permanently.

- Ensures MySQL data is not lost even if the pod restarts.

- The PersistentVolumeClaim (PVC) allows the pod to request storage dynamically.

**StatefulSet (`mysql-deployment.yaml`)**

- Used instead of Deployment because MySQL requires stable network identity and persistent storage.

- Ensures MySQL pods maintain the same hostname across restarts.

**Secret (`mysql-secret.yaml`)**

- Stores sensitive credentials securely (e.g., MySQL password).

- Used in MySQL deployment as an environment variable to avoid exposing secrets in YAML.

**ConfigMap (`nginx-configmap.yaml`)**

- Stores non-sensitive configuration data.

- Used to configure Nginx as a reverse proxy, mapping requests to Flask.

**Deployment (`flask-deployment.yaml` & `nginx-deployment.yaml`)**

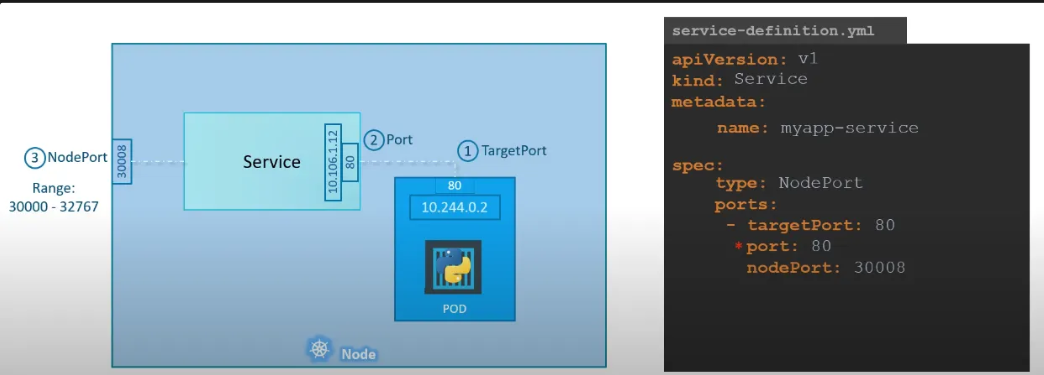
- Flask Deployment: Runs the Flask application.

- Nginx Deployment: Runs Nginx as a proxy for Flask.

**Service (`flask-service.yaml` & `nginx-service.yaml`)**

- **Flask Service**: Exposes Flask inside the cluster using `**ClusterIP**`.

- **Nginx Service**: Exposes Nginx externally using `**NodePort**`, allowing access from a browser.



Code link : [GitHub - jkbarathkumar/multi-tier-application](https://github.com/jkbarathkumar/multi-tier-application)

Commands:

After code setup enter these commands:

1. Build and push the docker image:
   1. Navigate to dockerfile folder
   2. docker build -t dockerhub\_username/flaskapp .
   3. docker push dockerhub\_username/flaskapp

1) cd flask

2) kubectl apply -f flask-deployment.yaml

3) kubectl apply -f flask-service.yaml

4) cd ..; cd mysql

5) kubectl apply -f mysql-deployment.yaml

6) kubectl apply -f mysql-pv.yaml

7) kubectl apply -f mysql-secret.yaml

8) cd ..; cd ngnix

9) kubectl apply -f nginx-configmap.yaml

10) kubectl apply -f nginx-deployment.yaml

11) kubectl apply -f nginx-service.yaml

12) Check Kubernetes pods,deployments,services running status

kubectl get all -o wide

13) To access the application:

Kubectl get nodes

Copy worker 1 IP or worker 2 IP and in browser enter http://<IP>:30007/

To access users http://<IP>:30007/users

14) It will return no users found or no db found:

**Access MySQL Inside the Pod and add the data**

kubectl exec -it mysql-0 -- mysql -u root -p

CREATE DATABASE mydb;

USE mydb;

SHOW TABLES;

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(100)

);

INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');

INSERT INTO users (name, email) VALUES ('Bob', 'bob@example.com');

SELECT \* FROM users;

GRANT ALL PRIVILEGES ON mydb.\* TO 'user'@'%';

FLUSH PRIVILEGES;

Errors:  
If you get Imagepullerror kindly pull the image manually in worker1 and worker2 vm