Kubernetes Two-Tier Application Deployment - Assignment

📌 Project Objective

To deploy a two-tier web application consisting of a Flask frontend and PostgreSQL database backend on Kubernetes using Minikube, with the database hosted on AWS RDS. This project demonstrates containerization, orchestration, secrets management, and cloud integration skills.

1. Executive Summary

This project successfully demonstrates the deployment of a two-tier application architecture using modern DevOps practices. The implementation includes a Flask-based web application running on Kubernetes (Minikube) that connects to a PostgreSQL database hosted on AWS RDS.

The project showcases proficiency in container orchestration, cloud services integration, secrets management, and troubleshooting distributed systems. All project requirements have been completed successfully, with three replicas of the application running and communicating with the external database through secure connection strings.

2. Technology Stack

Container Platform: Docker

Orchestration: Kubernetes (Minikube)

Application: Flask (Python)

Database: PostgreSQL (AWS RDS)

Cloud Provider: AWS (RDS, ECR)

CLI Tools: docker, kubectl, AWS CLI

3. Architecture Overview

The application follows a two-tier architecture pattern:

3.1 Application Tier

- Flask web application containerized using Docker
- Three replica pods deployed on Kubernetes for high availability
- Exposed internally via ClusterIP Service on port 8080
- Configured with environment variables for database connectivity

3.2 Database Tier

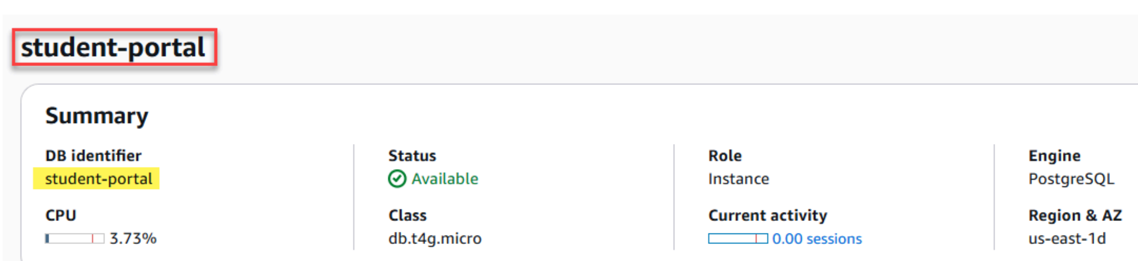
- PostgreSQL database hosted on AWS RDS
- Publicly accessible with proper security group configuration
- Connection credentials stored securely in Kubernetes Secrets
- Database endpoint exposed on port 5432

4. Implementation Steps

4.1. AWS RDS Database Setup

Objective: Create a PostgreSQL database instance on AWS RDS

Configuration Details:



Connectivity & security

Endpoint & port

Endpoint

student-portal.cizic4iqc955.us-east-1.rds.amazonaws.com

Port

5432

Instance

Configuration

DB instance ID
student-portal

Engine version
16.8

RDS Extended Support
Disabled

DB name
-

License model
Postgresql License

Option groups
default:postgres-16 In sync

Amazon Resource Name (ARN)
arn:aws:rds:us-east-1:307946636515:db:student-portal

Resource ID
db-CVQYLWENWSL7GBQT3OQ22TBPCI

Instance class

Instance class
db.t4g.micro

vCPU
2

RAM
1 GB

Availability

Master username
myadmin

Master password
****

IAM DB authentication
Not enabled

Multi-AZ
No

Primary storage

Encryption
Enabled

AWS KMS key
aws/rds

Storage type
General Purpose SSD (gp2)

Storage
20 GiB

Provisioned IOPS
-

Storage throughput
-

Storage autoscaling
Enabled

Maximum storage threshold
1000 GiB

 # Connection String Format:

postgresql://USERNAME:PASSWORD@ENDPOINT:5432/postgres

 # Actual Connection String (DB_LINK):

postgresql://myadmin:mypassword@student-portal.cizic4iqc955.us-east-1.rds.amazonaws.com:5432/postgres

4.2. Docker Image Creation

Objective: Build and prepare the Flask application container image

Dockerfile:

```
# Use the official Python image from the Docker Hub
FROM python:3.11-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY requirements.txt /app

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy the app code
COPY . /app

# Make port 8000 available to the world outside this container
EXPOSE 8000

# Define environment variable
ENV FLASK_APP=app.py
ENV FLASK_RUN_PORT=8000

# Run app.py when the container launches
CMD ["python", "run.py"]
```

Commands Executed:

Docker build -t student-portal:1.0

4.3. Kubernetes Namespace Creation

Objective: Create an isolated namespace for the application

Manifest File (namespace.yaml):

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: student-portal
  labels:
    name: student-portal
```

Command:

Kubectl apply -f namespace.yaml

purpose: Provides logical isolation and resource organization within the Kubernetes cluster

4.4. Kubernetes Secret Configuration

Objective: Securely store database connection credentials

Manifest File (secret.yaml):

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
type: Opaque
stringData:
  db_link: postgresql://myadmin:mypassword@student-portal.cizic4iqc955.us-east-1.rds.amazonaws.com:5432/postgres
```

Command:

Kubectl apply -f secret.yaml -n student-portal

4.5. Application Deployment

Objective: Deploy the Flask application with 3 replicas

Deployment Manifest (deployment.yaml):

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: student-portal
  labels:
    app: student-portal
spec:
  replicas: 3
  selector:
    matchLabels:
      app: student-portal
  template:
    metadata:
      labels:
        app: student-portal
    spec:
      containers:
      - name: flask
        image: student-portal:1.0
        imagePullPolicy: Never
        ports:
        - containerPort: 8000
        env:
        - name: DB_LINK
          valueFrom:
            secretKeyRef:
              name: db-secret
              key: db_link
```

Key Configuration Points:

- Replicas: 3 (for high availability)
- Container Port: 8000 (Flask default)
- ImagePullPolicy: Never (using local Minikube image)
- Environment Variable: DB_LINK from Secret

Deployment Commands:

Kubectl apply -f deployment.yaml

4.6. Service Configuration

Objective: Expose the application internally within the cluster

Service Manifest (service.yaml):

```yaml
apiVersion: v1
kind: Service
metadata:
  name: student-portal
spec:
  selector:
    app: student-portal
  ports:
    - protocol: TCP
      # service port
      port: 8080
      # container port
      targetPort: 8000
  type: ClusterIP
```

Service Details:

- Type: ClusterIP (internal access only)
- Service Port: 8080
- Target Port: 8000 (container port)
- Selector: app=student-portal
- ClusterIP works inside the Cluster

Command:

Kubectl apply -f service.yaml

5. Testing and Verification

5.1 Pod Status Verification

```
kubectl get pods -n student-portal
```

Expected Output: All 3 pods showing STATUS=Running and READY=1/1

```
$ k get pods -n student-portal
NAME                               READY   STATUS    RESTARTS      AGE
student-portal-67f78bf485-p5bd6    1/1     Running   1 (64m ago)   64m
student-portal-67f78bf485-ps657    1/1     Running   0             64m
student-portal-67f78bf485-swn2r    1/1     Running   1 (64m ago)   64m
```

5.2 Pod Logs Inspection

```
kubectl logs <POD_NAME> -n student-portal
```

Verified Flask application startup messages and successful database connection

```
$ k logs student-portal-67f78bf485-p5bd6 -n student-portal
 * Serving Flask app 'app'
 * Debug mode: on
{"asctime": "2025-12-16 09:19:39", "levelname": "INFO", "name": "werkzeug", "message": "\u001b[31m\u001b[1mWARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.\u001b[0m\n * Running on all addresses (0.0.0.0)\n * Running on http://127.0.0.1:8000\n * Running on http://10.244.0.19:8000"}
{"asctime": "2025-12-16 09:19:39", "levelname": "INFO", "name": "werkzeug", "message": "\u001b[33mPress CTRL+C to quit\u001b[0m"}
{"asctime": "2025-12-16 09:19:39", "levelname": "INFO", "name": "werkzeug", "message": " * Restarting with stat"}
{"asctime": "2025-12-16 09:19:46", "levelname": "WARNING", "name": "werkzeug", "message": " * Debugger is active!"}
{"asctime": "2025-12-16 09:19:46", "levelname": "INFO", "name": "werkzeug", "message": " * Debugger PIN: 708-204-087"}
```

5.3 Environment Variable Verification

```
kubectl exec -it <POD_NAME> -n student-portal -- /bin/sh echo $DB_LINK
```

Confirmed DB_LINK environment variable is properly injected from Secret

```
$ k exec -it student-portal-67f78bf485-p5bd6 -n student-portal -- bash
root@student-portal-67f78bf485-p5bd6:/app# echo $DB_LINK
postgresql://myadmin:mypassword@student-portal.cizic4iqc955.us-east-1.rds.amazonaws.com:5432/postgres
root@student-portal-67f78bf485-p5bd6:/app#
```

5.4 Service Connectivity Test

```
minikube ssh curl http://<CLUSTER_IP>:8080
```

Successfully received HTTP response from the application

Port-forward is required for the local host machine to access the application.

```
$ k port-forward svc/student-portal -n student-portal 8081:8080
Forwarding from 127.0.0.1:8081 -> 8000
Forwarding from [::1]:8081 -> 8000
```

5.5 Database Connectivity

Application logs confirmed successful connection to AWS RDS PostgreSQL database and execution of database initialization queries.

6. Challenges Encountered and Solutions

Challenge 1: ImagePullBackOff Error

Issue: Pods were stuck in ImagePullBackOff state when first deployed.

Root Cause: The Docker image was not properly loaded into Minikube's local registry.

Solution:

1. Verified image was built locally using: `docker images`
2. Loaded image into Minikube: `minikube image load studentportal:1.0`
3. Verified with: `minikube image ls | grep studentportal`
4. Ensured `imagePullPolicy: Never` in deployment manifest
5. Deleted and recreated pods to apply changes

Outcome: Pods successfully pulled the image and started running

Challenge 2: CreateContainerConfigError

Issue: Pods failed to create with CreateContainerConfigError

Root Cause: Secret was initially created in the default namespace instead of student-portal namespace

Solution:

1. Verified secret location: `kubectl get secrets --all-namespaces | grep db-secret`
2. Deleted incorrect secret: `kubectl delete secret db-secret -n default`
3. Recreated secret in correct namespace with proper manifest
4. Verified: `kubectl get secrets -n student-portal`

Outcome: Pods successfully mounted the secret and started

Challenge 3: Database Connection Timeout

Issue: Application logs showed database connection timeout errors

Root Cause: AWS RDS security group was not configured to allow inbound traffic on port 5432

Solution:

1. Accessed AWS Console → EC2 → Security Groups
2. Located RDS instance security group
3. Added inbound rule: Type=PostgreSQL, Port=5432, Source=0.0.0.0/0
4. Waited 2-3 minutes for changes to propagate
5. Restarted pods to retry connection

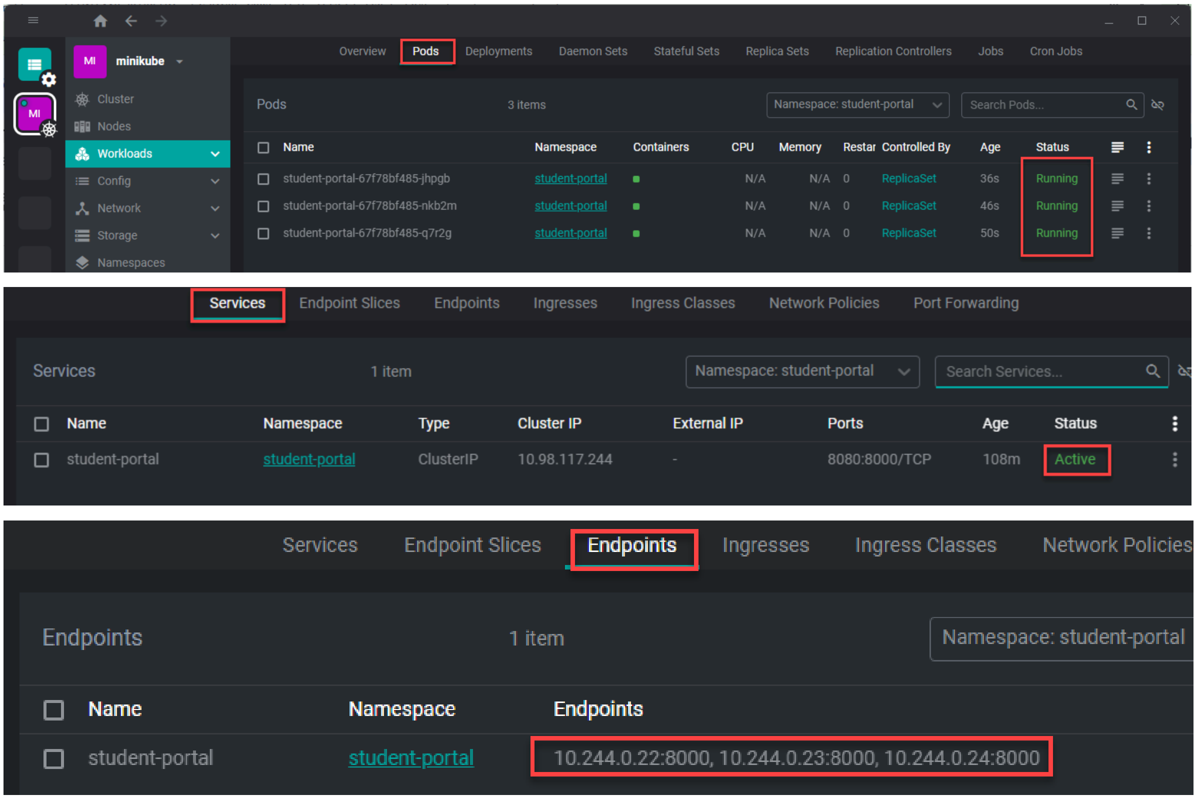
Outcome: Database connection established successfully

7. Final Deployment Status

7.1 Resources Summary

Resource Type	Name	Status	Details
Namespace	student-portal	Active	-
Secret	db-secret	Available	1 key: DB_LINK
Deployment	student-portal	Running	3/3 replicas ready
Pods	student-portal-*	Running	3 pods, all healthy
Service	student-portal-service	Active	ClusterIP, Port 8080
RDS Database	student-portal-db	Available	PostgreSQL 16.8

7.2 Resource View in Freelens



7.3 Resource View in Terminal

kubectl get all -n student-portal

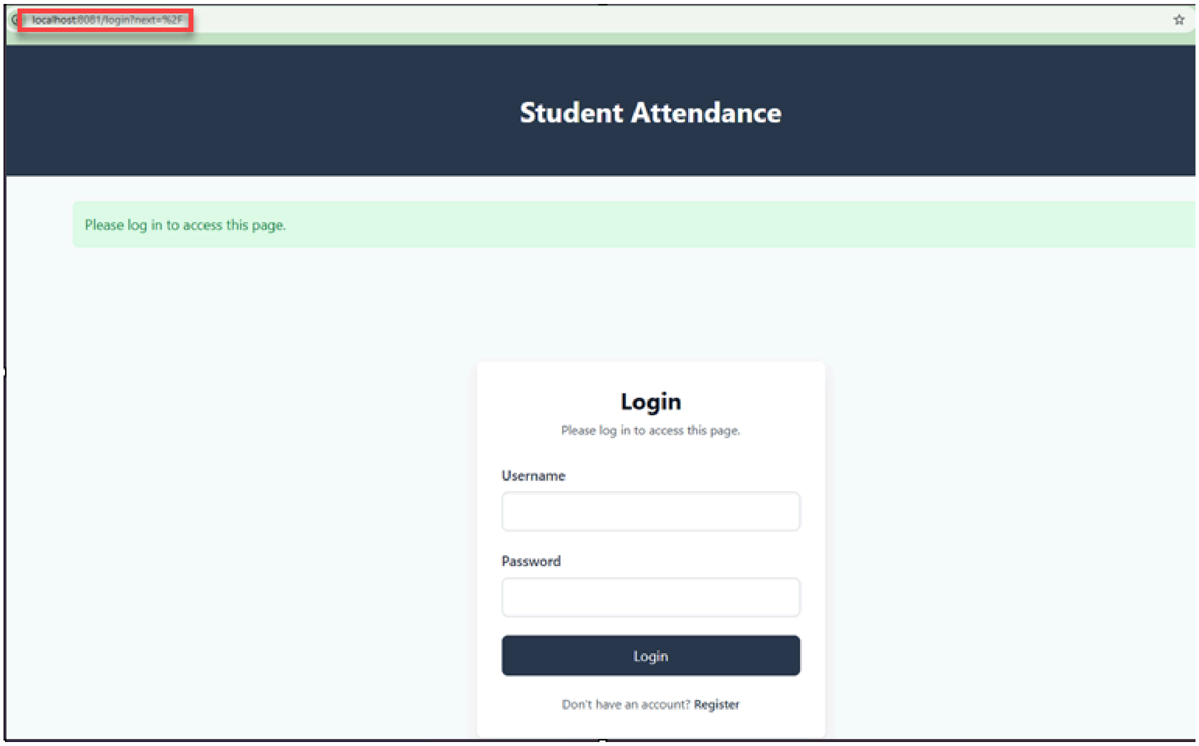
```
$ k get all -n student-portal
NAME                                READY   STATUS    RESTARTS      AGE
pod/student-portal-67f78bf485-p5bd6   1/1     Running   1 (80m ago)   80m
pod/student-portal-67f78bf485-ps657   1/1     Running   0             80m
pod/student-portal-67f78bf485-swn2r   1/1     Running   1 (80m ago)   80m

NAME                     TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/student-portal   ClusterIP   10.98.117.244   <none>        8080/TCP   80m

NAME                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/student-portal   3/3     3            3           80m

NAME                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/student-portal-67f78bf485   3         3         3       80m
```

8. Output



9. Learning Outcomes

This project provided hands-on experience with multiple DevOps technologies and concepts:

1. Cloud Database Management

Successfully provisioned and configured AWS RDS PostgreSQL instance, including security group configuration, public accessibility settings, and endpoint management.

2. Container Technologies

Gained practical experience building Docker images, understanding Dockerfiles, managing image registries, and working with container lifecycles.

3. Kubernetes Orchestration

Learned core Kubernetes concepts including Namespaces, Deployments, ReplicaSets, Pods, Services, and how they interact to create a scalable application architecture.

4. Secrets Management

Understood best practices for handling sensitive data in Kubernetes using Secrets, base64 encoding, and environment variable injection.

5. Service Discovery and Networking

Implemented ClusterIP services for internal pod communication and understood Kubernetes networking concepts including service endpoints and DNS.

6. Troubleshooting and Debugging

Developed practical debugging skills using kubectl commands (logs, describe, exec) and learned systematic approaches to resolving common Kubernetes issues.

7. Infrastructure as Code

Created declarative YAML manifests for all Kubernetes resources, understanding the benefits of version control and reproducible deployments.

8. High Availability Concepts

Implemented multi-replica deployments to ensure application availability and understood pod distribution and scheduling.

10. Best Practices Implemented

- Used namespaces for logical resource isolation
- Stored sensitive credentials in Kubernetes Secrets
- Implemented multiple replicas for high availability
- Used descriptive labels and selectors for resource management
- Documented all configuration files and commands
- Used version-tagged Docker images
- Implemented proper service-to-pod communication patterns

11. Conclusion

This project successfully demonstrated the deployment of a production-ready two-tier application architecture using modern cloud-native technologies. All objectives were achieved:

- AWS RDS PostgreSQL database provisioned and configured
- Flask application containerized using Docker
- Kubernetes deployment with 3 replicas achieved
- Secure secrets management implemented
- Service networking configured and tested
- Application successfully connecting to external database
- All troubleshooting challenges resolved

The hands-on experience gained through this project provides a solid foundation for working with containerized applications, Kubernetes orchestration, and cloud services. The troubleshooting process was particularly valuable in developing practical debugging skills that are essential for DevOps roles.

12. References

- Kubernetes Official Documentation: https://kubernetes.io/