Deployment of a Three-Tier Application on Kubernetes

Prerequisites

Before you begin, ensure the following:

- *A Kubernetes cluster is up and running (e.g., via Minikube, AWS EKS, GKE, or AKS).*
- *kubectl is installed and configured to interact with your cluster.*
- *Docker is installed for building container images.*
- *Basic knowledge of Kubernetes concepts like Pods, Services, Deployments, and Persistent Volumes.*

Step 1: Prepare Application Code

Ensure you have the source code for each tier of the application:

1. ***Frontend**: Typically a static website (HTML/CSS/JS).*
2. ***Backend**: A service built using a programming language (e.g., Node.js, Python).*
3. ***Database**: A database management system (e.g., MySQL, PostgreSQL).*

Organize the code for each tier in separate directories.

Step 2: Containerize the Application

Create Dockerfiles for each tier.

Example Dockerfile for Frontend:

FROM nginx:alpine

COPY ./frontend /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

Example Dockerfile for Backend:

```
FROM node:16
WORKDIR /app
COPY ./backend .
RUN npm install
EXPOSE 3000
CMD ["node", "server.js"]
```

Example Dockerfile for Database:

If using a managed database image:

```
FROM postgres:15
ENV POSTGRES_USER=myuser
ENV POSTGRES_PASSWORD=mypassword
ENV POSTGRES_DB=mydatabase
EXPOSE 5432
```

Build and push these images to a container registry (e.g., Docker Hub):

```
docker build -t <your_registry>/frontend:latest
./frontend
docker build -t <your_registry>/backend:latest
./backend
docker build -t <your_registry>/database:latest
./database
```

```
docker push <your_registry>/frontend:latest
docker push <your_registry>/backend:latest
docker push <your_registry>/database:latest
```

Step 3: Define Kubernetes Manifests

Create YAML files for each tier.

Frontend Deployment and Service:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: frontend
spec:
 replicas: 3
 selector:
   matchLabels:
     app: frontend
 template:
   metadata:
     labels:
       app: frontend
   spec:
     containers:
     - name: frontend
       image: <your_registry>/frontend:latest
       ports:
       - containerPort: 80
---
```

```
apiVersion: v1
kind: Service
metadata:
 name: frontend-service
spec:
 selector:
   app: frontend
 ports:
 - protocol: TCP
   port: 80
   targetPort: 80
 type: LoadBalancer
```

Backend Deployment and Service:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: backend
spec:
 replicas: 3
 selector:
   matchLabels:
     app: backend
 template:
   metadata:
     labels:
       app: backend
   spec:
     containers:
```

```
      - name: backend
        image: <your_registry>/backend:latest
        ports:
        - containerPort: 3000
        env:
        - name: DATABASE_URL
          value:
"postgresql://myuser:mypassword@database-service:5432/m
ydatabase"
---
apiVersion: v1
kind: Service
metadata:
 name: backend-service
spec:
 selector:
   app: backend
 ports:
 - protocol: TCP
   port: 3000
   targetPort: 3000
 type: ClusterIP
```

Database Deployment and Service:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: database
spec:
```

```
 replicas: 1
 selector:
   matchLabels:
     app: database
 template:
   metadata:
     labels:
       app: database
   spec:
     containers:
     - name: database
       image: <your_registry>/database:latest
       ports:
       - containerPort: 5432
       volumeMounts:
       - name: database-storage
         mountPath: /var/lib/postgresql/data
     volumes:
     - name: database-storage
       persistentVolumeClaim:
         claimName: database-pvc
---
apiVersion: v1
kind: Service
metadata:
 name: database-service
spec:
 selector:
   app: database
```

```
 ports:
 - protocol: TCP
   port: 5432
   targetPort: 5432
 type: ClusterIP
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: database-pvc
spec:
 accessModes:
 - ReadWriteOnce
 resources:
   requests:
     storage: 10Gi
```

Step 4: Apply the Manifests

Deploy the application to your Kubernetes cluster:

```
kubectl apply -f frontend.yaml
kubectl apply -f backend.yaml
kubectl apply -f database.yaml
```

Step 5: Verify the Deployment

1. *Check the status of all resources:*
2. `kubectl get all`
3. *Access the application via the external IP of the frontend-service (if LoadBalancer is used):*

```
4. kubectl get svc
```

5. *Use the EXTERNAL-IP of* `frontend-service` *in your browser.*

Step 6: Monitor and Scale

- *Monitor Logs:*

```
- kubectl logs -f deployment/frontend
- kubectl logs -f deployment/backend
```

- *Scale Pods:*