Kubernetes-Based Multi-Tier Architecture

# 1. Requirement Understanding

Design and deploy a Kubernetes-based multi-tier application including:  
- A microservice (API tier) that connects to a PostgreSQL database.  
- External access to the API via Ingress.  
- Internal-only access for the database.  
- Secure configuration using ConfigMap and Secret for DB credentials.  
- Persistent storage for the database.  
- No use of Pod IPs — access via Services.  
- Rolling updates enabled for the API; static setup for DB.

# 2. Assumptions

- Tech Stack: React (UI)+ Node.js + Express (API) + PostgreSQL.  
- Docker images stored on Docker Hub.  
- Hosted on GKE using default `standard` StorageClass.  
- Kubernetes version v1.28+.  
- Ingress Controller already installed.  
- DNS and Hosts File Point to Ingress Controller. Domain (e.g., api.example.com) or local /etc/hosts entry should resolve to the Ingress Controller's external IP.

# 3. Solution Overview

|  |  |
| --- | --- |
| **Component** | **Details** |
| React UI | Dockerized React app, consumes Api Microservice |
| API Microservice | Dockerized Node.js app, connects to PostgreSQL using env vars |
| PostgreSQL DB | Preloaded with a sample table and 5–10 records |
| ConfigMap | Stores DB host, user, and dbname |
| Secret | Stores DB password securely |
| API Deployment | 4 replicas, supports rolling updates |
| DB Deployment/StatefulSet | 1 replica, bound to PVC for data persistence |
| Ingress | Exposes API externally |

# A screenshot of a computer screen AI-generated content may be incorrect.4. Resource Justification

|  |  |  |
| --- | --- | --- |
| Component | Resource Type | Justification |
| UI Tier | Deployment (2 Pods) | Handles load and supports rolling updates |
| API Tier | Deployment (4 Pods) | Handles load and supports rolling updates |
| DB Tier | Deployment (1 Pod) | Single instance for data consistency |
| PostgreSQL PV | PVC (1Gi) | Ensures data durability and prevents data loss |
| Config | ConfigMap | Decouples DB settings from code |
| Credentials | Secret | Keeps sensitive DB credentials secure |
| Access Layer | ClusterIP (DB), Ingress (API) | Reliable internal and external access via DNS |

# 5. Summary of Best Practices

|  |  |  |
| --- | --- | --- |
| Feature | Implemented? | Details |
| Connection pooling | Yes | `pg.Pool` used in Node.js app for DB connection reuse |
| Config separation | Yes | DB settings stored in ConfigMap |
| Secret management | Yes | DB password stored in Secret |
| Persistent DB storage | Yes | PVC attached to DB Pod |
| Stateless API with updates | Yes | 4 replicas managed with Deployment |
| External API access | Yes | Exposed via Ingress |
| Avoided direct Pod access | Yes | Internal access managed via Services |