Kubernetes and Docker Advanced Assignment

# Requirement Understanding

### Design

* The project follows a cloud-native architecture using microservices deployed on Kubernetes.
* Includes an API tier and a backend database, designed for scalability and fault tolerance.
* Configuration management is externalized via Kubernetes ConfigMaps and Secrets.
* All communication between tiers uses Kubernetes service discovery, avoiding Pod IPs.
* API tier is externally exposed via Kubernetes Ingress to enable secure and scalable access.

### Implementation

* The system is containerized using Docker to ensure consistency across environments.
* Persistent volumes (PVs) and persistent volume claims (PVCs) are used to retain database data even if the corresponding pod is re-deployed.
* The build, deployment, and lifecycle processes are automated using scripts and container orchestration, improving reliability and reproducibility.
* Infrastructure as Code (IaC) principles are applied to provision and configure environments in a consistent and version-controlled manner.

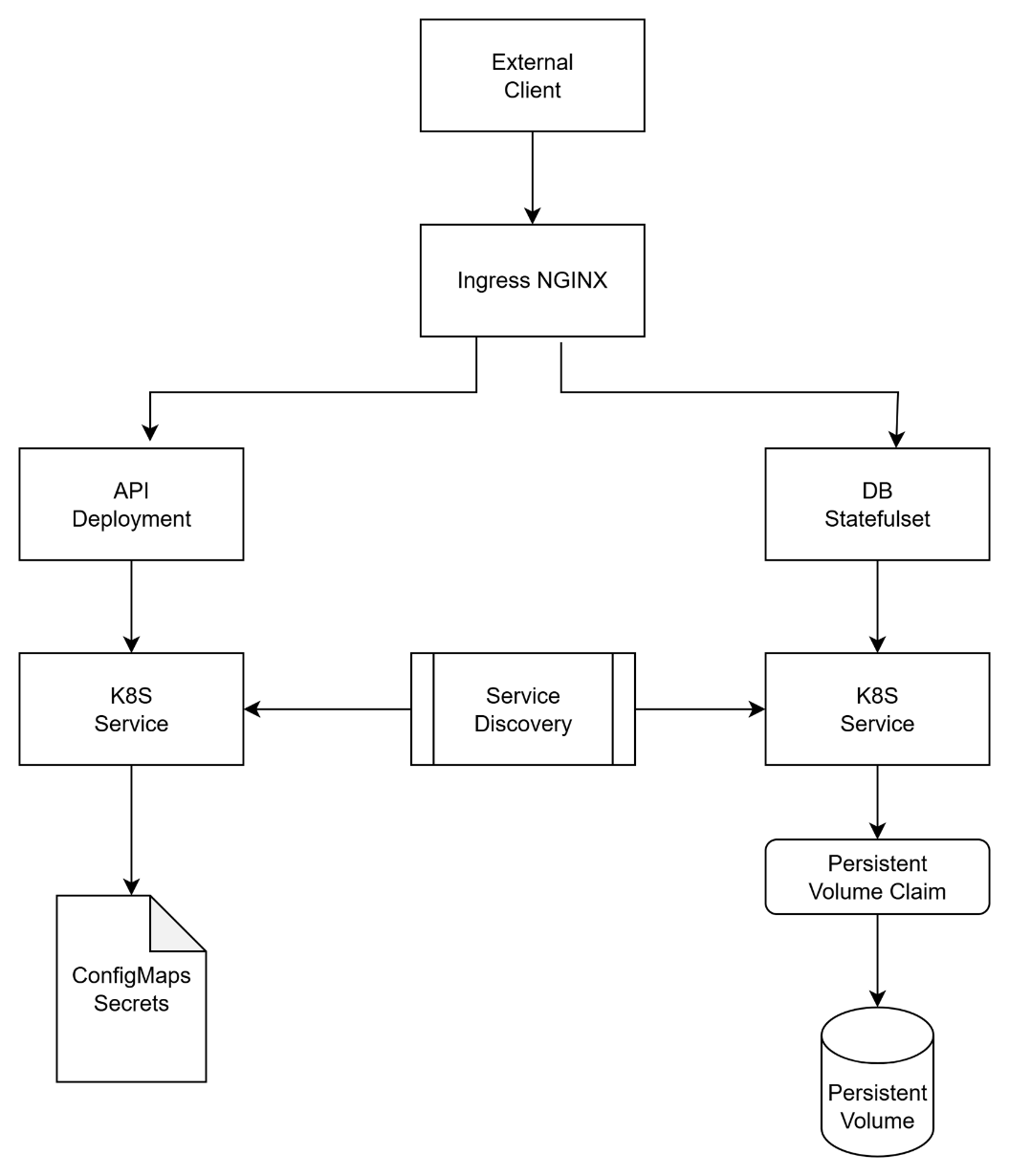
### Operations

* Database resilience is achieved by decoupling state from pods, ensuring data persists independently.
* Inter-tier service discovery and load balancing are managed natively by Kubernetes Services.
* Deployment automation reduces manual errors and simplifies ongoing maintenance.
* IaC enables teams to replicate environments and apply consistent configurations during updates or scaling activities.
* Ingress controllers help manage traffic routing and facilitate observability and security at the API entry point.

# Assumptions

* The Kubernetes cluster is pre-provisioned and accessible.
* Docker images are built and pushed to Docker Hub.
* NGINX Ingress controller is installed in the cluster.
* The user has access to update their local hosts file for Ingress testing.
* Persistent storage is available for the database.

# Solution Overview



* API and DB are deployed as separate pods using Deployments/StatefulSets.
* All configuration (including DB credentials) is externalized using ConfigMaps and Secrets.
* API tier is exposed externally using an Ingress resource, not a LoadBalancer.
* Pods are managed by Kubernetes, ensuring self-healing and high availability.
* Database data persisted using a PersistentVolumeClaim.
* Docker images are built from the provided Dockerfile and hosted on Docker Hub.

# Justification for the Resources Utilized

* Deployments/StatefulSets:
  + Used for managing stateless (API) and stateful (DB) workloads, ensuring scalability and resilience.
* ConfigMaps & Secrets:
  + Enable secure, environment-specific configuration without code changes.
* ClusterIP Services:
  + Used for internal communication, following best practices to avoid Pod IPs.
* Ingress:
  + Provides a single-entry point for external traffic, supporting host-based routing and TLS termination.
* PersistentVolumeClaim:
  + Ensures database data is retained across pod restarts or rescheduling.
* NGINX Ingress Controller:
  + Industry-standard, widely supported, and easy to automate in any cloud environment.