Task1: Backend - Identity Reconciliation

Task Overview:

Dr. Chandrashekar, affectionately known as Doc, is currently stranded in the year 2023, engrossed in fixing his time machine to embark on a journey back to the past and save a dear friend. As an avid patron of the popular online store Zamazon.com, Doc exercises great caution by using different email addresses and phone numbers for each purchase to avoid unwanted attention to his ambitious project.

Moonrider, known for its expertise in enhancing customer experiences, has decided to integrate its cutting-edge technology into Zamazon.com. This integration is designed to collect and manage contact details from shoppers, providing a personalised customer experience.

However, given Doc's unique approach to anonymity, Moonrider faces a distinctive challenge: linking different orders made with different contact information to the same individual.

Your Mission:

1. Develop a web service capable of processing JSON payloads with "email" and "phoneNumber" fields, creating a shadowy infrastructure that consolidates contact information across multiple purchases.
2. Craft a response mechanism that returns an HTTP 200 status code, along with a JSON payload containing consolidated contact details. The payload should be cunningly structured with a "primaryContactId," "emails," "phoneNumbers," and "secondaryContactIds."
3. Be prepared for a scenario where no existing contacts match the incoming request. In such cases, your service should craft a new entry in the database, treating it as a discreet individual with no affiliations.
4. Implement a strategy where incoming requests matching existing contacts trigger the creation of "secondary" contact entries, providing a covert mechanism for storing new information.
5. Exercise caution as primary contacts can seamlessly transform into secondary contacts if subsequent requests reveal overlapping contact information. This dual-purpose functionality adds an extra layer of complexity to your mission.

Schema reference:

```
Unset
{

      id          Int

 phoneNumber      String?

 email         String?

 linkedId       Int? // the ID of another Contact linked to this one

 linkPrecedence    "secondary"|"primary" // "primary" if it's the
first Contact in the link

 createdAt       DateTime

 updatedAt       DateTime

 deletedAt       DateTime?

}
```

Requirements:

1. Meticulously implement the **`/identify`** endpoint, ensuring that it operates with utmost discretion and precision.
2. Execute the creation of a new "Contact" entry with **`linkPrecedence="primary"`** when no existing contacts match the incoming request.
3. Employ a covert strategy for creating "secondary" contact entries when incoming requests match existing contacts and introduce new information.
4. Maintain the integrity of the database state, executing updates seamlessly with each incoming request.

Bonus Points:

- Craft an error handling system that misdirects potential threats and provides misleading error responses.
- Employ covert optimization techniques for database queries and operations to operate under the radar.
- Execute a covert unit testing strategy to validate the functionality of your service without revealing its true nature.
- Anticipate and handle edge cases with the finesse of a seasoned operative.

MOONRIDER

Submission guidelines:

1. Make the github repo public and add a readme file, with proper steps for execution.
2. FIll the form and paste the github links.
3. Record a video of the code running and explain the code.
4. Write proper comments in the code.

Task2: DevOps - Containerization, Version Control, and Scalable Deployment

Overview

In this assignment, you will containerize a microservice, manage its deployment with version control, and implement scalable deployment strategies. This exercise mimics tasks common in tech companies like Google, Uber, and Amazon, focusing on end-to-end microservice lifecycle management.

Objectives

1. **Containerization:** Package a microservice into a lightweight, secure, and production-ready Docker container.
2. **Version Control:** Implement version management with Git branches and tags.
3. **Scalable Deployment:** Use Kubernetes/Minikube for scalable deployment with resource constraints.
4. **Automation:** Automate CI/CD pipelines for seamless deployment.

Scenario

You are part of a DevOps team at an e-commerce company tasked with building a microservice for product catalogue management. The team needs to ensure scalability, security, and observability while deploying multiple versions of the microservice to a Kubernetes cluster.

MOONRIDER

Assignment Tasks

1. Microservice Preparation

- **Repository Setup:**
  - Clone the sample repository: [link]
  - The microservice is a REST API for managing a product catalogue.

---

2. Containerization

- Write a `Dockerfile` to containerize the microservice.
- **Requirements:**
  - Use a base image such as `alpine` or language-specific slim images to minimize size.
  - Employ multi-stage builds for production readiness.
  - Ensure environment variables and secrets are managed securely (e.g., via `.env` files or Docker secrets).
  - Include health checks for container monitoring.

---

3. Version Management

- Create three versions of the microservice:
  1. **v1.0:** A base version with `/health` and `/products` endpoint.
  2. **v1.1:** Add a `/products/search` endpoint for searching by keyword.
  3. **v2.0:** Enhance the search endpoint with query parameters and error handling.
- Tag versions in Git using semantic versioning (e.g., `v1.0.0`, `v1.1.0`).
- Automate document changes in a `CHANGELOG.md` file.

---

4. Deployment

- Deploy the microservice to Kubernetes with the following configurations:
  - **Namespace isolation:** Deploy each version in a separate namespace.
  - **Scalability:** Use Horizontal Pod Autoscaler (HPA) to handle traffic.
  - **Resource Management:** Define CPU and memory limits in `Deployment` YAML files.
  - **Ingress Controller:** Use NGINX or Traefik for routing requests to different versions based on URL path (`/v1`, `/v1.1`, `/v2`).

  Note: Minikube or any other kubernetes environment is acceptable.

---

5. CI/CD Pipeline

- Implement a CI/CD pipeline using GitHub Actions or Jenkins.
- Automate the following:
    - Build and push Docker images to Docker Hub or a private registry.
    - Deploy containers to Kubernetes.
    - Run integration tests post-deployment.

---

6. Documentation

- Include a `README.md` with:
    - Deployment instructions for local and Kubernetes environments.
    - CI/CD pipeline setup and execution steps.
    - Logging and monitoring setup guide.
- Add a `SYSTEM_DESIGN.md` to explain architectural decisions.

Bonus Tasks

- Perform a vulnerability scan of the Docker image.
- Configure RBAC policies in Kubernetes to restrict permissions.
- Use TLS for secure communication with the microservice.
- Use Terraform to provision the Kubernetes cluster.

Submission Requirements

1. **GitHub Repository** containing:
    - Microservice source code and configurations. ()
    - Dockerfile, Kubernetes manifests, and CI/CD pipeline files.
    - Documentation (`README.md`, `CHANGELOG.md`, `SYSTEM_DESIGN.md`).
2. **Screenshots or Video:**
    - Running containers for all versions.
    - CI/CD pipeline execution.