Introduction

This is a practical test for a DevSecOps Engineer candidate at Bank J. Safra Sarasin. The goal is to solve the following *hypothetical scenario*:

Our company has an app used across multiple platforms, and our Business Team wants to identify the most frequently used device types.

You are responsible for implementing this functionality.

Instructions

Test Instructions:

1. **Timeframe:** Complete within 7 days (1 week) from receipt.
2. **Individual Effort:** Work independently; no collaborations allowed.
3. **Version Control & Sharing:**
    - Utilize a public Git repository of your choice.
    - Share the repository link with us upon creation.
4. **Technical Choices:**
    - Select one programming language/framework: Node.js, .NET, Python, or Java Spring Boot.
    - Choose one database from the approved list (any version):
        - Oracle Express Edition (XE)
        - Microsoft SQL Server Express
        - MongoDB
        - MariaDB
        - PostgreSQL
        - Elasticsearch
        - Redis
5. **Commit Guidelines:**
    - Aim for at least one commit per day.
    - **Note:** Commits made after the 7-day period will be excluded from evaluation.
6. **Evaluation Outcome:**
    - **Successful:** Receive notification and invitation to the next round, including a review discussion.
    - **Unsuccessful:** Notification will be provided, concluding the process.

Web API Applications

Develop two Web APIs to manage device registration and statistics, adhering to DevSecOps practices.

The goal of these APIs is to store and retrieve the amount of deviceTypes (iOS/Android/Watch/TV).

1. StatisticsAPI

   Should contain two exposed endpoints:

   a. Store information about user login event

   POST /Log/auth

   Input:

   - userKey: string
   - deviceType: string

   Output:

   - statusCode: integer
   - message: string

   Behavior:

   - This method should:
     - Receive the input;
     - Do any treatment that you think it's ok for the entire operation;
     - Do a call to the DeviceRegistrationAPI, method /Device/register;
     - Return:
       - If it's ok:
         - statusCode: 200
         - message: success
       - If it's not ok:
         - statusCode: 400
         - message: bad_request

   b. Retrieve Device Registrations by Type

   GET /Log/auth/statistics

   Input:

   - deviceType: string

   Output:

   - deviceType: string
   - count: integer

   Behavior:

- This method should:
    - Receive the input
    - Do any treatment that you think it's ok for the entire operation
    - Do a connection to the chosen Database
    - Retrieve the amount of devices registered given the received Device Type
    - Return:
        - If it's ok:
            - deviceType: {{DEVICE_TYPE_RECEIVED}}
            - count: {{AMOUNT_OF_DEVICETYPES_REGISTERED}}
        - If it's not ok:
            - deviceType: {{DEVICE_TYPE_RECEIVED}}
            - count: -1

2. DeviceRegistrationAPI

   One exposed endpoint:

   a. Register a Device Type for a given User

   POST /Device/register

   Input:

   - userKey: string
   - deviceType: string

   Output:

   - statusCode: integer

   Behavior:

   - This method should:
       - Receive the input
       - Do any treatment that you think it's ok for the entire operation
       - Do a connection to the chosen Database
       - Add the received information to the Database
       - Return:
           - If it's ok:
               - statusCode: 200
           - If it's not ok:
               - statusCode: 400

Deliverables

1. For the APIs described at “Web API Application” Section:
    a. Source Code;
    b. Docker Files containing the steps to generate the proper runnable image for the APIs described at “Web API Application” Section.
        i. The Docker registry should be: DockerHub;
2. Create the Kubernetes/Openshift/Docker-Compose deployment solution for the entire ecosystem (APIs and Database):
    a. Should contain all resources to deploy that you think it’s ok;
    b. Should contain the database configuration to run as a container;
        i. The Docker registry should be: DockerHub;
    c. Should contain the security measures that you think it’s ok;
    d. Should contain the communication across the applications that you think it’s ok;
        i. Additional requirement for communication if you will deploy a Kubernetes/Openshift setup:
            1. API 1 (StatisticsAPI), should be publicly accessible, outside the cluster;
            2. API 2 (DeviceRegistrationAPI) should be internally accessible, just inside the cluster.