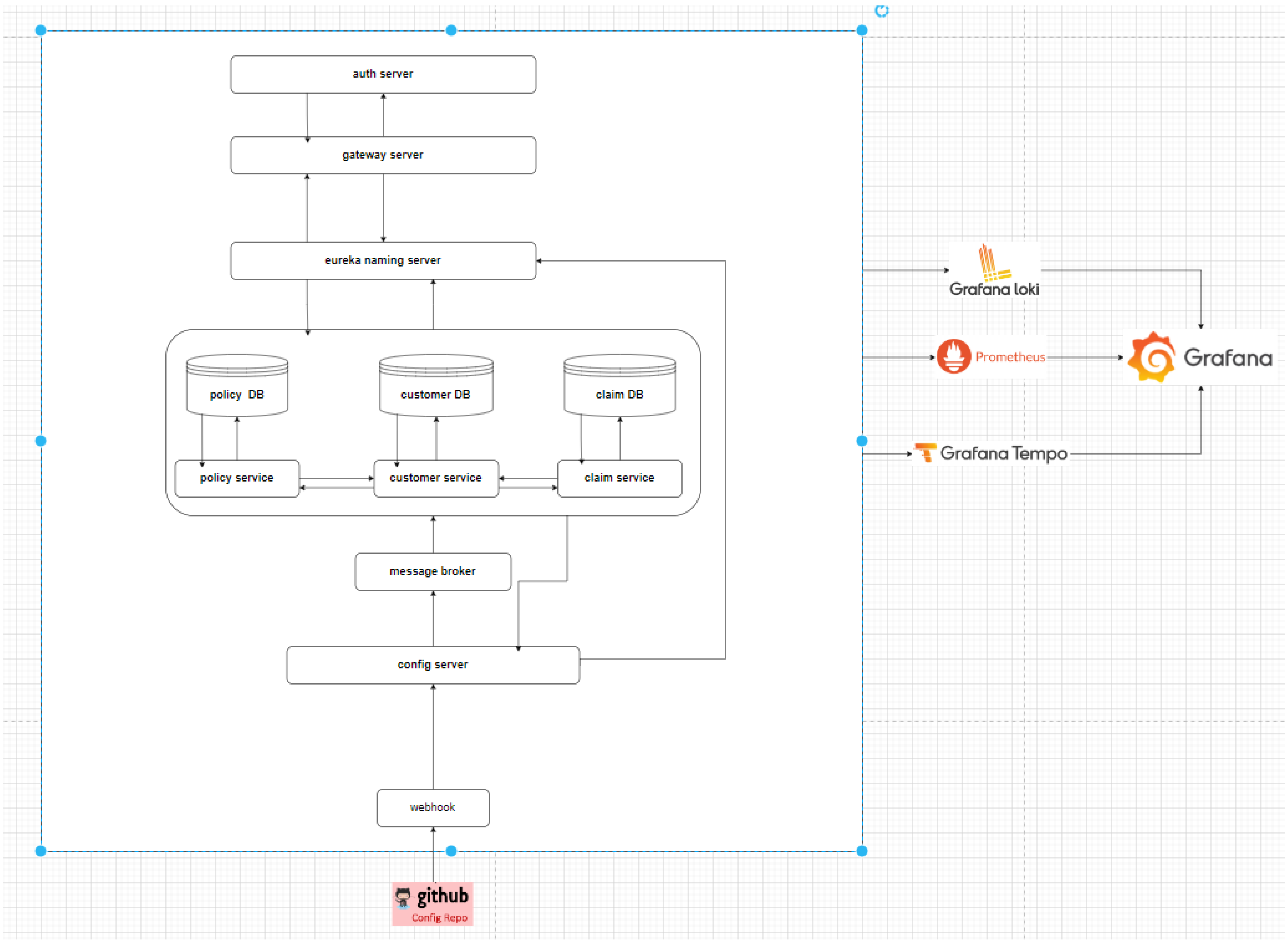
Harel Task

Q1 microservices design for an insurances system

1. Architectural Overview
   The system will be based on **microservices architecture**, where each microservice will be responsible for a specific domain, such as customer management, policy management, and claims processing. Each service will have its own database, promoting loose coupling and independent scalability.



2. The best practice for inter service communication and resilience is to use **Gateway server**, The Gateway Server is built using Spring Cloud Gateway, providing an effective API routing mechanism. It also handles cross-cutting concerns such as security, monitoring, metrics and resiliency, ensuring a robust and efficient communication layer between clients and microservices.
   The service gateway sits as the gatekeeper for inbounded traffic to microservice calls within our application, with a service gateway in place our service clients never directly call the URL of an individual service, but instead place all calls to the service gateway.
   For fault tolerance we are going to use resilience4j library, it's offering the following patterns for increasing fault tolerance due a network problems or failure of any of the multiple services:
    - **Circuit braker** - used to stop making requests when invoked is failing.
    - **Fallback** - alternative paths to failing requests.
    - **Retry** - used to make reties when retries when a service has temporarily failed.
    - **Rete limit** – limits the number of calls that a service receives at a time.
    - **Bulkhead** – limits the number of outgoing concurrent requests to a service to ovoid over loading.

   And to use **Eureka naming service**, Service Discovery is a crucial aspect of distributed systems and microservice architectures. The Eureka Naming Service facilitates dynamic discovery of services within the system. It enables microservices to register and deregister themselves as they scale or fail, allowing seamless communication between services. Here are the key components and concepts associated with service discovery:

    - Service Registry
    - Service Provider
    - Service Consumer
    - Dynamic Updates
    - Load Balancing
    - Decentralized Communication
    - Health Checking

3. Security:
    - **OAuth2.1 / OpenID Connect** will be used for secure authentication and authorization.
    - OAuth2 allows applications to access resources on behalf of a user without sharing credentials.
    - OpenID Connect builds on OAuth2 to provide identity information via **ID tokens**.
    - **Mutual TLS (mTLS)** will secure communication between microservices, ensuring both client and server authenticate each other. We will manage certificates with Harel acting as its own Certificate Authority (CA).

Q3 dependency management in maven

1. The pom.xml file is a crucial component in a Maven project. It is an XML file that defines various aspects of a project.
    - **project Information** - like the version of the project, spring version, packaging, and more.
    - **dependencies** - file is where you declare the libraries and frameworks your project depends on.
    - **build Configuration** – for example you can define to use google jib to create docker images.
    - **properties** - The pom.xml file allows you to define properties that can be reused across the file. This allows for version management, project-specific settings, and reusable configurations like java version, spring cloud version and otle version.
2. In Maven, dependencies are categorized based on when they are needed during the project lifecycle. These categories determine how and when a dependency is made available to the project.
    - **compile** - The compile scope is the **default scope** for dependencies. This means that the dependency is available **during the compile time, runtime, and test time**.
    - Provided - The provided scope means that the dependency is required for **compiling** and **testing.**
    - runtime - The runtime scope means that the dependency is **not needed for compiling** the project, but it is required **at runtime**. It is available during the execution of your application.

Q4 caching in spring boot

We are going to implement a system that uses Redis for temporary storage and MySQL for long-term storage.

Redis as a Cache Layer:

- **Role:** Redis will act as a fast cache to temporarily store data.
- **TTL (Time To Live):** Each piece of data stored in Redis will have a TTL value (e.g., two days). After the TTL expires, the data will be automatically removed from Redis.
- **Why TTL:** The TTL ensures that the cache is fresh and up-to-date, so Redis won't store stale data. Once the data expires, it can be retrieved from MySQL or reprocessed through external services.

MySQL for Long-Term Storage:

- **Role:** MySQL will act as the primary, persistent data storage where the data will be saved long-term. Once data expires from Redis it will be transferred to MySQL.

- **Data Transfer:** When data is transferred from Redis to MySQL, the TTL should be reset or extended, and the data will stay in MySQL for as long as it is needed for historical analysis or compliance purposes.

Process:

1. Whenever a request is made to the system for a specific data, the system first checks Redis to see if the data is already cached. If data exists in Redis and is not expired, it is used directly from Redis, else the system proceeds to the next step.
2. If data is not found in Redis or is expired, the system will then connect to external services to retrieve the latest data.
3. Once the data is obtained from external services, it will be stored in Redis for a defined TTL.
4. After the TTL expires or when the system reaches a predefined point, the data is then moved from Redis to MySQL for long-term storage.

Q5 SQL query

```
SELECT c.id, c.name, c.email

FROM customers c

JOIN policies p ON c.id = p.customer_id

WHERE p.status = 'ACTIVE'

GROUP BY c.id, c.name, c.email

HAVING COUNT(p.id) >= 2;
```

Q6 query optimize

Indexing the status column will speed up lookups by reducing the number of rows that need to be scanned.

Q7 aggregation query

```
SELECT p.status, CAST(SUM(p.premium) AS DECIMAL(10, 2)) AS total_premium

FROM policies p

GROUP BY p.status;
```

Q9 Kubernetes deployment YAML

configmaps.yaml

```
apiVersion: v1

kind: ConfigMap

metadata:

  name: harel-configmap

data:

  SPRING_PROFILES_ACTIVE: prod

  SPRING_CONFIG_IMPORT: "configserver:http://configserver:8071/"

  EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: "http://eurekaserver:8070/eureka/"

  CONFIGSERVER_APPLICATION_NAME: configserver

  EUREKA_APPLICATION_NAME: eurekaserver

  POLICY_APPLICATION_NAME: policy-service

  CUSTOMER_APPLICATION_NAME: customer-service

  CLAIM_APPLICATION_NAME: claim-service

  GATEWAY_APPLICATION_NAME: gatewayserver

  KEYCLOAK_ADMIN: admin

  KEYCLOAK_ADMIN_PASSWORD: admin

  SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_JWK-SET-URI:
http://keycloak:7080/realms/master/protocol/openid-connect/certs
```

Policy.yaml

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: policy-deployment

  labels:

    app: policy

spec:
```

```
replicas: 1
selector:
  matchLabels:
    app: policy
template:
  metadata:
    labels:
      app: policy
  spec:
    containers:
      - name: policy
        image: yoramnagdocker/policy:0.0.1-SNAPSHOT
        ports:
          - containerPort: 8080
        env:
          - name: SPRING_APPLICATION_NAME
            valueFrom:
              configMapKeyRef:
                name: harel-configmap
                key: POLICY_APPLICATION_NAME
          - name: SPRING_PROFILES_ACTIVE
            valueFrom:
              configMapKeyRef:
                name: harel-configmap
                key: SPRING_PROFILES_ACTIVE
          - name: SPRING_CONFIG_IMPORT
            valueFrom:
              configMapKeyRef:
                name: harel-configmap
              key: SPRING_CONFIG_IMPORT
        - name: EUREKA_CLIENT_SERVICEURL_DEFAULTZONE
          valueFrom:
            configMapKeyRef:
              name: harel-configmap
              key: EUREKA_CLIENT_SERVICEURL_DEFAULTZONE
---
apiVersion: v1
kind: Service
metadata:
  name: policy
spec:
  selector:
    app: policy
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 8000
      targetPort: 8080
```