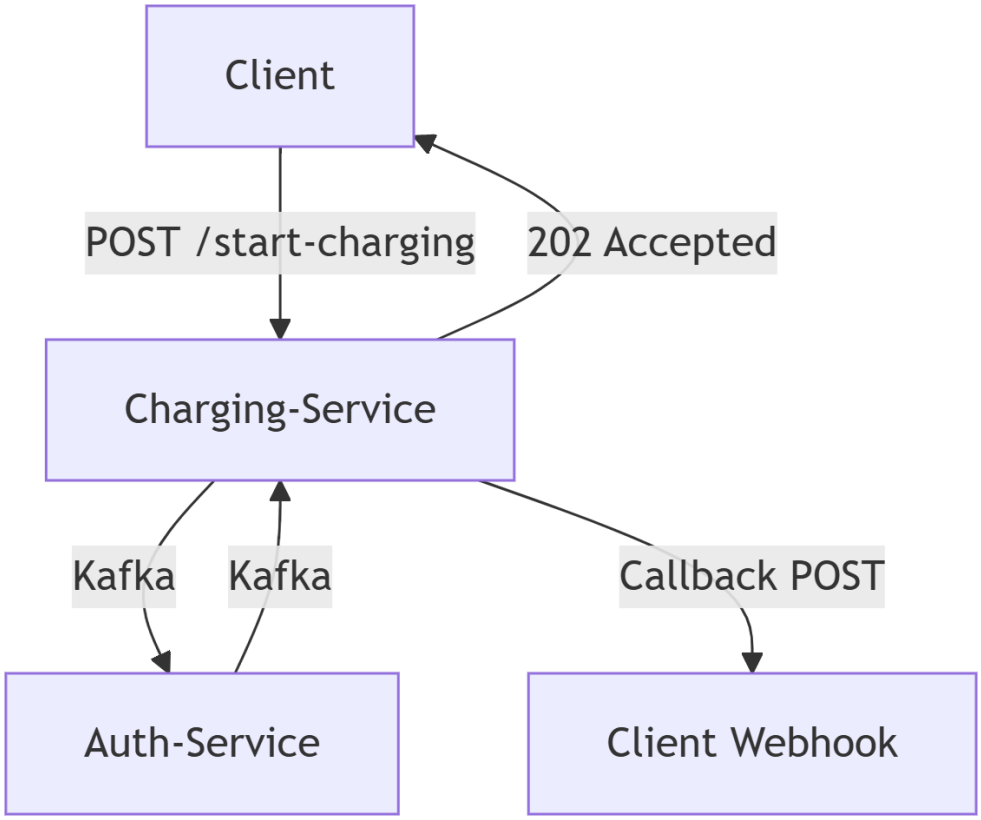
1. Overview

A simple Kotlin, Springboot microservice for managing electric vehicle charging sessions with:

- Asynchronous request processing
- Kafka-based event-driven architecture
- REST API with callback notifications

2. System Architecture



Components

1. **Charge-Service (8082) -** Client request the first request(producer) and consumes the later validated/ authorized response from the auth-service and send the response to callback service.
2. **Auth-Service (8084) –** Consume the request and validate the charge-request (Internal ACL service). Later again send the validated response back to the Auth-Service as a producer.
3. **Kafka (9092) –** Act as a message brocker in between asynchronous commiunication between Charge-Service and Auth-Service
4. **H2 database –** Simple in memory database to persist the result and request.\

3. API Specification

Start Charging Session

Endpoint: POST /api/v1/charging/start-charging

Request: Valid

```
{
  "stationId": "123e4567-e89b-12d3-a456-426614174000",
   "driverToken": "validDriverToken-new-123",
   "callbackUrl": "http://localhost:8082/api/v1/callback/get-callback"
}
```

Response (202 Accepted):

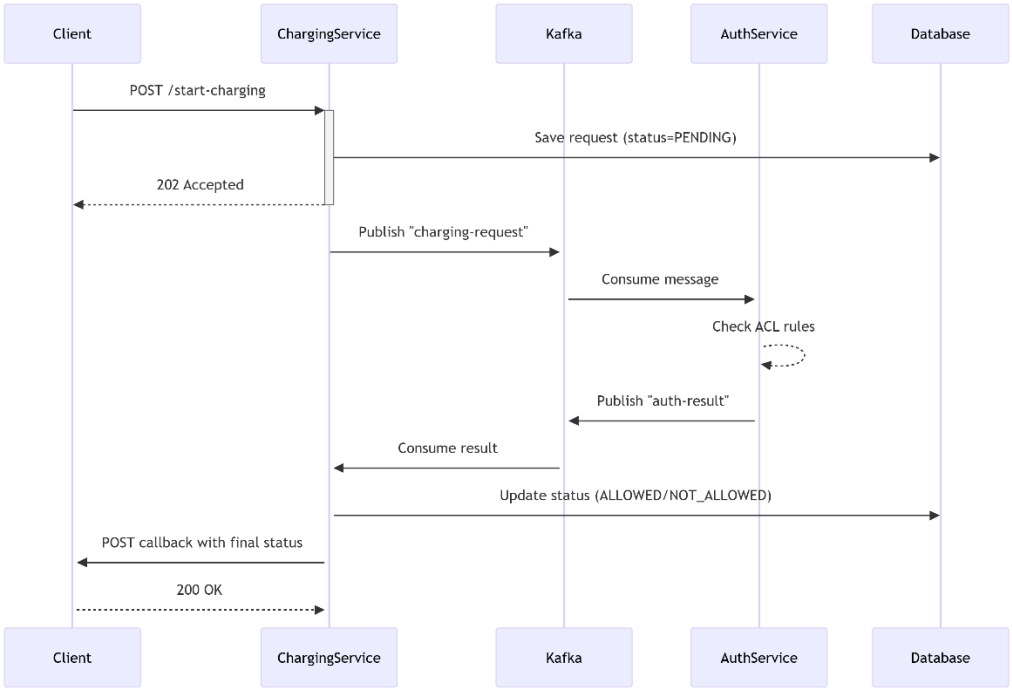
```
{
  "status": "accepted",
  "message": "Request is being processed. The result will send to callback Url"
}
```

You can able to test the api via swagger (/swagger-ui/index.html#/charging-controller/startCharging) or via Postman.

Curl

```
curl -X 'POST' \
  'http://localhost:8082/api/v1/charging/start-charging' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "stationId": "123e4567-e89b-12d3-a456-426614174000",
    "driverToken": "validDriverToken-new-1233",
    "callbackUrl": "http://localhost:8082/api/v1/callback/get-callback"
}'
```

4. WorkFlow Diagram



4. Key Features

1. **Asynchronous Processing**
    - Immediate 202 response
    - Results delivered via sample/simple callback service
2. **Fault Tolerance**
    - Default "UNKNOWN" status on timeouts
3. **Validation**
    - Driver Token: Uppercase letters (A-Z), Lowercase letters (a-z) Digits (0-9), Hyphen (-), period (.), underscore (_), and tilde (~).

      ```
      @field:NotBlank
      @field:Size(min = 20, max = 80)
      @field:Pattern(regexp = "^[a-zA-Z0-9\\-\\._~]+$")
      val driverToken: String
      ```

    - StationId : Should be a UUID

      ```
      @field:NotNull
      val stationId: UUID,
      ```

    - CallBackURL : Should be a valid Http or Https endpoint

      ```
      @field:NotBlank
      @field:Pattern(regexp = "https?://.+")
      val callbackUrl: String
      ```

5. Solution Design

Problem Solved

- Prevents service overload during peak demand
- Decouples authorization checks from API layer
- Provides real-time notifications without polling

6. Deployment Guide

Prerequisites

- Docker 20+
- JDK 17

You can download the repository from https://github.com/charithellawala/sample-async-project and run “docker-compose up –build”.

Steps

1. Build and run:

```
docker-compose up --build
```

2. Verify services:

```
docker-compose ps
```

3. Test API:

```
curl -X 'POST' \
  'http://localhost:8082/api/v1/charging/start-charging' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "stationId": "123e4567-e89b-12d3-a456-426614174000",
  "driverToken": "validDriverToken-new-1233",
    "callbackUrl": "http://localhost:8082/api/v1/callback/get-callback"
}'
```

7. Technical Choices

Kafka: Guaranteed message delivery

H2: Lightweight and simple persistence for the sample solution

Spring Boot: Rapid Development with Kotlin

8. Limitations of the Solution.

Kafka Related Limitations :

- No built-in retries for failed callbacks: Lost callbacks if client endpoint is down/ didn't implement as the scope is unknown.
- Retry Mechanism still to be designed/implemented for failure scenarios. (implementing a DLT)
- Single Kafka Cluster.

Database Limitations :

- Lack of historical data.
- Bottleneck for high-volume stations.
- Data loss on abrupt shutdown

Architectural Limitations :

- Increased latency during peaks
- Cascading failures if Kafka/AuthService is down
- Hardcoded timeoouts

Security Limitations :

- Potential data leakage in Kafka
- Callback URL spooling/ malicious redirects
- No ratelimits or Gateway

Operational Limitations :

- Hard to debug cross-service flows/ No standard tracing.
- Manual scaling

Business Logic Limitations :

- Duplicate charges possible.
- No proper ACL service.
- No pricing model available/ mapped