Charge Service - Request Handling System

Developed by: Arian Ghoochani

Contents

1 Overview

Charge Service is a **fully dockerized** and **scalable** Django-based service designed to handle **electric vehicle (EV) charging authorization** in a **synchronous** manner. The system utilizes **Kafka** for queuing requests and **Django REST Framework (DRF)** to manage API interactions. **NGINX** is used as the web server, and **Docker Compose** ensures a seamless deployment experience, eliminating manual dependency management.

2 Tech Stack

- **Backend**: Django + DRF
- **Queuing**: Apache Kafka
- **Web Server**: NGINX
- **Containerization**: Docker & Docker Compose

3 How to Use

3.1 1. Clone the Repository

```
git clone https://github.com/arianghoochani/charge-service.git
```

3.2 2. Update the Docker Configuration

Navigate to the `docker-compose.yml` file:

```
cd charge-service/dockerized_server
```

Modify the Kafka Consumer Service section (line 59):

```
kafka_consumer:
  build:
    context: ./charge_server/chargedjango
  container_name: kafka_consumer
  environment:
    - KAFKA_BROKER=kafka:9092
    - DJANGO_API_URL=http://YOUR_SERVER_IP/api/checkauthority/
  command: sh -c "python kafka/kafka_consumer.py"
  depends_on:
    - kafka-init
    - chargebackend
  volumes:
    - ./charge_server/chargedjango/db.sqlite3:/backend/db.sqlite3
```

Replace `http://YOUR_SERVER_IP` with your actual server IP.

3.3 3. Build & Run the Service

```
docker compose build
docker compose up -d
# For older Docker versions:
docker-compose build
docker-compose up -d
```

3.4 4. First Use - Insert an ACL Entry

Before making any charging requests, **you must first insert authorized station and driver tokens** into the Access Control List (ACL) using the following API:

```
curl -X POST http://YOUR_SERVER_IP/api/insertACL/ \
     -H "Content-Type: application/json" \
     -d '{
           "station_id": "531111111111111111111111111111111111",
           "driver_token": "driver_token_2025_valid"
         }'
```

Once the ACL is added successfully, you can proceed with sending a charging request using `/api/chargingRequestValidator/`.

4 System Architecture

This project handles **asynchronous charge authorization** using **Kafka** for queuing. The architecture ensures **scalability, reliability, and efficiency** by separating concerns into distinct components.

4.1 Component Breakdown

1. **User/Client**: Sends a charging request via an API endpoint.
2. **Django Backend**: Validates the request and passes it to Kafka Producer for queuing.
3. **Kafka Producer**: Pushes the request into a Kafka Topic (`charging_requests`).
4. **Kafka Consumer (Django)**: Listens to the Kafka queue, processes messages, and checks authorization.
5. **Authorization API**: Determines whether the request is `allowed` or `denied`.
6. **Decision Log DB**: Stores request decisions for future reference and monitoring.

5 API Endpoints

Endpoint	**Method**
`/api/chargingRequestValidator/`	POST
`/api/checkauthority/`	POST
`/api/insertACL/`	POST
`/api/getrequestlog/`	GET

6 Methods Description

6.1 Charging Request Validator

Description: This endpoint receives charging requests and pushes them to Kafka for processing.

6.2 Check Authority

Description: Internal API to check authorization based on ACL.

6.3 Insert ACL

Description: Adds new authorized users to the system.
=

7 Conclusion

This document provides a structured overview of the Charge Service, detailing its architecture, API endpoints, and implementation. The system ensures secure, scalable, and efficient handling of EV charging requests.