IoT Platform for Managing Electric Vehicle Charging Infrastructure

Denis Bahnari
59878

Internet of Things Project

November 30, 2025

System Architecture

The system architecture is based on a cloud plantform where several containers work together to support the full Internet of Things workflow for electric vehicle charging sessions. Inside this environment there is a container running the MQTT broker. This broker is responsible for organizing all publish and subscribe services and managing the distribution of messages across topics. A second container runs the main application that processes all logic of the system. A third container hosts the PostgreSQL database used to store all session information. The fourth container is named *ml_processor* which is responsible for training and performing machine learning predictions. A Node RED container is also part of the architecture that receives data from the system and presents it to the user through a dashboard.

There is an additional folder called *client_1*. This folder contains a simulation script that sends charging session data at fixed time intervals.

Client Simulation Publish

The client simulation component is designed to emulate real datasets coming from an electric vehicle charging station. Its main purpose is to provide the system with continuous, realistic data for testing the database, machine learning processor, and dashboard.

The simulation code reads each row from the CSV file, generates a random start time for the charging session, and calculates the corresponding end time based on the session duration. The simulator then creates a structured message including the user ID and all session information, converts it to a JSON format, and publishes it to the MQTT broker at regular intervals. The connection to the broker is secured using a CA certificate, and authentication is performed using a username and password.

MQTT Broker

The MQTT broker container was obtained from the official Docker image. To improve security, the broker uses username and password authentication, with credentials stored in a hashed format to prevent unauthorized access. The communication port is 8883, allowing TLS encryption to be applied. The TLS certificates were generated as self-signed certificates by the MQTT broker server, ensuring encrypted communication between all clients and containers.

The broker handles two main topics in the system. The first topic is dedicated to incoming online charging sessions, which allows the main application to publish real-time session data as it arrives from clients. The second topic is responsible for updating the Node RED dashboard with the current status of the system, including session summaries, cluster assignments, and other relevant metrics. This separation of topics ensures that each component receives only the information it needs, improving efficiency and reducing unnecessary traffic.

Database

The PostgreSQL database uses the official Docker image. This database was selected because it is simple to use, efficient and already familiar. The system has two main tables, one table stores EV charging sessions and the second table stores information about charging stations. Some extra columns were added to store the results of the machine learning models, such as the cluster of each session.

The database module includes simple functions that insert new data into the tables. The function that inserts session data reads a JSON message, converts the values into the correct types and saves them in the `ev_session` table. A similar function inserts station information into the `ev_station` table. Both functions ignore repeated entries to avoid duplicates.

There is also a function that updates the cluster values of each session after the machine learning model finishes the predictions. This function writes the new cluster values directly into the database.

Other helper functions allow the system to read all stored sessions, get the most recent session and obtain basic statistics about the data. These results are used by the main application and by the dashboard.

In general the database acts as the central storage of the project. It keeps all the session data, the station information and the machine learning results in a simple and organized way.

Machine Learning Processor

The machine learning processor is implemented as a Flask server running inside the *ml_processor* container. It is prepared to receive training requests and prediction requests. The training process starts by converting all raw sessions into a set of features. The processor generates session features and also aggregated features at user level and station level.

User features include total number of sessions, average energy consumed, average duration, average charging rate and number of stations used. The system also calculates the preferred time period of each user. Station features include number of sessions, average session duration, average energy, number of users and estimations of daily and weekly energy.

These features are merged to create a single dataset. Time values are converted into one hot columns. The processor scales all features and trains three models. The models are KMeans for clustering, DBSCAN for density clustering for anomaly detection.

When a prediction request is received the processor creates a feature vector from the raw session and applies the same transformations. The result contains the predicted KMeans cluster and the DBSCAN cluster.

Node RED Dashboard

The Node RED component is deployed using the official Docker image. It receives all dashboard updates from the main application through the MQTT broker, using secure TLS communication. Whenever new data arrives, it is processed by a set of function nodes that clean, filter and restructure the values before they reach the visual elements of the dashboard.

The flow is organised in several processing blocks. Each block extracts a specific type of information such as summary statistics, session trends, time-of-day usage, charging cost, user behaviour and cluster profiles. After being processed, these values are sent to the corresponding dashboard charts and indicators. This structure ensures that each panel in the dashboard receives only the information it needs, keeping the interface clean and responsive.

The dashboard is automatically updated every time new data is inserted into the system. When online charging sessions arrive in real time, the values are recalculated and the interface reflects the latest information almost instantly.

To access the dashboard, the user only needs to open the following address in a browser:

```
http://127.0.0.1:1880/ui
```

This provides direct access to all charts, statistics and visual indicators that describe the behaviour of the charging infrastructure.

Main Application

The main application works as the central coordinator of the whole system. It connects all parts of the project: the database, the machine learning processor and the Node RED dashboard.

When the application starts, it first loads the offline dataset included in the project. All charging sessions and station information are inserted into the database so the system begins with a complete set of historical data.

After this initial loading, the application sends the full dataset to the machine learning module so the models can be trained. Once the training is finished, the application requests predictions for all stored sessions and updates the database with the new cluster labels.

With the database updated, the application prepares the information needed by the dashboard. It collects general statistics such as daily and weekly trends, time of day usage and cluster summaries. These results are sent through MQTT to the Node RED dashboard, which immediately shows the updated information.

Once everything is initialized, the application switches into real-time mode. It listens for new online charging sessions sent by MQTT. Each time a new session arrives, it is saved in the database and sent to the machine learning processor for an instant prediction. The result is then stored and the dashboard is updated again so the user always sees the most recent activity.

In short, the main application keeps the whole system running: it loads the initial data, communicates with the machine learning models, updates the dashboard and continuously processes new charging sessions in real time.

Challenges and Difficulties

Several challenges were encountered during the development of this project. Integrating secure TLS communication for MQTT required generating certificates and correctly configuring all clients, which proved non-trivial. Creating the feature engineering pipeline for the machine learning models was also complex, as it involved merging data from different entities such as sessions, users, and stations.

The most significant difficulties arose within the *ml_processor* container. Training the machine learning models was particularly challenging, since generating features that produced meaningful predictions required careful consideration. Often, clusters showed poor separation, and in some cases, a single cluster contained the vast majority of data while the others were sparsely populated, making it difficult to obtain balanced and informative results.

Implementing the Node RED dashboard correctly was another major challenge. Structuring the flows and ensuring that data was accurately sent and processed took considerable time. Processing issues and the need to construct precise queries to extract relevant information added to the complexity. In particular, implementing the component that calculates the average and total charging duration for user-selected periods was especially difficult beacuse of the Node RED flow interface, where restricting and aggregating data according to dynamically selected time intervals was a challenge. Ensuring that each dashboard component received properly formatted data and responded correctly to user input demanded significant attention to detail.

In addition, managing multiple containers and ensuring reliable communication among them demanded careful configuration. Real-time updates to the dashboard also posed challenges, as the system needed to automatically push new values whenever a new session arrived.

Despite these challenges, the final system operates as intended, providing a complete and functional IoT pipeline for managing electric vehicle charging data.