# Charger sessions and telemetry

## Coding test

I have attached the coding test in DBC format (Readable from Databricks) and in HMTL format (Readable from all the web browsers). I have used Spark because the data is big enough. Code is commented and all the solutions are clearly represented.

## Architecture Questions

**How would you store the data so that it could accommodate different use cases**

**(visualization, ad-hoc analysis, EDA, predictive analytics)**

I will choose a columnar storage format like Parquet due to following reasons:

* The data is mostly going to be analyzed in an OLAP (Online Analytical Processing).
* Widely used format in a lot of tools/ use cases.
* Events have a lot of fields that are only useful to unequivocally identify the event and to know when that event was generated. Most of these fields are not going to be queried in normal queries, with Parquet only the searched columns will be used in the query.
* Powerful data compression and improved read performance.
* The schema evolution is limited but there are evolutions of the format like (Databricks Delta) that could be used.

**How would you design a data pipeline that ingests and processes all these events in**

**near-real-time to generate charging profile dashboards for each charging session?**

**Imagine we could show these metrics in a small dashboard in our myWallbox**

**portal/app.**

I will implement a Kappa architecture in which all the data (events in this case) will be published into a distributed storage queue like Kafka. The data will then be enriched (e.g., by performing aggregations over a certain time frame) and published back again to the messaging system. The enriched data will then be stored in a Data Warehouse or Data Lake and will be used by all the different systems that may need it (In this case, BI tools to display dashboards).

I have decided to enrich the data before making it available for all data consumers because the number of generated events could be very large and wrong events or outliers could be generated because many different devices will send data to the messaging system at the same time. By adding the enrichment step, all these outliers/ erroneous events will be removed before consuming them and the total number of events to handle will be decreased by using aggregations.

**How would you manage to avoid schema changes breaking down data pipelines?**

According to previous point, the architecture will be an event-driven architecture. Taking that into account, I will define a schema structure that could vary according to a field in that schema (Example: *metadataVersion*). That field will point unambiguously to a certain version of the schema structure (the different schema structures will be defined in an internal wiki), so in order to identify the schema to be used by the event, that field will have to be consulted first.

The different versions of the schema have to be known by all the possible users of them (Like I said, these versions will be defined in a common wiki), and if a new version is created, because none of the already defined could be used in a certain use case, all the possible users will be notified.

The concept is similar to what Microsoft describes for the [Event Grid Event Schema](https://docs.microsoft.com/en-us/azure/event-grid/event-schema).

In these events, it could be attached information of the schema change of a certain table in a certain pipeline and what to do to persist these changes.

**How would you do to generate some anomaly detection models in near-real-time**

**based on the received telemetry data?**

Depending on the data volume (I will assume that it will be in the order of terabytes per day), I will implement a Streaming job with Spark Streaming. That Streaming job will always be running and reading new events to a Kafka queue, with all those new events, it will apply some aggregations in a certain timeframe to see if previously defined anomalies (defined by rules) are identified (Example: If the temperature of a certain charger has increased more than a 10% comparing it with the data of the previous ten minutes).

Other options that could also be used: Amazon Kinesis, Azure Stream Analytics