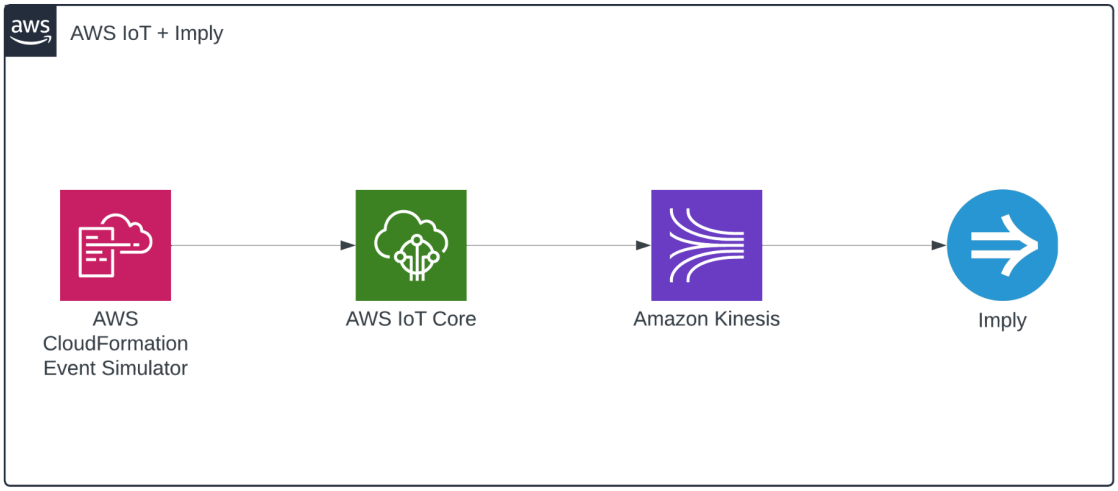
IoT Workshop for EV RT Analytics

Prerequisites

1. Make sure the conductors have your email addresses
2. Make sure that you are added to a Polaris project for the workshop. See inbox for the invite.
3. Make sure you have the AWS Kinesis IAM ARN from the workshop organizers.
4. Familiarize yourself with the architecture for the workshop and also read through the steps before attempting them.

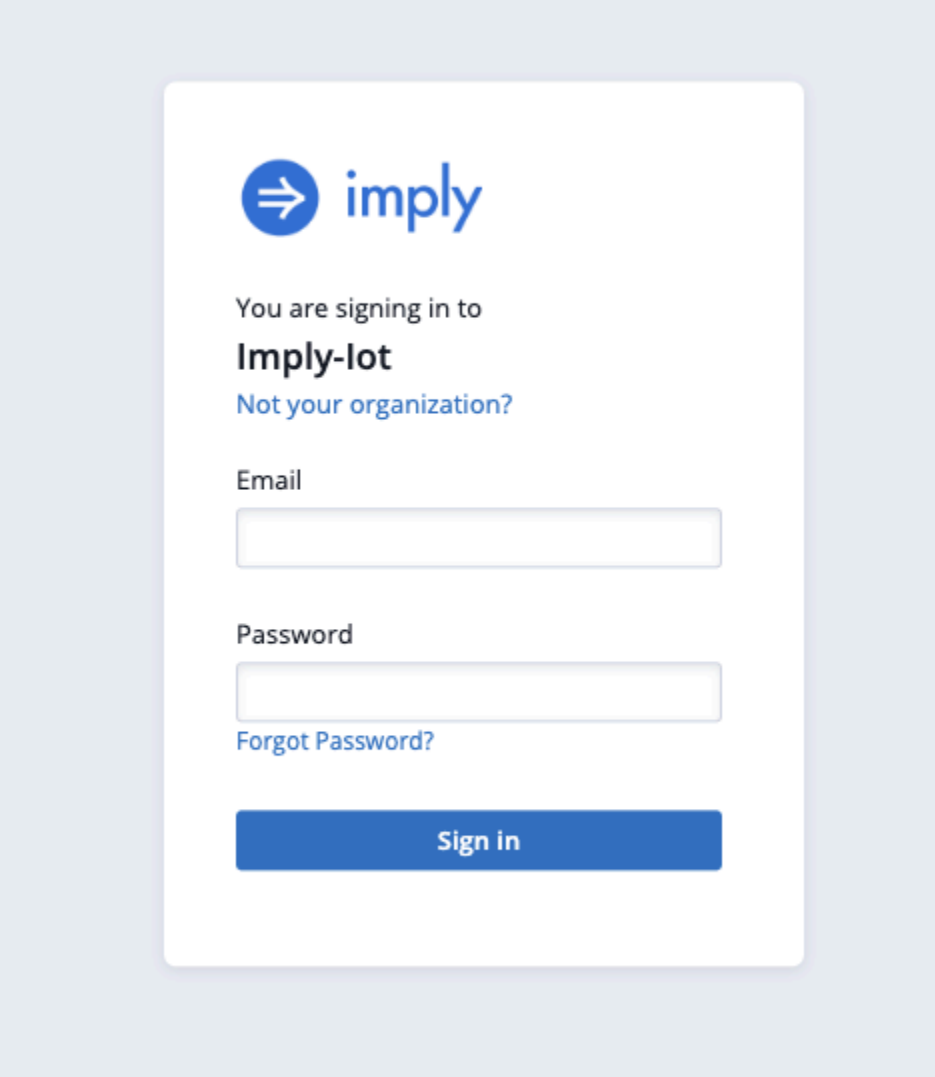
Architecture Walkthrough

The below picture provides an overview of the architecture that has been used to build this workshop. More information about the architecture can be found here.



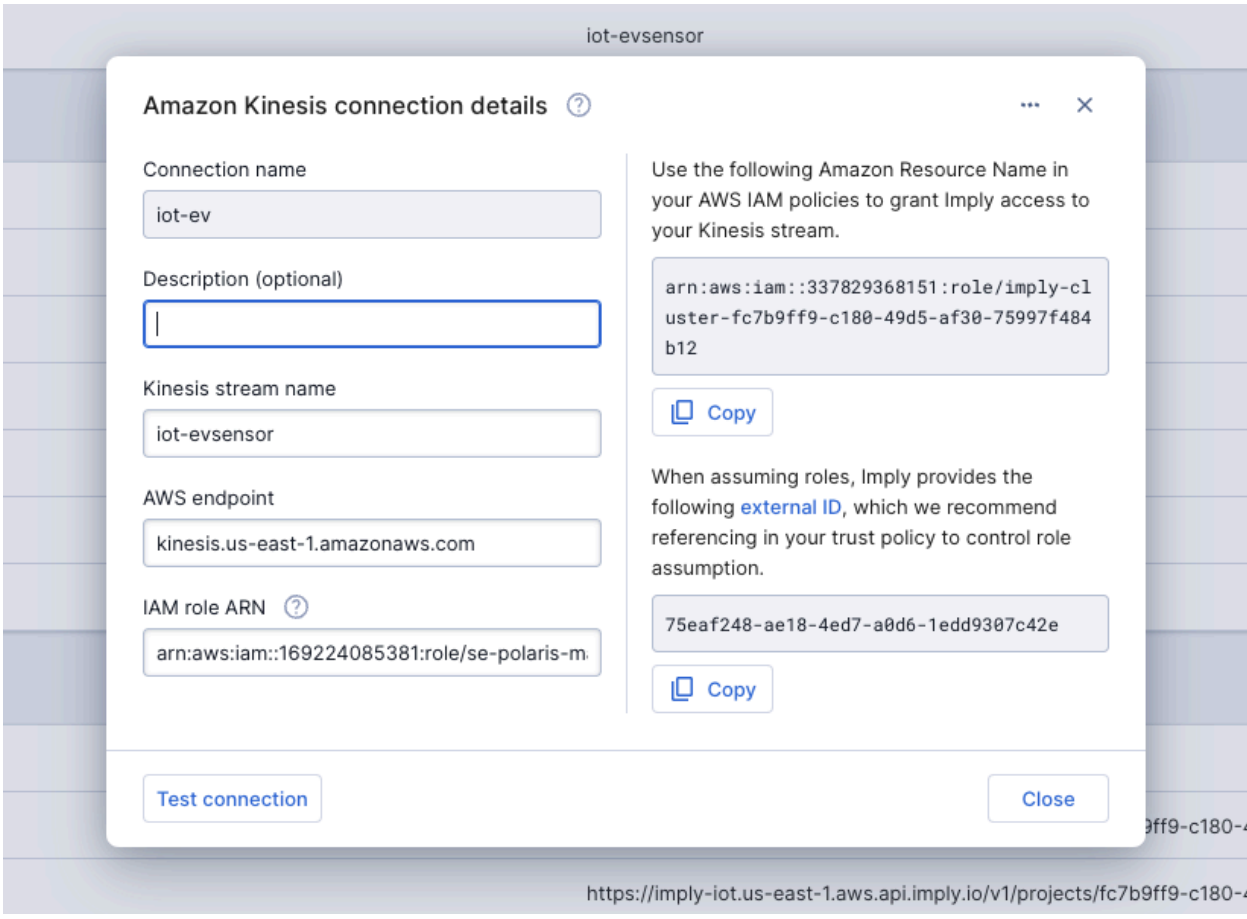
Connect Imply Polaris to Kinesis, and Create a Table

1. Sign into your Imply account



2. Click on “Sources” on the left nav, then at the top right click “Create source”, then “Amazon Kinesis”
   Sources → Create Source → Amazon Kinesis

   - For a comprehensive guide, here are the docs.
     https://docs.imply.io/polaris/ingestion-guide-kinesis



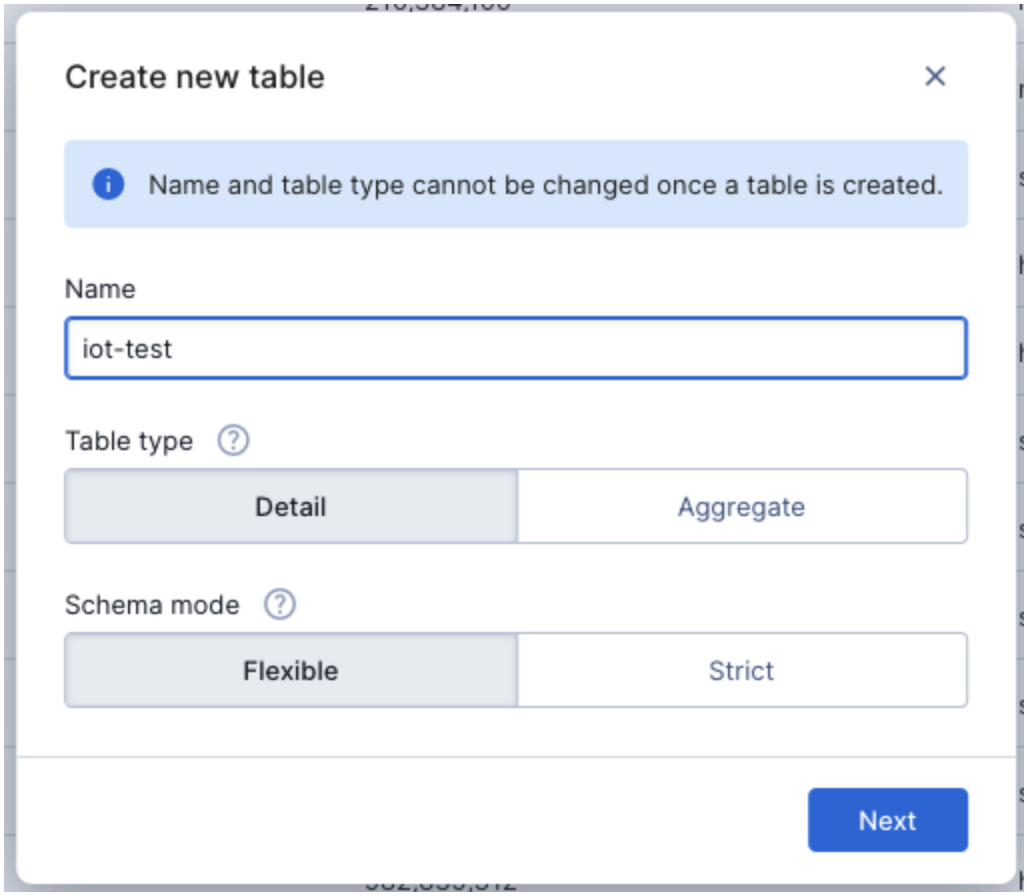
Since the connection name has to be unique, please suffix your initials at the end of the “connection name” in the “new connection” pop-up window.

Use the details that were shared with you by the organizers for connection to AWS Kinesis. For example:

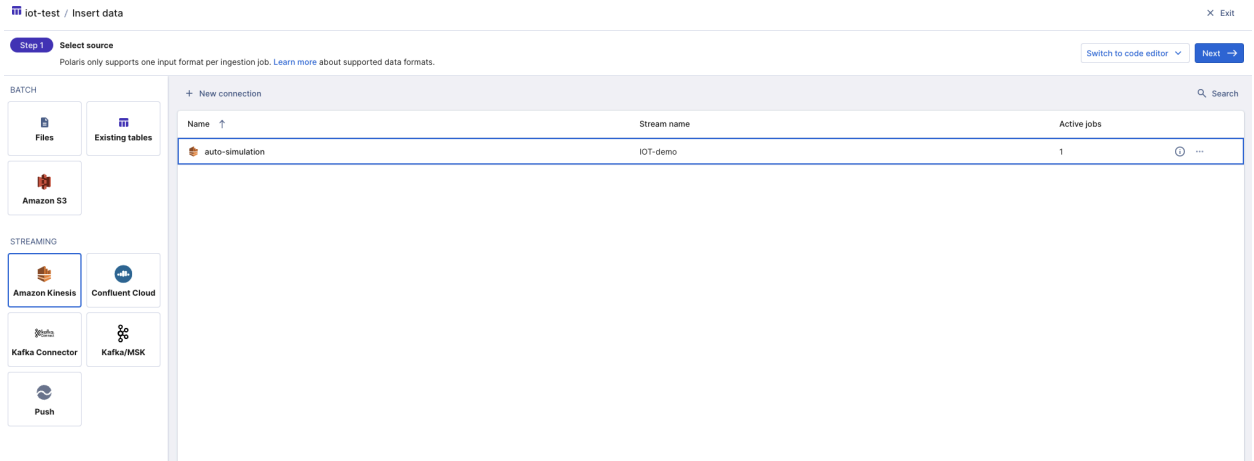
```
Unset
      Kinesis stream name: iot-evsensor
      AWS endpoint:  kinesis.us-east-1.amazonaws.com
      IAM Role ARN: arn:aws:iam::169224085381:role/se-polaris-marc
```

Click “Test connection” and if successful, click “Create connection”.

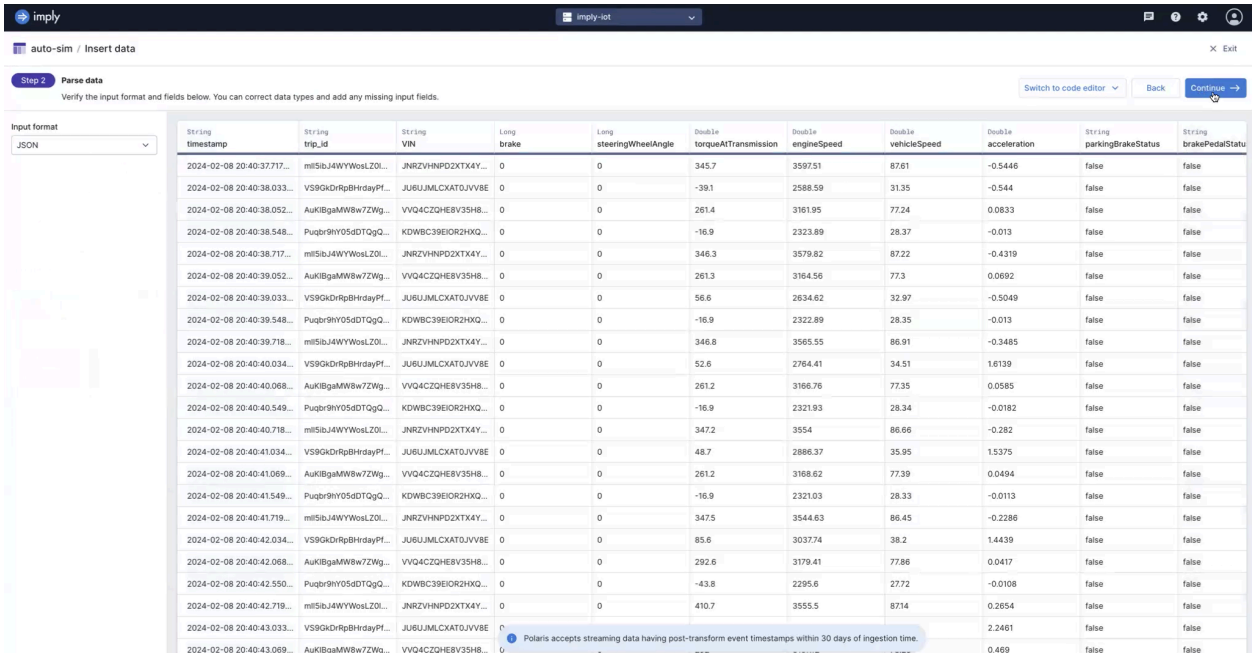
3. Create a table in Imply Polaris by clicking on the “Tables” tab and click “Create table” at the top right. Choose Detail and Flexible. (see tooltip and docs for more info). Since the table name has to be unique, please suffix your initials at the end of the “connection name” in the “new connection” pop-up window.



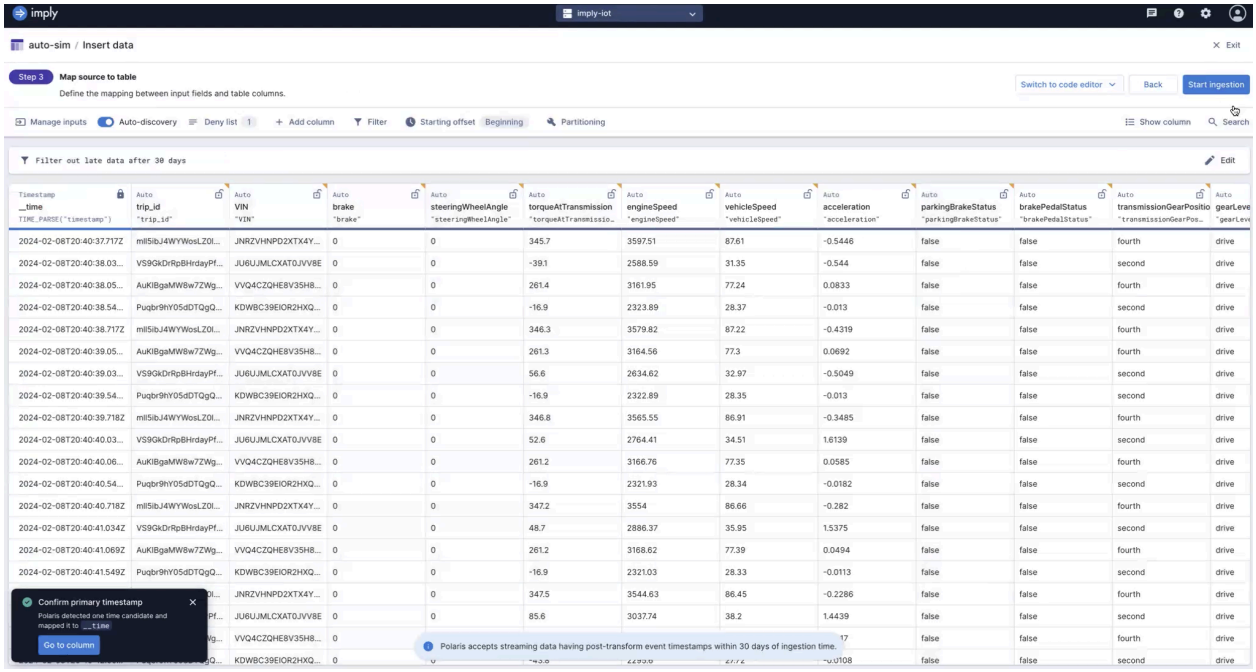
4. Click the “Load data” button on the top right corner and then click “Insert data”. Select “Amazon Kinesis” on the left pane, and then select the source that you created in the previous step. Then click “Next”.



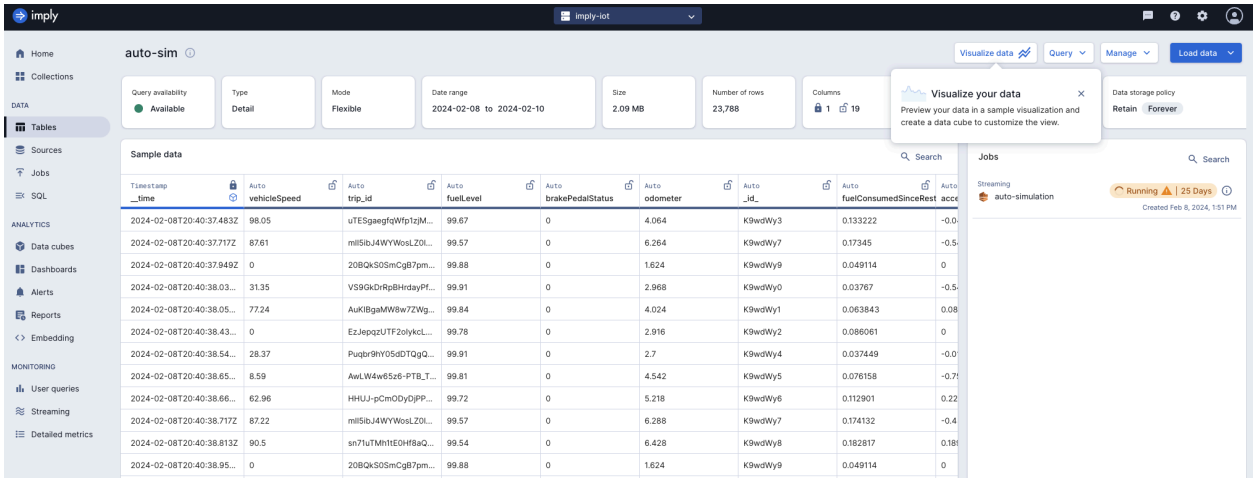
5. Polaris will auto identify the schema, click “Continue”. If the schema is not identified, please explicitly choose JSON as the “input format”.



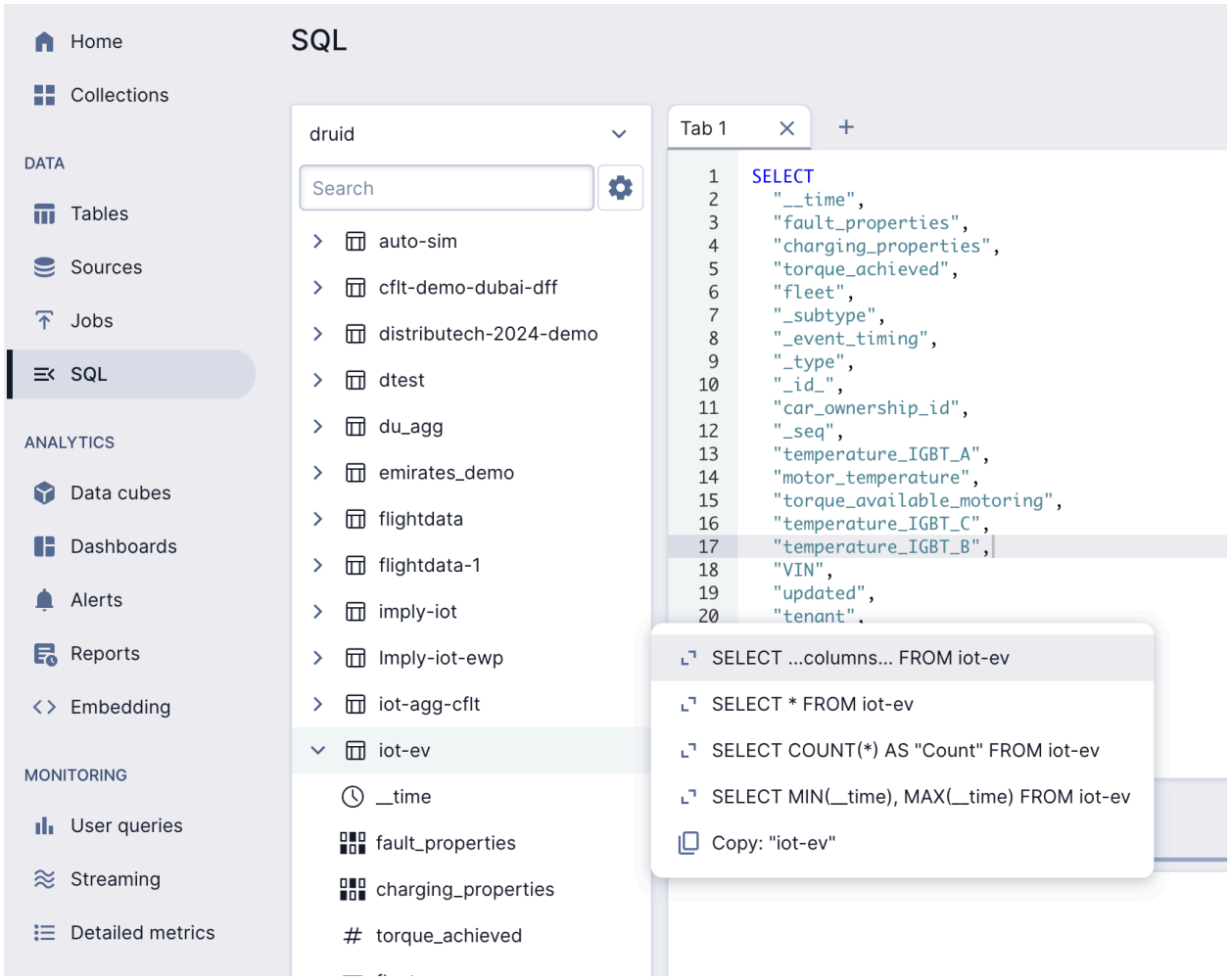
6. You can customize the table further, but for now in this test click “Start Ingestion”



7. Your table is now created, **congratulations!!!** You have created an end-to-end data application pipeline. You can now query your IoT event data from the UI or via API.

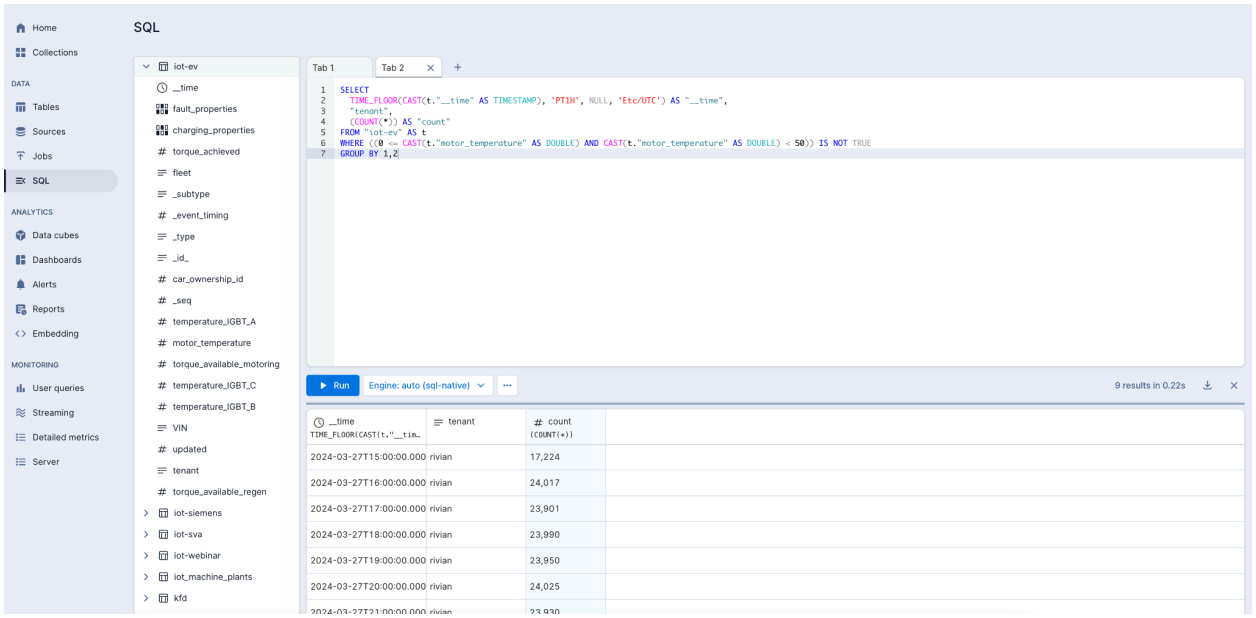


8. Lets query the table by following the below steps:
    - From the left pane, click on “SQL”. Then click the newly created table name and choose the first query option from the pop up.
        i. SELECT … columns … FROM iot-ev
    - Click on “Run”.



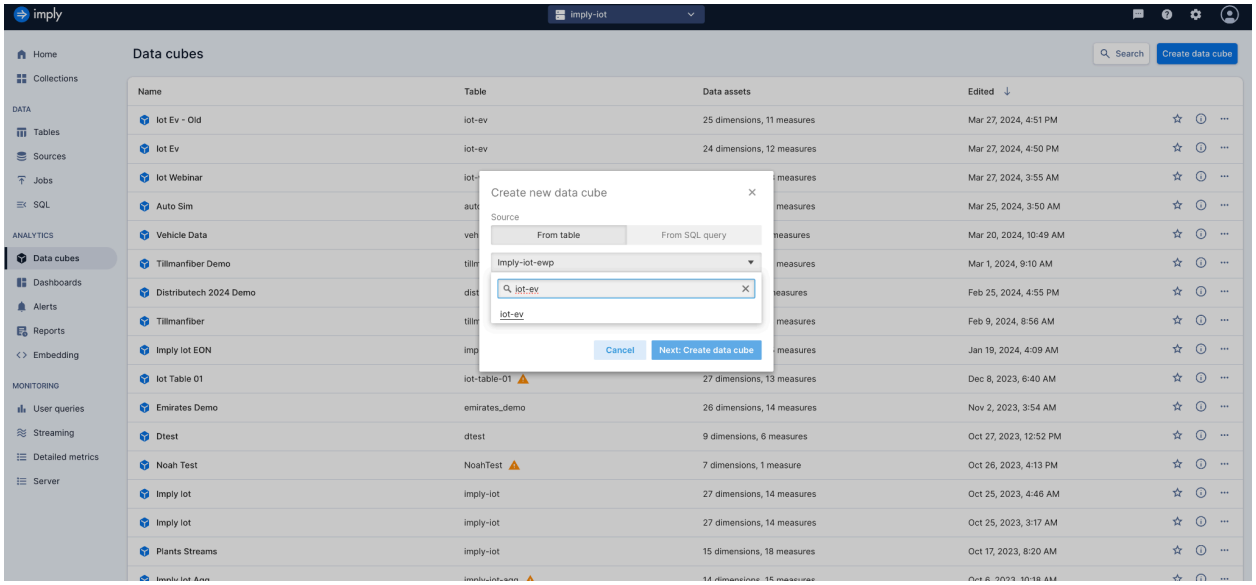
- Click on ‘+’ above the query editor pane to get a new tab.
- Below are some example queries you can test in the SQL UI:

```
Unset
### In English, this query answers "For each hour, for each tenant, how many events showed the motor temperature outside of normal operating range?"

SELECT
  TIME_FLOOR(CAST(t."__time" AS TIMESTAMP), 'PT1H', NULL, 'Etc/UTC') AS "__time",
  "tenant",
  (COUNT(*)) AS "count"
FROM "iot-ev" AS t
WHERE ((0 <= CAST(t."motor_temperature" AS DOUBLE) AND CAST(t."motor_temperature" AS DOUBLE) < 50)) IS NOT TRUE
GROUP BY 1,2


## How many events came in from different fleet categories each hour?

SELECT
  TIME_FLOOR(CAST(t."__time" AS TIMESTAMP), 'PT1H', NULL, 'Etc/UTC') AS "__time",
  CAST(t."fleet" AS VARCHAR) AS "fleet",
  (COUNT(*)) AS "count"
FROM "iot-ev" AS t
WHERE CAST(t."fleet" AS VARCHAR) IN ('Fleet Van', 'Passenger Car', 'Fleet Truck', 'Passenger Truck', 'Fleet Car')
GROUP BY 1, 2
```



Real time Analytics & Dashboarding

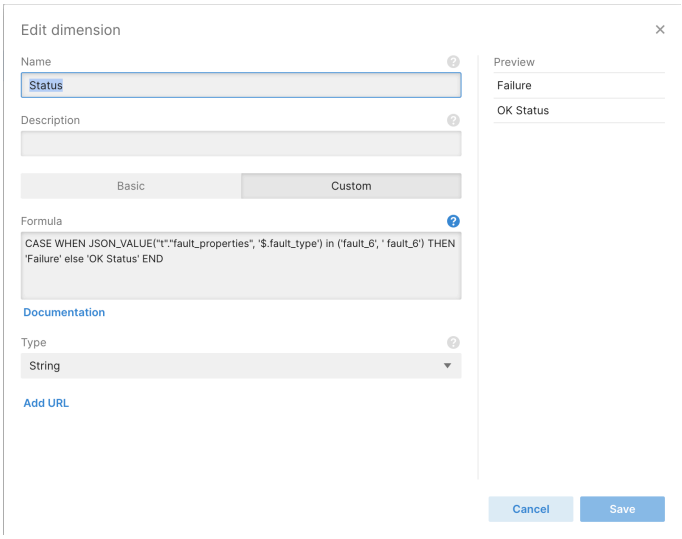
1. Creating a data cube can be done in just a couple short steps:
    a. On the left menu, click “Data Cubes”, then at the top right “Create data cube”



b. From the drop down select your new table we created “iot-test”, and click “Next: Create data cube”. Leave the “Auto-fill dimensions and measures” checked

c. Now let’s modify the **status** dimension and replace the default expression with this new one, it will help to simulate a failure status only when fault_type = ‘fault_6’:

```
CASE WHEN JSON_VALUE("t"."fault_properties", '$.fault_type') in ('fault_6', ' fault_6') THEN 'Failure' else 'OK Status' END
```



d. Now let’s modify the **fault_type** dimension and replace the default expression with this new one in order to decode the fault type descriptions :

```
CASE WHEN JSON_VALUE("t"."fault_properties", '$.fault_type') = 'fault_1' THEN 'charging level 1' WHEN JSON_VALUE("t"."fault_properties", '$.fault_type') = 'fault_2' THEN 'charging level 2' WHEN JSON_VALUE("t"."fault_properties", '$.fault_type') = 'fault_3' THEN 'fast charging' WHEN JSON_VALUE("t"."fault_properties", '$.fault_type') = 'fault_4' THEN 'low battery' WHEN JSON_VALUE("t"."fault_properties", '$.fault_type') = 'fault_5' THEN 'charging issue' WHEN JSON_VALUE("t"."fault_properties", '$.fault_type') = 'fault_6' THEN 'engine overheating' END
```

Edit dimension

Name

Fault Type

Description

Basic | Custom

Formula

```
CASE WHEN JSON_VALUE("t"."fault_properties", '$.fault_type') = 'fault_1' THEN 'charging level 1' WHEN JSON_VALUE("t"."fault_properties", '$.fault_type') = 'fault_2' THEN 'charging level 2' WHEN JSON_VALUE("t"."fault_properties", '$.fault_type') = 'fault_3' THEN 'fast charging' WHEN JSON_VALUE("t"."fault_properties", '$.fault_type') = 'fault_4' THEN 'low
```

Documentation

Type

String

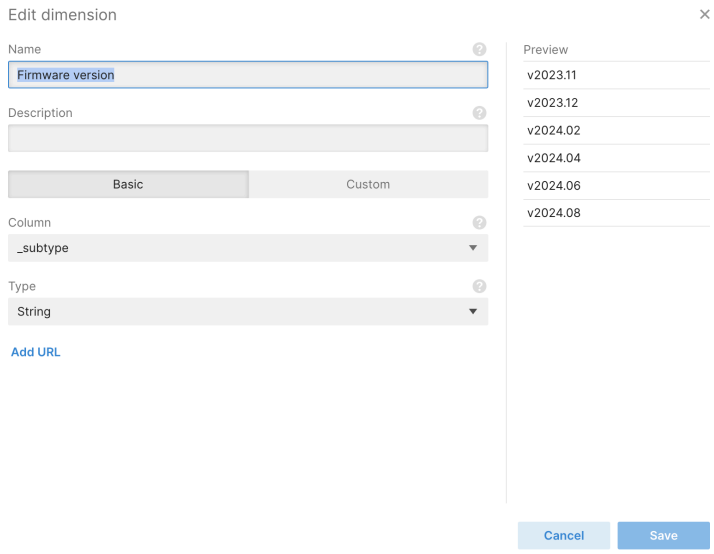
Add URL

Preview

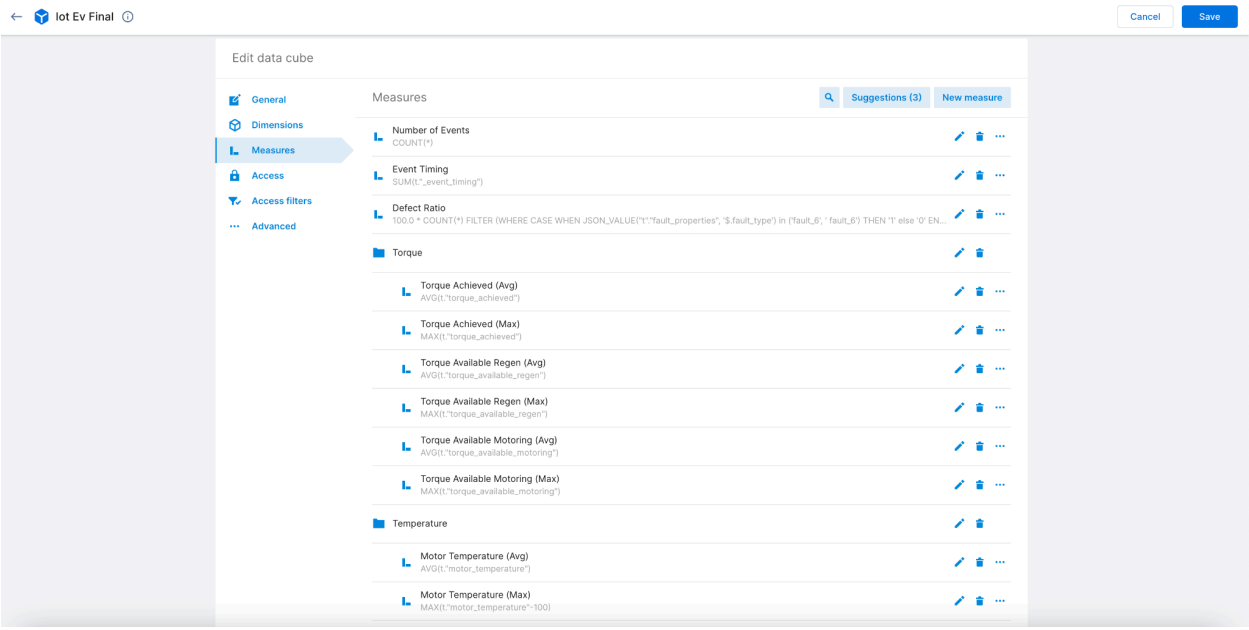
- charging issue
- charging level 1
- charging level 2
- engine overheating
- fast charging
- low battery

Cancel | Save

e. Now, let's rename the **subtype** dimension into **Firmware version** :



f. Now let's add/modify some custom measures to the datacube.



- for each **Torque and Temperature** measure, only keep/add an **AVG** and a **MAX** expression :

Torque Achieved (Avg)
AVG(t."torque_achieved")

Torque Achieved (Max)
MAX(t."torque_achieved")

Torque Available Regen (Avg)
AVG(t."torque_available_regen")

Torque Available Regen (Max)
MAX(t."torque_available_regen")

Torque Available Motoring (Avg)
AVG(t."torque_available_motoring")

Torque Available Motoring (Max)
MAX(t."torque_available_motoring")

Motor Temperature (Avg)
AVG(t."motor_temperature")

Motor Temperature (Max)
MAX(t."motor_temperature"-100)

Temperature IGBT A (Avg)
AVG(t."temperature_IGBT_A")

Temperature IGBT A (Max)
MAX(t."temperature_IGBT_A"-100)

Temperature IGBT B (Avg)
AVG(t."temperature_IGBT_B")

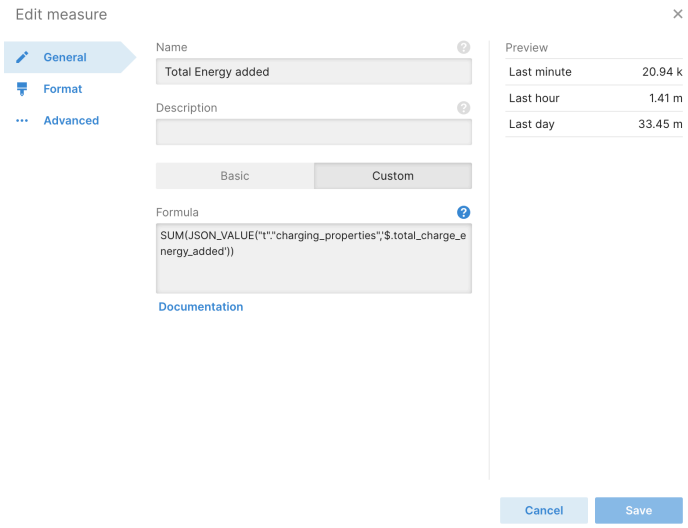
Temperature IGBT B (Max)
MAX(t."temperature_IGBT_B"-100)

Temperature IGBT C (Avg)
AVG(t."temperature_IGBT_C")

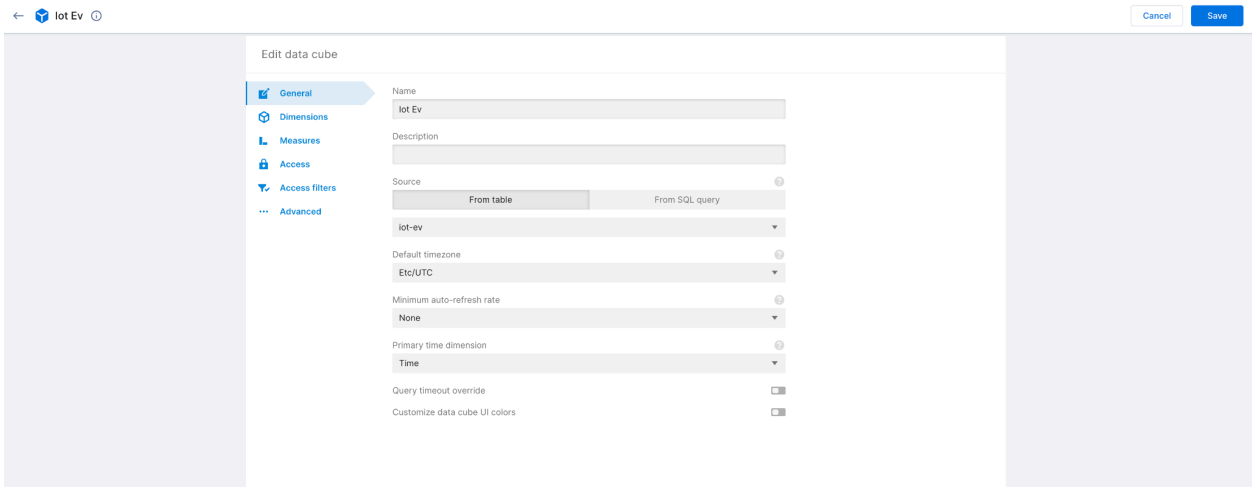
Temperature IGBT C (Max)
MAX(t."temperature_IGBT_C"-100)

g. finally , add a last measure named “**Total Energy added**” with following SQL expression:

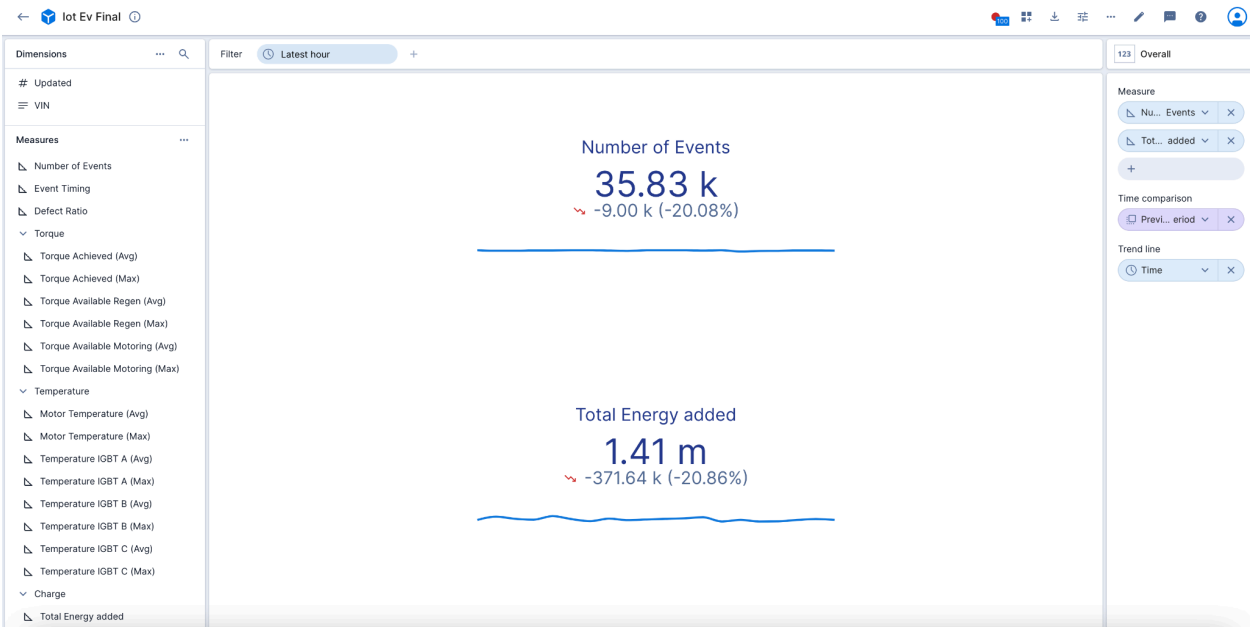
```
SUM(JSON_VALUE("t"."charging_properties",'$.total_charge_energy_added'))
```



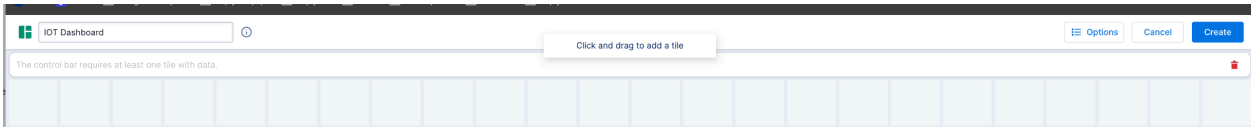
h. Give your datacube a name. And click “Save” :



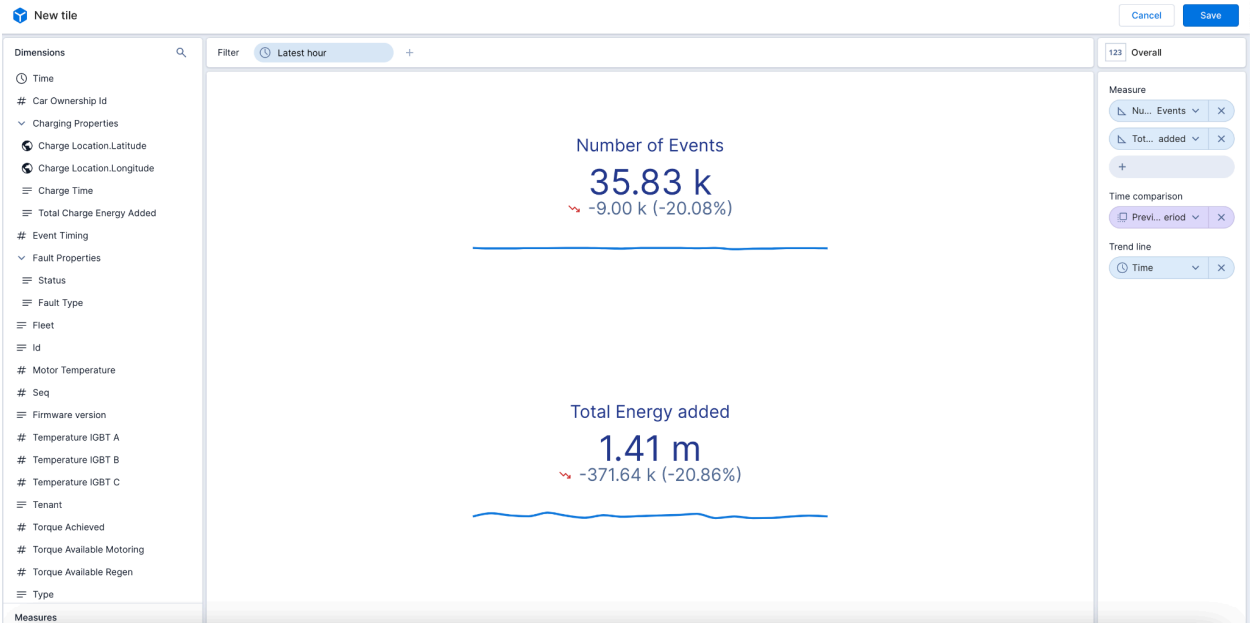
i. Now you can use the UI to drag and drop and analyze the data with existing dimensions and measures, for example, drag “Total Energy added” onto the measure lists. You can do this with multiple measures and change the measures according to your need. From here you can click  at the top right, to add this to a dashboard, but we’ll get into that further in a second. Go ahead and try it. We’ll get into creating dashboards next.



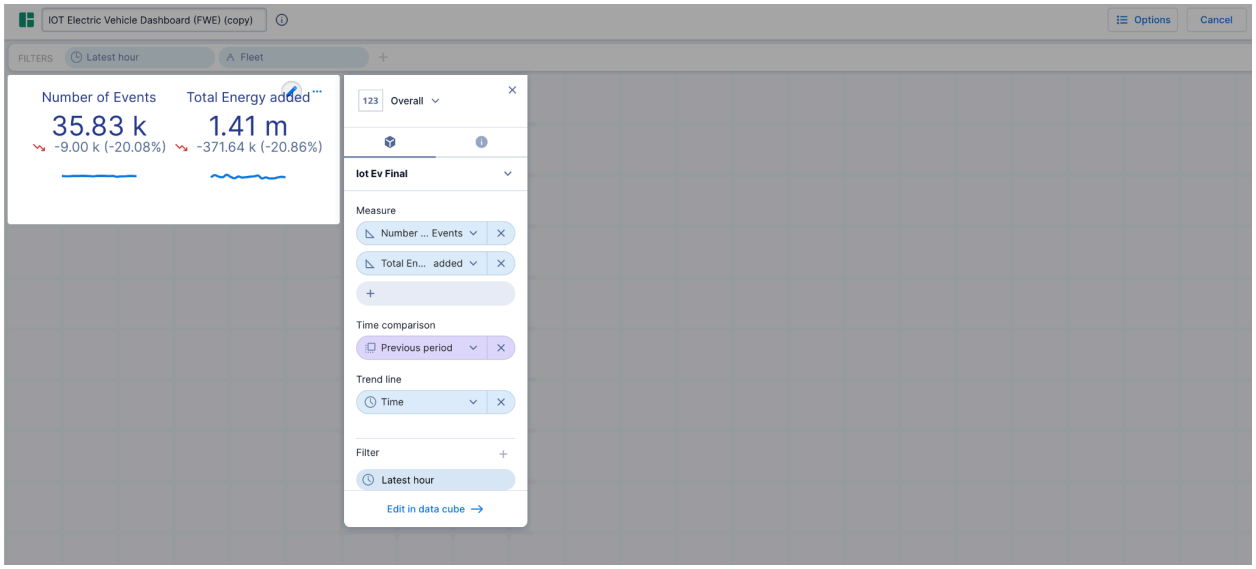
2. Creating a dashboard from a datacube is easy:
    a. Navigate back to the home screen and click “Dashboards” and “Create dashboard” at the top. Name the dashboard as “IOT Dashboard”



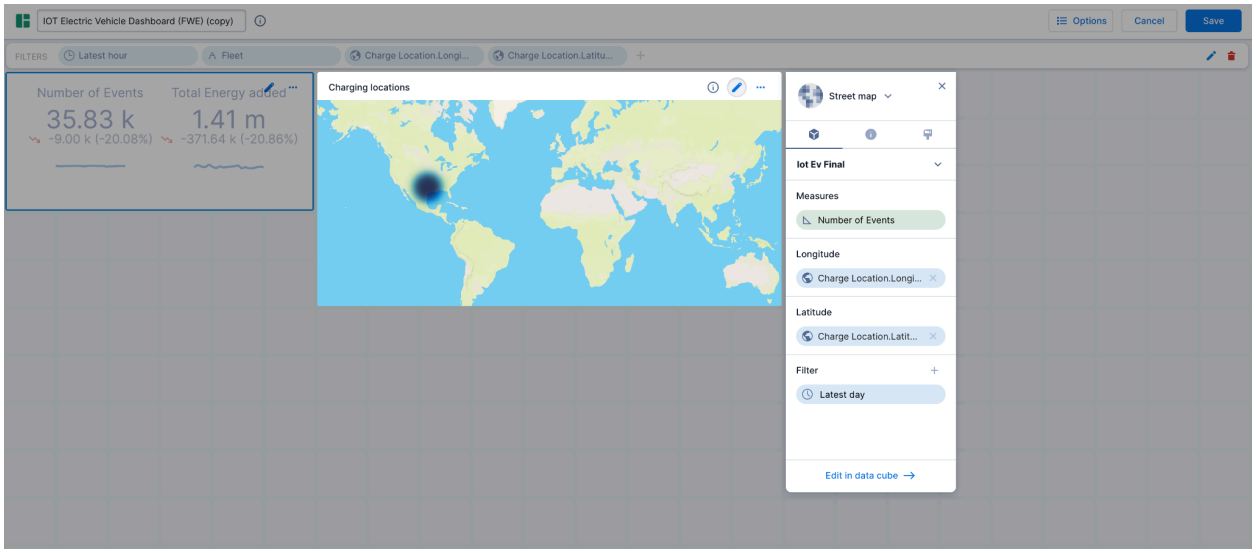
    b. Click and select an area on the dashboard, in the drop down select the cube you just created, it may be selected by default if you only have one cube in your polaris project. For chart type, select “Overall”, and click “Edit in data cube”, just create the same Overall visualization as previous step, then save :



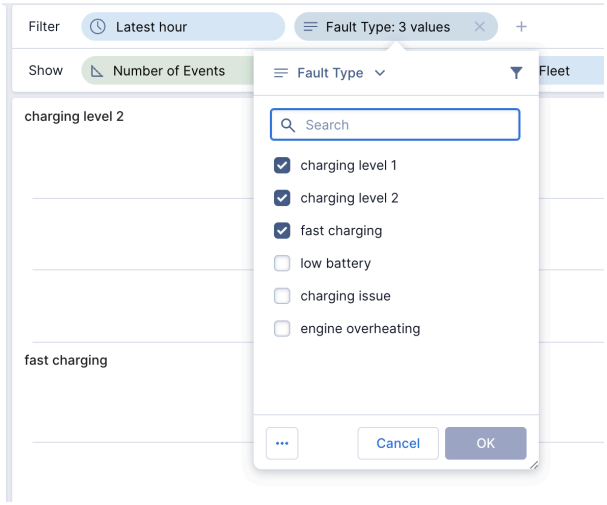
Visualization is added to the dashboard :



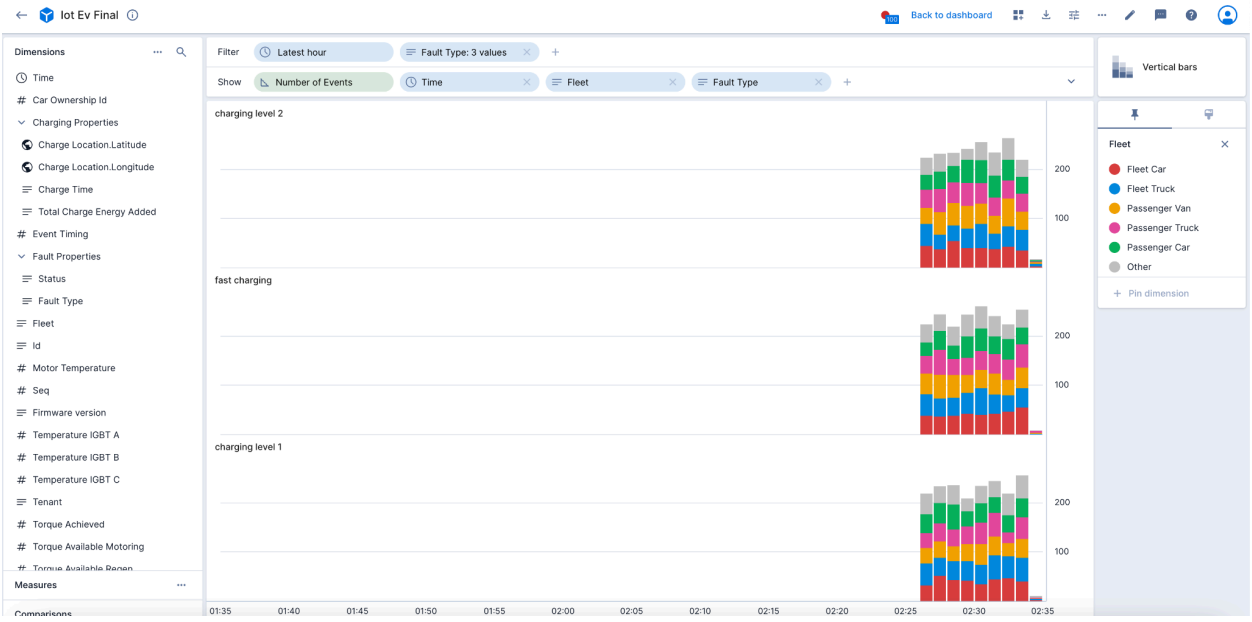
c. Now, let's create a new chart against the same datacube, select the **street map** visualization, it will automatically propose to use the 2 existing geo dimensions **LAT** and **LON** :



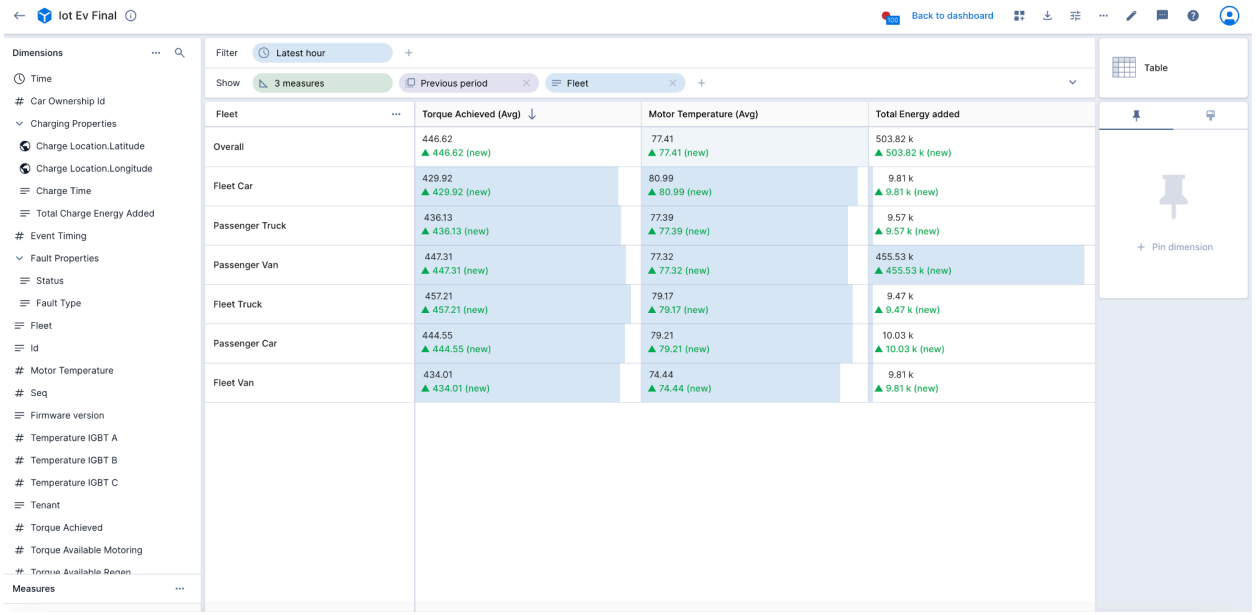
d. Now, let's create a new chart on our dashboard against the same datacube to monitor the charging status per fleet type, select vertical bars chart type, then add **time**, **fleet** and **fault type**, also add a filter on **fault type** restricted to the 3 charging types below :



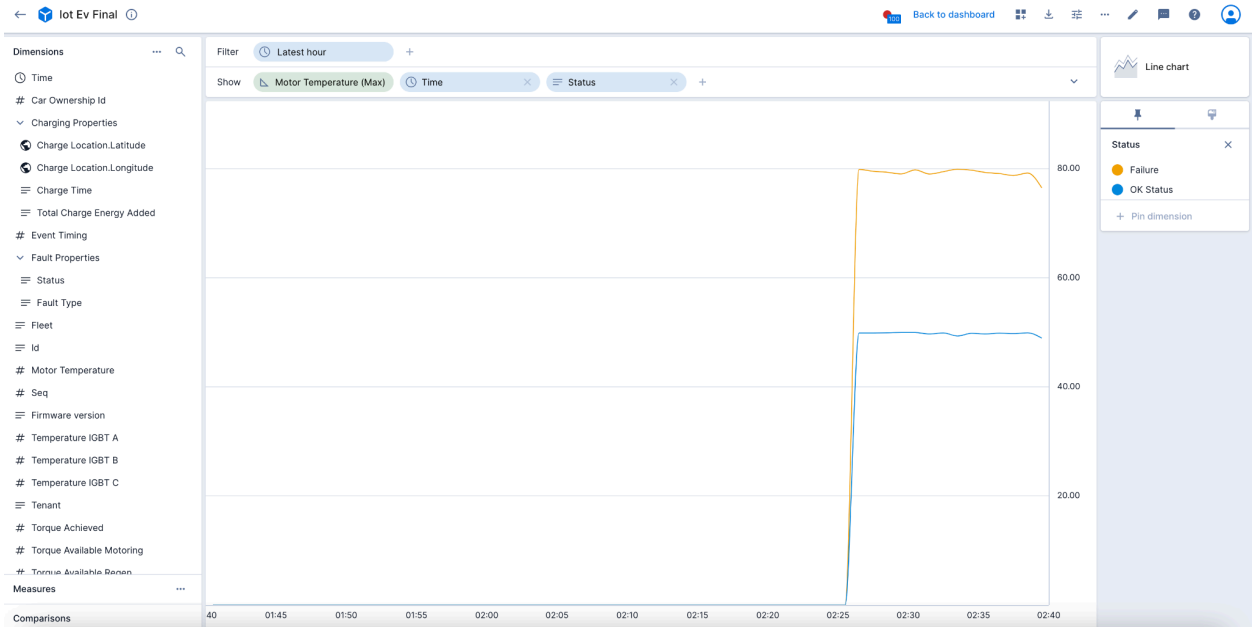
Your report should look like this before you add it to the dashboard :



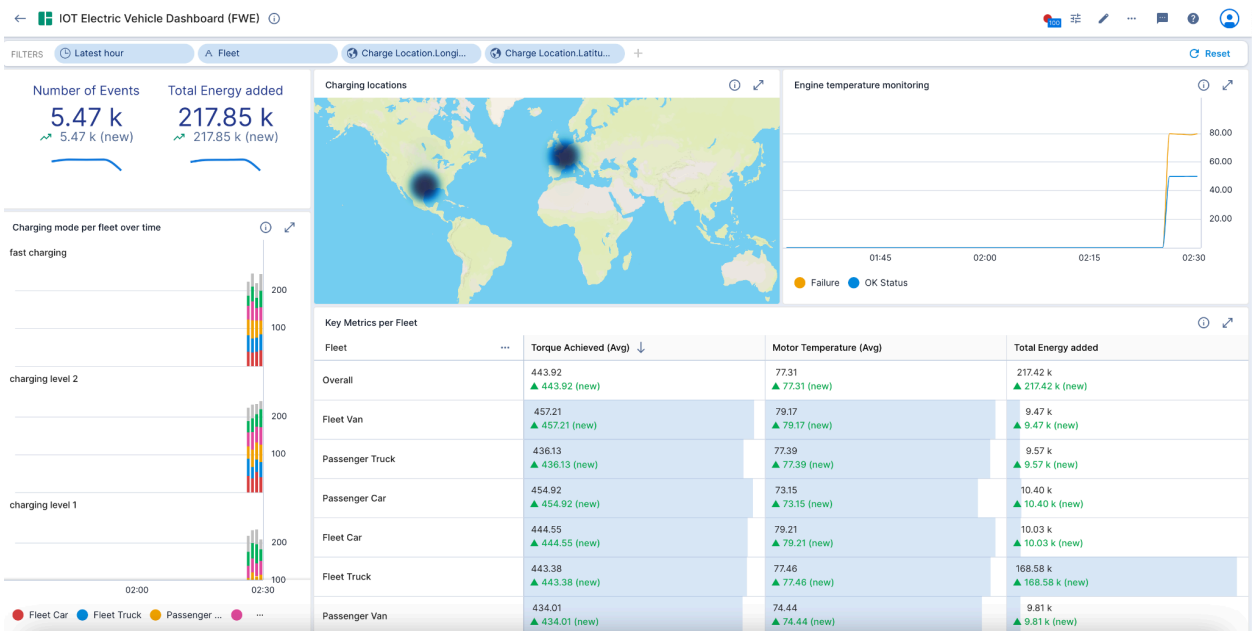
e. Now let's add a new chart to monitor Avg Torque, Avg Motor Temperature and Total energy added by selecting a Table visualization, adding these 3 measures and adding the previous period for comparison, report should look like this before you add it to the dashboard :



f. Finally, let's add one final chart to the dashboard to measure the Max Motor temperature per Status by selecting a Line chart visualization, and adding **Motor Temperature (Max)**, **Time** and **Status** to the show bar, your report should look like this before your add it to the dashboard :



g. Finally rearrange your tiles to fit the screen, edit predefined filters (time to filter on latest hour only, fleet) and save the dashboard. It should look like the example below. You can now click and interact with the dashboard, and filter it. You can also share the dashboard URL with other users, as you filter it the URL preserves the state of the filter. You can click the “Reset” button at the top right and revert to the unfiltered version. For more tips and trick see https://docs.imply.io/polaris/create-a-dashboard



3. For instructions on how to to create alerts see: https://docs.imply.io/polaris/set-up-alerts/
4. For instructions on how to create embedded visualizations see: https://docs.imply.io/polaris/embed-visualizations

General documentation around querying polaris via the API and all the building blocks is all available here.
https://docs.imply.io/polaris/query
https://docs.imply.io/polaris/api-query
https://docs.imply.io/api/polaris/api-reference