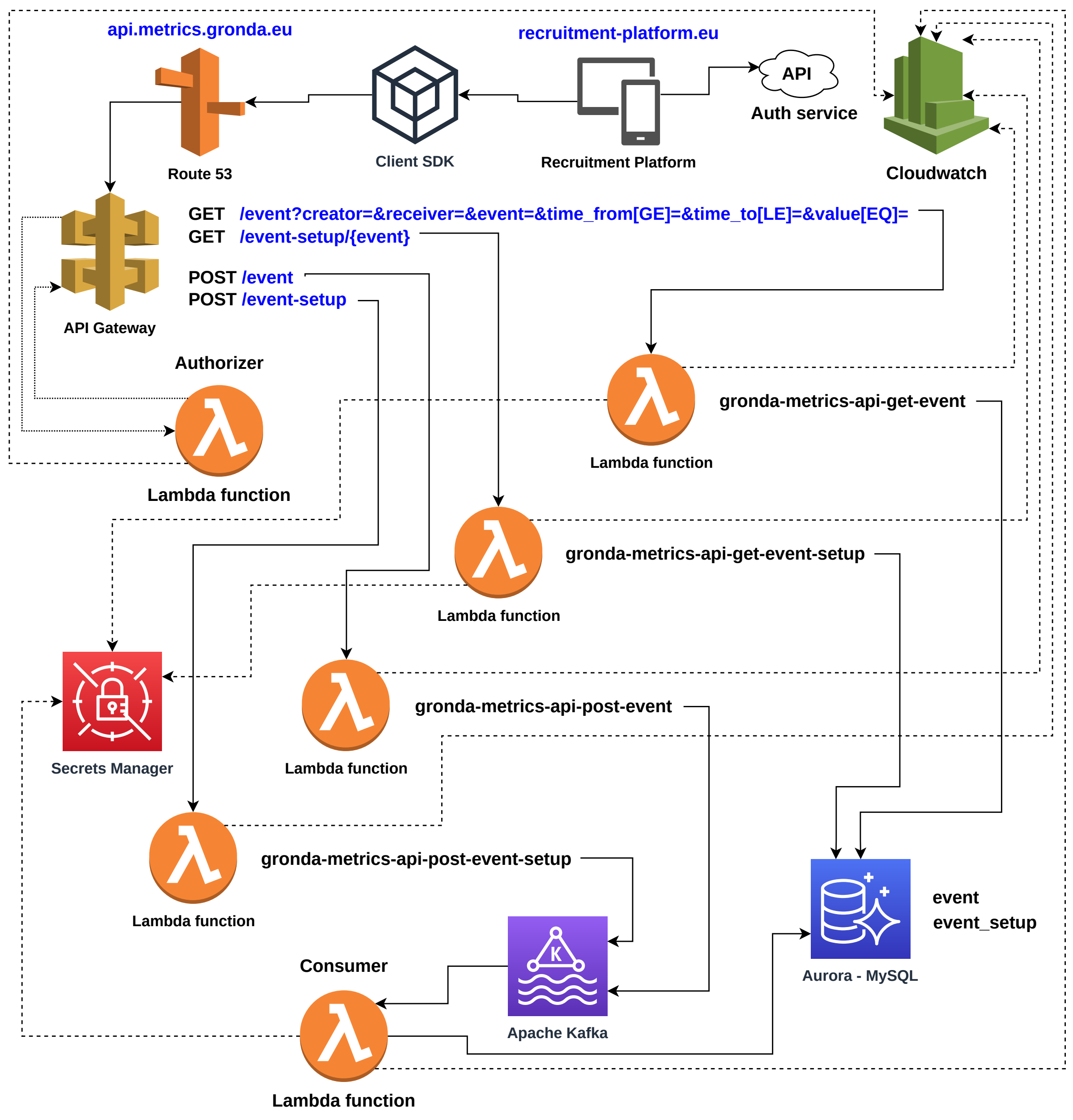
Architecture

Flowchart



Endpoints

GET /event

> Endpoint to extract all possible ways of the data stored in **event** table

Parameters (QueryString)

- **creator**: string
  - Browser session of the user triggered the event

- **receiver**: string
  - The element that receives the event, ex.: (bottomContactButtonFromClientXJobPost)
- **event**: string
  - Event-triggered by some action of a user, page, post, ex.: (click, view, play, impression)
- **time_from**: string
  - Date of the initial date range to use to filter the time
- **time_to**: string
  - Date of the final date range to use to filter the time

> *Available indexes to the QueryString **time_from[GE]** and **time_to[LE]***
>
> ***EQ**: Equal*
> ***GT**: Greater than*
> ***GE**: Greater than or equals*
> ***LT**: Less than*
> ***LE**: Less than or equal*

- **value**: string
  - Value received from the event trigger, ex.: onchange:salary (5000)

> *Available indexes to the QueryString **value[GE]***
>
> ***LK:** Contain*
> ***NL:** Not contain*
> ***EQ:** Equal*
> ***NE:** Not equal*
> ***LL:** Starts with*
> ***RL:** Ends with*

GET /event-setup/{event}

> Endpoint to get the event setup
> This setup will be used to manage how the event will be stored

Parameters

- **event**: string
    - Event stored in **event** table to associate the setup

POST /event

> Endpoint to store the events received
>
> **When the event has been storing:**
>
> **Rules:**
> - Check the setup of the event from the **event_setup** table before storing
> - If there isn't a specific event setup, the default value to use as **event_receive_option** is: **unlimited**

Payload

```
interface EventPayload {
  creator: string;
  receiver: string;
  event_type: string;
  time: string;
  value?: string;
}
```

PUT /event-setup

> Endpoint to store the setup of the event
> This setup will be used to manage the storing of event

> **Rules:**
> \- **unlimited**: Store events received unlimited
> \- **once_per_receiver**: Store events received once per receiver, if there is more than one event sent for the same receiver and creator, ignore then
> \- **once_per_event**: Store events received once per event, if there is more than one event sent for the same event and creator, ignore then

Payload

```
type EventReceiveOption = 'unlimited'
  | 'once-per-receiver'
  | 'once-per-event'

interface EventSetupPayload {
  event_type: string;
  event_receive_option: EventReceiveOption
}
```

Alternatives Considered

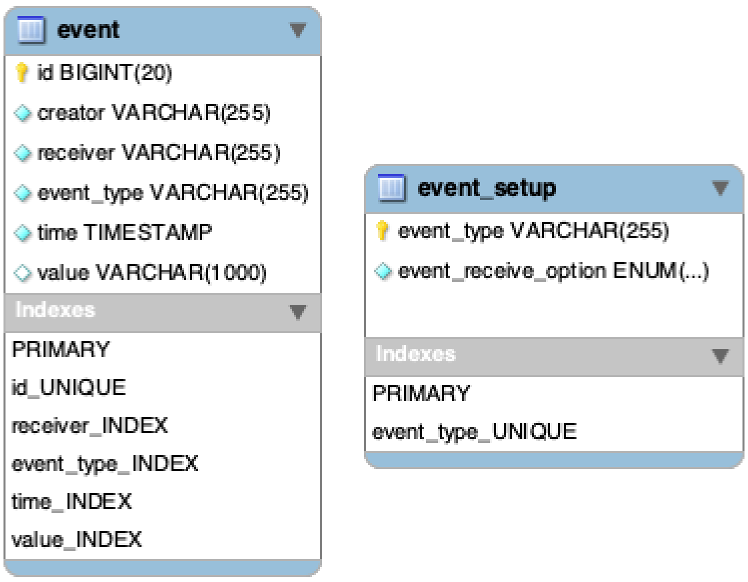
- ✔️ Relational database (Aurora)
  - Amazon Aurora is up to five times faster than standard MySQL databases and three times faster than standard PostgreSQL databases. It provides the security, availability, and reliability of commercial databases at 1/10th the cost.
  - High availability across AWS Regions with Aurora global databases.
  - https://aws.amazon.com/rds/aurora
- ❌ NoSQL database: DynamoDB
  - High cost of usage, it will be thousands of inputs per minutes
  - Terrible to manage customized queries, ex.: Query a range of dates
  - https://aws.amazon.com/dynamodb/

Database Modeling

Structure of data model

The tables don't have any relation between themselves, because, the field to relate the tables (**event**) is varchar. It was left open to store any kind of event to be flexible and customizable. The **event_setup** is not required, because, when the event has been storing, if there isn't any setup for the event, the default setup will be used, *event_receive_option**: unlimited***

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/fc93c2c1-bab1-48ae-835b-cd6997ec467d/Gronda_Metrics_API.mwb



> `event_setup`
>
> The **PK** of this table is the **event_type** field, because, there should be only one setup per **event_type**, and we could use the **PUT** verb to store the setup easily
>
> **event_receive_option:**
> ENUM(

> 'unlimited', → **default**
> 'once_per_receiver',
> 'once_per_event',
>)