Holded Senior Data Engineer Challenge

The team wants to process a new data source consisting of the following fields:

- *data: struct*
  - *id: string*
  - *amount: float*
- *attributes: struct*
  - *type: string*
  - *targetSubscription: string*
  - *X-Message-Stamp-Holded\\Shared\\Infrastructure\\Messenger\\Stamp\\NewContextStamp: struct*
    - *accountId: string*
    - *userId: string*
    - *requestPlatform: (Optional)*
  - *X-Message-Stamp-Holded\\Core\\Messaging\\Messenger\\Stamp\\TimestampStamp: string*
  - *X-Message-Stamp-Holded\\Core\\Messaging\\Messenger\\Stamp\\MessageIdStamp: string*
  - *X-Message-Stamp-Holded\\Shared\\Infrastructure\\Messenger\\Stamp\\AmplitudeTrackStamp: struct (Optional)*
    - *id: uint*
    - *trackedDevice: string (Optional)*

The data is received in JSON format through a REST API (HTTP POST /collect), which you must implement.

The ***/collect*** endpoint receives a package of events in JSON format (the array size is variable), it must generate a unique key to identify each of the events, and it must be possible to know in the future which events belong to the same package.
This producer must send each event separately to a Kafka topic (***events.raw***) and validate against a schema previously registered in Schema Registry. In case of receiving an invalid event, it must be sent to a Kafka topic (***events.dead.letter***) in JSON format and also recorded in Minio (our local S3 service) to be analyzed later.

Later, we will implement an Apache Beam Streaming application (use DirectRunner) that connects to the topic (***events.raw***) and validates it against the Schema Registry service. Once validated, it ingests the JSON and adds the following metadata:

- ***metadata***
  - ***correlationId***: Unique identifier of the event (remember that this identifier must allow you to regroup all those events that were sent together in the same package in the future)
  - ***publishTime***: The timestamp at which the event was ingested into Kafka
  - **schemaId**: Identifier of the schema against which the event has been generated and validated.

Once this field is added to the previous ones, you should have an event with the following format:

- *data: struct*
    - *id: string*
    - *amount: float*
- *attributes: struct*
    - *timestampStamp: string*
    - *messageIdStamp: string*
    - *type: string*
    - *targetSubscription: string*
    - *newContextStamp: struct*
        - *accountId: string*
        - *userId: string*
        - *requestPlatform: (Optional)*
    - *amplitudeTrackStamp: struct (Optional)*
        - *id: uint*
        - *trackedDevice: string (Optional)*
- *metadata*
    - *correlationId: string*
    - *publishTime: string (This is the timestamp of ingestion in Kafka)*
    - *schemaId: long*

This new event enriched with metadata must be saved in Minio in JSON format.

Note: You can use Avro Schema or Protobuf for your schema definitions

Finally, the enriched event must sink into a postgresql table (use default schema).

```
Unset
CREATE TABLE events (
    timestamp_stamp TIMESTAMP WITH TIME ZONE NOT NULL,
    message_id_stamp UUID NOT NULL,
    correlation_id UUID NOT NULL,
    type TEXT NOT NULL,
    target_subscription TEXT NOT NULL,
    new_context_account_id UUID NOT NULL,
    new_context_user_id UUID NOT NULL,
    data JSONB NOT NULL,
)
PARTITION BY RANGE (timestamp_stamp);
CREATE INDEX events_type_idx ON events (type);
```

As a summary, the following tasks must be performed:

- Draft the architecture you are planning to implement. You'll use this for the next part of the interview.
- Implement HTTP POST /collect method with a Kafka producer that sends valid events to ***events.raw***, validate schema against Schema Registry, and send invalid events to ***events.dead.letter*** include a header that contains the error message.
- Create an Apache Beam Streaming application (using DirectRunner) to enrich the **events** with **metadata**, validate them against Schema Registry and save all enriched events to Minio in JSON format.
- Write the enriched event to postgresql (follow table schema).
- Create a bash script to create topics in Kafka by reading the /topics/main-topics file where you will find the topics you must create with the number of partitions and replicas. (Optional)
- Create a bash script to create Kafka connectors if you consider that you should use Kafka Connect. (Optional)
- Create a bash script to register schemas in Schema Registry. (Optional)

You are provided with a docker-compose.yaml with all the necessary infrastructure. To deploy the platform, you must have Docker Desktop installed and run the following command in the Terminal:

```
Unset
$ docker compose up -d --build
```

You have kafka-ui (web interface to manage Kafka) to initially create topics, connectors, schemas,... whatever you want. So you can leave the implementation of the bash scripts for later and also you can visualize if the events are being inserted correctly in Kafka.

Once you have implemented the REST API, you can start the server:

```
Unset
$ cd api
$ poetry shell
$ poetry install --no-root
$ fastapi run api/main.py
```

The default value for port is ***8000***

You can send data using the event generator located in the *./generator* folder:

```
Unset
$ cd generator
$ poetry shell
$ poetry install --no-root
$ python generator.py
```

The default values for the hostname and filename arguments are ***localhost:8000*** and ***events.json***.

Finally, to shutdown the platform, run the following command in Terminal:

```
Unset
$ docker compose down
```

You can use the libraries you want as long as you explain what you are using them for.

Remember that your code should be as close to what a production release would be. Tests are appreciated.

Requirements

- Docker Desktop
- poetry or another virtualenv

Things to evaluate

- All code works
- Clarity, cleanliness and good practices applied
- Tests
- Explain the decisions you make

IMPORTANT! Remember, behind this process there are people to evaluate your test, be clear and explain why you do what you do and above all try to ensure that the test can run without problems on any computer.