Tracking Project

Please see high_level_diagram.png for the diagram layout of the tracking project.

Specifications

-> Capability to continuously receive data (1 payload per second) from various data sources

-> Uniform output

-> Date format: *YYYY-MM-DD HH:mm:ss*

-> Speed unit: *mph*

-> Alarm code: *Common form*

Non-functional specifications

-> Minimum latency

-> High availability

-> Read and Write heavy (reading incoming data, 1000s of user queries)

Project flow layout

1. Separate flows are created to simulate the data that will be incoming from different sources and ensure that a message is sent every second from the respective data source.
2. Data will be sent to a specific topic based on the data input format.
   An assumption is made that all the respective data formats will have the same schema. This will ensure that Kafka consumers only have to listen to topics based on the data format instead of topics for every company. This also ensures the data transformation is per input format instead of per company. Currently there are only two companies that send data in different formats; this could easily change to 100 different companies with only a few formats between them.
3. An Kafka consumer listens to the specific topics and consumes data as soon as it lands on the topic (using the earliest offset to ensure data is read from the beginning).
4. The incoming data is immediately stored in Minio (which can be any data storage like s3) to preserve the raw data.
5. The transformation for the data is:
   CSV files
   Get converted to JSON; general conversion is used, in other words, no specific schema; I assumed that the header is present in each payload.
   The date field gets extracted as a flowfile attribute.
   A JoltTransform processor is used to transform the data into the desired schema as well as the desired data formats for specific fields like the timestamp.
   A LookupRecord processor is used to extract the current alarm code, look it up in a simple key lookup service, and replace it with the value returned by the lookup service.
   JSON files
   Extract the date field as a flowfile attribute.
   A JoltTransform processor is used to transform the data into the desired schema as well as the desired data formats for specific fields like the timestamp.

6. The transformed data gets combined and sent as a message to a *trackingdata.output.transformed* Kafka topic.
7. Druid is configured to monitor the Kafka topic and pull the data from the Kafka topic into real-time nodes for segmenting.
8. I assumed a type of backend server would be configured to query the data from Druid. Specific routes will either get all the data or a specific track ID, and some parameters can be added to get data over a specific timeframe. More routes can be added; this is just an initial list.
9. Clients/users will be able to query the data through the server, and I assumed that they would only have read access.

Additional notes

- I ran an independent Spark analysis to ensure that the two date fields in the JSON payload are always the same. From there, I chose one field and saved the other field in its original format in the metadata object that gets created.
- Currently, it is easier to use jolt to transform the records because there are only two formats and each contains one jolt specification per format. An alternative would be to create read and write schemas in an AvroSchema registry and convert the data accordingly. I decided not to use this route because of the schema overhead. Depending on how the project grows and how the pipeline will be scaled and adjusted to accommodate more formats, I can revisit this decision.
- The NiFi pipeline only catches the failed flowfiles of the processors that do not have a default error attribute written to the flowfile. The processors with the default error attribute get logged.
- Only the common fields in the data are added to the main JSON body; all the uncommon fields between the two payloads get added to a metadata object along with a sourceTopic field that can be used to filter the data based on the specific input format.
- The required date format is YYYY-MM-DD, uppercase DD stands for day of the year, when implemented can be quite confusing for example 2021-11-68. I assumed and implemented it as lowercase dd.

Please note that this is the first time that I have worked with Apache Druid and time series databases in general. My experience is in relational databases. I am unsure if the schema format, especially the metadata object, is correctly implemented. In Elasticsearch, object data field types can contain different fields to ensure that all the data is preserved. For example, on one project I worked on, there were 3 different selections based on the user's input; the users could choose all of them or only a specific selection. The output would contain common fields that should always be returned and then an object with the information of the selected input, which is not necessarily the same for all outputs.