**Tasks to be performed:**

* Using Auto Loader to read streaming data from object storage.
* Perform streaming transformation.
* Write data to sink.
* Preparing your Structured Streaming Code for production.
* Read data from delta lake, transform, and write to delta lake.
* Read data from Kafka, transform, and *write to Kafka*.
* Visualization in Power BI.

**Building a Realtime Reporting Channel using Databricks Structured Streaming, Power BI**

**Use case Overview:**

Using Databricks for reading streaming data form data sources and then implementing Medallion Architecture, where first raw data is ingested from the source in raw format in this case JSON which is refer to as bronze stage, where table data is in raw format without any transformations. Then we transform incoming data and save the transformed data in silver stage and finally we make business ready data in gold stages according to the requirements.

All the architecture is orchestrated using Delta Live Tables, which gives us the ability to Extract, Transform, and Load the data as a pipeline.

**Desirable outcome:**

* Ingestion of Event Stream.
* Performing conversions according to the requirements.
* Storing data, ability to perform UPSERT operations.
* Data visualization.

**Concepts used:**

* Structured Streaming

You can use Databricks for near real-time data ingestion, processing, machine learning, and AI for streaming data.

Databricks offers numerous optimizations for streaming and incremental processing. For most streaming or incremental data processing or ETL tasks, Databricks recommends Delta Live Tables.

Most incremental and streaming workloads on Databricks are powered by Structured Streaming, including Delta Live Tables and Auto Loader.

* Delta Live Table

Delta Live Tables is a declarative framework for building reliable, maintainable, and testable data processing pipelines. You define the transformations to perform on your data and Delta Live Tables manages task orchestration, cluster management, monitoring, data quality, and error handling.

Instead of defining your data pipelines using a series of separate Apache Spark tasks, you define streaming tables and materialized views that the system should create and keep up to date. Delta Live Tables manages how your data is transformed based on queries you define for each processing step. You can also enforce data quality with Delta Live Tables expectations, which allow you to define expected data quality and specify how to handle records that fail those expectations.

* Auto Loader

You can use Auto Loader in your Delta Live Tables pipelines. Delta Live Tables extends functionality in Apache Spark Structured Streaming and allows you to write just a few lines of declarative Python or SQL to deploy a production-quality data pipeline with:

* + Autoscaling compute infrastructure for cost savings
  + Data quality checks with expectations
  + Automatic schema evolution handling
  + Monitoring via metrics in the event log

You do not need to provide a schema or checkpoint location because Delta Live Tables automatically manages these settings for your pipelines. See Load Data with Delta Live Tables.

**Example Code:** Loading data from cloud storage, using structured streaming and then adding it to Delta Live Table.

import dlt

@dlt.table()

def load\_data(checkpoint\_path, table\_name):

return spark.readStream

.format("cloudFiles")

.option("cloudFiles.format", "json")

.option("cloudFiles.schemaLocation", checkpoint\_path)

.load(file\_path)

.select("\*", input\_file\_name().alias("source\_file"), current\_timestamp().alias("processing\_time"))

.writeStream

.option("checkpointLocation", checkpoint\_path)

.trigger(availableNow=True)

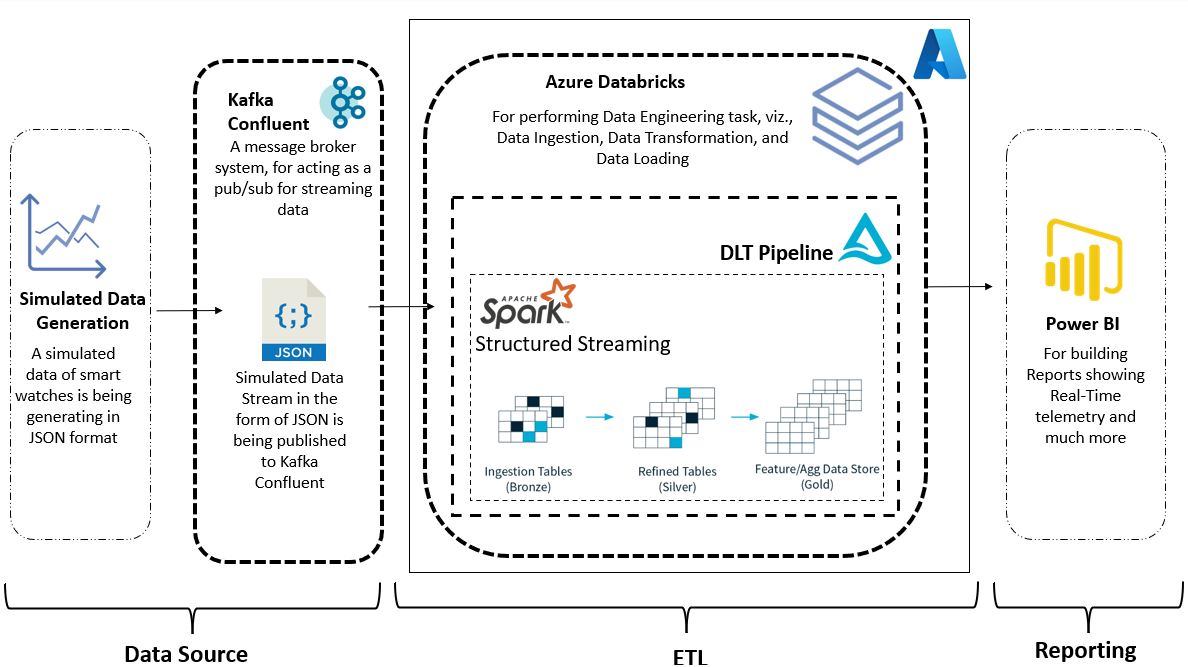
.toTable(table\_name))

* Confluent Kafka

Confluent Platform is a full-scale data streaming platform that enables you to easily access, store, and manage data as continuous, real-time streams. Built by the original creators of Apache Kafka®, Confluent expands the benefits of Kafka with enterprise-grade features while removing the burden of Kafka management or monitoring. Today, over 80% of the Fortune 100 are powered by data streaming technology – and most of those leverage Confluent.

**Workflow**

**Architecture Overview:**



* **Defining Data Source**

Here we have tried to simulate heartbeat data which is generated from smart watches. Which is done with the help of libraries like Faker and colorama.

Format of data being generated:

A screenshot of a computer

Description automatically generated with medium confidence*{"Ip": Ip, "time”: time, "version”: version, "model": tracker, "color": color, "heart\_bpm":bpm, "kcal":kcal}*

As shown above the format of data is in the form of JSON file format, and this data is being send to Kafka Confluent message broker service.

With this we can simulate a Kafka data source with real-time-like data flowing through it.

* **Data Ingestion**

As we have setup our data source, now we need a way to seamlessly ingest that data so that we can perform any transformations on it, to do that we are using Apache Structured Streaming service in Azure Databricks, which is which is a product of Apache, and it gives us the ability to ingest data from various sources such as Kafka. With the help of structured streaming, we can ingest our streaming data without having to deal with a lot of manual configurations or in case any kind of error occurs the structured streaming has some protocols which can deal with that or if there is any kind of schema differences it also has the option to deal with it.

To be able to connect with Kafka we need few credentials for example bootstrap server, API key and API secret and with these configurations we can use Kafka connector provided in Databricks to connect with Kafka confluent for ingesting or accessing the real time streaming data.

**Example Code:**

raw\_kafka\_events = (spark.readStream

.format("kafka")

.option("subscribe", TOPIC)

.option("kafka.bootstrap.servers", KAFKA\_BROKER)

.option("kafka.security.protocol", "SASL\_SSL")

.option("kafka.sasl.jaas.config", "kafkashaded.org.apache.kafka.common.security.plain.PlainLoginModule required username='{}' password='{}';".format(confluentApiKey, confluentSecret))

.option("kafka.ssl.endpoint.identification.algorithm", "https")

.option("kafka.sasl.mechanism", "PLAIN")

.option("failOnDataLoss", "false")

.option("startingOffsets", "earliest")

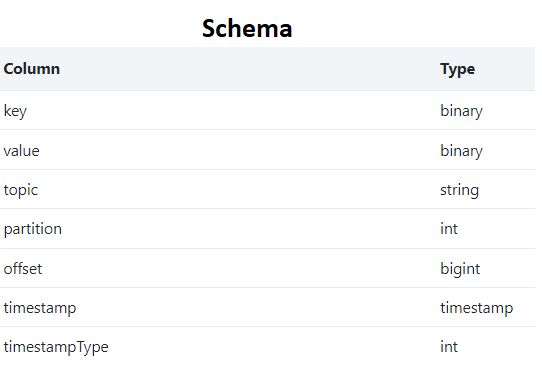
.load()

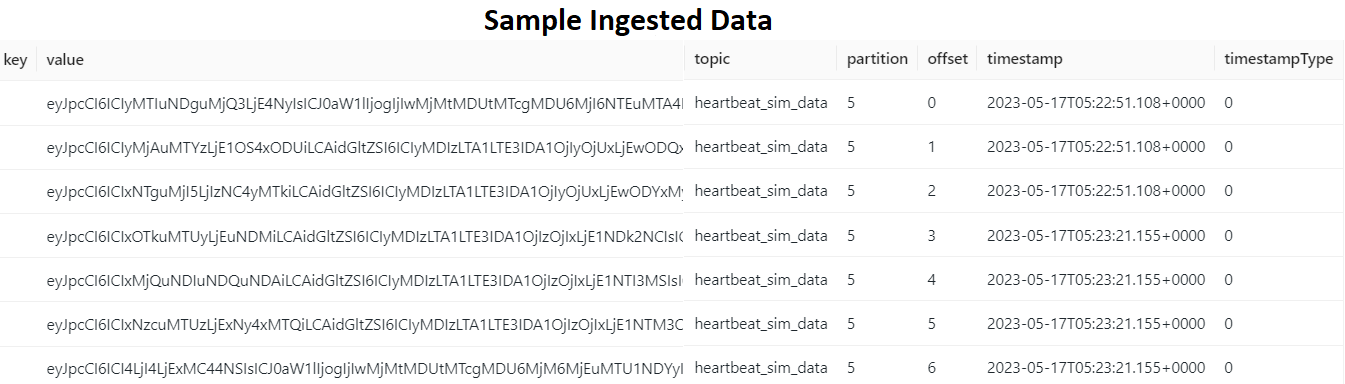
)

* **Data Transformation**

As we are ingesting our data in JSON format so it is not going to be human readable form we need to convert it into human readable form and then perform required transformations so that we can just fulfill our purpose or the requirements that we want for this demonstration.

**Raw Ingested data:**





As it can be seen the ingested data does not have human readable format or any transformation cannot be performed on this kind of data when we import JSON Data we have a key and a value contains the payload which is encrypted former we have to explicitly yeah extract the Jason data from this raw data that we have ingested from Kafka and then we performed some required transformation so that it can be converted into a table which can be used for further processing or any other task.

As the industry data does not have any schema so no operations can be performed on it in the next step what we are going to do we are going to assign a schema to this ingested data so that every time any Kafka payload is ingested it is converted automatically in a pipeline.

**Code:** for creating structured data

@dlt.table(comment="real schema for Kakfa payload",

table\_properties={"pipelines.reset.allowed": reset},

temporary=True)

def bpm\_raw():

return (

# kafka streams are (timestamp,value) with value containing the kafka payload, i.e. event

# no automatic schema inference!

dlt.read\_stream("kafka\_events")

.select(col("timestamp"),

from\_json(col("value").cast("string"), event\_schema).alias("event"))

.select("timestamp", "event.\*")

)

After providing a struct data type we will get a schema which will look like this:

A screenshot of a computer

Description automatically generated with low confidence

Now we can perform transformations on this data and convert the ingested data according to our needs and requirements.

Transforming data according to the need.

**Code:** here we are just selecting the columns that we need according to the requirements, and we also are implementing some data quality factors so that we won't get null values or the values which will put any kind of bottleneck and analysis.

@dlt.table()

@dlt.expect\_or\_drop("heart rate is set", "bpm IS NOT NULL")

@dlt.expect\_or\_drop("event time is set", "time IS NOT NULL")

@dlt.expect\_or\_drop("human has pulse >0", "bpm > 0")

@dlt.expect\_or\_drop("tracker is valid", "model <> 'undef' ")

def bpm\_cleansed(table\_properties={"pipelines.reset.allowed": reset}):

return (

dlt.read\_stream("bpm\_raw")

.withColumnRenamed("heart\_bpm", "bpm")

.select("time", "bpm", "model")

)

First streaming analysis purposes and also for visualization what we have done is as we are getting different events, we have aggregated them into one single average value and then what we have done is we have a recorded the number of events that have happened and what is the average heart rate which is being calculated as the data is being injected and for the same dashboard is also created

**Code:**

@dlt.table(name="global\_stat")

def bpm\_global():

return (

dlt.read\_stream("bpm\_cleansed").groupBy()

.agg(count('bpm').alias('eventsTotal'),

max('bpm').alias('max'),

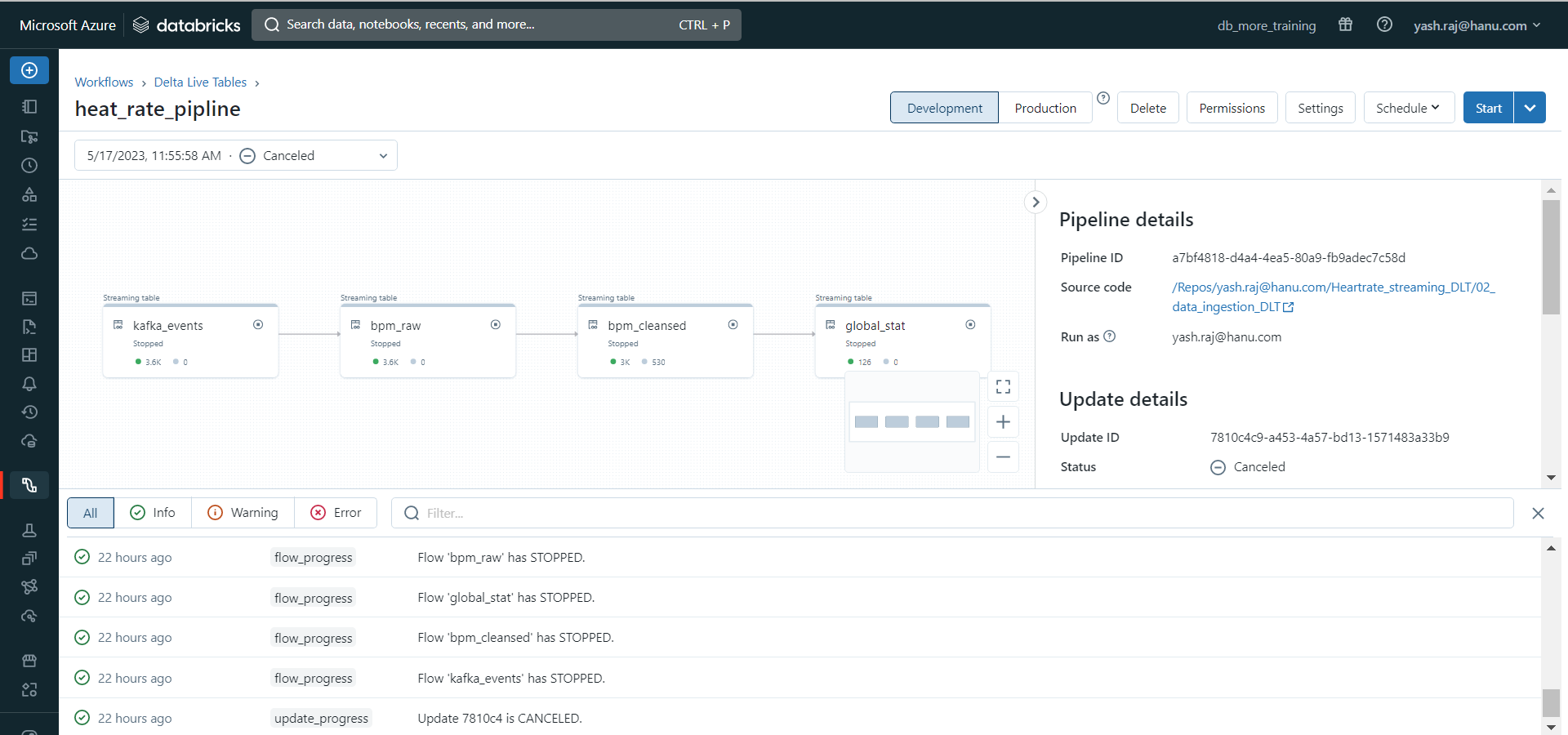
min('bpm').alias('min'),

avg('bpm').alias('average'),

stddev('bpm').alias('stddev')

)

)

* Now on the final data step what we are going to do as we have aligned all the delta live tables now we are going to create a delta live table pipeline in UI. The final output or the presentation of how data is flowing is shown in the image below.
* **Data Visualization**

As the transformation is being performed in Databricks what we have done we have a used delta table as a sink for the final output and we have connected that delta table to our power BI so that a dashboard can be created for seeing the real time changes which is being coming along the streams and we have used a feature called *auto refresh page where in every 5 seconds* our page is being refreshed so that as the new data is coming so we are we only see the updated data in our dashboard.