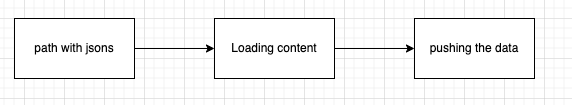
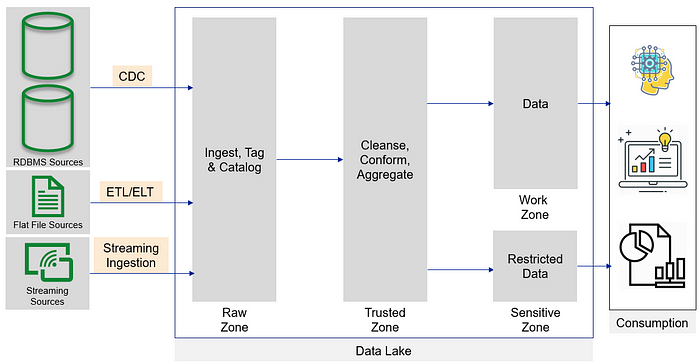
Assignment 2: Coding skills assessment:

High level:



I will use CDC – change data capture, it’s a technology use to determine and track data that has changed, so an action can be take.



Links for CDC explanation:

<https://en.wikipedia.org/wiki/Change_data_capture>

<https://towardsdatascience.com/change-data-capture-cdc-for-data-ingestion-ca81ff5934d2>

* I will contain all jsons in certain path
* When new json will add the CDC will detect a change in the data in the path and triger an action
* I will load the json with pyspark into a data frame, and will prepper it for next stage
* Pushing the data into Postgress

Complete code :

import time

from watchdog.observers import Observer

from watchdog.events import FileSystemEventHandler

from pyspark.sql import SparkSession

from pyspark.sql.functions import explode

from pyspark.sql.types import StringType, StructField, StructType

# Initialize SparkSession

spark = SparkSession.builder \

.appName("CDCDataLoader") \

.getOrCreate()

# Define schema for objects\_detection\_events

objects\_detection\_schema = StructType([

StructField("vehicle\_id", StringType(), True),

StructField("detection\_time", StringType(), True),

StructField("detections", StringType(), True)

])

# Define schema for vehicle\_status

vehicle\_status\_schema = StructType([

StructField("vehicle\_id", StringType(), True),

StructField("report\_time", StringType(), True),

StructField("status", StringType(), True)

])

# Define a class for file system event handling

class FileHandler(FileSystemEventHandler):

def \_\_init\_\_(self, spark):

self.spark = spark

def on\_created(self, event):

if event.is\_directory:

return

filename = event.src\_path

if 'objects\_detection\_events' in filename:

objects\_detection\_df = self.spark.read.json(filename, schema=objects\_detection\_schema)

objects\_detection\_df = objects\_detection\_df.withColumn("detections", explode(objects\_detection\_df.detections))

objects\_detection\_df = objects\_detection\_df.select(

"vehicle\_id",

"detection\_time",

"detections.object\_type",

"detections.object\_value"

)

# Write objects\_detection DataFrame to PostgreSQL

objects\_detection\_df.write \

.format("jdbc") \

.option("url", "jdbc:postgresql://your\_postgresql\_host:5432/your\_database") \

.option("dbtable", "objects\_detection\_table") \

.option("user", "your\_username") \

.option("password", "your\_password") \

.save()

elif 'vehicle\_status' in filename:

vehicle\_status\_df = self.spark.read.json(filename, schema=vehicle\_status\_schema)

# Write vehicle\_status DataFrame to PostgreSQL

vehicle\_status\_df.write \

.format("jdbc") \

.option("url", "jdbc:postgresql://your\_postgresql\_host:5432/your\_database") \

.option("dbtable", "vehicle\_status\_table") \

.option("user", "your\_username") \

.option("password", "your\_password") \

.save()

# Set up file system event handler

event\_handler = FileHandler(spark)

observer = Observer()

observer.schedule(event\_handler, path='/path/to/watch', recursive=False)

observer.start()

try:

while True:

time.sleep(1)

except KeyboardInterrupt:

observer.stop()

observer.join()

# Stop the SparkSession

spark.stop()

This code continuously monitors the specified directory for new file arrivals. When new files are created, the event handler triggers the processing and loading of the file's data into PostgreSQL using PySpark. The application remains active and responds to new file arrivals until it is explicitly stopped.

DB:

**Why I using SQL and not NoSQL?**

* **Structured data and predefined queries:** The scenario mentions tables with well-defined schemas and emphasizes common user queries. This suggests that the data is mostly structured and requires specific queries, which are strengths of relational databases like PostgreSQL. NoSQL databases are better suited for flexible and evolving data structures where queries are less predictable.
* **Focus on data integrity and ACID transactions:** The script emphasizes ensuring data integrity through database transactions and schema management. NoSQL databases often offer weaker consistency guarantees compared to relational databases, which prioritize ACID (Atomicity, Consistency, Isolation, Durability) properties.

Main reasons I would have choose NoSQL:

* **Highly scalable data ingest with real-time analytics**
* **Unstructured or semi-structured data**

NoSQL options like Cassandra or MongoDB could be considered if the scenario significantly changes towards handling massive, real-time data or highly unstructured data models.

**Why did I choose Postgress?**

* **Structured data :** The scenario involves well-defined tables with specific data types, which aligns perfectly with PostgreSQL's ability to handle structured data and enforce schema constraints.
* **Data integrity and ACID transactions:** Ensuring data consistency, PostgreSQL upholds ACID properties, guaranteeing the atomicity, consistency, isolation, and durability of your data.
* **Robustness and scalability :** 
  1. **Handles large datasets:**PostgreSQL is known for its ability to handle large and complex datasets efficiently. This makes it a reliable choice for storing and managing the potentially large volumes of data coming from the Mobileye cars
  2. **Scales to your needs:** PostgreSQL can be scaled horizontally to accommodate increasing data volumes and user demands. This allows you to adapt the database infrastructure as your system grows.

**Why I used Pyspark ?**

* **Data Processing:** PySpark is used to read JSON files containing objects\_detection\_events and vehicle\_status data. The spark.read.json() function is employed to create DataFrames from these JSON files, utilizing the specified schemas to ensure the data is correctly parsed.
* **Database Interaction:** PySpark's DataFrame API is utilized to interact with PostgreSQL. DataFrames containing the transformed data are written to PostgreSQL tables using the write.format("jdbc") function. This allows for efficient bulk insertion of data into the database.
* **Parallel Processing:** PySpark enables parallel processing of data, which can improve performance when dealing with large datasets. Spark distributes the processing tasks across multiple executors, allowing for faster data processing and loading.