Taxi Casestudy

- **Team:**

1- Mohamed Ahmed Fathy

2- Mohamed Abdelsalam

3- Dina Hosny

4- Kareem Elbeltagy

5- Sara Salah

- **Introduction :**

The Taxi Casestudy project revolves around taxis in New York City and involves the implementation of a data pipeline using various AWS services. This project was developed as part of the Data Management track at ITI.
The decision to utilize AWS for our pipeline was driven by the numerous advantages offered by the AWS Cloud. By leveraging AWS services, we were able to efficiently manage and process large volumes of data while ensuring scalability, reliability, and security.

The pipeline starts by uploading the source files to Amazon S3, a highly scalable object storage service. To simulate real-time data, we employed AWS Lambda, a serverless compute service. The Lambda application reads the source files and randomly selects a number of records ranging from 1 to 100. These selected records are saved in Amazon DynamoDB, serving as a checkpoint to prevent duplicate data transmission in the future. Additionally, the selected records are subtracted from the DynamoDB table and saved back to S3.

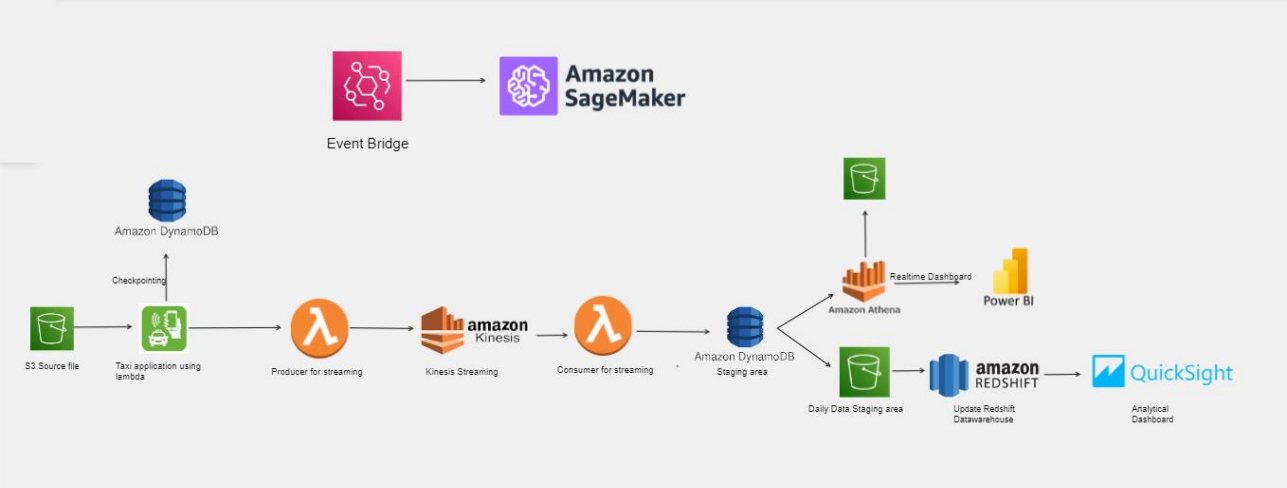
Once the records are saved in S3, a Lambda producer is triggered, capturing the arrival of the file. It then sends the data to an Amazon Kinesis stream, which acts as a data ingestion layer. A consumer process retrieves the data from Kinesis and stores it in a DynamoDB table, which serves as a staging area and NoSQL database for real-time streaming.

The pipeline supports two main streams: real-time streaming and analytical streaming. For real-time streaming, we utilize Amazon Athena, an interactive query service, to fetch data from the DynamoDB table. The retrieved data is backed up on S3, and Power BI, a business analytics tool, automatically refreshes every 30 seconds using the data within Athena. This enables the creation of a real-time dashboard for monitoring purposes.

In contrast, the analytical dashboard focuses on historical data. At the end of each day, a Lambda function is triggered to extract the data specific to that day. This data is then stored in S3. Another Lambda function retrieves this data from S3 and

writes it into an Amazon Redshift table, a powerful data warehousing solution. QuickSight, a cloud-based business intelligence tool, copies the data from Redshift and creates an analytical dashboard, providing insights into historical patterns and trends.

By combining the capabilities of various AWS services, the Taxi Casestudy project delivers a robust and scalable data pipeline for processing taxi-related data in real-time and generating analytical insights for informed decision-making.



- **Application :**

The application code provided is a Python script that serves as the AWS Lambda function for the Taxi Casestudy project. Here's a brief description of how it works:

The script starts by setting up the AWS clients for S3 and DynamoDB using the boto3 library.
It specifies the S3 bucket and object key where the source data file is stored and the DynamoDB table name for checkpointing.
The script generates a random range of records to select from the source data file.
It reads the Parquet file from S3 using the specified bucket and object key, converting it to a Pandas DataFrame.
The script checks if there are any records in the new DataFrame. If it's empty, an exception is raised.
NaN and None values in the DataFrame are converted to 0, and all columns are converted to strings for consistency.
It retrieves the existing data from the DynamoDB table and stores it in a DataFrame.
If the existing data is empty, all records from the new DataFrame are inserted into the DynamoDB table with a unique ID generated for each item.
If there is existing data, the script compares the data in both DataFrames and retrieves the unique records.
The unique records are converted to the desired data types using a dictionary mapping of column names to their respective data types.
The first 100 records from the unique DataFrame are selected for further processing.
The selected records are saved as a Parquet file in S3.
The DataFrame records are converted to a list of dictionaries, and a unique ID is generated for each item.
The records are then inserted into the DynamoDB table using batch writing.
Overall, the application code demonstrates how the Lambda function processes the data, performs data type conversions, and ensures the uniqueness of records

before storing them in the DynamoDB table and saving a subset of records as Parquet files in S3.

- **Producer:**

AWS Lambda function for the producer component in the Taxi Casestudy project that should take data from the S3 file and stream it into Kinesis, it has a trigger on the S3 that once application writes any file, it will trigger automatically and stream it into kinesis!
Here's a brief description of how it works:

The script imports the necessary libraries and sets up the AWS clients for S3 and Kinesis using the boto3 library.
It specifies the S3 bucket and object key where the Parquet file is stored, as well as the Kinesis Data Stream name to which the records will be sent.
The Parquet file is retrieved from S3 using the specified bucket and object key. It is then read into a Pandas DataFrame.
The script checks if there are any records in the DataFrame. If it's empty, an exception is raised.
Each record in the DataFrame is sent to the Kinesis Data Stream using a loop. The record is converted to JSON format and encoded before being sent.
The put_record method of the Kinesis client is used to put the record into the specified stream. The partition key is set to a constant value in this example.
The lambda_handler function is called manually to trigger the Lambda function.
In summary, the producer code reads the Parquet file from S3, checks for records, and sends each record as a JSON-encoded message to the specified Kinesis Data Stream.

Consumer:

Consumer has a trigger on kinesis, once data arrives on kinesis it will automatically work and make all the required transformation needed for data then save it in dynanmoDB.

Here's a brief description of how it works:
The script imports the necessary libraries and sets up the AWS client for DynamoDB using the boto3 library.
It defines a helper function, get_last_id, which retrieves the last ID value from a DynamoDB table.
The lambda_handler function is the main entry point for the Lambda function.
It initializes the DynamoDB client and specifies the table name where the data will be stored.
The event parameter contains the records received from the Kinesis Data Stream.
For each record in the event, the script decodes the data and parses it as JSON.
The JSON data is converted to a Pandas DataFrame, specifically for the Green Taxi data.
The script performs various transformations on the DataFrame to prepare it for insertion into DynamoDB. This includes converting datetime columns, calculating trip duration, mapping values to descriptive labels, dropping unnecessary columns, and adding additional attributes such as ingestion date.
The script retrieves the last ID value from the DynamoDB table to ensure unique IDs for the incoming records.
Each record in the DataFrame is transformed into an Item dictionary compatible with DynamoDB's expected attribute types.
The transformed Item is added to the batch_items list for later batch writing to DynamoDB.
The batch_items list is divided into smaller batches of 25 items each (maximum batch size for DynamoDB).
Each batch of items is written to DynamoDB using the batch_write_item method of the DynamoDB client.

Finally, the Lambda function returns a response indicating the successful write operation.
In summary, the consumer code processes the records received from the Kinesis Data Stream, transforms the data into a suitable format, and performs batch writing of the records into the DynamoDB table.

DynamoDB table :

In the Taxi Casestudy project, we utilize Amazon DynamoDB as a key component of our data pipeline. DynamoDB is a fully managed NoSQL database service provided by AWS, offering fast and scalable storage for structured data.

We employ DynamoDB as a staging area and NoSQL database for real-time streaming data. It acts as a reliable and efficient storage solution for the incoming taxi-related records, allowing us to process and analyze the data in real-time.

The DynamoDB table in our case study stores the taxi-related records, including information such as vendor details, trip dates, payment types, distances, fares, and more. It ensures the persistence and durability of our data, enabling us to capture and retain the necessary information for further analysis and reporting.

By leveraging DynamoDB, we benefit from its automatic scalability and high performance, allowing us to handle the influx of streaming data efficiently. The use of DynamoDB in our pipeline contributes to the real-time processing and analysis of taxi data, facilitating the generation of insights and supporting decision-making processes.

Analytical Dashboard:

- **DailyDWH lambda code**

This code copies data from DynamoDB at the end of the day depends on the insert data of the record (we have a new column we made for that) and save it into S3

Here's a summary of how it works:

- The script imports the necessary libraries and sets up the AWS clients for DynamoDB and S3 using the boto3 library.
- The lambda_handler function is the main entry point for the Lambda function.

The current date is obtained in the desired format using the datetime library.
DynamoDB and S3 clients are initialized.

- Records are retrieved from DynamoDB based on a filter expression for the current date.
- The S3 bucket and key where the file will be stored are defined.
- The order of fields in the records is set according to the desired order.
- The records are written to a CSV file, with the file being temporarily stored in '/tmp/green_data.csv'.
- The CSV file is uploaded to the specified S3 bucket and key.
- The Lambda function returns a response indicating the successful copying of data to the S3 file.

- **Redshift copying lambda :**

This code will take that S3 file into Redshift at the end of the day , it has trigger over the S3 that the DailyDWH script runs

- Here's a summary of how it works:

- The lambda_handler function is the main entry point for the Lambda function.

Inside a try-except block, the function attempts to connect to the Redshift cluster using the provided connection details.

- A cursor is created to execute SQL commands.

- The COPY command is constructed, specifying the table (greendwh), the source file in S3 (s3://greendwh/green_data.csv), the IAM role that grants access to the S3 bucket, and the format of the data (CSV).
- The COPY command is executed by calling cur.execute(copy_command). After executing the command, the changes are committed, and the connections to Redshift are closed.
- In summary, this script establishes a connection to the Redshift data warehouse and executes the COPY command to load data from the CSV file in the S3 bucket. This process facilitates the ingestion of the extracted and formatted data from DynamoDB into the Redshift data warehouse, enabling further analysis and reporting capabilities on the stored taxi data.

- **Redshift**

- Amazon Redshift serves as the data warehouse for storing and analyzing the taxi-related data. Here are a few key points about Redshift in our project:

- **Storage and Performance:** Redshift provides a scalable and high-performance storage solution for handling large volumes of taxi data. It leverages columnar storage and parallel processing to efficiently store and process the data, ensuring fast query performance.

- **Data Loading:** Redshift enables the seamless loading of data from various sources, such as CSV files stored in S3. We utilize the Redshift COPY command to efficiently load the transformed taxi data from the CSV file in the S3 bucket into the Redshift table.

- **Data Structure:** The data in Redshift is organized in tables, where each table corresponds to a specific aspect of the taxi data, such as vendor details, trip information, fares, and more. The structure of these tables is designed to optimize query performance and facilitate data analysis.

- **Integration with Analytics Tools:** Redshift integrates seamlessly with popular business intelligence (BI) tools, such as **Amazon QuickSight**. This integration allows us to create interactive dashboards, perform ad-hoc queries, and gain insights from the taxi data stored in Redshift.

- **Data Analysis and Reporting:** By leveraging the power of Redshift, we can perform complex analytical queries on the taxi data. We can uncover patterns, trends, and anomalies, enabling us to generate reports and visualizations that provide valuable insights for decision-making and business optimization.

- **Scalability and Cost-Effectiveness:** Redshift's ability to scale up or down based on demand ensures that we can handle increasing data volumes and query loads efficiently. This scalability, combined with Redshift's pay-as-you-go pricing model, allows us to manage costs effectively by only paying for the resources we need.

• Quicksight

In our Taxi Casestudy project, Amazon QuickSight serves as the business intelligence (BI) tool used to create analytical dashboards over the data stored in Amazon Redshift.

• Realtime Dashboard

we have implemented a real-time dashboard using Amazon Athena and Power BI. Here's a description of how it works:

Athena Table: We have created an Athena table that is connected to the DynamoDB table acting as a staging area for real-time streaming data. The Athena table allows us to query and analyze the data stored in DynamoDB in real-time. The data is partitioned and organized in a way that optimizes query performance and facilitates efficient data retrieval.

Real-Time Data Refresh: Power BI is integrated with Athena to create a real-time dashboard. Power BI connects directly to the Athena table, allowing us to visualize and analyze the real-time data in a visually appealing and interactive manner. As new data is ingested into DynamoDB and reflected in the Athena table, the Power BI dashboard automatically refreshes to display the most up-to-date information.

Dashboard Design and Visualization: Power BI provides a user-friendly interface that enables us to design and customize the real-time dashboard according to our specific requirements. We can create visually appealing charts, graphs, and other visualizations to represent key metrics and insights derived from the streaming data. The dashboard layout and design can be tailored to meet the needs of different stakeholders.

Automated Refresh: Power BI offers automated data refresh capabilities, allowing us to define the frequency at which the dashboard should refresh. In our case, the dashboard is configured to refresh every 30 seconds. This ensures that the dashboard always displays the latest data from the Athena table, providing real-time insights to users without manual intervention.

Interactive Analysis: Power BI empowers users to interact with the real-time dashboard and perform ad-hoc analysis. Users can apply filters, drill-down into specific data subsets, and explore different visualizations to gain deeper insights into the taxi data. This interactivity enhances the usability of the real-time dashboard and enables users to derive meaningful insights on the fly.

- **Orchestration:**

In our Taxi Casestudy project, we have implemented an orchestration mechanism to ensure the smooth and timely execution of our pipeline. Here's how it works:

- **SageMaker for Application Execution:** We have utilized Amazon SageMaker to run our application code. We have created a SageMaker notebook instance that serves as our execution environment. The notebook instance runs twice in succession, with a 30-second wait in between, to ensure that the application code is executed every 30 seconds. This interval allows for near real-time processing of data.

- **EventBridge for Scheduled Execution:** To automate the execution of the SageMaker notebook instance at the desired interval, we have leveraged Amazon

EventBridge. EventBridge is configured to trigger the SageMaker notebook instance every minute, ensuring that our application code runs consistently every 30 seconds. This orchestration ensures the timely processing of data and keeps the pipeline running smoothly.

- **SageMaker for Daily Data Warehouse Update:** In addition to the real-time processing, we have also implemented a separate SageMaker notebook instance to run the Daily Data Warehouse (DWH) script. This script is responsible for taking all the data accumulated in DynamoDB throughout the day and copying it into Redshift for analytical purposes. The SageMaker notebook instance executes the DWH script once, triggered by the EventBridge rule, at the end of the day, precisely at 11:59 PM.

- **EventBridge for Daily DWH Execution:** Amazon EventBridge is again utilized to schedule the execution of the Daily DWH script. The EventBridge rule is configured to trigger the SageMaker notebook instance running the DWH script at the designated time of 11:59 PM. This ensures that the data from DynamoDB is extracted and transferred to Redshift for analysis on a daily basis, providing a complete and up-to-date view of the taxi data.

- By combining SageMaker for application execution and the scheduling capabilities of EventBridge, we have established an orchestrated pipeline that runs our application code every 30 seconds for real-time processing. Additionally, the Daily DWH script runs at the end of each day to transfer the accumulated data from DynamoDB to Redshift for comprehensive analysis. This orchestration mechanism ensures the timely and automated execution of critical steps in our pipeline, enabling us to derive valuable insights from the taxi data in a systematic and efficient manner.

Conclusion :

In conclusion, our Taxi Casestudy project demonstrates the utilization of various AWS services and technologies to build an end-to-end data pipeline for analyzing taxi-related data. We have successfully leveraged the strengths of each component to create a comprehensive solution. Here are the key highlights:

AWS Cloud Advantage: We chose AWS as our cloud platform due to its scalability, reliability, and wide range of services that meet our project requirements. AWS provides the infrastructure and tools necessary to implement a robust and flexible data pipeline.

Data Management with AWS Services: We utilized multiple AWS services to manage our data effectively. The data pipeline starts with uploading source files to Amazon S3, where data is stored securely and can be easily accessed. DynamoDB acts as a staging area and NoSQL database for real-time streaming, while Amazon Redshift serves as our data warehouse for analytical purposes.

Real-Time and Analytical Streaming: Our pipeline supports both real-time and analytical streaming. We leveraged AWS Lambda and Kinesis Data Streams for real-time processing and data ingestion. Athena enables querying and analysis of real-time data, while Power BI and QuickSight provide interactive and visually appealing dashboards for real-time and analytical insights.

Orchestration and Automation: We implemented orchestration and automation using Amazon SageMaker and EventBridge. The SageMaker notebook instances execute our application code every 30 seconds, ensuring near real-time processing. EventBridge triggers the SageMaker instances at specified intervals and schedules the daily transfer of data from DynamoDB to Redshift.

Scalability and Flexibility: AWS services offer scalability, allowing our pipeline to handle growing data volumes efficiently. We can easily adapt and modify our pipeline as per changing business needs and data requirements.

Overall, our Taxi Casestudy project showcases the power of AWS cloud services in building a robust and scalable data pipeline. By leveraging AWS services for data

management, real-time streaming, analytics, orchestration, and automation, we have created a comprehensive solution that enables us to derive valuable insights from taxi-related data. The project highlights the importance of leveraging cloud technologies to optimize data processing, visualization, and decision-making processes.