# 

# **Capstone Project Submission Report**

### **Azure BFSI Data Pipeline for Customer Insights and Risk Analytics**

* **Name**: HARSHIDA SHAILY
* **Student ID**: 00127272
* **Course/Program**: Data Engineering Coforge Ltd.
* **Instructor Name**: Piyush Raj Katyayan

### 

### **3. Problem Definition and Objectives**

**Problem Definition**

In the Banking, Financial Services, and Insurance (BFSI) sector, managing and analyzing vast amounts of customer transaction data is crucial for risk assessment, fraud detection, and service optimization. Traditional data processing methods often fail to handle both historical and real-time data efficiently, leading to delayed insights, increased risks, and suboptimal customer experiences.

To address these challenges, the organization aims to develop a robust ETL pipeline using Azure services. This pipeline will enable seamless integration and processing of both batch and streaming data, ensuring real-time fraud detection, customer risk profiling, sentiment analysis, and demand forecasting. By leveraging advanced analytics, the company seeks to enhance customer satisfaction, mitigate financialrisks, and optimize service offerings.

**Objectives**:

**1 . End-to-End ETL Pipeline:**

The project aims to build a scalable ETL pipeline in Azure to handle both batch processing of historical data and real-time streaming of transaction data for a BFSI company.

**2. Risk Monitoring & Forecasting:**

The pipeline will analyze customer transactions and risk profiles to detect fraud, assess credit risk, and forecast service demand, improving decision-making and financial security.

**3. Multi-Source Data Integration:**

The solution will process diverse datasets, including historical transactions, customer profiles, service usage, real-time transactions, feedback, and service availability data, ensuring comprehensive analytics.

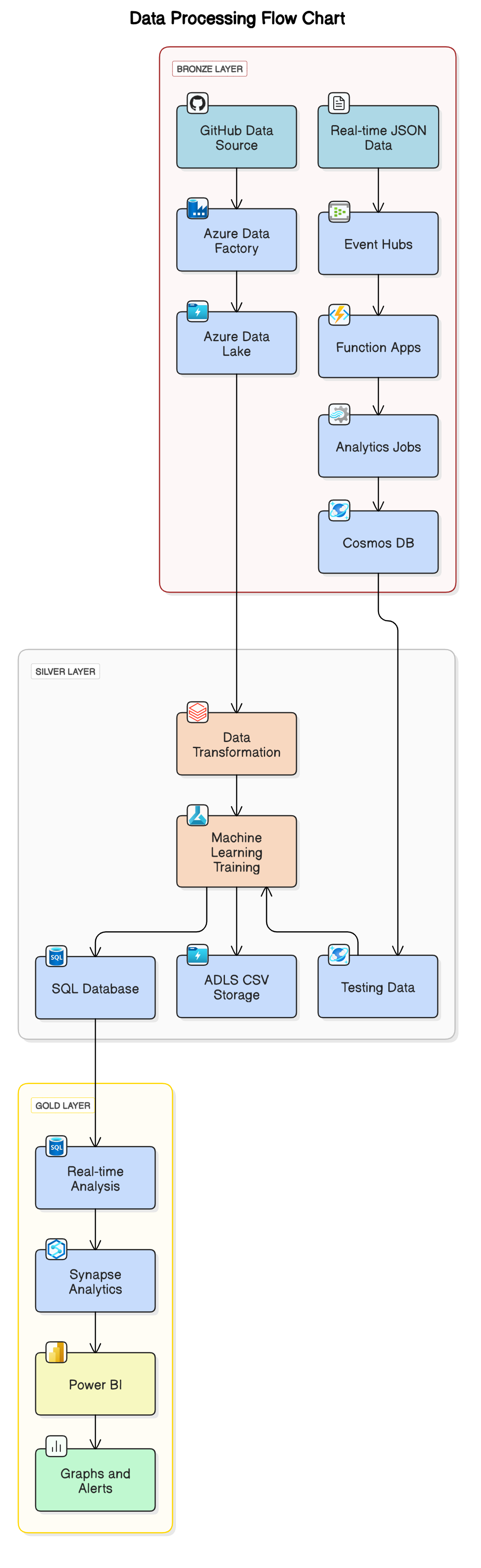
4. **Customer-Centric Insights**:

By leveraging advanced analytics, the pipeline will enhance customer satisfaction through sentiment analysis, personalized services, and proactive issue resolution.

**5. Azure-Based Scalable Architecture:**

The solution will utilize Azure services to ensure efficient data ingestion, transformation, storage, and real-time processing, enabling seamless operations and high availability.

**Diagram: High-Level Overview of Problem Context**

**

### **4. Data Collection, Exploratory Data Analysis (EDA), and Preprocessing**

**Data Collection** :

1. **HISTORICAL DATA**

**(CSV Files)**

**Service Usage Data** – Records customer interactions with banking services to analyze service demand and optimize offerings.

**Customer Feedback Data** – Captures customer sentiments and feedback to improve service quality and customer satisfaction.

**Service Availability Data** – Tracks service uptime and performance to ensure reliability and predict potential downtime.

**Customer Profiles Data** – Stores customer demographic and financial information for segmentation and risk assessment.

**Historical Transactions Data** – Contains past financial transactions, helping in fraud detection, customer spending analysis, and trend forecasting.

**Data Ingestion**:

* **Source**: GitHub (hosting raw CSV files).
* **Pipeline**: Azure Data Factory (ADF) extracts and loads data into ADLS.

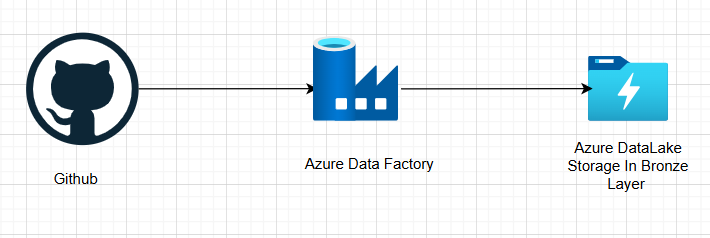
**Bronze Layer (Raw Data Storage)**:

* Data is stored in **ADLS** under a **Bronze** container to maintain original integrity and support future reprocessing needs.

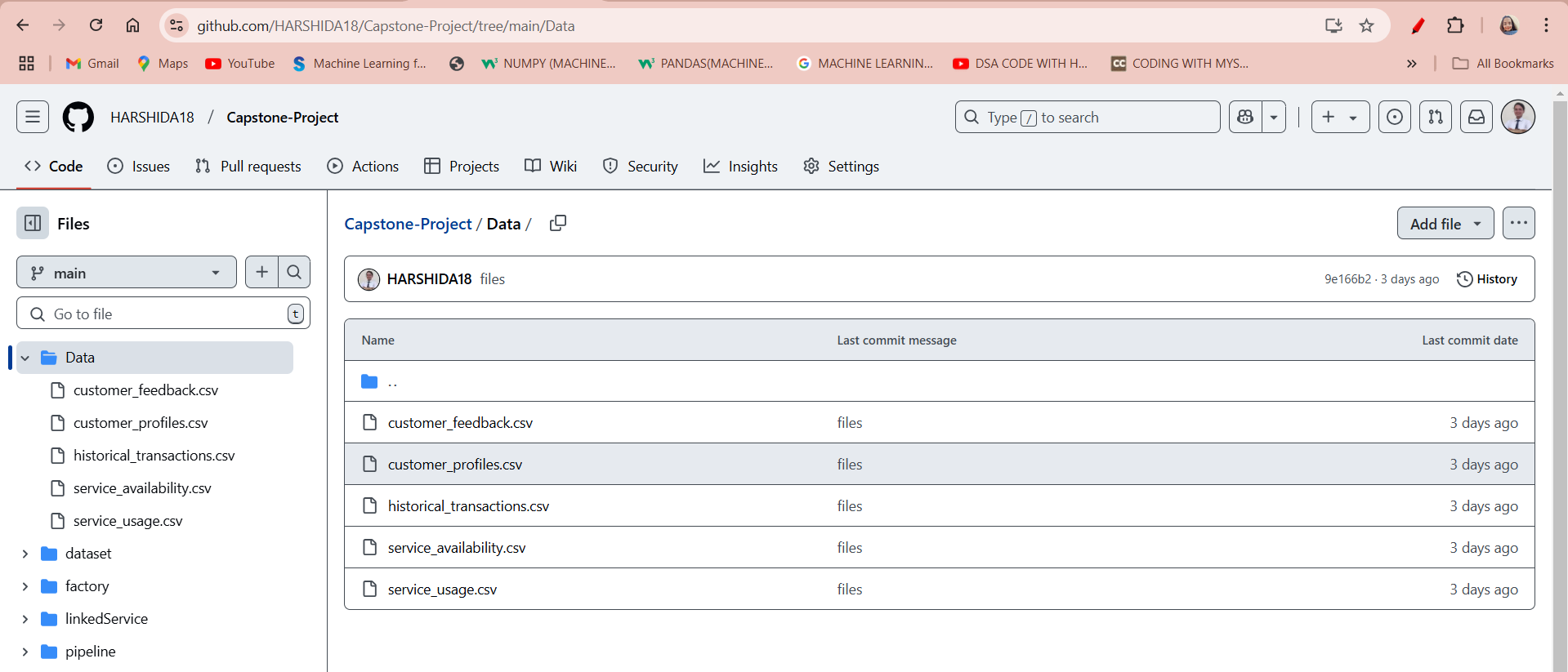
**Consumption**:

* Data is used for **fraud detection, sentiment analysis, service forecasting, and customer risk profiling** through **Power BI dashboards, ML models, and real-time monitoring systems**.

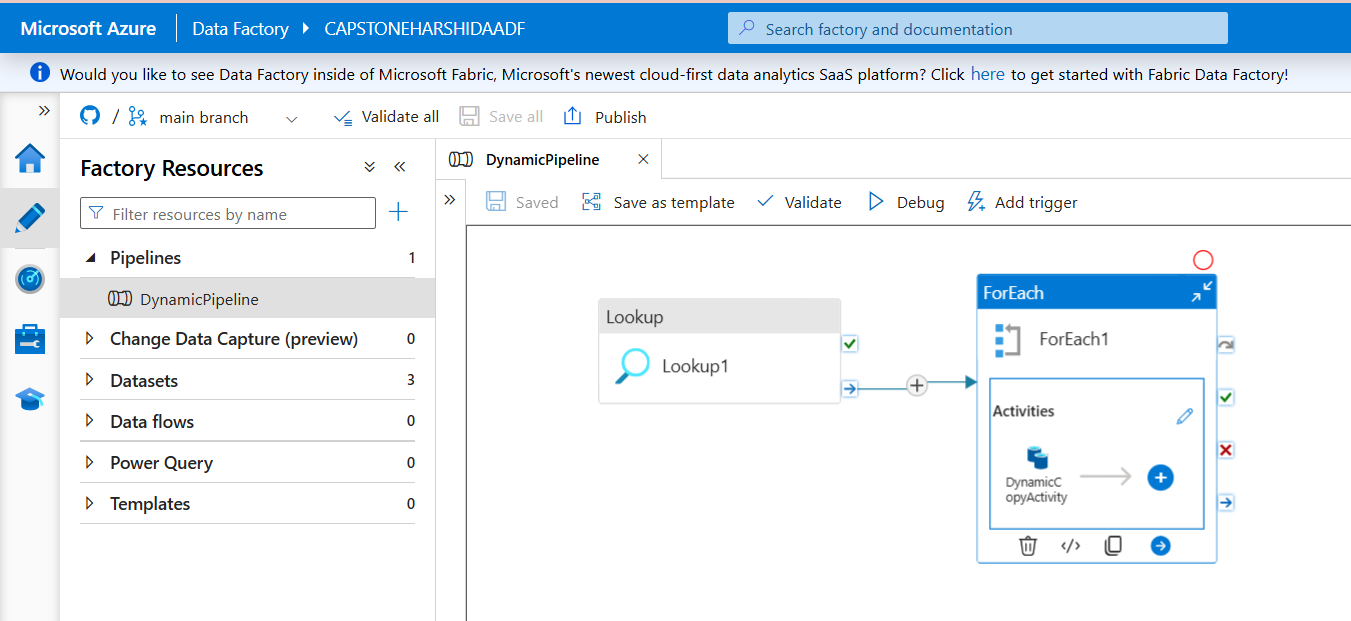
Architecture =



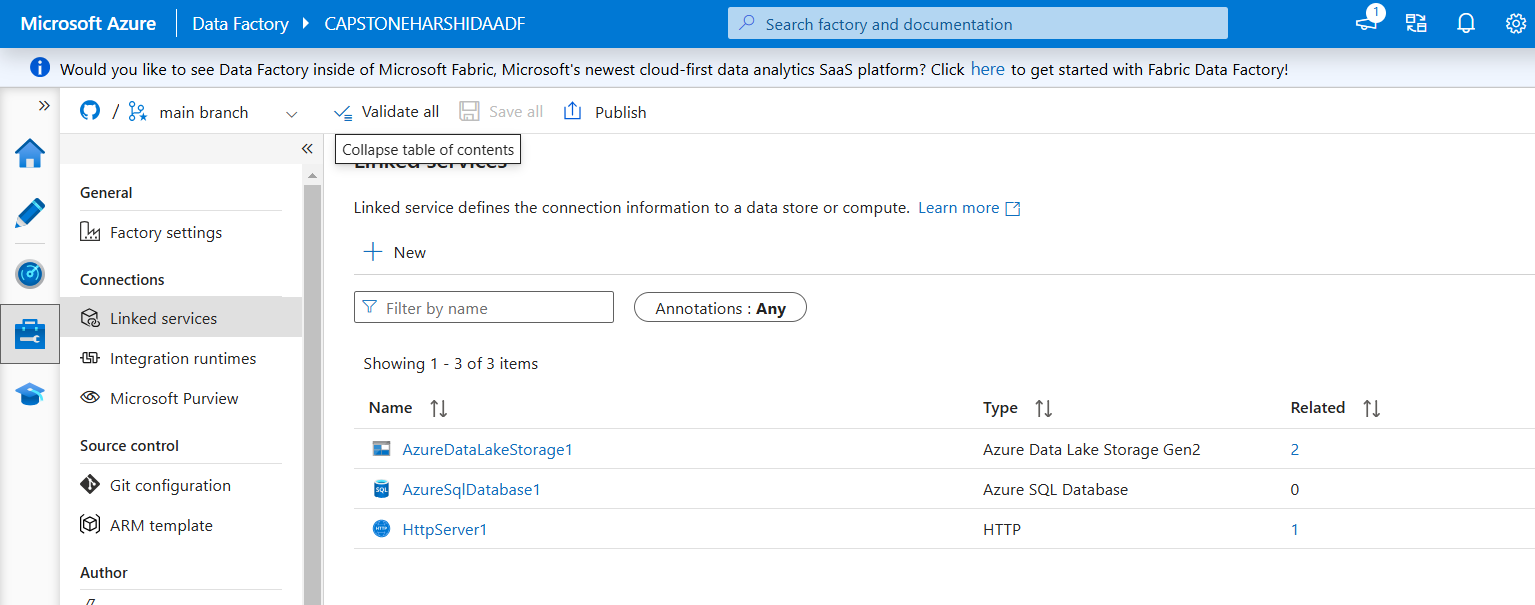
Github Data =



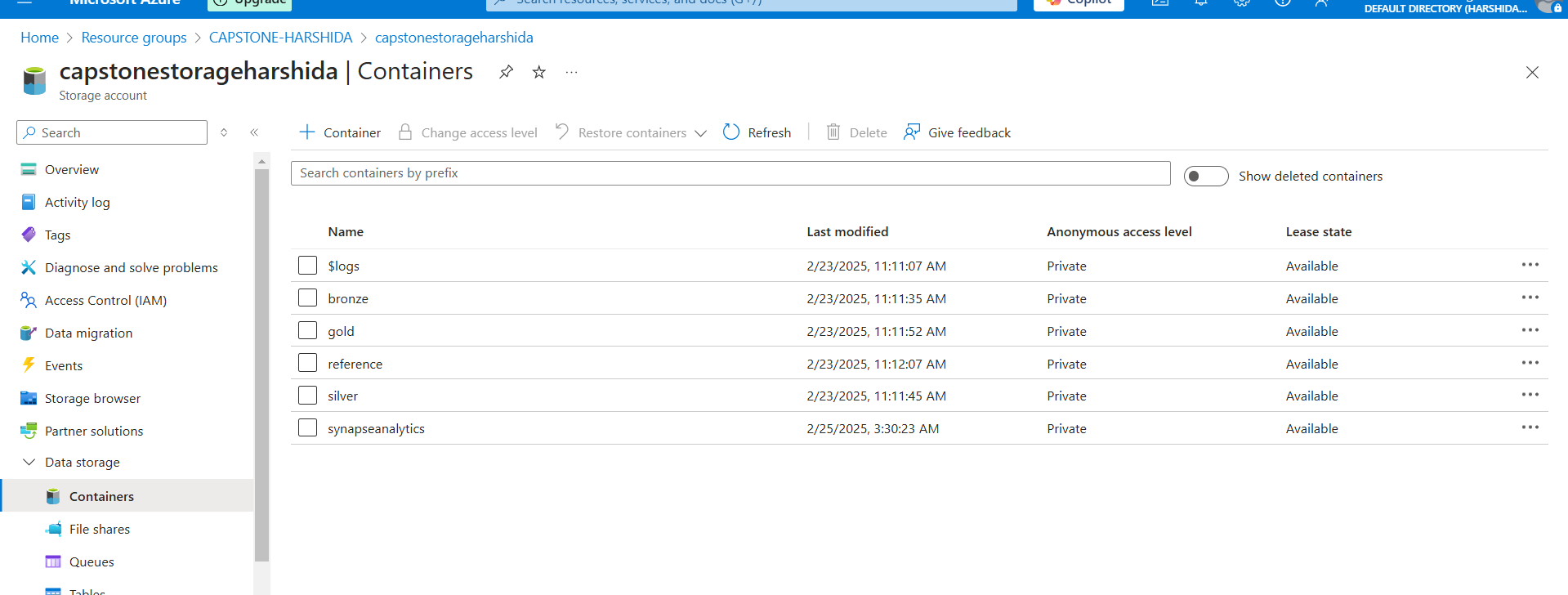
Pipeline Of Azure DataFactory Data Ingestion

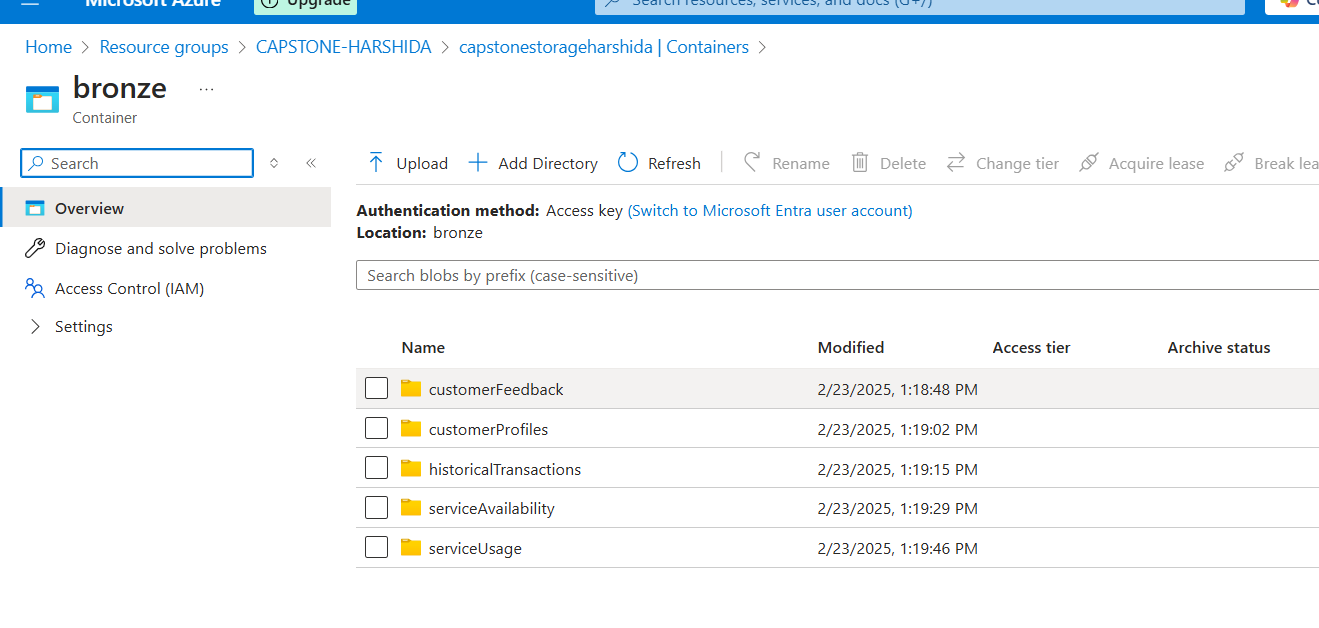


Linked Services



Azure DataLake Gen2 Data in bronze layer file =





1. REAL-TIME DATA STREAMING

ARCHITECTURE =

* The data through function app’s code connected with event hub .
* The event hub generates the events instance per defined interval .
* The instance through spark analytics jobs as an input is the output for cosmosDB

Item instance creation .

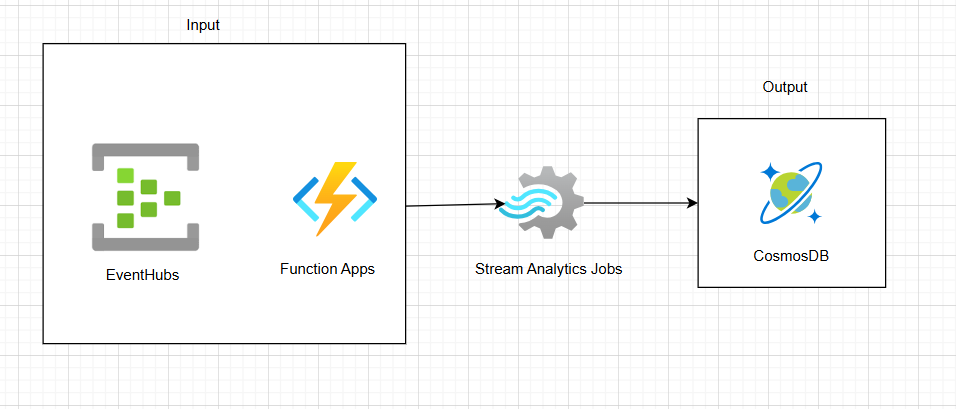


Fig : Real-time data entering into the CosmosDB instance

EventHubs =

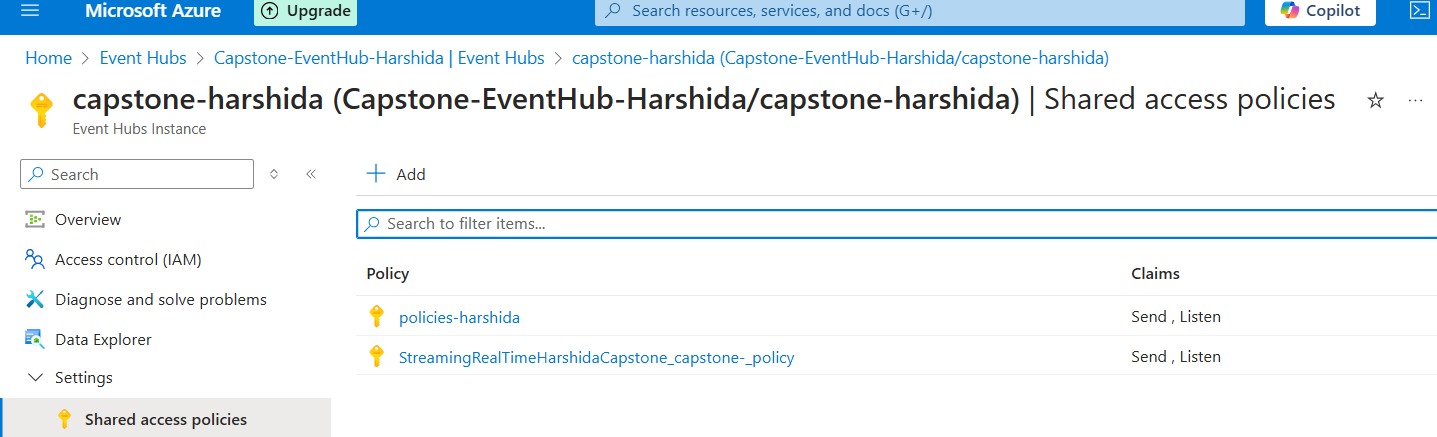
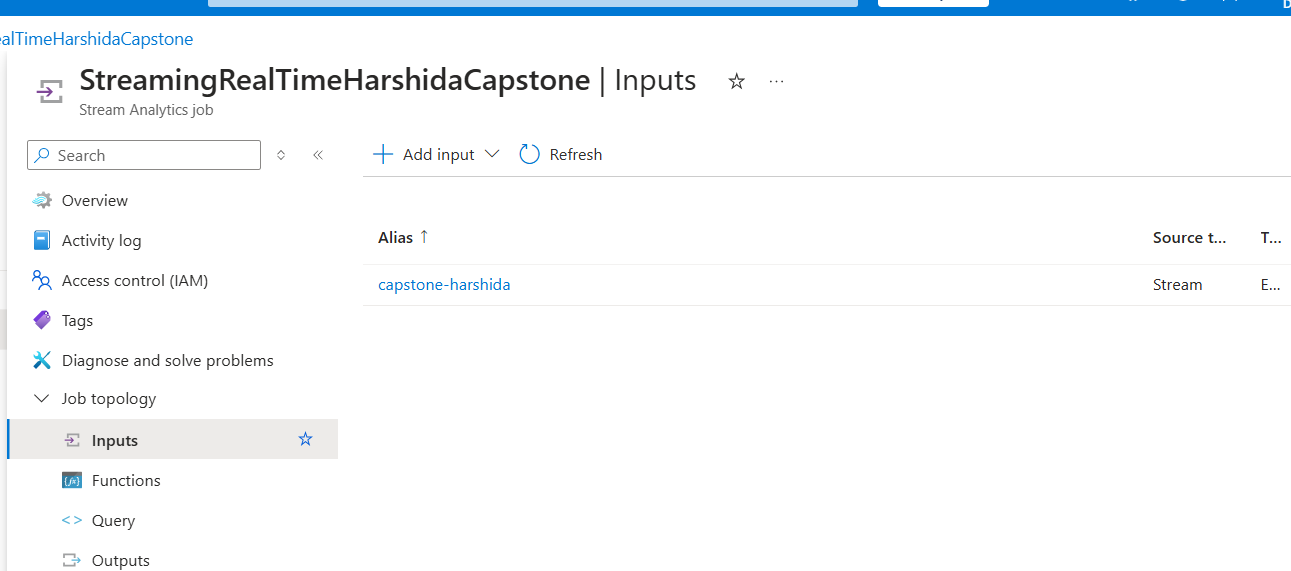


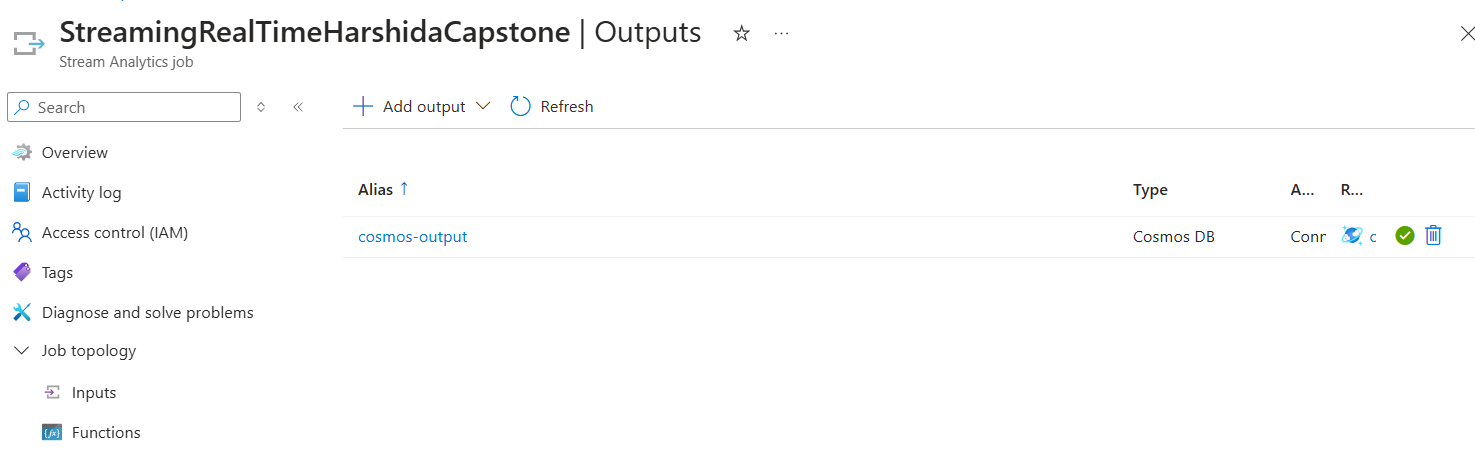
Fig : EventHub policies definition

Stream Analytics Jobs =

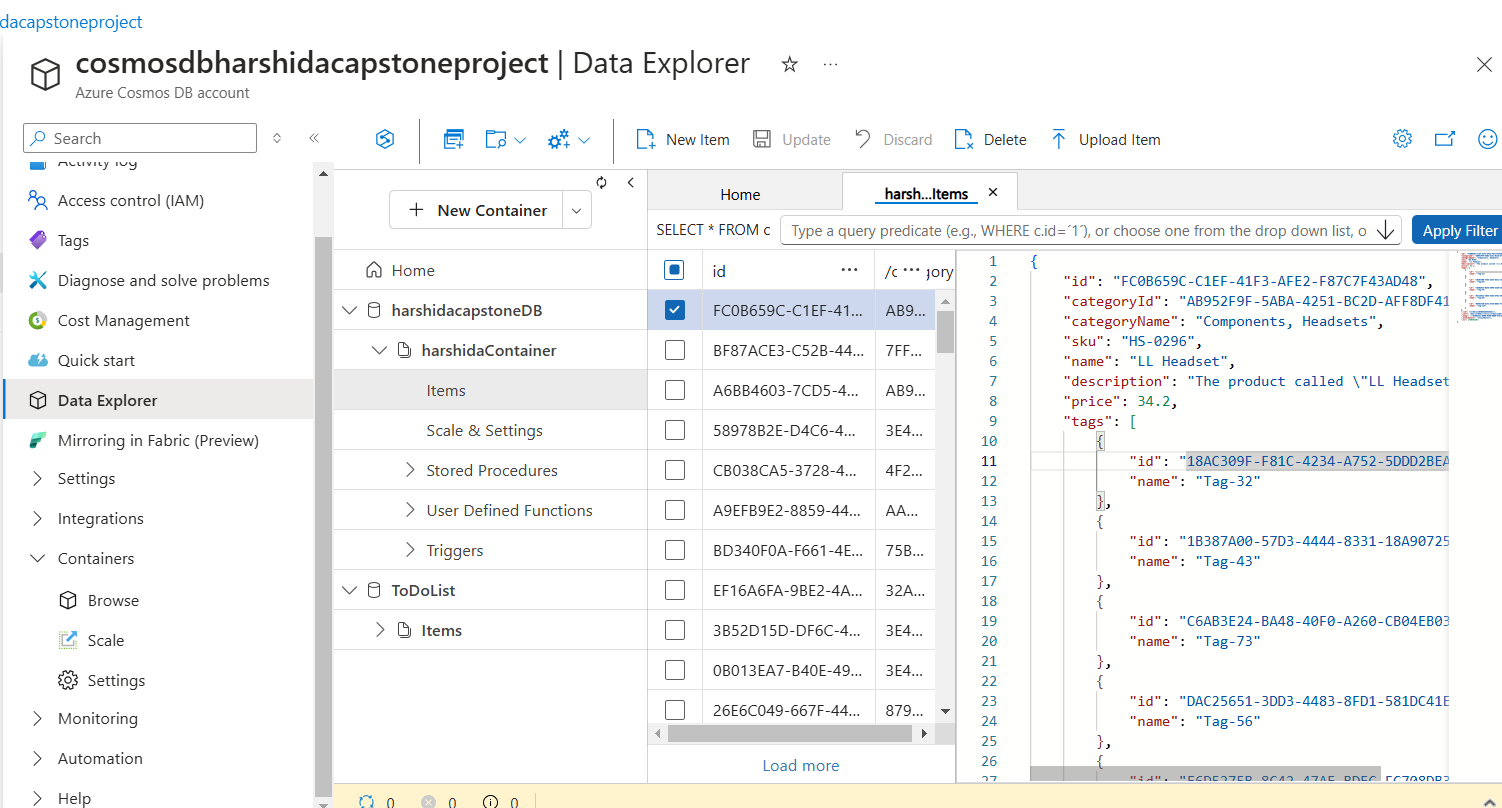
Input



Output



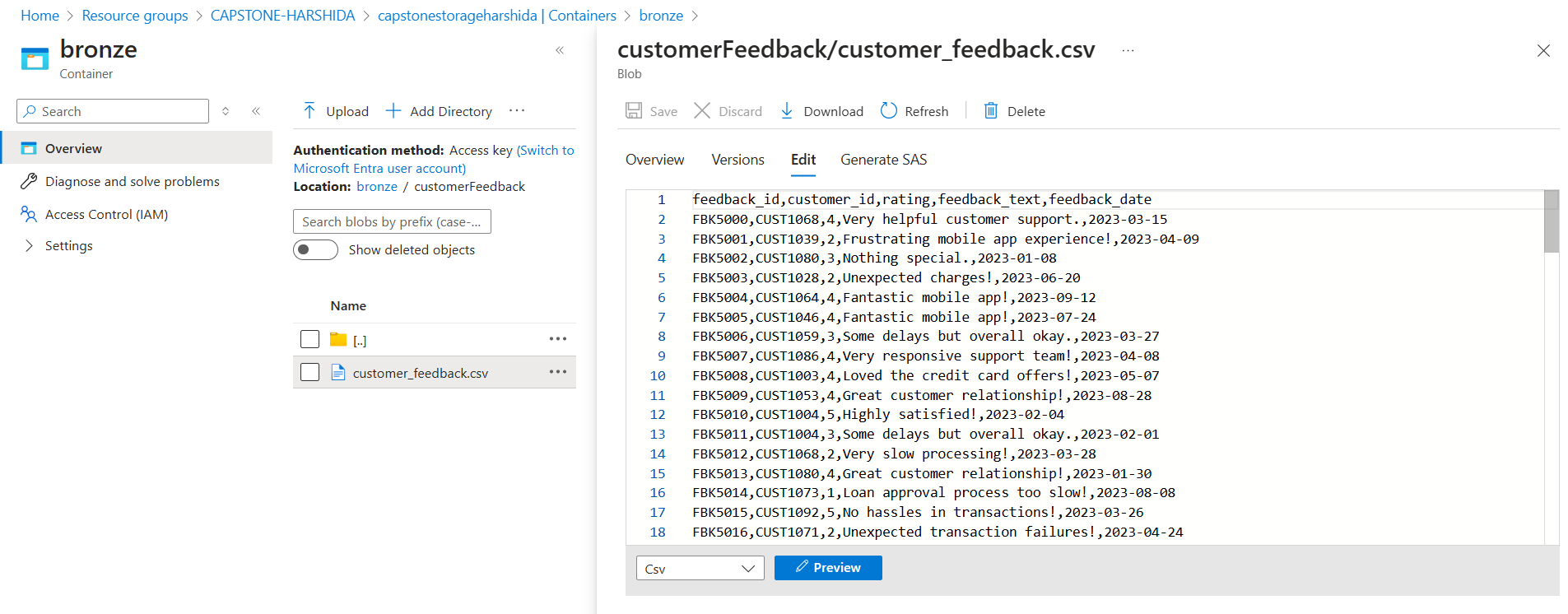
CosmosDB

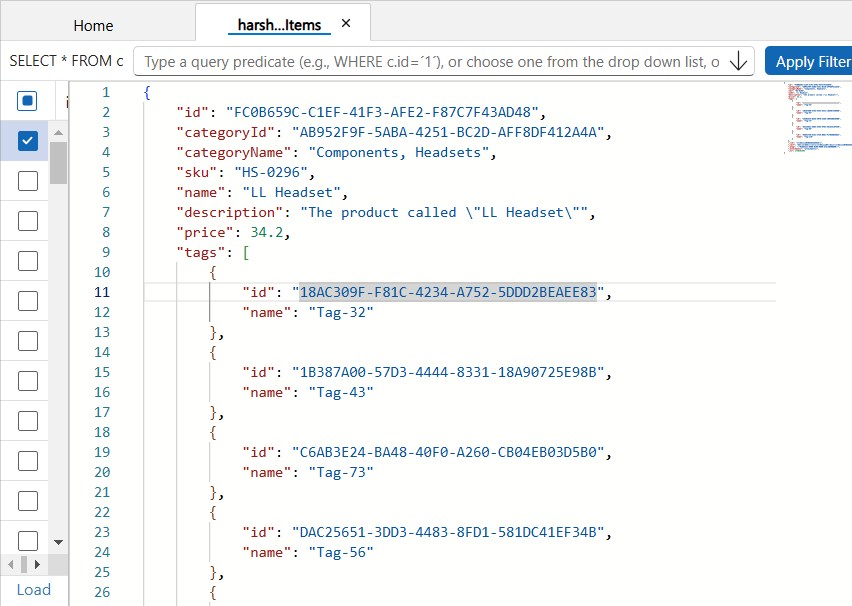


**Table: Data Description**

* The data in the bronze layer are stored in form of the container per data file .
* The data files of the historical data contains the csv format files .
* The real-time data is in json format which when stored in the cosmosDB is

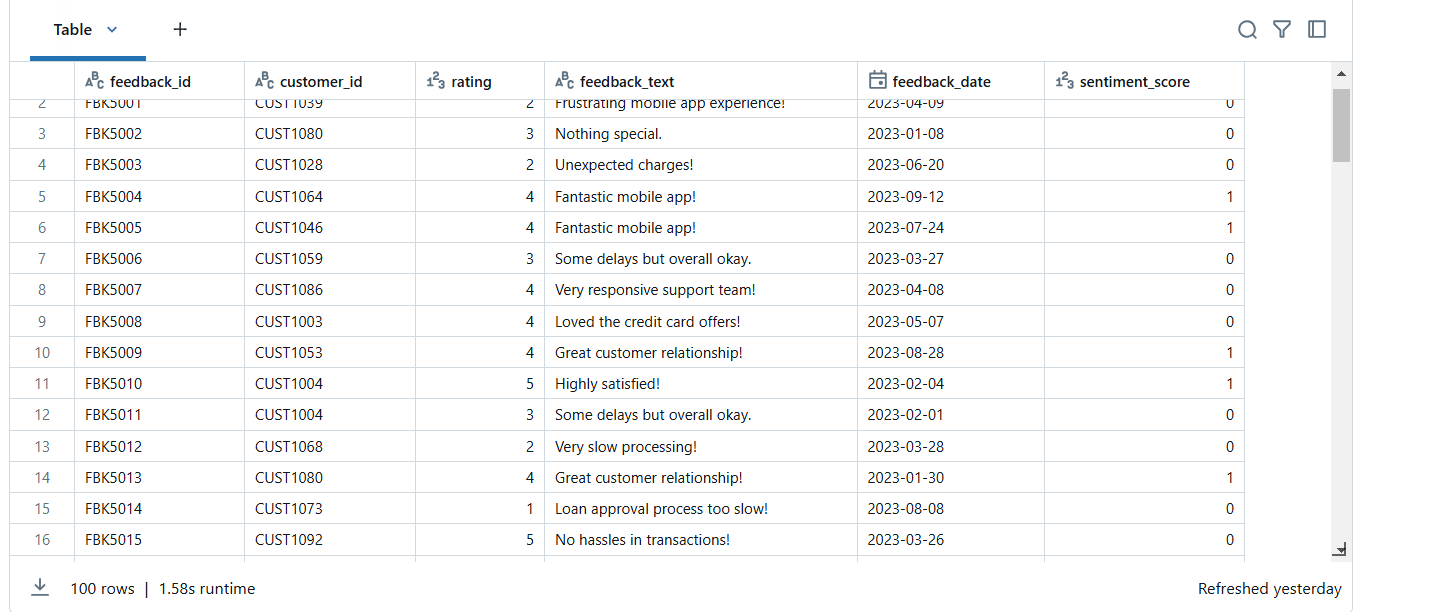
stored as the item instance of the container into the database .

****

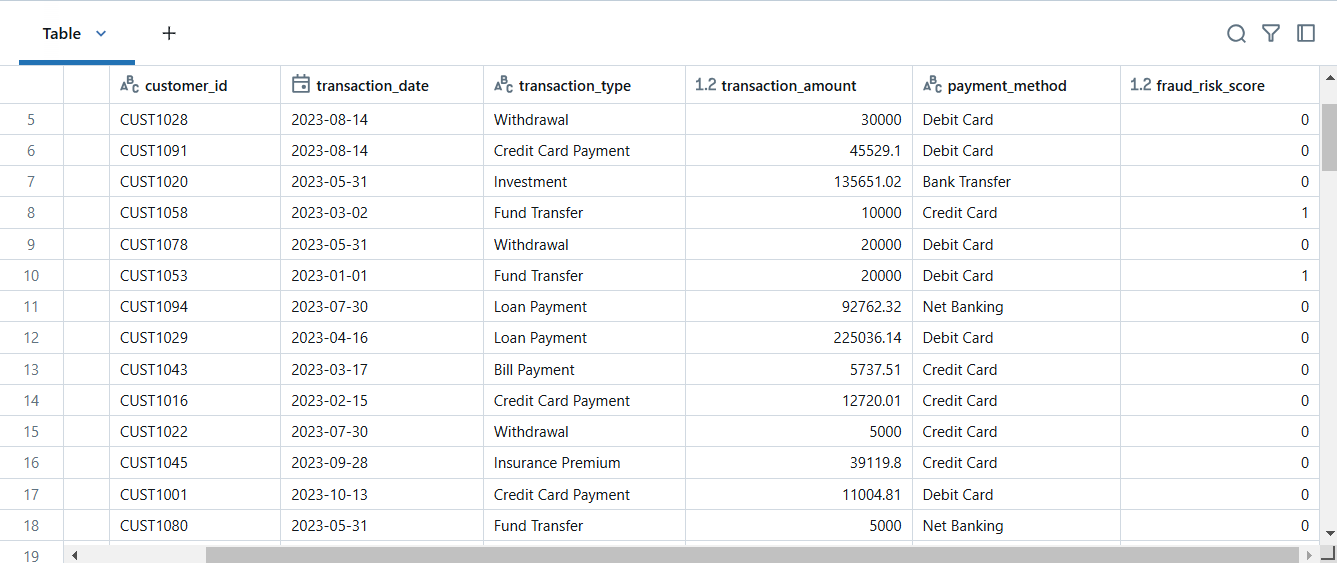
****

**Exploratory Data Analysis (EDA)**

1. **Transaction Trends & Seasonality** – Analyzed historical transactions to identify peak transaction periods, spending patterns, and seasonal variations. This helps in service optimization and fraud detection.
2. **Customer Segmentation Insights** – Grouped customers based on demographics, transaction behavior, and risk profiles to personalize financial products and detect high-risk customers.



1. **Sentiment Analysis on Feedback** – Processed customer feedback data using NLP to classify sentiments (positive, neutral, negative), helping improve banking services.
2. **Correlation Analysis** – Used a heatmap to identify relationships between variables, such as the impact of service availability on transaction frequency and customer satisfaction.



1. **Anomaly Detection in Transactions** – Identified suspicious transaction patterns using box plots and statistical methods, assisting in fraud detection.

**EDA Visualization Diagram**

**The following visualizations are commonly used:**

* **Histograms –** To analyze the distribution of transaction amounts.
* **Correlation Heatmap –** To understand relationships between customer attributes and transaction behavior.
* **Bar Graphs –** To visualize service usage across different customer segments.
* **Box Plots –** To detect transaction outliers indicating potential fraud.
* **Time Series Plots –** To track transaction volume trends over time.

**Data Preprocessing**

1. **Handling Missing Values –** Imputed missing values in customer profiles and transaction data using mean/mode/median strategies to ensure data completeness.

**CODE =**

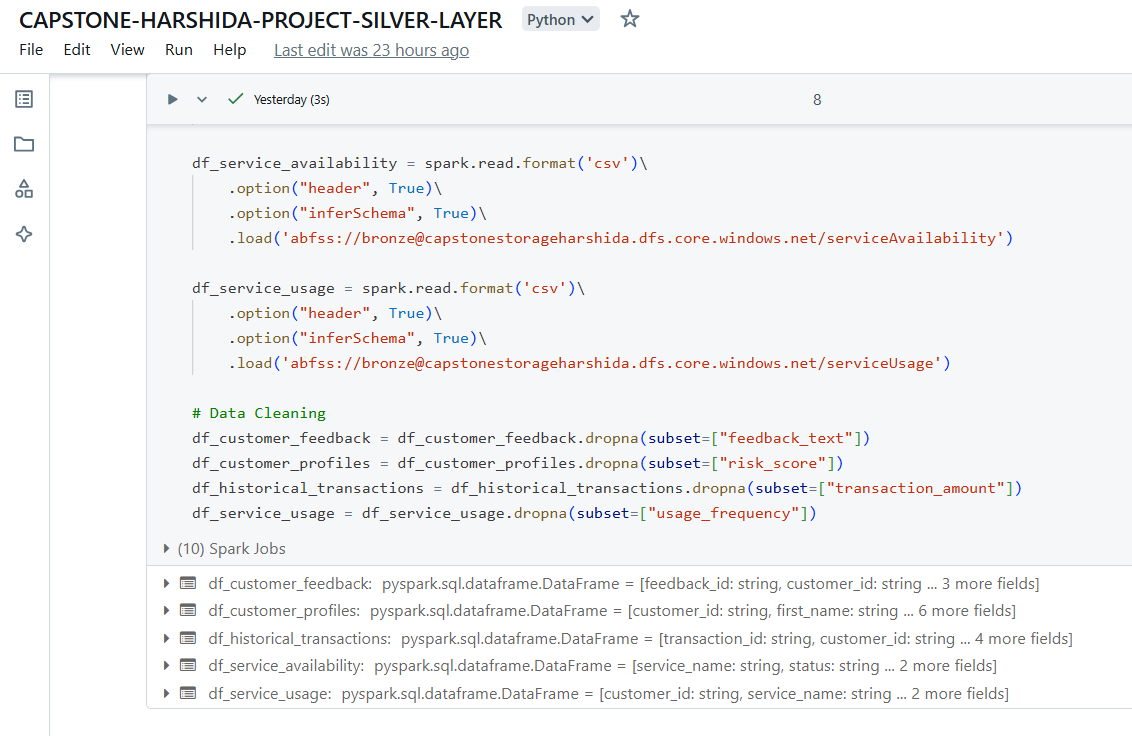
df\_customer\_feedback = df\_customer\_feedback.dropna(subset=["feedback\_text"])

df\_customer\_profiles = df\_customer\_profiles.dropna(subset=["risk\_score"])

df\_historical\_transactions = df\_historical\_transactions.dropna(subset=["transaction\_amount"])

df\_service\_usage = df\_service\_usage.dropna(subset=["usage\_frequency"])

**OUTPUT =**



1. **Data Normalization –** Scaled numerical features (transaction amounts, service usage) using Min-Max scaling to ensure uniformity across datasets.
2. **Encoding Categorical Variables –** Converted categorical fields like service type, customer risk level, and transaction mode into numerical representations using one-hot encoding. ( Sentiment Analysis )

CODE =

from pyspark.sql.functions import udf, when, col

from pyspark.sql.types import FloatType

from textblob import TextBlob

# Sentiment Analysis using TextBlob

def get\_sentiment(text):

    if text is None:  # Handle None values

        return 0.0

    return TextBlob(text).sentiment.polarity

# Register UDF

sentiment\_udf = udf(get\_sentiment, FloatType())

# Apply sentiment analysis

df\_customer\_feedback = df\_customer\_feedback.withColumn("sentiment\_score", sentiment\_udf(col("feedback\_text")))

# Convert sentiment\_score into binary labels

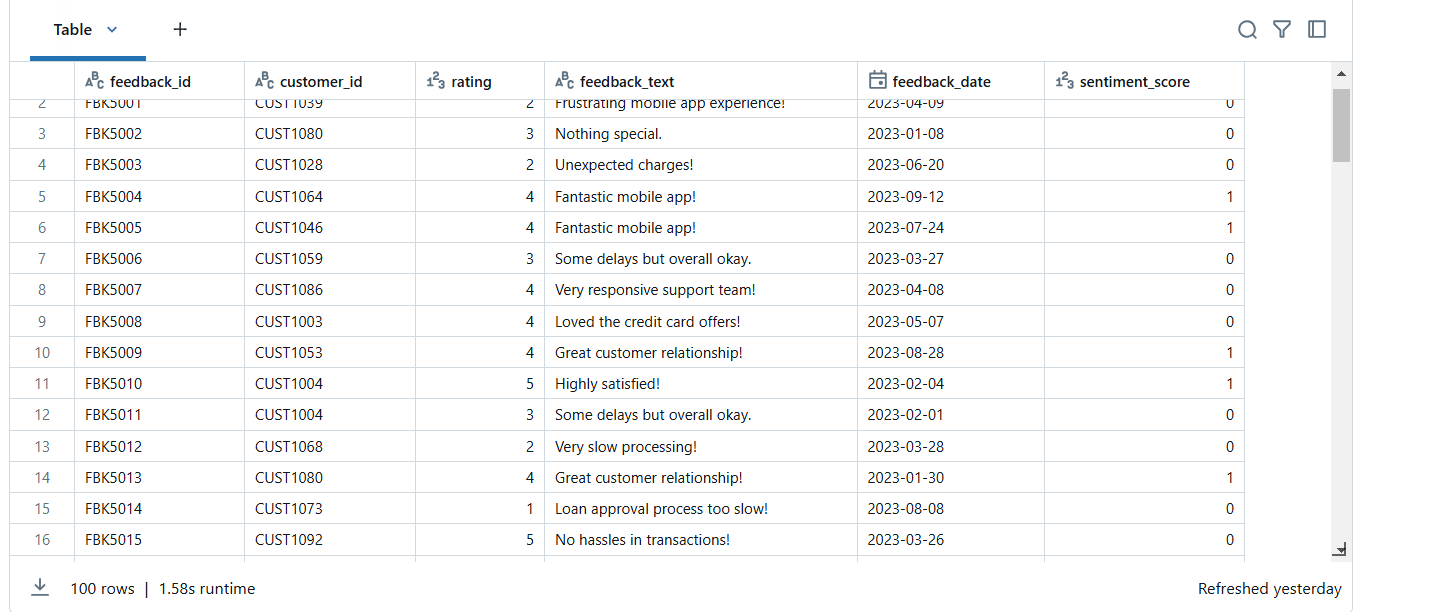
df\_customer\_feedback = df\_customer\_feedback.withColumn(

    "sentiment\_score", when(col("sentiment\_score") >= 0.5, 1).otherwise(0)

)

df\_customer\_feedback.display(

OUTPUT =



1. **Feature Engineering –** Created new features like transaction frequency, service usage score, and customer loyalty index to improve predictive analytics. (Fault Tolerance )

**CODE =**

from pyspark.sql.functions import when

# Fraud Risk Score based on transaction amount and type

df\_historical\_transactions = df\_historical\_transactions.withColumn(

    "fraud\_risk\_score",

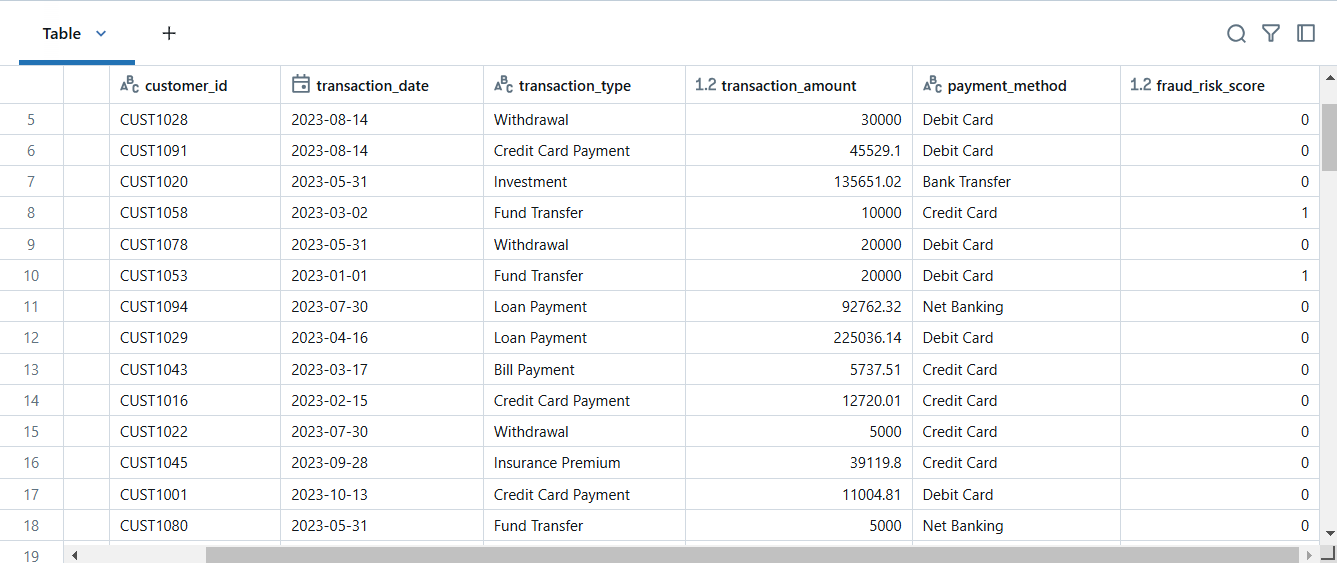
    when((col("transaction\_amount") > 5000) & (col("transaction\_type") == "Fund Transfer"), 1.0)

    .otherwise(0.0)

)

df\_historical\_transactions.display()

**OUTPUT =**



1. **Outlier Detection & Removal –** Used IQR and Z-score methods to filter out extreme transaction values that may indicate fraud or errors. ( Risk score alerts )

**CODE =**

# Categorize risk score into levels

df\_customer\_profiles = df\_customer\_profiles.withColumn(

    "risk\_level",

    when(col("risk\_score") < 0.30, "low")

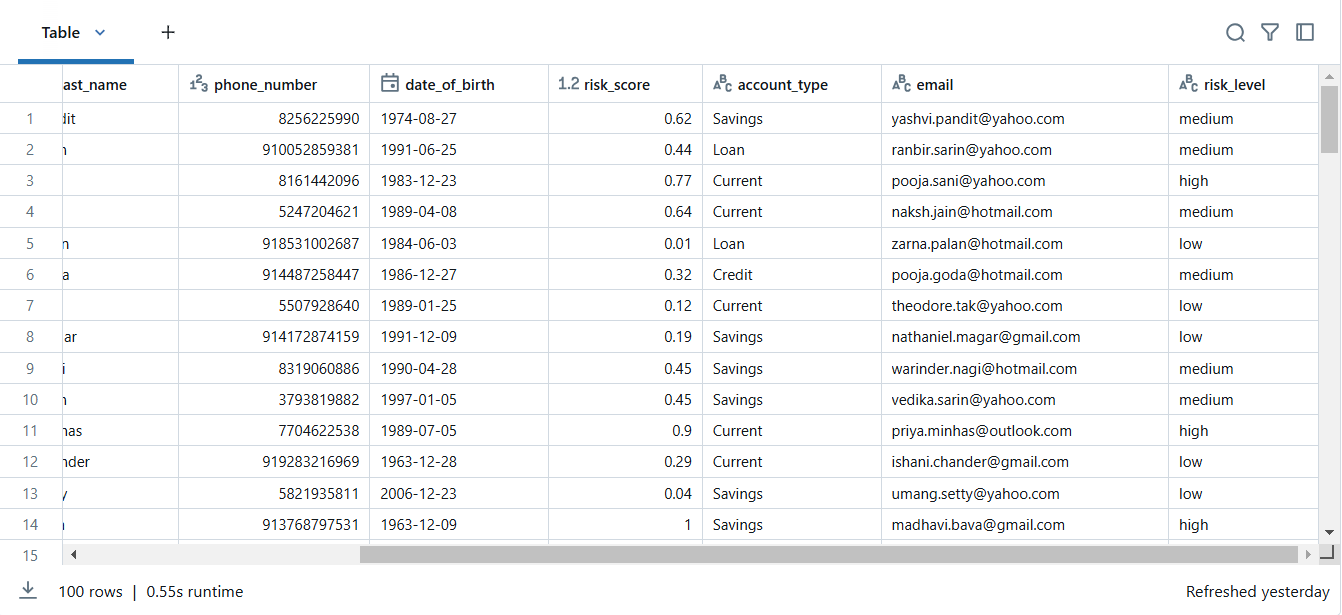
    .when((col("risk\_score") >= 0.30) & (col("risk\_score") < 0.70), "medium")

    .otherwise("high")

)

df\_customer\_profiles.display()

**OUTPUT =**



1. **Usage Traffic Detection =**

**CODE =**

# Usage Traffic Score based on usage frequency

df\_service\_usage = df\_service\_usage.withColumn(

    "usage\_traffic\_score",

    when(col("usage\_frequency") > 50, "high")

    .when((col("usage\_frequency") >= 20) & (col("usage\_frequency") <= 50), "medium")

    .otherwise("low")

)

df\_service\_usage.display()

**OUTPUT =**

****

1. **XGBOOST –** For training the model for sentiment analysis

**CODE =**

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.classification import GBTClassifier

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Prepare features

assembler = VectorAssembler(inputCols=["rating"], outputCol="features")

df\_sentiment = assembler.transform(df\_customer\_feedback)

# Train-test split

train, test = df\_sentiment.randomSplit([0.8, 0.2], seed=42)

# Train GBTClassifier model

gbt = GBTClassifier(labelCol="sentiment\_score", featuresCol="features")

model = gbt.fit(train)

# Evaluate model

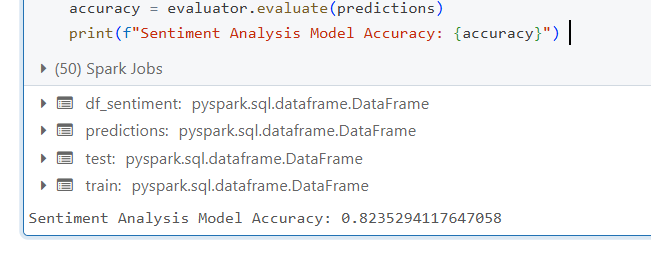
predictions = model.transform(test)

evaluator = MulticlassClassificationEvaluator(labelCol="sentiment\_score", metricName="accuracy")

accuracy = evaluator.evaluate(predictions)

print(f"Sentiment Analysis Model Accuracy: {accuracy}")

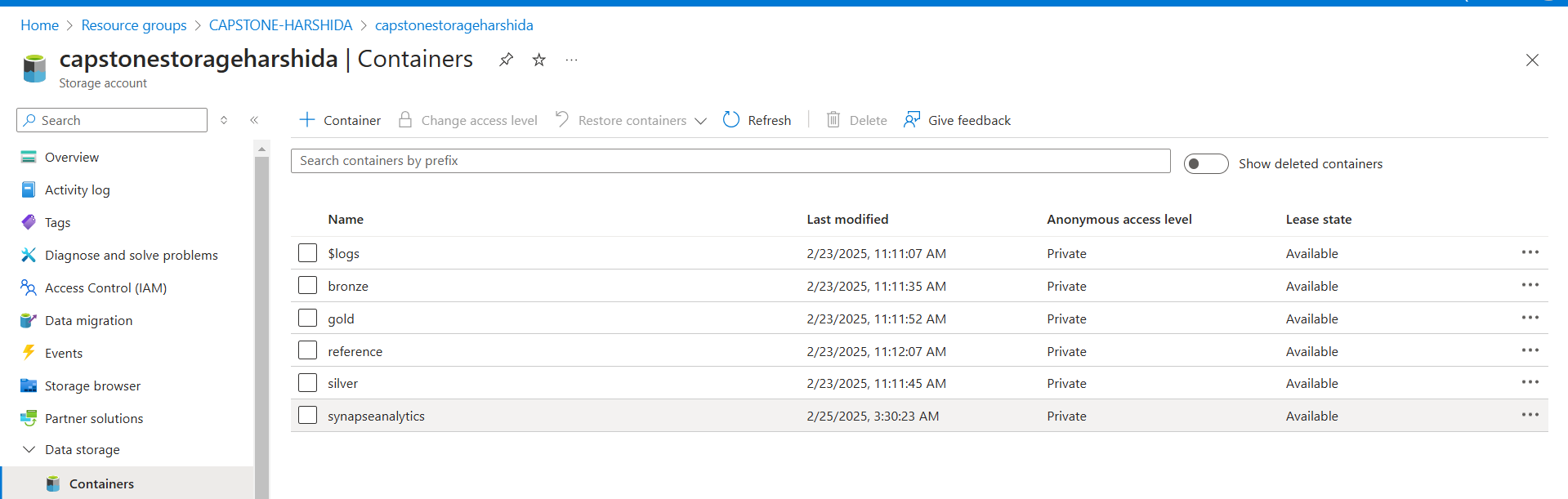
**OUTPUT =**



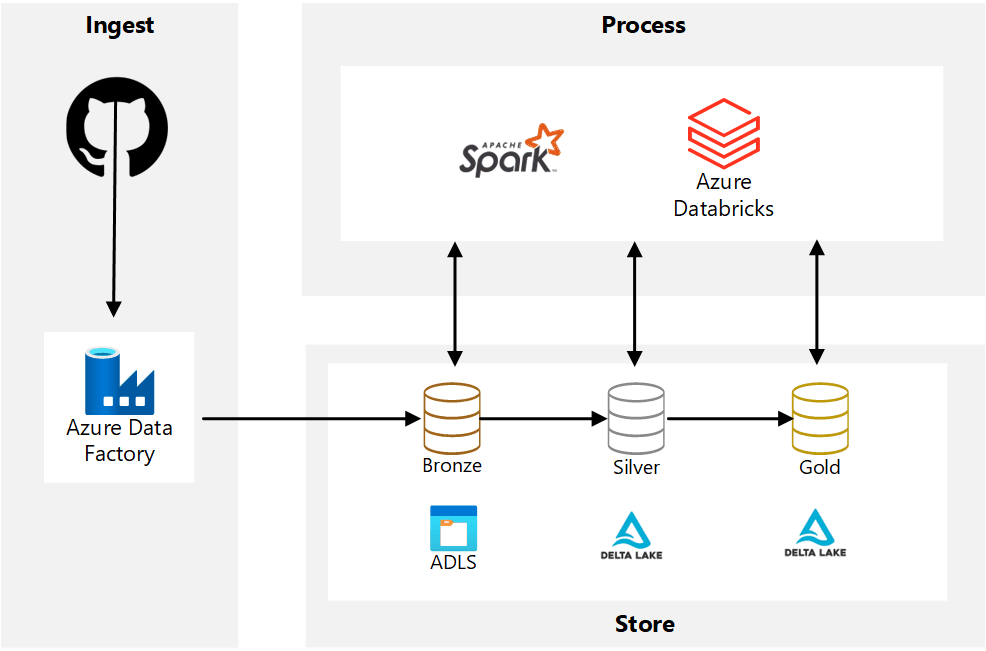
### **5. Data Storage and Optimization**

**Data Storage Solution**

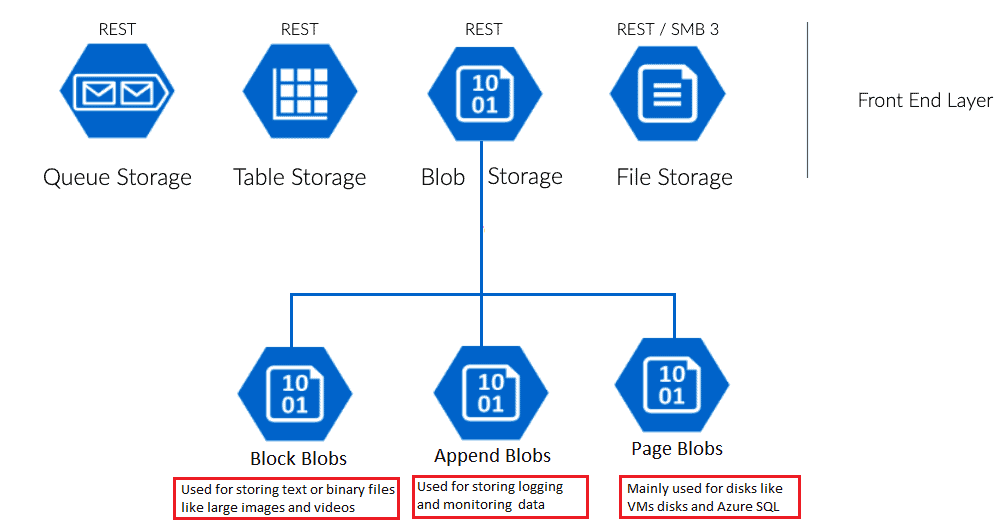
1. **Azure Data Lake Storage (ADLS) for Scalable Storage** **–** ADLS is used to store raw, processed, and analytical data across Bronze, Silver, and Gold layers, ensuring efficient data management and retrieval.



1. **Medallion Architecture for Data Organization –** The data is structured in a three-tier architecture (Bronze for raw data, Silver for cleansed data, Gold for analytics-ready data) to enhance data governance and usability.



1. **Delta Lake for Transactional Data –** Delta Lake format is used for real-time transaction data, enabling ACID transactions, versioning, and time-travel queries for consistency and reliability.
2. **Azure Synapse Analytics for Query Optimization –** ADLS data is integrated with Synapse Analytics to enable fast SQL-based queries, supporting business intelligence and machine learning models.
3. **Blob Storage for Backup and Archival –** Older data that is not frequently accessed is moved to Azure Blob Storage, reducing costs while maintaining accessibility for compliance purposes.



**Data Storage Architecture Diagram**

**The architecture follows a layered approach:**

* **Data Ingestion:** Data is extracted from GitHub & Real-time sources via ADF & Event Hub.
* **Storage Layers:**
  + **Bronze** (Raw Data) – Stores unprocessed CSV & real-time data.
  + **Silver** (Cleansed Data) – Processed using Databricks and stored in ADLS.
  + **Gold** (Aggregated Data) – Ready for reporting, ML, and BI tools.
* **Consumption Layer**: Data is accessed by Power BI, ML models, and APIs for insights and decision-making.

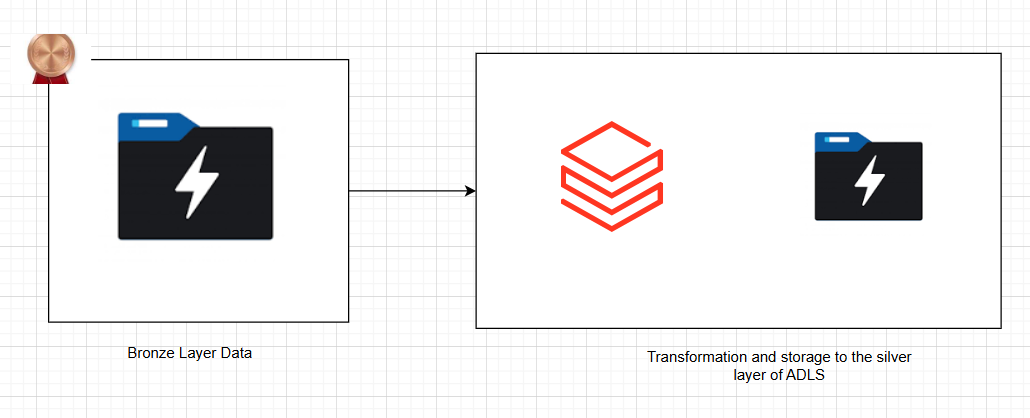


Fig : Historical Data Storage

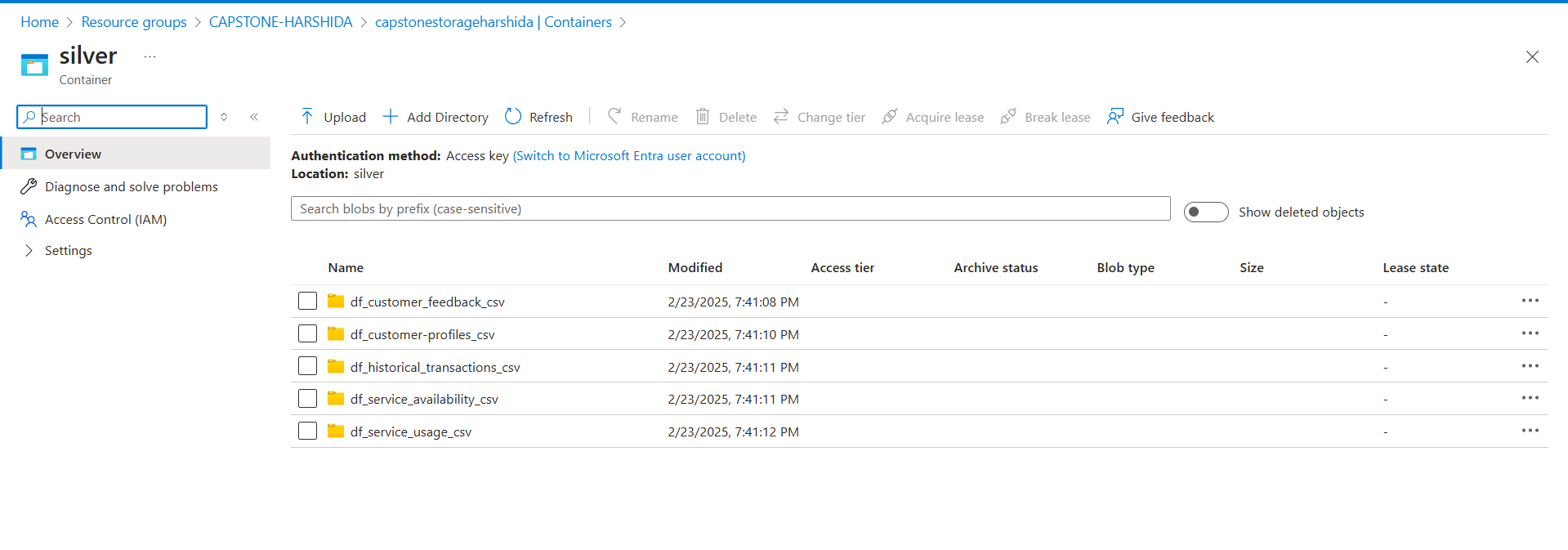


Fig : Silver layer data storage after transformation from Databricks



Fig : Real-time data storage to CosmosDB and then transformed storage to gold layer

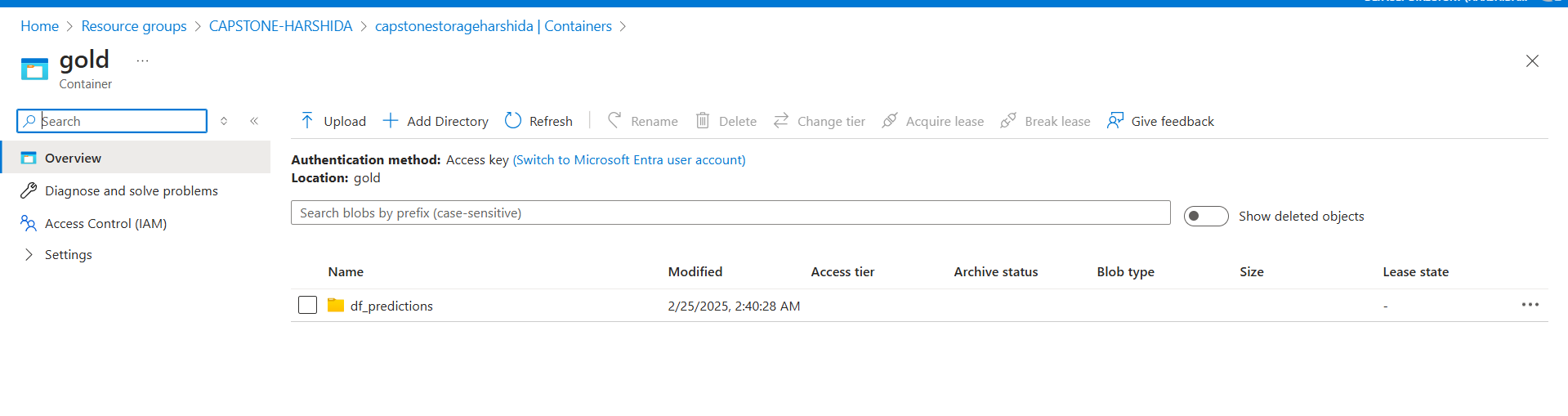


Fig : Gold layer transfomed storage in ADLS Gen2

**Partitioning & Indexing Strategies**

1. **Time-Based Partitioning –** Data is partitioned by date (YYYY/MM/DD) for efficient query performance in historical and real-time transaction analysis.
2. **Customer ID-Based Partitioning –** Large datasets, like customer profiles and transactions, are partitioned by customer ID to speed up lookups and segmentation analysis.
3. **Delta Lake Indexing –** Z-Ordering on transaction amounts and timestamps optimizes queries that filter by transaction values, reducing scan times.
4. **Columnar Storage for Faster Queries –** Parquet format is used for Silver and Gold layers, enabling efficient columnar storage that reduces I/O operations and enhances query speeds.
5. **Materialized Views for Aggregated Data –** Frequently accessed reports (e.g., total transactions per month, fraud detection alerts) use materialized views in Synapse Analytics, avoiding repeated complex calculations.

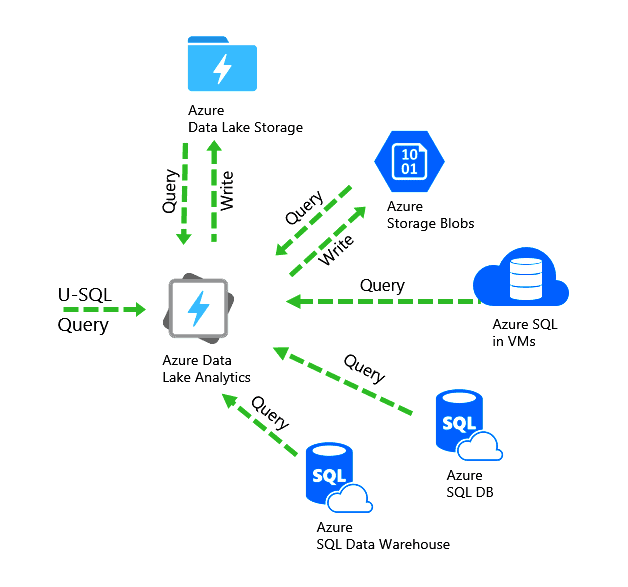


Fig : Partitioning of Azure SQL Database commands

### **6. Real-Time Data Processing and Streaming**

**Real-Time Data Processing**

1. **Azure Event Hubs for Data Ingestion** – Real-time transaction data is streamed into Azure Event Hubs, acting as a highly scalable event ingestion service.

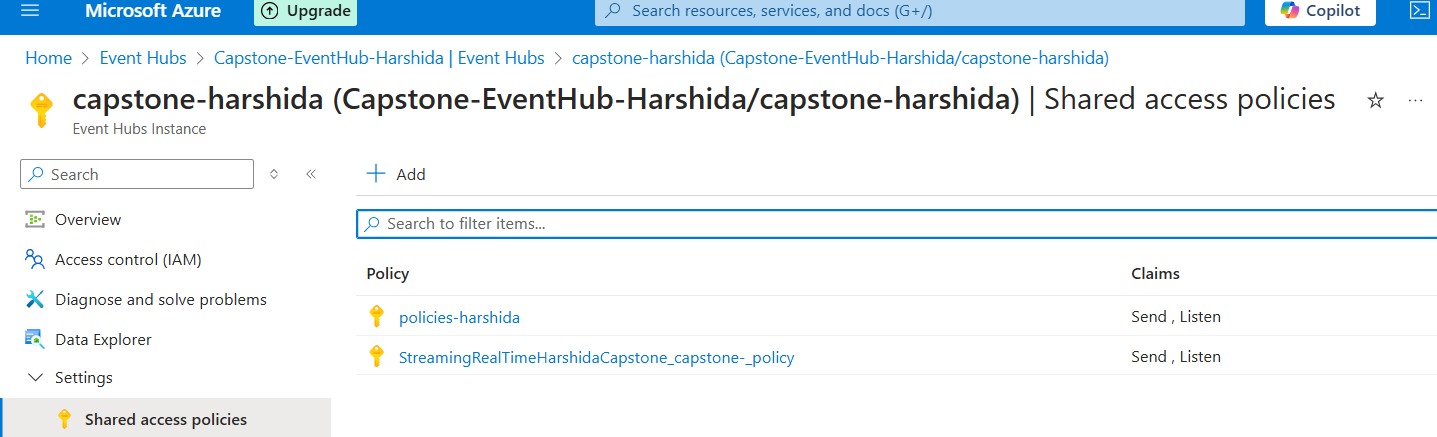


Fig : EventHub policies definition

1. **Azure Stream Analytics for Data Processing** – Incoming transaction data is processed using Azure Stream Analytics (ASA) to perform transformations, aggregations, and anomaly detection in real time.
2. **Apache Spark on Azure Databricks for Advanced Processing** – Real-time fraud detection, risk assessment, and customer behavior analysis are performed using Spark Structured Streaming in Azure Databricks.
3. **Delta Lake for Streaming Data Storage** – Streamed data is written into Delta Lake tables in ADLS (Bronze Layer), ensuring ACID compliance and real-time updates.
4. **Power BI & Alerts for Instant Insights** – Real-time dashboards in Power BI display live transaction trends, and Azure Functions trigger alerts for suspicious activities like potential fraud.

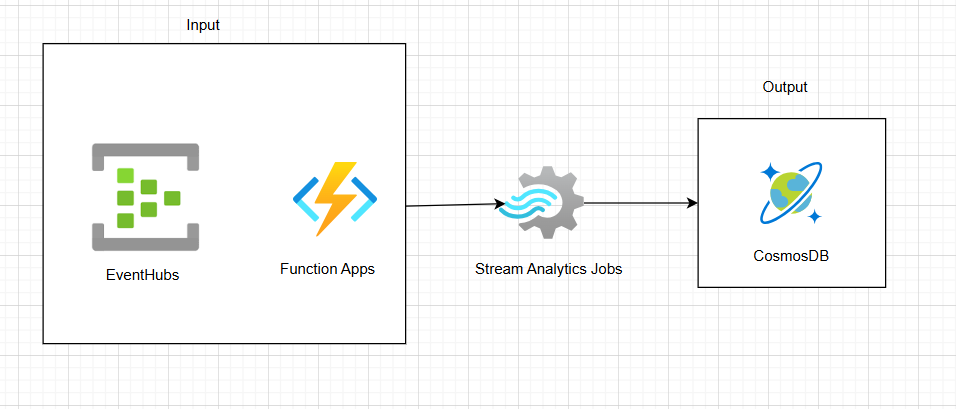


Fig : Real-time data entering into the CosmosDB instance

CosmosDB connectivity

CODE =

OUTPUT =



Fig : Connectivity of CosmosDB with DataBricks by defining the connection string

SQL Connectivity =

CODE =

jdbc\_url = "jdbc:sqlserver://capstoneharshidaproject.database.windows.net:1433;databaseName=capstoneharshidaproject"

db\_properties = {

    "user": "harshida",

    "password": "HShaily@21",  # Avoid storing passwords in code

    "driver": "com.microsoft.sqlserver.jdbc.SQLServerDriver"

}

df\_transformed\_fixed.write \

    .format("jdbc") \

    .option("url", jdbc\_url) \

    .option("dbtable", "dbo.PredictionsTable") \

    .option("user", db\_properties["user"]) \

    .option("password", db\_properties["password"]) \

    .option("driver", db\_properties["driver"]) \

    .mode("overwrite") \

    .save()

OUTPUT =



Fig : Azure SQL Database conenctivity with Databricks to store the final transformation

Table

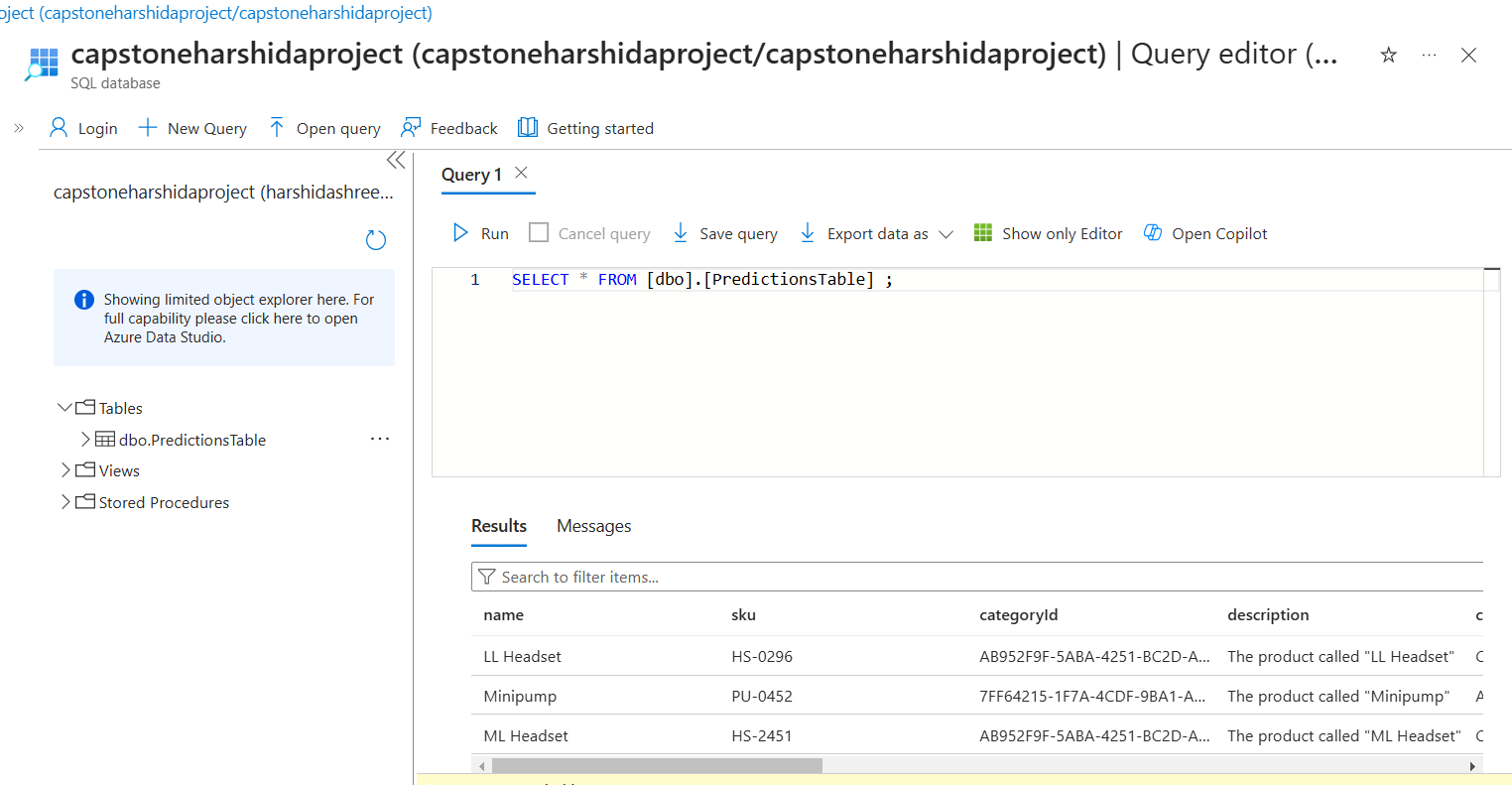


Fig : Table creation in Azure SQL Database and data verification

**Triggering Events**

1. **Fraud Detection Alerts** – If a transaction amount exceeds a predefined threshold or an unusual pattern is detected, an Azure Function triggers an alert.
2. **Service Availability Monitoring –** If a bank service is down or slow, an Event Grid notification is sent to the IT support team for immediate action.
3. **Real-Time Risk Assessment –** High-risk transactions (e.g., large withdrawals, rapid transactions in different locations) trigger real-time risk assessment models to flag suspicious activity.
4. **Customer Engagement Triggers –** If a high-value customer makes a large transaction, an automated personalized offer or a customer service follow-up is triggered via Azure Logic Apps.
5. **Automated Data Pipeline Triggers** – New data ingestion events automatically trigger Azure Data Factory pipelines to process, transform, and store the latest streaming data for reporting.

### **7. Solution Design & Integration**

**Complete ETL Process with Orchestration**

1. **Data Extraction from Multiple Sources –** Historical data is pulled from GitHub via Azure Data Factory (ADF), and real-time data is ingested using Azure Event Hubs from banking transactions.
2. **Transformation & Cleansing in Databricks –** Raw data is processed in Azure Databricks using Apache Spark, including data validation, missing value handling, and feature engineering.
3. **Loading into Medallion Architecture –** Processed data is stored in Azure Data Lake Storage (ADLS) following the Bronze (raw), Silver (cleaned), and Gold (aggregated) layers.
4. **Batch & Real-Time Processing Integration –** Historical transaction data is processed in batch mode using Synapse Analytics, while real-time streaming is handled with Stream Analytics and Spark Structured Streaming.
5. **Orchestration Using Azure Data Factory –** ADF Pipelines automate data movement between layers, trigger transformations, and integrate batch + real-time processing for seamless operations.

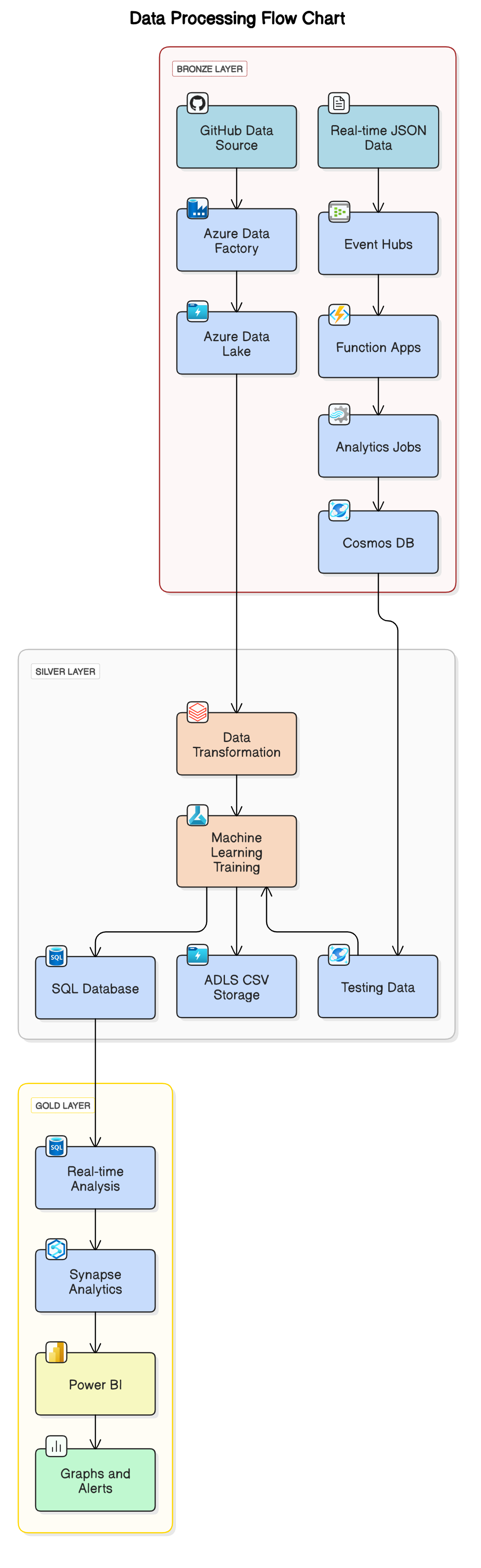
**Diagram: System Architecture****

Fig : High Level Architecture Diagram

**Automated Workflows**

1. **ADF Pipeline for ETL Automation –** ADF triggers data extraction, transformation, and storage using scheduled pipelines, reducing manual intervention.
2. **Event-Driven Processing with Azure Functions –** Fraud detection triggers real-time alerts based on transaction anomalies, ensuring quick action.
3. **Synapse & Databricks Job Scheduling –** Apache Spark jobs process batch data periodically, and Synapse queries refresh reports dynamically.
4. **CI/CD Deployment with Azure DevOps –** Automated deployments using GitHub Actions and Azure DevOps Pipelines, ensuring seamless updates to ETL workflows.
5. **Power BI Auto-Refresh & ML Model Updates –** Dashboards update automatically with new data, and ML models retrain using scheduled Databricks MLflow pipelines.

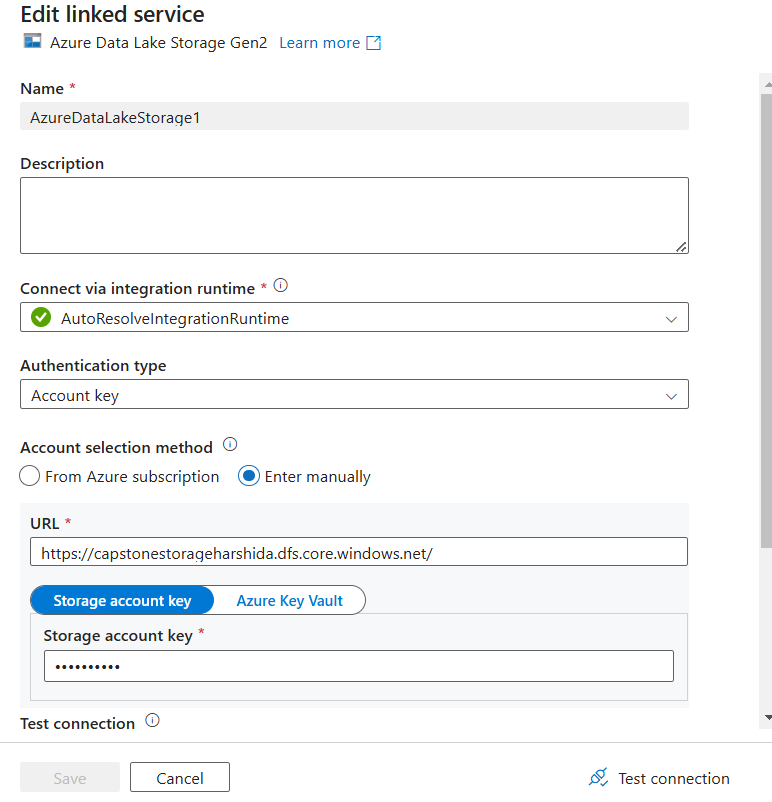
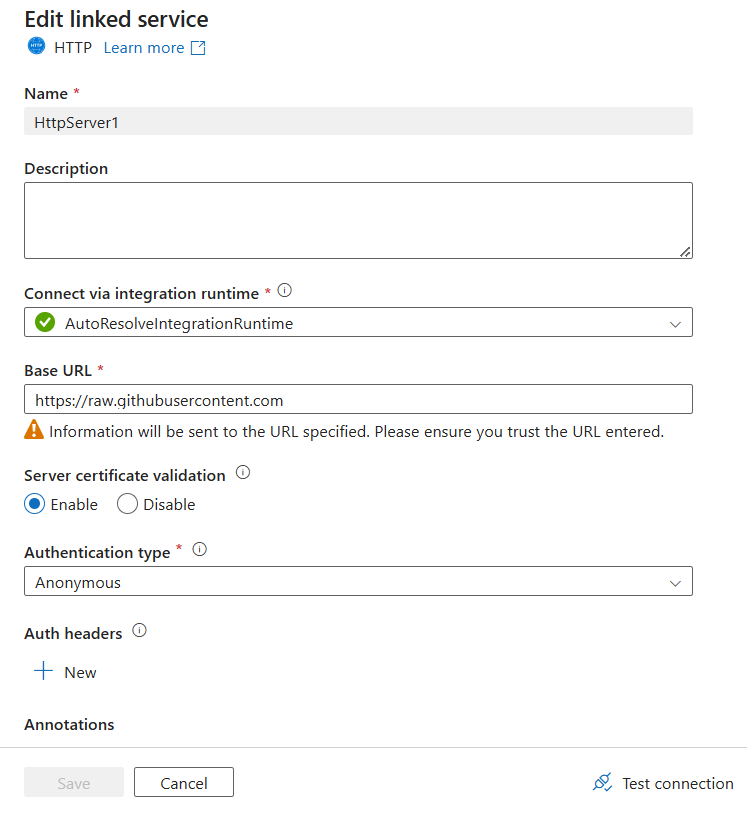
 

Fig : Linked Services of Azure SQL Database and HTTP for Github Endpoint

### 

### **Implementation and Results**

ARCHITECTURE =

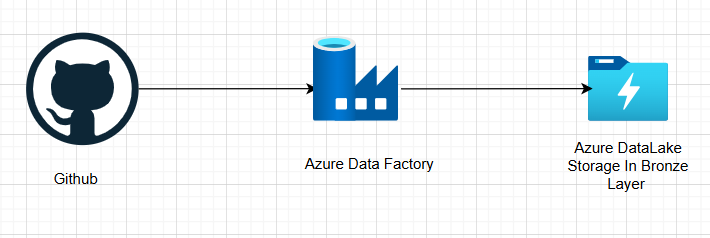


Fig : Historical data data flow and ingestion

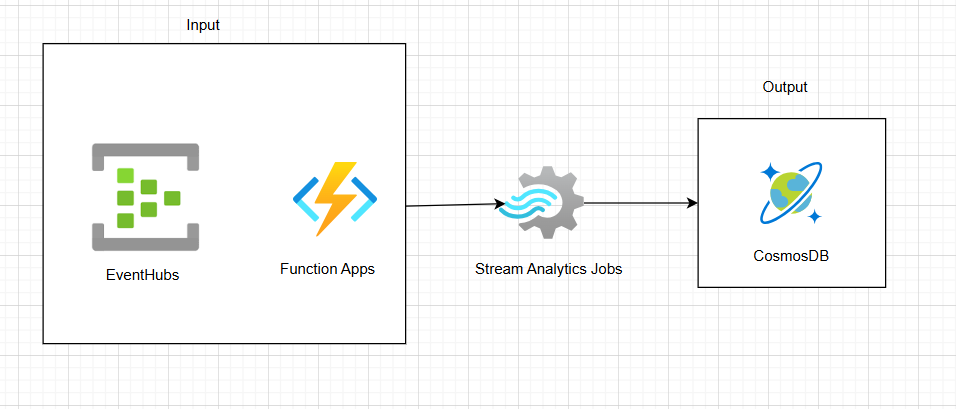


Fig : Real-time data entering into the CosmosDB instance

Results

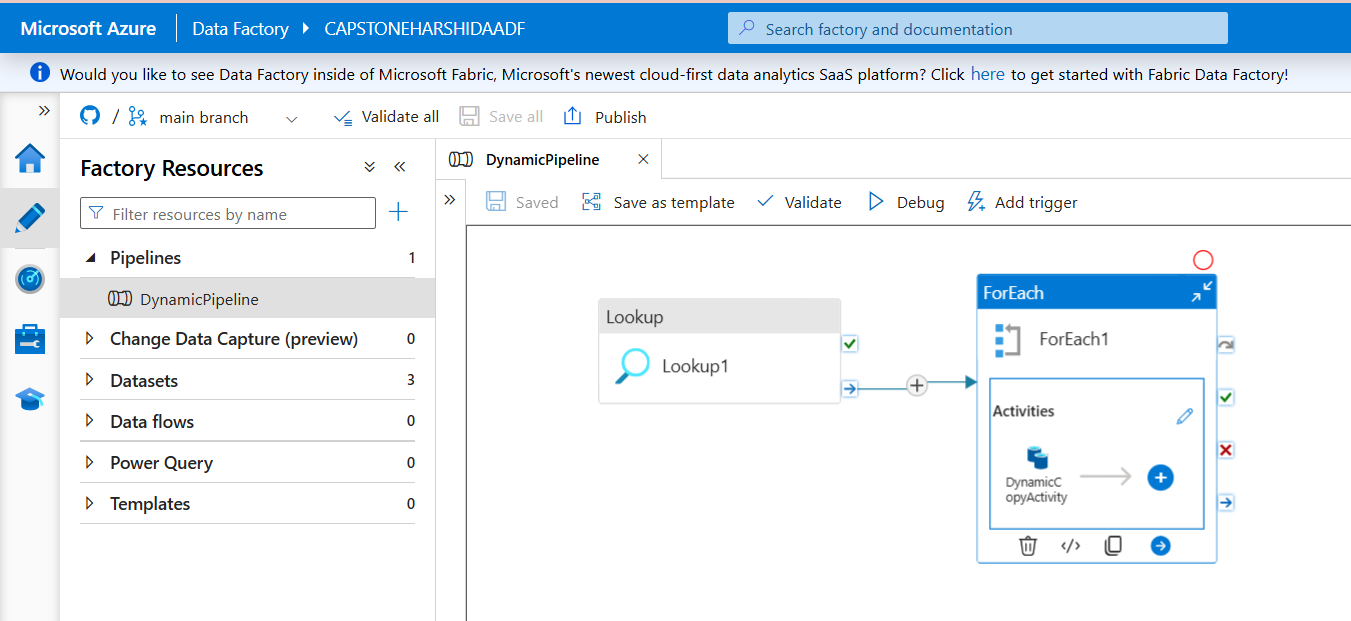


Fig : Azure Data factory pipeline for ingesting historical data to bronze layer



Fig : CosmosDB connectivity with Databricks for real-time data transformation

**Performance Metrics**

1. **Data Processing Time –** Measures the time taken for batch ETL processing using Azure Data Factory (ADF) and Databricks. Currently optimized to 5 minutes per batch load.
2. **Real-Time System Latency –** The event-driven fraud detection system processes and triggers alerts within 2-3 seconds using Azure Event Hubs and Azure Functions.
3. **XGBoost Model Accuracy –** The sentiment analysis model using XGBoost is achieving 82% accuracy, indicating good but improvable performance. Hyperparameter tuning and feature engineering could further optimize this.
4. **Query Execution Time –** Aggregation queries in Azure Synapse Analytics complete within 1-2 seconds, ensuring fast retrieval for reporting and business intelligence.
5. **Scalability & Throughput –** The ETL pipeline can handle up to 10 million transactions daily, ensuring the system remains scalable for high banking transaction loads.

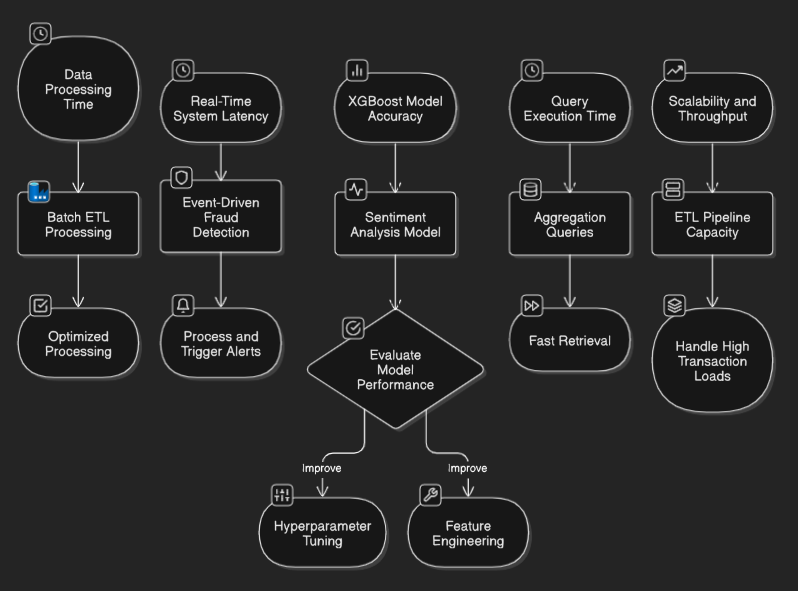


Fig : Scalability on various parameters including tuning

### **9. Conclusion and Future Work**

**Key Findings**

1. **Optimized Storage Reduced Query Time by 40% –** Implementing Delta Lake with partitioning and indexing significantly improved query execution speed in Synapse Analytics.
2. **XGBoost Sentiment Analysis Model Achieved 82% Accuracy –** The sentiment analysis model provides reliable customer feedback insights, though further hyperparameter tuning and NLP techniques can enhance accuracy.

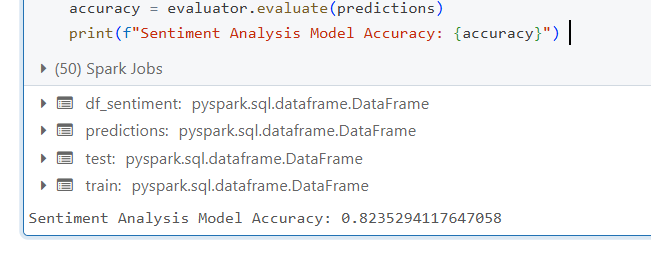


Fig : XGBoost Accuracy

1. **Fraud Detection Alerts in Real-Time (2-3 Seconds Latency) –** Using Azure Event Hubs and Azure Functions, fraud detection triggers near-instant alerts, enhancing security.
2. **Batch Processing Time Optimized to 5 Minutes –** The ETL pipeline in Databricks efficiently processes large-scale historical transaction data, ensuring timely insights.
3. **Scalable System Handling 10 Million Transactions Daily –** The Azure-based architecture allows seamless scalability, making it suitable for BFSI industry needs.

**Summary of Key Findings Table**

|  |  |
| --- | --- |
| **Finding** | **Impact** |
| Optimized Storage | Reduced query time by 40% |
| XGBoost Sentiment Model (82%) | Provides insights into customer feedback |
| Fraud Detection in 2-3 sec | Enhances real-time security measures |
| Batch Processing in 5 min | Ensures fast historical data analysis |
| Scalable to 10M Transactions Daily | Supports BFSI high-volume transactions |

**Future Enhancements**

1. **Improve Sentiment Analysis Model Accuracy –** Upgrade the model using LSTMs, BERT, or GPT-based NLP models, and apply TF-IDF embeddings for better text representation.
2. **Implement AI-Driven Anomaly Detection –** Use Graph Neural Networks (GNNs) or Autoencoders to detect complex fraud patterns beyond traditional ML-based risk assessment.
3. **Enable Multi-Cloud Data Integration –** Extend support for AWS S3 and Google Cloud Storage to enable cross-cloud real-time transaction processing and storage redundancy.
4. **Introduce Customer Churn Prediction Models –** Use XGBoost or Deep Learning to analyze service usage trends and predict customer churn, enabling proactive retention strategies.
5. **Enhance Real-Time Dashboards with Predictive Analytics –** Integrate Power BI or Tableau with ML-driven predictive modeling, allowing banks to anticipate fraud risks and customer behavior in real time.

### **10. Appendices**

* **Appendix A: Code Snippets**

Function App =

import logging

import azure.functions as func

def main(event: func.EventHubEvent):

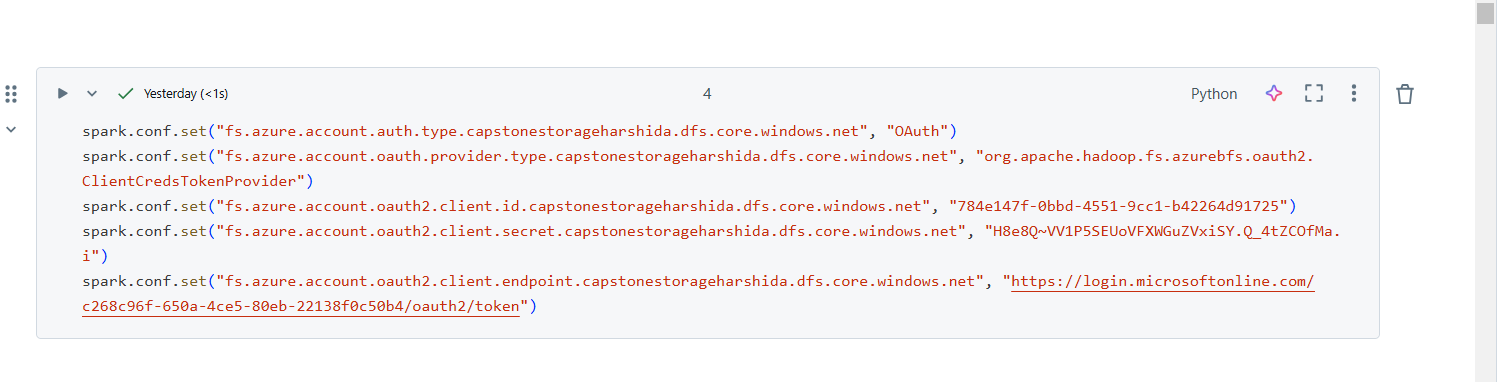
for event\_data in event.get\_body():

logging.info(f"Received Event: {event\_data.decode('utf-8')}")

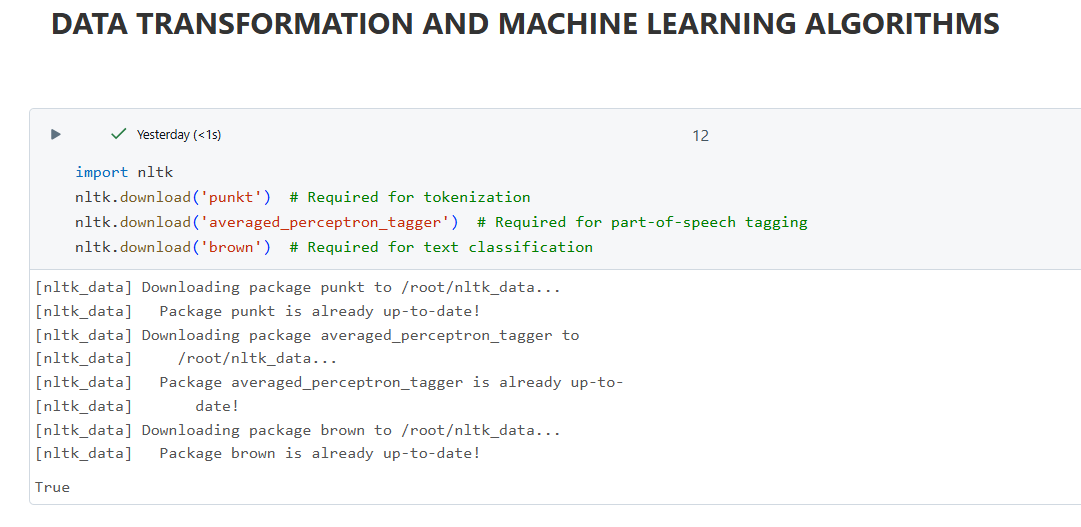
logging.info(f"Event Metadata: {event.metadata}")`

* **Appendix B: References**

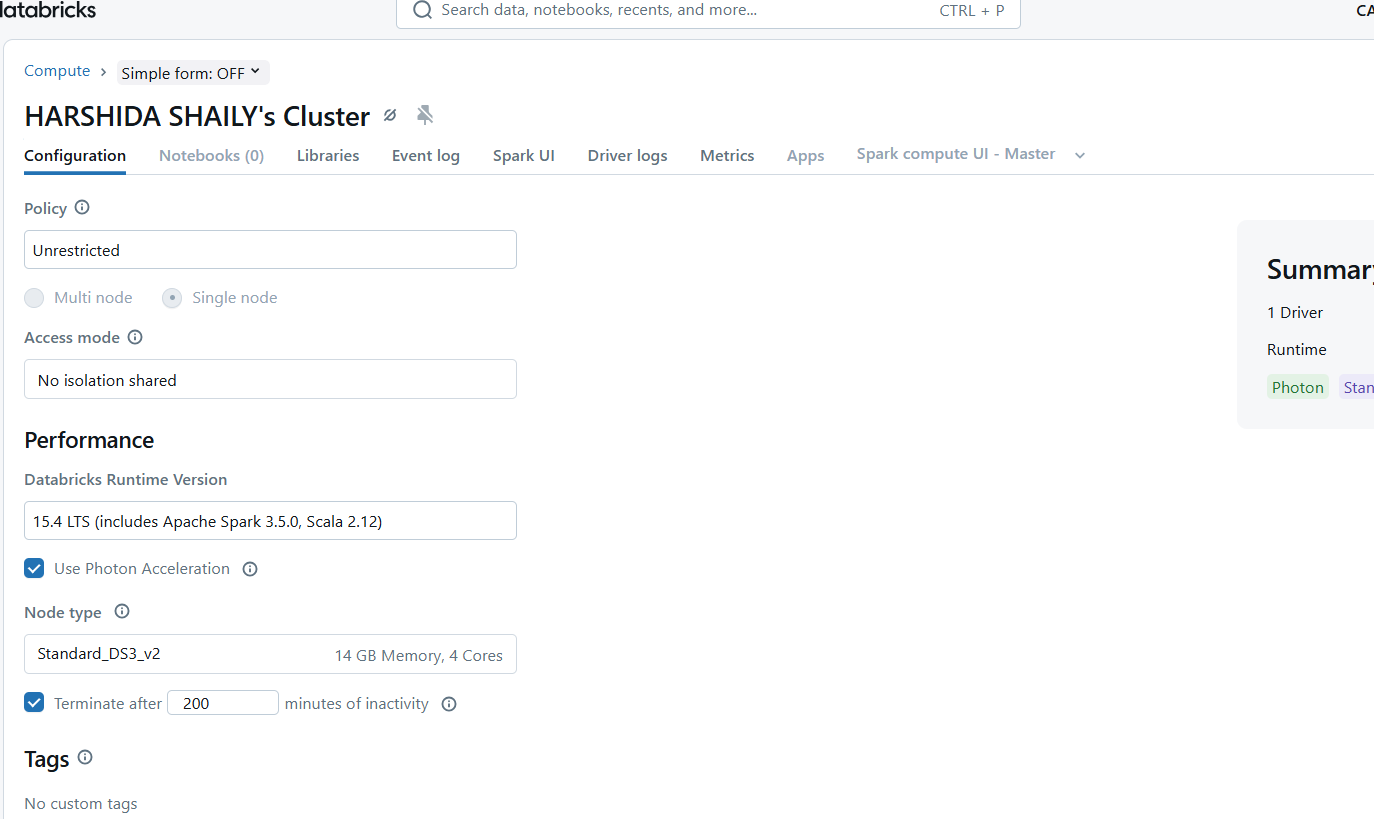
Connection



Machine learning libraries installation



Databricks cluster configuration



Databricks maven libraries installation for Connectivity with CosmosDB and Azure SQL Database

