Project: Log Anomalies Detection in Spark

Team Members-

Subrat Shukla, DE-1
Aniroop Gupta, DE-1
Harish ER, DE-1

- ## Project Overview:
  Utilize Azure Data Factory to ingest log data into Azure Storage, and leverage Azure Databricks for log analysis using Apache Spark for insights such as error rates, trends, and anomalies.

- ## Project Description:
  The **Log Anomaly Detection** project is designed to provide a scalable solution for ingesting, analyzing, and monitoring system logs to gain actionable insights such as error rates, trends, and anomalies. The architecture leverages **Azure Data Factory** (ADF) for automated data ingestion, **Azure Storage** for centralized storage, and **Azure Databricks** powered by Apache Spark for processing and analyzing log data.

- ## Data Overview:
  The file "Application Logs.csv" contains log data with 3,305 entries and 5 columns, focusing on application-level logs. Below are the key details:
  - **Columns:**
    - **Level:** Represents the severity of the log (e.g., Information, Error, Warning).
    - **Date and Time:** Captures the timestamp of each log entry.
    - **Source:** Indicates the application or process generating the log (e.g., MSSQLSERVER, edgeupdate).
    - **Event ID:** A unique identifier associated with the log event.
    - **Task Category:** Provides additional context or details about the log event.

- **Summary:**
  - The dataset has some missing entries, as indicated by the non-null counts (e.g., 3284 out of 3305 rows have valid data in most columns).
  - The **Source** column is of numerical type but includes floating-point values, potentially requiring standardization for analysis.
- **Sample Data:**

```
   Level         Date and Time         Source   Event ID   Task Category
```

- `Information 19-12-2024,09:29:56 MSSQLSERVER  17890 Server log context`
- `Information 19-12-2024,09:28:32 edgeupdate     0.0 Service stopped.`
- `Information 19-12-2024,09:26:40 MsiInstaller 1035  Windows Installer logs`

- ## How it works:

  We are referring to log files taken from our local system.

  File name: Application Logs

  Type: CSV

  Preview:

A2 | fx | Information

	A	B	C	D	E	F	G	H	I
1	Level	Date and Time	Source	Event ID	Task Category				
2	Information	19-12-2024 9.29	MSSQLSERVER	17890	Server	A significant part of sql server process memc			
3	Information	19-12-2024 9.28	edgeupdate	0	None	Service stopped.			
4	Information	19-12-2024 9.26	MsiInstaller	1035	None	Windows Installer reconfigured the product.			
5	Information	19-12-2024 9.26	MsiInstaller	1035	None	Windows Installer reconfigured the product.			
6	Information	19-12-2024 9.26	MsiInstaller	1035	None	Windows Installer reconfigured the product.			
7	Information	19-12-2024 9.26	MsiInstaller	1035	None	Windows Installer reconfigured the product.			
8	Information	19-12-2024 9.26	MsiInstaller	1035	None	Windows Installer reconfigured the product.			
9	Information	19-12-2024 9.26	MsiInstaller	1035	None	Windows Installer reconfigured the product.			
10	Information	19-12-2024 9.26	MsiInstaller	1035	None	Windows Installer reconfigured the product.			
11	Information	19-12-2024 9.26	MsiInstaller	1035	None	Windows Installer reconfigured the product.			
12	Information	19-12-2024 9.26	MsiInstaller	1035	None	Windows Installer reconfigured the product.			
13	Information	19-12-2024 9.26	MsiInstaller	1035	None	Windows Installer reconfigured the product.			
14	Information	19-12-2024 9.26	MsiInstaller	1035	None	Windows Installer reconfigured the product.			
15	Information	19-12-2024 9.26	MsiInstaller	1035	None	Windows Installer reconfigured the product.			
16	Information	19-12-2024 9.26	MsiInstaller	1035	None	Windows Installer reconfigured the product.			
17	Information	19-12-2024 9.26	MsiInstaller	1035	None	Windows Installer reconfigured the product.			

- **Execution Overview:**

  The Hybrid Cloud Data Movement project involves creating an end-to-end pipeline to move and process data using Azure Data Factory and Azure Databricks.

  Steps to Implement the Solution:

  1. Understanding Requirements
    - Identify the log sources (e.g., system logs, application logs) from on-premises servers or cloud platforms.
    - Store raw logs in Azure Blob Storage or Azure Data Lake for further processing.
    - Parse unstructured log data into structured formats (JSON, CSV).
    - Extract critical insights like error rates, trends, and anomalies using Spark.

  2. Setting Up Azure Environment
    - Create a new ADF instance to orchestrate log ingestion pipelines.
    - Set up a Blob Storage container or Data Lake to hold raw and processed log files.
    - Deploy and configure a Databricks workspace for data analysis and log processing.
    - Establish secure connections to log sources using a **Self-hosted Integration Runtime**, ensuring seamless data ingestion from on-premises systems.

  3. Data Movement (ADF)
    - **ADF Pipelines:**
      - Create a pipeline with the following tasks:
        1. **Extract Logs:** Pull log files from the source using Copy Data activities or custom connectors.
        2. **Store in Azure:** Load the extracted data into Blob Storage or Data Lake containers in raw format.

- **Triggers:**
    - Implement scheduled triggers for regular ingestion (e.g., hourly or daily).
    - For real-time monitoring, use event-based triggers to respond dynamically to log file updates.

4. Data Processing (Azure Databricks)

- **Databricks Notebooks:**
    - Develop notebooks to process raw logs, performing tasks such as:
        - Filtering by severity (e.g., ERROR, WARN).
        - Aggregating logs to analyze trends over time.
        - Detecting anomalies using Spark MLlib (e.g., Isolation Forest, DBSCAN).
    - Example Spark Code:
      logs_df = spark.read.text("path_to_logs")
      parsed_logs = logs_df.withColumn("timestamp", ...) # Extract relevant columns
- **Integration with ADF:**
    - Use **Notebook Activities** in ADF to trigger log processing scripts on Databricks dynamically.

5. End-to-End Workflow

- **Pipeline Composition:**
    - Combine data movement and processing pipelines into a cohesive ADF workflow:
        1. **Data Ingestion Pipeline:** Extract logs and load them into Azure Blob Storage.
        2. **Data Processing Pipeline:** Trigger Databricks notebooks for log analysis.
- **Error Handling and Resilience:**
    - Include proper error handling and retry mechanisms in ADF for robustness.
    - Log metadata about failed runs or exceptions to Azure Monitor for proactive resolution.

- **Azure resources used for this project:**
  1. Azure Data Factory (ADF)
  2. Azure Blob Storage
  3. Azure Databricks
  4. Azure Monitor

- **Project Requirements:**
  1. Create new Azure Data Factory
  2. Create a new container from Storage accounts, to store data.
  3. Create a Databricks notebook to load data
  4. Download Microsoft Integration Runtime to create pipeline.
  5. Download logs file from any open source.

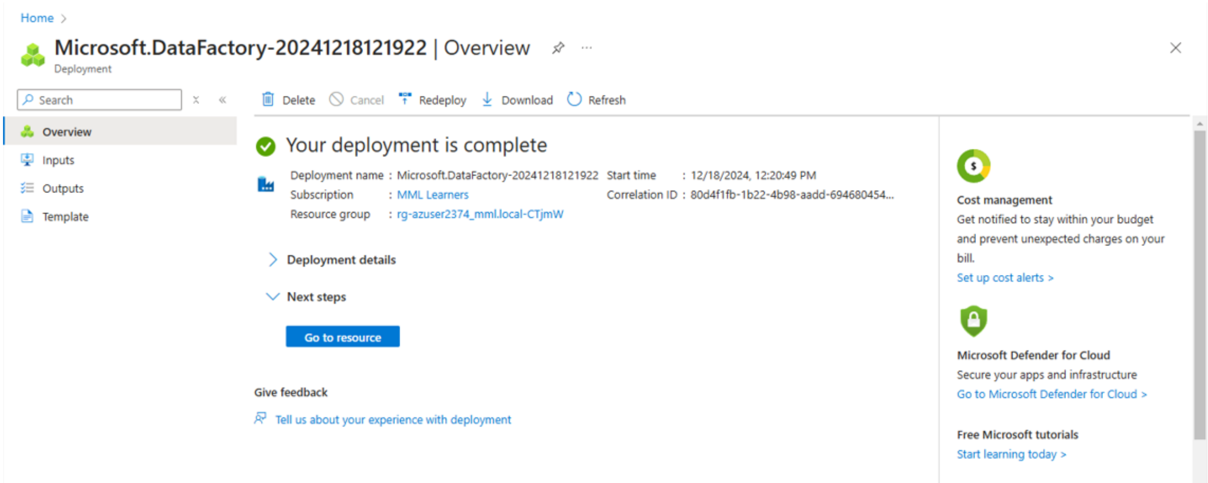
- **Tasks Performed:**
  1. **Configured Azure Data Factory (ADF):** Set up ADF and created linked services to connect with on-premises log data sources and Azure Blob Storage.
  2. **Set up Integration Runtime:** Deployed a self-hosted integration runtime for secure and reliable data movement between on-premises log sources and Azure services.
  3. **Built Pipelines in ADF:** Designed and implemented pipelines to extract raw log data from source systems and load it into Azure Blob Storage.
  4. **Scheduled and Tested Pipelines:** Scheduled pipelines using time-based triggers and tested them to ensure seamless and consistent log data ingestion.
  5. **Integrated Databricks with ADF:** Connected Azure Databricks with ADF for triggering and orchestrating data analysis workflows.
  6. **Developed Databricks Notebooks:** Created notebooks to preprocess logs (e.g., parsing, cleaning) and perform advanced analysis such as calculating error rates, detecting trends, and identifying anomalies.
  7. **Performed Data Processing:** Implemented Spark-based transformations and analytics to process logs into structured data and generate actionable insights.

8. **Validated Results:** Conducted rigorous testing to verify data processing accuracy, detect anomalies effectively, and ensure insights matched system expectations.

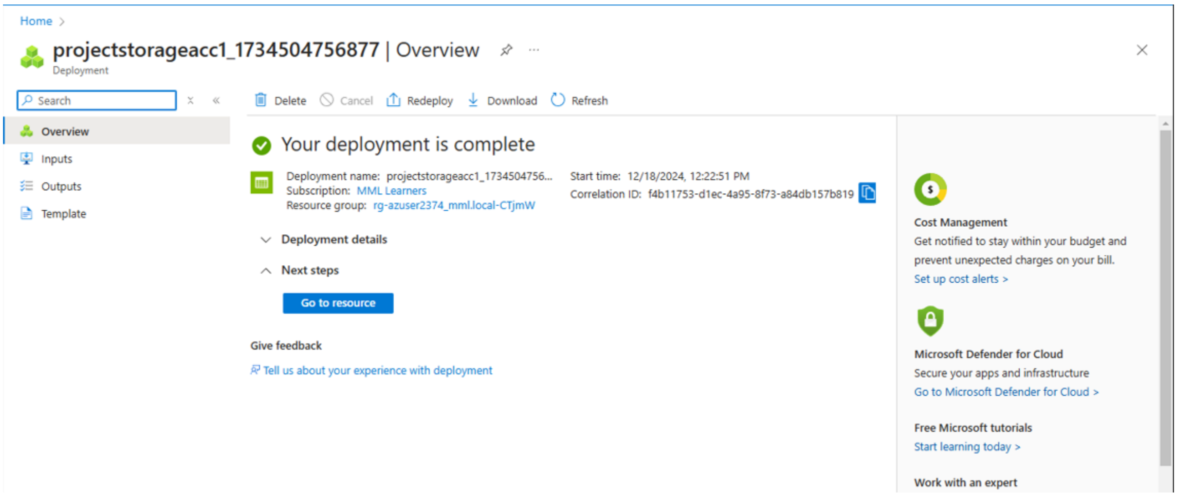
This implementation created a resilient and scalable solution for log anomaly detection, leveraging Azure's data and analytics ecosystem. It streamlined hybrid log data management, providing actionable insights to maintain system health and improve operational efficiency.

- ## Analysis Results:

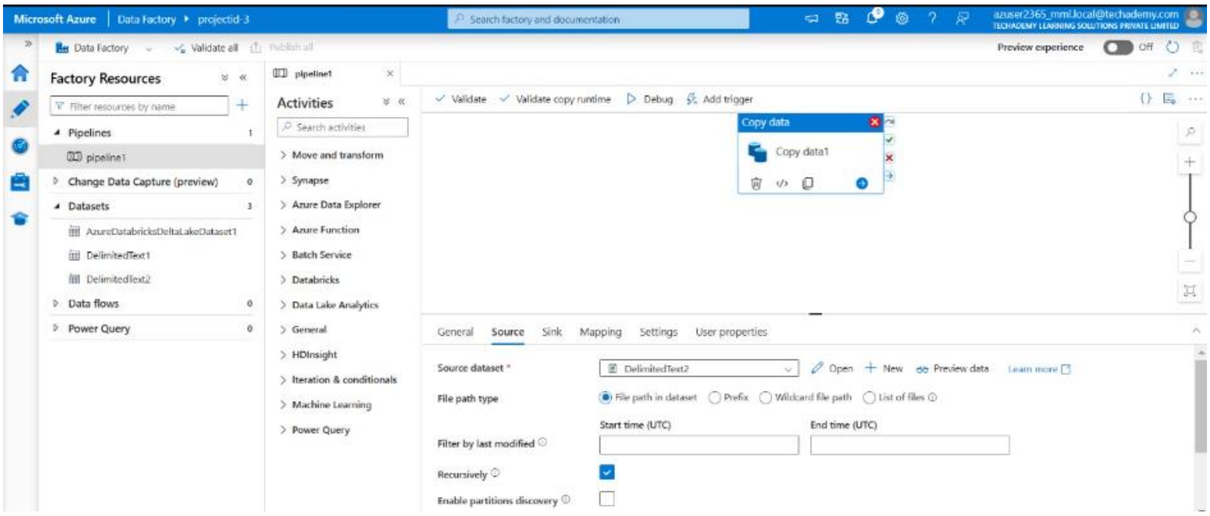
1. Creating new Data Factory:



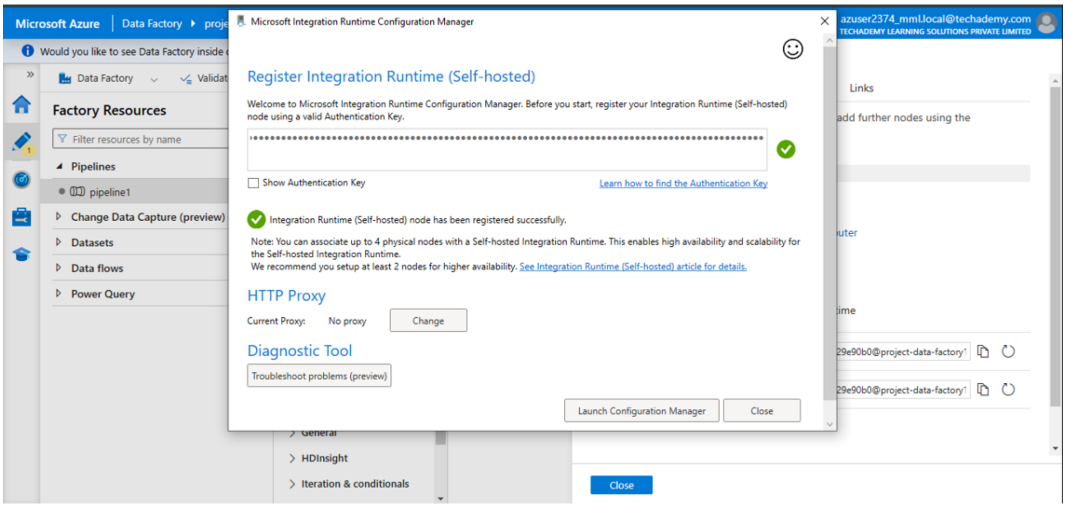
2. Creating storage account:

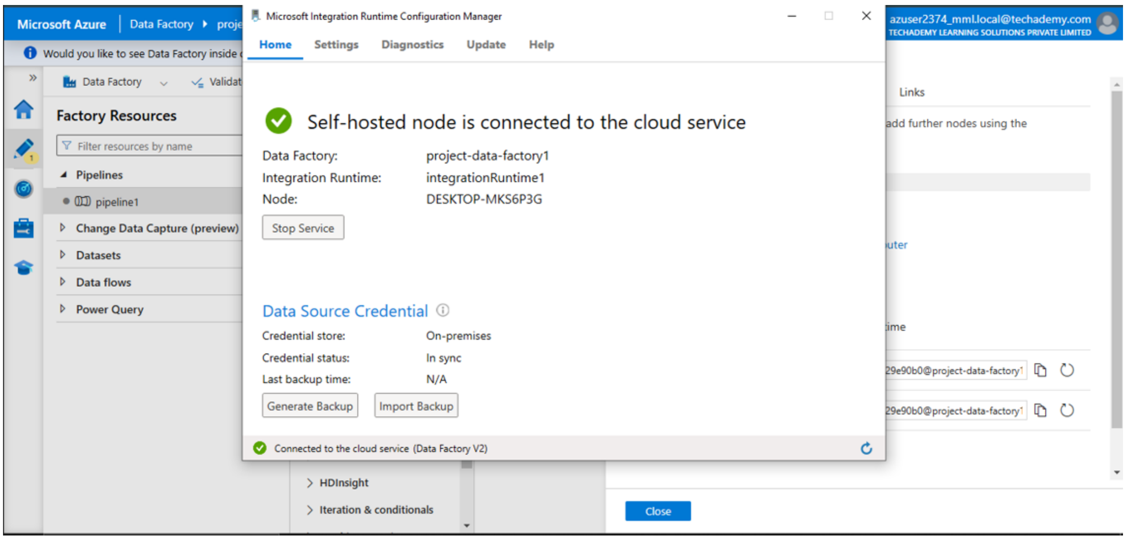


3. Creating Pipeline to load the data:

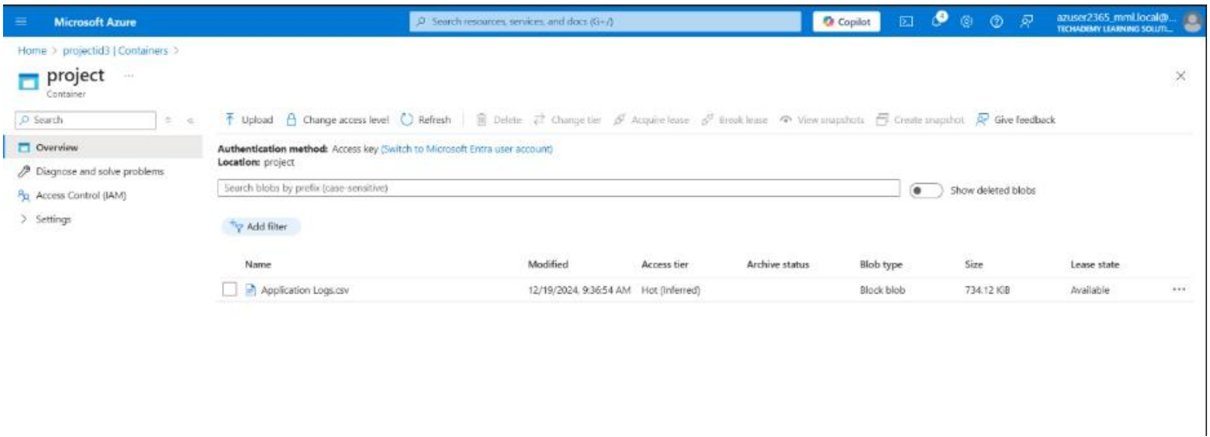


4. Integrating Integration Runtime to connect self-host node with cloud:

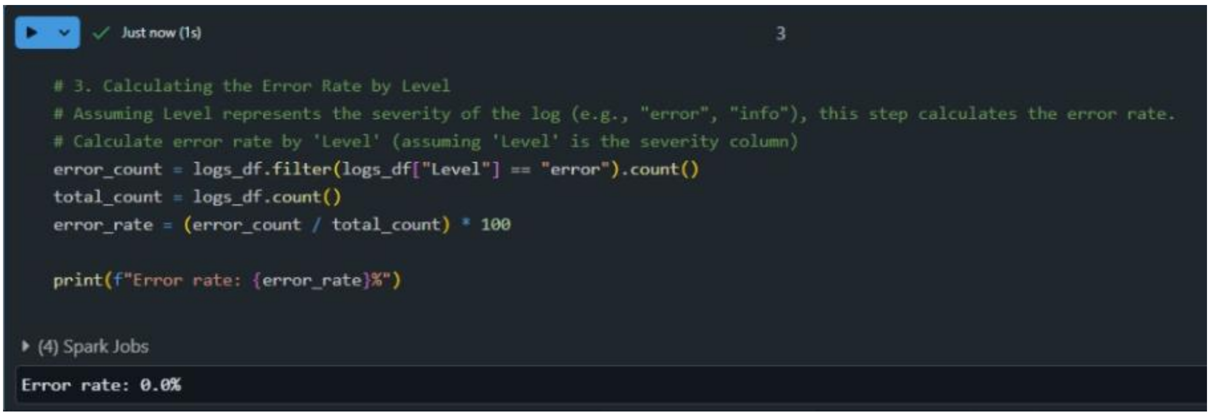




5. Migrated data to the Storage container:



6. Integrate Databricks with ADF for automated workflow:



Just now (1s) 4

```python
# 4. Trend Analysis Over Time (Using Date and Time Column)
# This step extracts the date and hour components from the Date and Time column to analyze the error trends over time.
from pyspark.sql.functions import col, to_date, hour

# Extract date and hour directly from 'Date and Time' column
logs_df = logs_df.withColumn("date", to_date(col("Date and Time")))
logs_df = logs_df.withColumn("hour", hour(col("Date and Time")))

# Group by date and hour to find trends
trend_df = logs_df.groupBy("date", "hour", "Level").count().orderBy("date", "hour")
trend_df.show(10)
```

(2) Spark Jobs

logs_df: pyspark.sql.dataframe.DataFrame = [Level: string, Date and Time: string ... 5 more fields]
trend_df: pyspark.sql.dataframe.DataFrame = [date: date, hour: integer ... 2 more fields]

```
+----+----+--------------------+-----+
|date|hour|               Level|count|
+----+----+--------------------+-----+
|NULL|NULL|C:\WINDOWS\system...|    7|
|NULL|NULL|App: C:\Program F...|    7|
|NULL|NULL|\t -s ""MSSQLSERV...|    7|
|NULL|NULL|13: 8a292df8-d653...|    7|
|NULL|NULL|Application Id=55...|    7|
|NULL|NULL|dbv = 1568.230.50...|    1|
|NULL|NULL|    10.0.22621.4601"|    7|
|NULL|NULL|[16] 0.001881 -0....|    1|
|NULL|NULL|[1] 0.040380 -0.0...|    1|
|NULL|NULL|               [5] -|    1|
+----+----+--------------------+-----+
only showing top 10 rows
```

Just now (1s) 5

```python
# 5. Anomaly Detection Based on Error Count Spikes
# Here, we detect anomalies by flagging times when the error count exceeds a predefined threshold (e.g., twice the average error count).
from pyspark.sql import functions as F

# Aggregating error counts by date and hour
error_counts = logs_df.filter(logs_df.Level == "error") \
    .groupBy("date", "hour") \
    .agg(F.count("*").alias("error_count"))

# Calculate the average error count
avg_error_count = error_counts.agg(F.avg("error_count")).collect()[0][0]

# Check if the average value is None (no errors in the logs), and set a default threshold if needed
if avg_error_count is None:
    print("No error logs found.")
    threshold = 10  # Set a reasonable default threshold if no errors are found
else:
    # Define a threshold for anomaly detection (e.g., twice the average error count)
    threshold = avg_error_count * 2  # Double the average count

# Flag anomalies where error count exceeds the threshold
anomalies = error_counts.filter(error_counts.error_count > threshold)
anomalies.show(10)
```

(3) Spark Jobs

anomalies: pyspark.sql.dataframe.DataFrame = [date: date, hour: integer ... 1 more field]
error_counts: pyspark.sql.dataframe.DataFrame = [date: date, hour: integer ... 1 more field]

```
No error logs found.
+----+----+-----------+
|date|hour|error_count|
+----+----+-----------+
+----+----+-----------+
```

Just now (3s) 6

```python
from pyspark.sql.functions import regexp_extract

# Extract timestamp, log level, and message
df_parsed = logs_df.withColumn("timestamp", regexp_extract("Level", r"(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})", 1)) \
                .withColumn("log_level", regexp_extract("Level", r"\b(INFO|ERROR|WARN|DEBUG)\b", 1)) \
                .withColumn("message", regexp_extract("Level", r"(?:INFO|ERROR|WARN|DEBUG)\s+(.*)", 1))

df_parsed.show(truncate=False)

output_path = "dbfs:/user/hive/warehouse/processed_logs"

# Save as Parquet
df_parsed.write.mode("overwrite").parquet(output_path)

# Save as CSV
df_parsed.write.mode("overwrite").csv(output_path)
```

(3) Spark Jobs

df_parsed: pyspark.sql.dataframe.DataFrame = [Level: string, Date and Time: string ... 8 more fields]

```
|Information|19-12-2024 09:26:40|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:40|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:40|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:40|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:40|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:40|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:40|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:40|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:40|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:40|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:40|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:40|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:40|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:40|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:40|MsiInstaller|1033    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:40|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:39|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
|Information|19-12-2024 09:26:39|MsiInstaller|1035    |None          |NULL|NULL|      |      |    |
+-----------+-------------------+------------+--------+--------------+----+----+------+------+----+
only showing top 20 rows
```

Just now (1s) 8

```python
# Define the output path for processed data
output_path = f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/processed_data/"

# Write the processed DataFrame back as a CSV file, overwrite existing data if necessary
logs_df.write.mode("overwrite").option("header", "true").csv(output_path)

# Print confirmation
print(f"Data successfully written to: {output_path}")
```

(1) Spark Jobs

```
Data successfully written to: wasbs://project@projectid3.blob.core.windows.net/processed_data/
```

3 minutes ago (1s) 1

```python
# Set Azure Storage configurations
storage_account_name = "projectid3"
container_name = "project"
storage_account_key = "Sk4YdDVh5/tdr13VMFs09NGRrmwZTMFawnqXBq40hXi6qG2zh7cWLLxMLTf9WES0vlwWGVd3RCkg+AStmYEfQA=="

# Configure Spark to access Azure Blob Storage using the account key
spark.conf.set(
    f"fs.azure.account.key.{storage_account_name}.blob.core.windows.net",
    storage_account_key
)

# Define the file path to the CSV file in Azure Blob Storage
file_path = f"wasbs://{container_name}@{storage_account_name}.blob.core.windows.net/Application Logs.csv"

# Read the CSV file into a DataFrame
logs_df = spark.read.option("header", "true").csv(file_path)

# Show the first few rows to confirm data is loaded correctly
logs_df.show(5)
```

(2) Spark Jobs

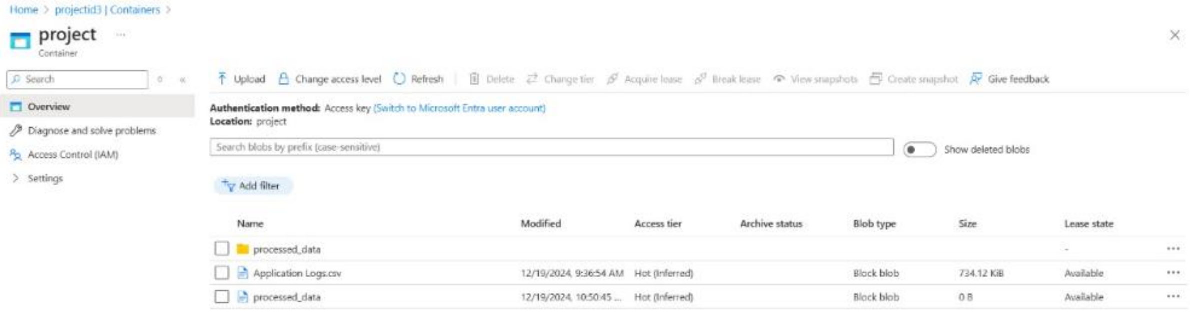
logs_df: pyspark.sql.dataframe.DataFrame = [Level: string, Date and Time: string ... 3 more fields]

```
+-----------+-------------------+------------+--------+-------------+
|      Level|      Date and Time|      Source|Event ID|Task Category|
+-----------+-------------------+------------+--------+-------------+
|Information|19-12-2024 09:29:56| MSSQLSERVER|   17890|       Server|
|Information|19-12-2024 09:28:32|  edgeupdate|       0|         None|
|Information|19-12-2024 09:26:40|MsiInstaller|    1035|         None|
|Information|19-12-2024 09:26:40|MsiInstaller|    1035|         None|
|Information|19-12-2024 09:26:40|MsiInstaller|    1035|         None|
+-----------+-------------------+------------+--------+-------------+
only showing top 5 rows
```

This ETL process ensures the data first gets converted into processed form and then migrates the processed data to the Blob Storage Container.

Let us see the final processed data folder in Blob Storage.

7. Processed Data migrated to Blob Storage:



- **Conclusion:**

  The Log Anomalies Detection project successfully implemented a scalable and efficient solution for managing and analyzing log data. Leveraging Azure's robust ecosystem, we automated data ingestion from diverse sources using Azure Data Factory and securely stored raw logs in Azure Blob Storage. With Azure Databricks, we harnessed the power of Apache Spark to preprocess, transform, and analyze log data, extracting key insights such as error rates, trends, and anomalies. The integration of these tools provided seamless workflows, automated processes, and accurate anomaly detection.

Submitted By-

Subrat Shukla, DE-1
Harish ER, DE-1
Aniroop Gupta, DE-1

--Thank You!