Case Study: Implementing a Data Pipeline Using Azure Data Factory, Databricks, and Synapse Analytics

Overview

This case study outlines the implementation of a scalable data pipeline that integrates Azure services: Azure Data Factory (ADF) for data ingestion, Azure Databricks for ETL (Extract, Transform, Load) processes, and Azure Synapse Analytics for storage and analytics. The pipeline extracts data from an HTTP API, processes it through a series of transformations, and stores it in Synapse dedicated SQL pools.

Solution Architecture

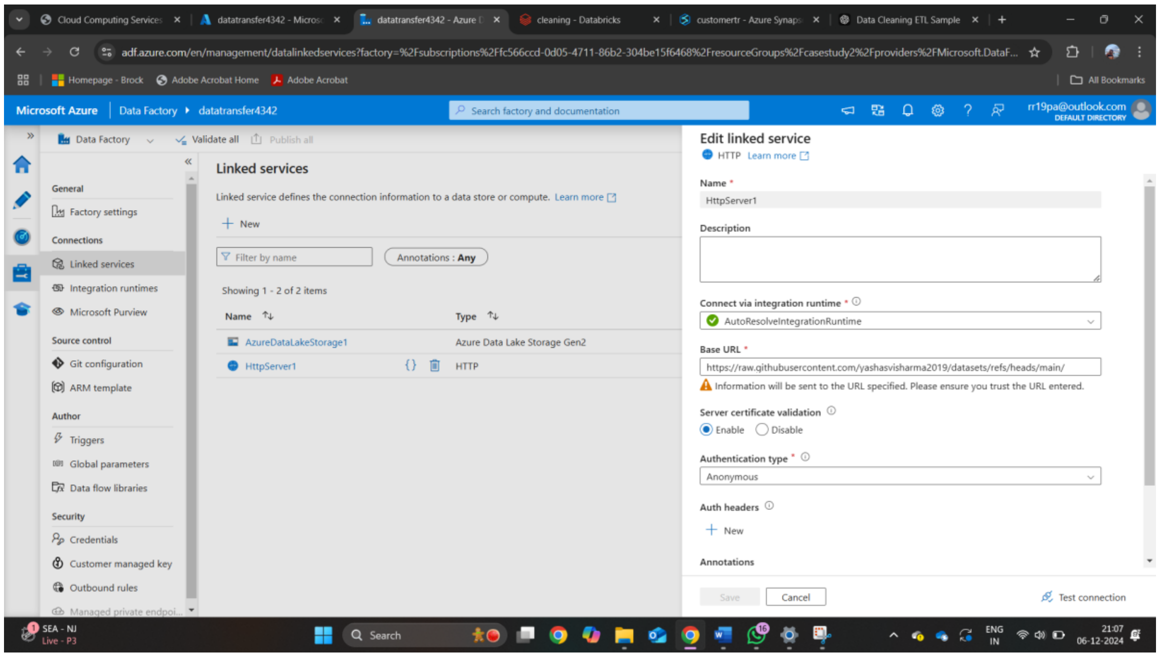
1. **Data Sources**: HTTP API.
2. **Ingestion Layer**: Azure Data Factory pipelines.
3. **Transformation Layer**: Azure Databricks.
4. **Storage Layers**:
   - **Raw Zone**: Data Lake.
   - **Processed Zone**: Data Lake.
   - **Gold Zone**: Data Lake and Synapse Dedicated SQL Pools.

Implementation Steps

1. Data Ingestion with Azure Data Factory

Objective: Pull data from an external HTTP API and store it in Azure Data Lake's raw zone.

Steps:
- Created an HTTP Linked Service in ADF to connect to the API.
- Developed an ADF pipeline using a Copy Data Activity to fetch JSON data from the API and store it in Azure Data Lake in the raw zone.
- Configured parameters for API keys and pagination to handle dynamic data extraction



adf.azure.com/en/management/datalinkedservices?factory=%2Fsubscriptions%2Ffc566ccd-0d05-4711-86b2-304be15f6468%2FresourceGroups%2Fcasestudy2%2Fproviders%2FMicrosoft.DataF...

Microsoft Azure | Data Factory ▸ datatransfer4342

Linked services

Linked service defines the connection information to a data store or compute. Learn more

Showing 1 - 2 of 2 items

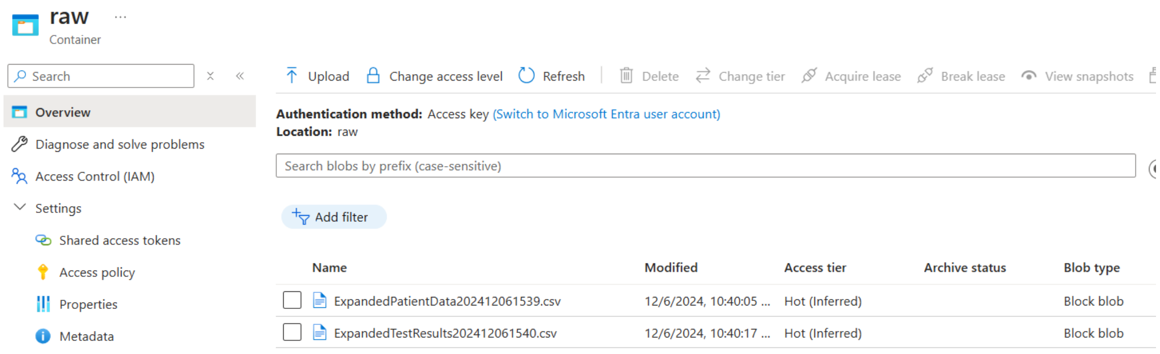
Name	Type
AzureDataLakeStorage1	Azure Data Lake Storage Gen2
HttpServer1	HTTP

Edit linked service

HTTP Learn more

Name *	HttpServer1
Description	
Connect via integration runtime *	AutoResolveIntegrationRuntime
Base URL *	https://raw.githubusercontent.com/yashasvisharma2019/datasets/refs/heads/main/
Server certificate validation	[✓ Enable] Disable
Authentication type *	Anonymous

Information will be sent to the URL specified. Please ensure you trust the URL entered.



raw

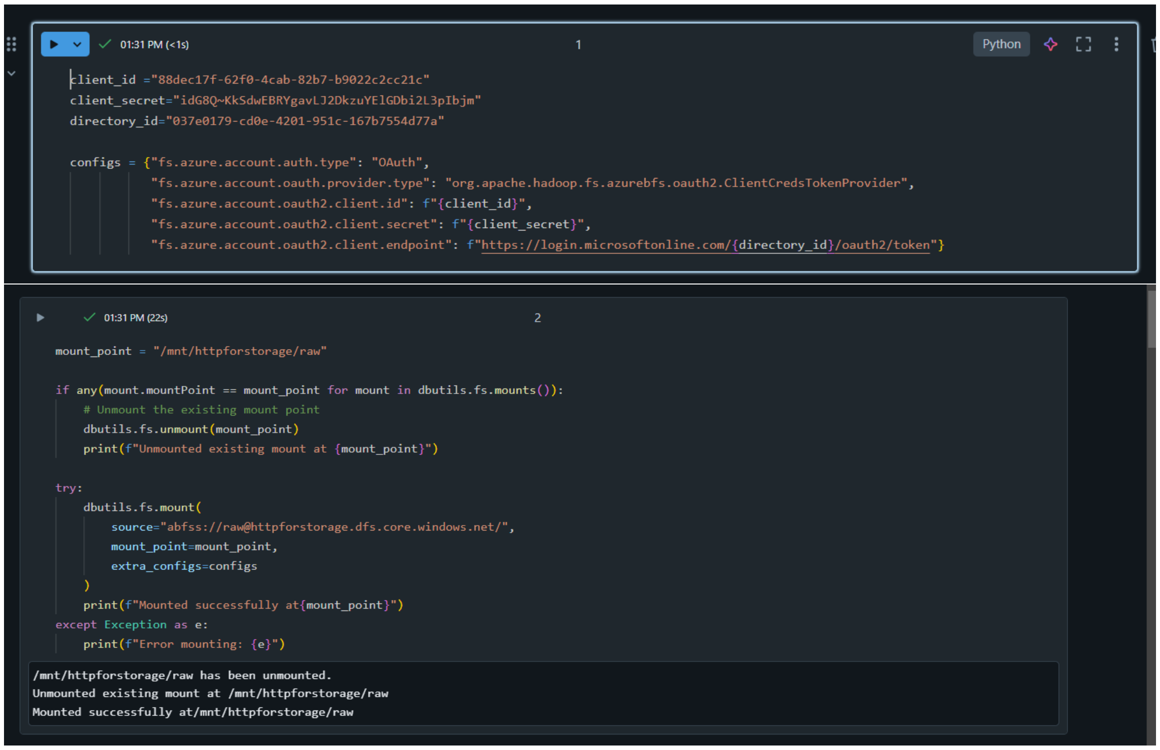
Container

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: raw

Name	Modified	Access tier	Archive status	Blob type
ExpandedPatientData202412061539.csv	12/6/2024, 10:40:05 ...	Hot (Inferred)		Block blob
ExpandedTestResults202412061540.csv	12/6/2024, 10:40:17 ...	Hot (Inferred)		Block blob

2. Data Transformation Using Azure Databricks

Objective: Clean, transform, and prepare data for analytics by moving data across zones.

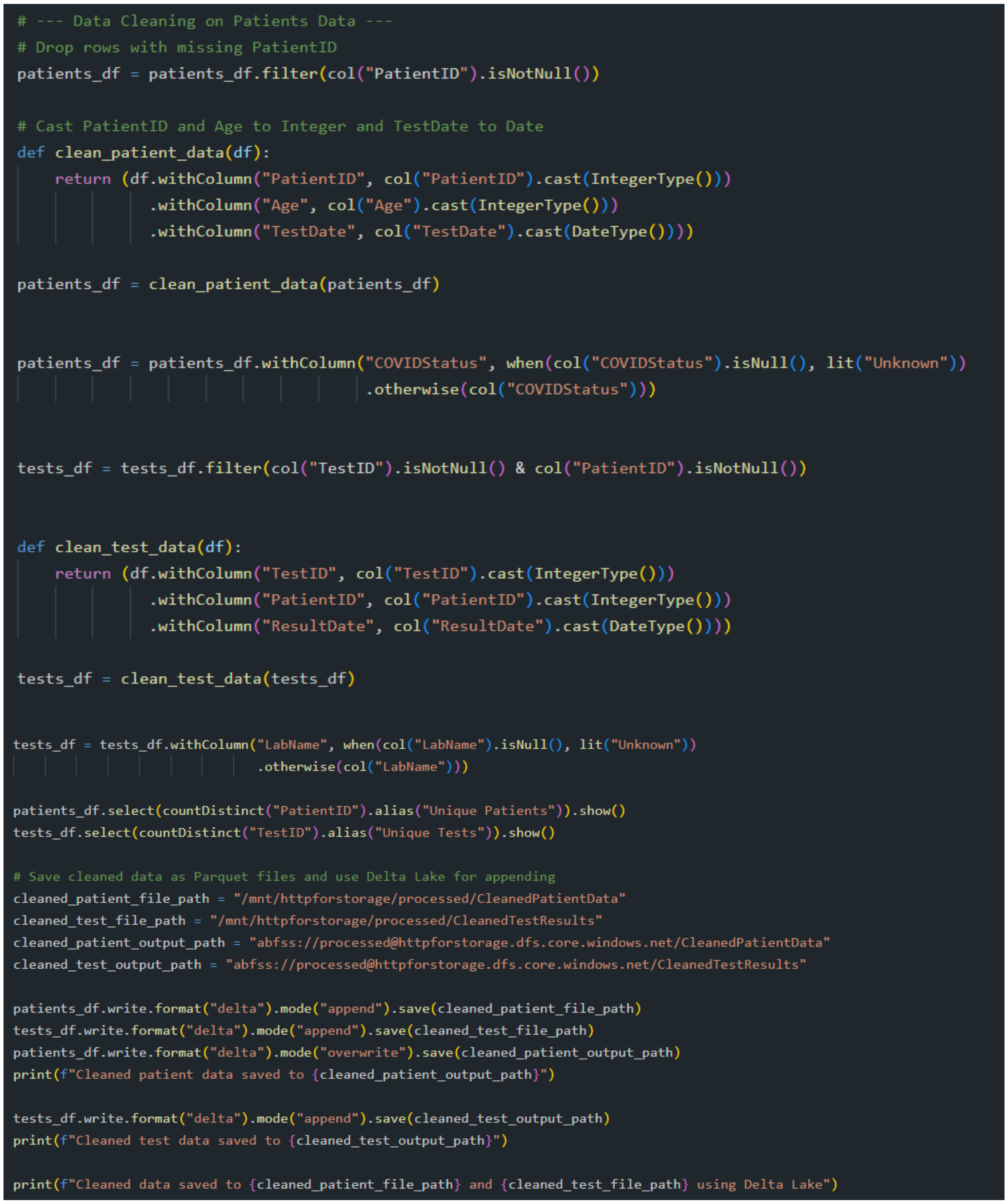


```python
client_id ="88dec17f-62f0-4cab-82b7-b9022c2cc21c"
client_secret="idG8Q~KkSdwEBRYgavLJ2DkzuYElGDbi2L3pIbjm"
directory_id="037e0179-cd0e-4201-951c-167b7554d77a"

configs = {"fs.azure.account.auth.type": "OAuth",
           "fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
           "fs.azure.account.oauth2.client.id": f"{client_id}",
           "fs.azure.account.oauth2.client.secret": f"{client_secret}",
           "fs.azure.account.oauth2.client.endpoint": f"https://login.microsoftonline.com/{directory_id}/oauth2/token"}
```

```python
mount_point = "/mnt/httpforstorage/raw"

if any(mount.mountPoint == mount_point for mount in dbutils.fs.mounts()):
    # Unmount the existing mount point
    dbutils.fs.unmount(mount_point)
    print(f"Unmounted existing mount at {mount_point}")

try:
    dbutils.fs.mount(
        source="abfss://raw@httpforstorage.dfs.core.windows.net/",
        mount_point=mount_point,
        extra_configs=configs
    )
    print(f"Mounted successfully at{mount_point}")
except Exception as e:
    print(f"Error mounting: {e}")
```

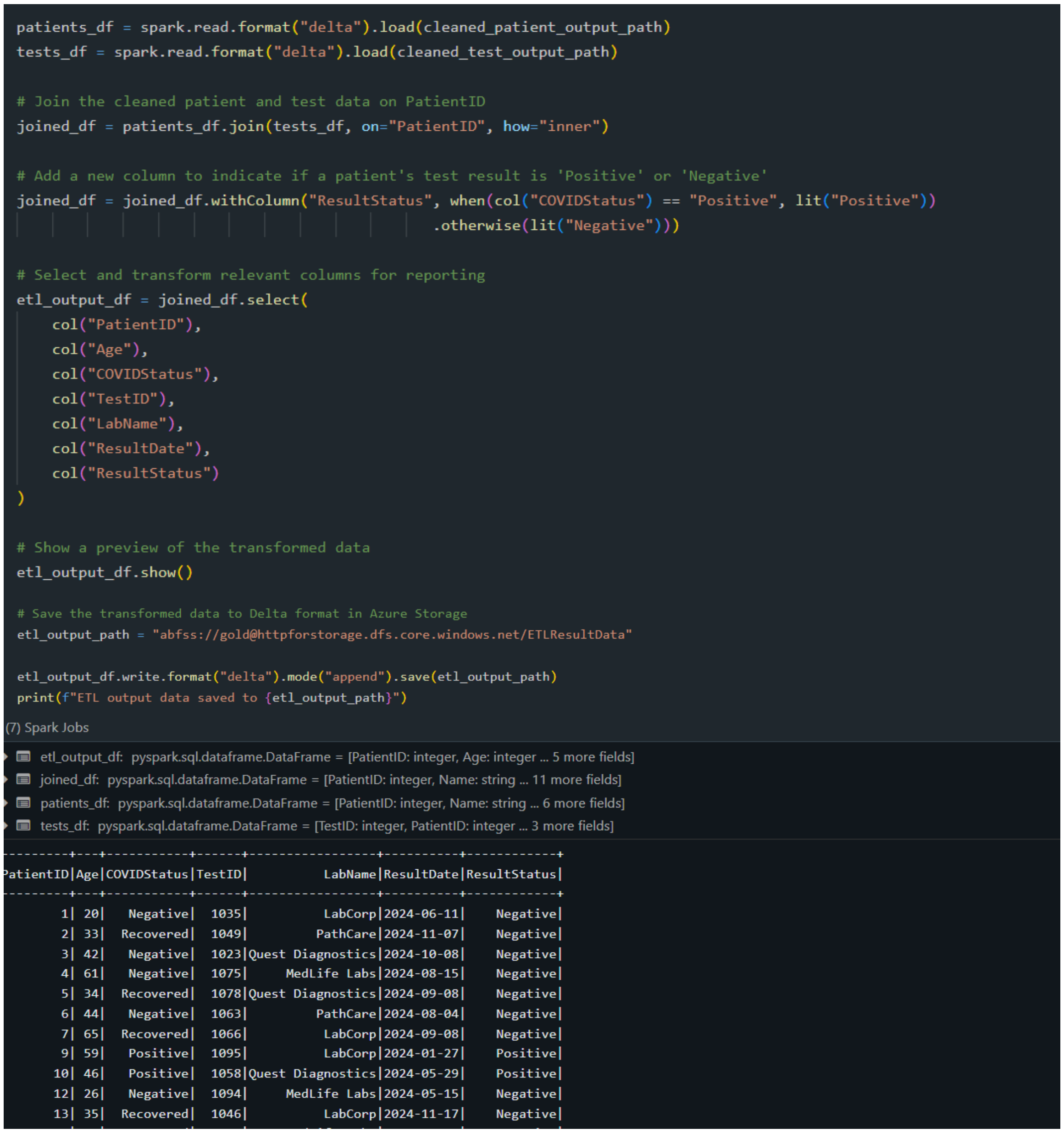
```
/mnt/httpforstorage/raw has been unmounted.
Unmounted existing mount at /mnt/httpforstorage/raw
Mounted successfully at/mnt/httpforstorage/raw
```

Raw to Processed Zone:
- Loaded the raw CSV files into Databricks notebooks using PySpark.
- Applied data cleaning: null value handling, duplicate removal, and schema standardization.
- Transformed the data into a structured Parquet format and stored it in the processed zone.



```python
# --- Data Cleaning on Patients Data ---
# Drop rows with missing PatientID
patients_df = patients_df.filter(col("PatientID").isNotNull())

# Cast PatientID and Age to Integer and TestDate to Date
def clean_patient_data(df):
    return (df.withColumn("PatientID", col("PatientID").cast(IntegerType()))
            .withColumn("Age", col("Age").cast(IntegerType()))
            .withColumn("TestDate", col("TestDate").cast(DateType())))

patients_df = clean_patient_data(patients_df)


patients_df = patients_df.withColumn("COVIDStatus", when(col("COVIDStatus").isNull(), lit("Unknown"))
                                     .otherwise(col("COVIDStatus")))


tests_df = tests_df.filter(col("TestID").isNotNull() & col("PatientID").isNotNull())


def clean_test_data(df):
    return (df.withColumn("TestID", col("TestID").cast(IntegerType()))
            .withColumn("PatientID", col("PatientID").cast(IntegerType()))
            .withColumn("ResultDate", col("ResultDate").cast(DateType())))

tests_df = clean_test_data(tests_df)


tests_df = tests_df.withColumn("LabName", when(col("LabName").isNull(), lit("Unknown"))
                               .otherwise(col("LabName")))

patients_df.select(countDistinct("PatientID").alias("Unique Patients")).show()
tests_df.select(countDistinct("TestID").alias("Unique Tests")).show()

# Save cleaned data as Parquet files and use Delta Lake for appending
cleaned_patient_file_path = "/mnt/httpforstorage/processed/CleanedPatientData"
cleaned_test_file_path = "/mnt/httpforstorage/processed/CleanedTestResults"
cleaned_patient_output_path = "abfss://processed@httpforstorage.dfs.core.windows.net/CleanedPatientData"
cleaned_test_output_path = "abfss://processed@httpforstorage.dfs.core.windows.net/CleanedTestResults"

patients_df.write.format("delta").mode("append").save(cleaned_patient_file_path)
tests_df.write.format("delta").mode("append").save(cleaned_test_file_path)
patients_df.write.format("delta").mode("overwrite").save(cleaned_patient_output_path)
print(f"Cleaned patient data saved to {cleaned_patient_output_path}")

tests_df.write.format("delta").mode("append").save(cleaned_test_output_path)
print(f"Cleaned test data saved to {cleaned_test_output_path}")

print(f"Cleaned data saved to {cleaned_patient_file_path} and {cleaned_test_file_path} using Delta Lake")
```

Processed to Gold Zone:
- Enhanced the dataset with business logic: aggregations, key metrics, and dimensional modeling.
- Stored the transformed data in the gold zone as Parquet files.



```python
patients_df = spark.read.format("delta").load(cleaned_patient_output_path)
tests_df = spark.read.format("delta").load(cleaned_test_output_path)

# Join the cleaned patient and test data on PatientID
joined_df = patients_df.join(tests_df, on="PatientID", how="inner")

# Add a new column to indicate if a patient's test result is 'Positive' or 'Negative'
joined_df = joined_df.withColumn("ResultStatus", when(col("COVIDStatus") == "Positive", lit("Positive"))
                                               .otherwise(lit("Negative")))

# Select and transform relevant columns for reporting
etl_output_df = joined_df.select(
    col("PatientID"),
    col("Age"),
    col("COVIDStatus"),
    col("TestID"),
    col("LabName"),
    col("ResultDate"),
    col("ResultStatus")
)

# Show a preview of the transformed data
etl_output_df.show()

# Save the transformed data to Delta format in Azure Storage
etl_output_path = "abfss://gold@httpforstorage.dfs.core.windows.net/ETLResultData"

etl_output_df.write.format("delta").mode("append").save(etl_output_path)
print(f"ETL output data saved to {etl_output_path}")
```

```
(7) Spark Jobs
etl_output_df:  pyspark.sql.dataframe.DataFrame = [PatientID: integer, Age: integer ... 5 more fields]
joined_df:  pyspark.sql.dataframe.DataFrame = [PatientID: integer, Name: string ... 11 more fields]
patients_df:  pyspark.sql.dataframe.DataFrame = [PatientID: integer, Name: string ... 6 more fields]
tests_df:  pyspark.sql.dataframe.DataFrame = [TestID: integer, PatientID: integer ... 3 more fields]
```

```
--------+---+-----------+------+-----------------+----------+------------+
atientID|Age|COVIDStatus|TestID|          LabName|ResultDate|ResultStatus|
--------+---+-----------+------+-----------------+----------+------------+
       1| 20|   Negative|  1035|          LabCorp|2024-06-11|    Negative|
       2| 33|  Recovered|  1049|         PathCare|2024-11-07|    Negative|
       3| 42|   Negative|  1023|Quest Diagnostics|2024-10-08|    Negative|
       4| 61|   Negative|  1075|     MedLife Labs|2024-08-15|    Negative|
       5| 34|  Recovered|  1078|Quest Diagnostics|2024-09-08|    Negative|
       6| 44|   Negative|  1063|         PathCare|2024-08-04|    Negative|
       7| 65|  Recovered|  1066|          LabCorp|2024-09-08|    Negative|
       9| 59|   Positive|  1095|          LabCorp|2024-01-27|    Positive|
      10| 46|   Positive|  1058|Quest Diagnostics|2024-05-29|    Positive|
      12| 26|   Negative|  1094|     MedLife Labs|2024-05-15|    Negative|
      13| 35|  Recovered|  1046|          LabCorp|2024-11-17|    Negative|
```

3. Analytics with Azure Synapse

Objective: Load the gold-zone data into a Synapse Dedicated SQL Pool for optimized analytics.

Steps:
- Created a Synapse dedicated pool and configured table schemas.
- Leveraged Synapse to define external tables pointing to the gold-zone files for easier integration.

Error Handling and Optimizations

1. Parameter Errors in ADF

Issue: Incorrect parameter passing for API endpoints resulted in data fetch failures.

Resolution:
- Used debug mode in ADF pipelines to identify incorrect values.
- Introduced default values and validation logic in pipeline parameters to handle edge cases.

2. ETL Performance in Databricks

Issue: High latency in transformations due to inefficient Spark queries.

Resolution:
- Optimized transformations by enabling DataFrame caching.
- Used partition pruning to process only the required data.
- Tuned the cluster configuration (e.g., increased node size and parallelism).

Key Learnings and Best Practices

1. **Parameterize Pipelines**: Always parameterize data pipelines for flexibility and reusability.
2. **Monitor and Debug**: Leverage ADF's monitoring tools to track pipeline execution and quickly resolve errors.
3. **Optimize Transformations**: Optimize Spark jobs with efficient partitioning, caching, and query tuning.
4. **Synapse Integration**: Use dedicated SQL pools for structured analytics and external tables for large file processing.

Conclusion

The project demonstrated how Azure services could be seamlessly integrated to build a robust, scalable, and efficient data pipeline. The solution streamlined data ingestion, transformation, and analytics while addressing common challenges with parameter handling and performance optimization. This pipeline serves as a foundation for future advanced analytics and machine learning use cases.