Data Engineering Exercise

AZURE

Document Version: [V2]

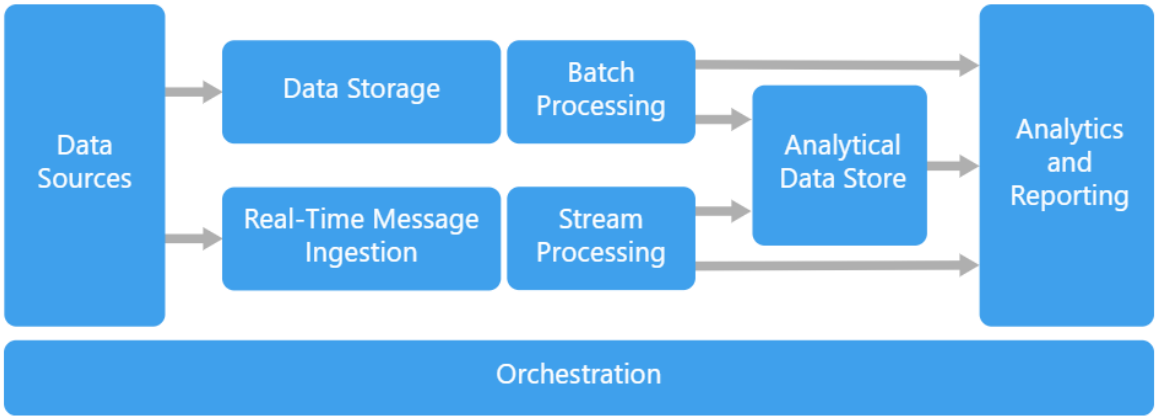
Date: [05/24/2021]

Contents

1. Big data architecture style

This section describes the common Big data architecture patten for big data on public cloud (Azure).

A big data architecture is designed to handle the ingestion, processing, and analysis of data that is too large or complex for traditional database systems



Most of Big data solutions typically involve one or more of the following types of workload:

- Batch processing of big data sources at rest.
- Real-time processing of big data in motion.
- Interactive exploration of big data.
- Predictive analytics and machine learning.

Use Case -

- Store and process data in volumes too large for a traditional database.
- Transform unstructured data for analysis and reporting.
- Capture, process, and analyze unbounded streams of data in real time, or with low latency.
- Use Azure Machine Learning or Microsoft Cognitive Services.

Benefits -

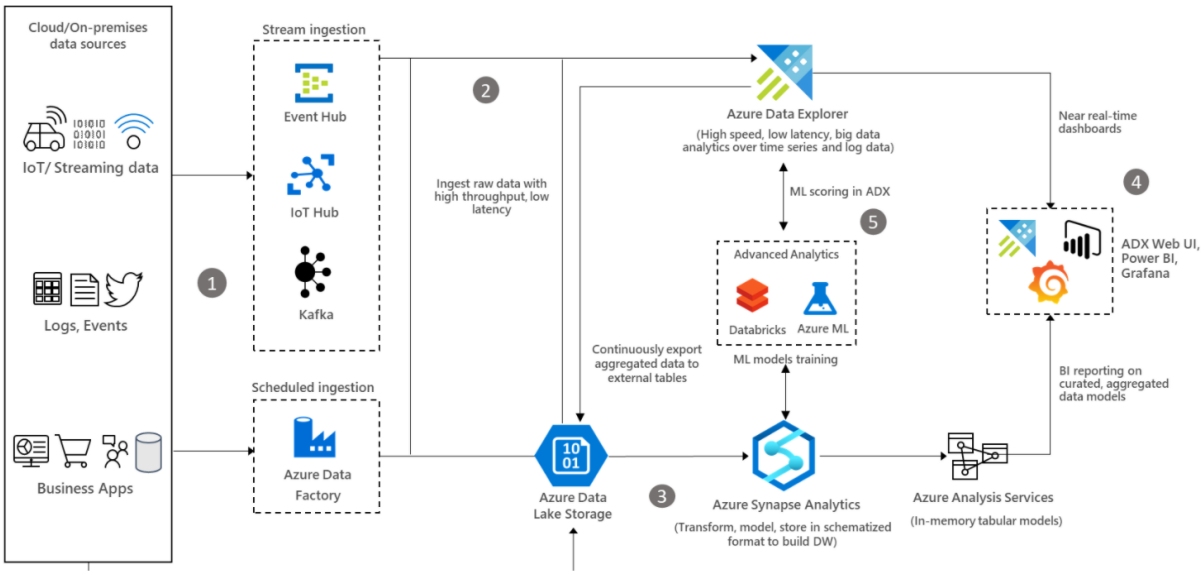
Technology choices- You can mix and match Azure managed services and Apache technologies in HDInsight clusters to capitalize on existing skills or technology investments.

Performance through parallelism- Big data solutions take advantage of parallelism enabling high-performance solutions that scale to large volumes of data.

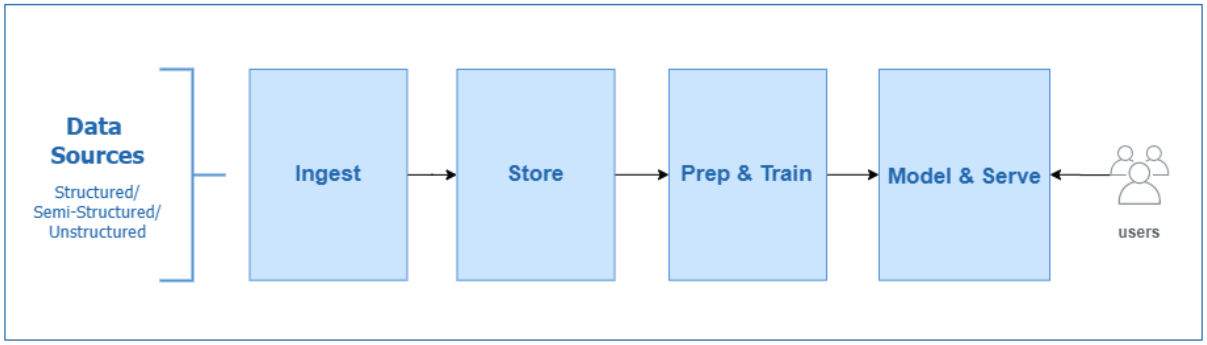
Elastic scale- All the components in the big data architecture support scale-out provisioning so that you can adjust your solution to small or large workloads and pay only for the resources that you use.

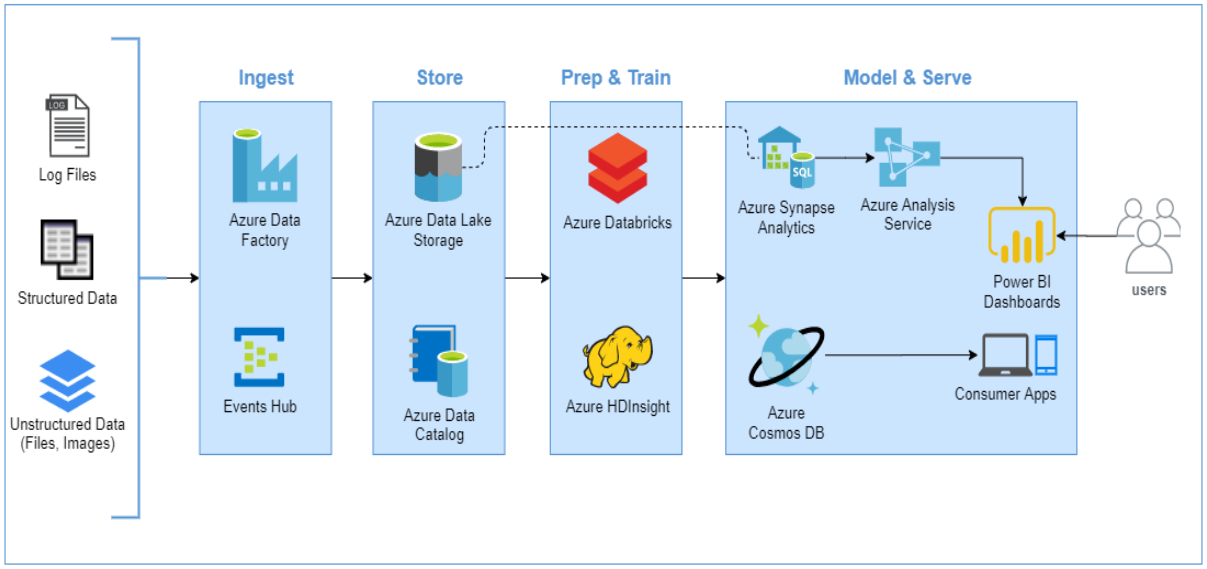
Interoperability with existing solutions- The components of the big data architecture are also used for IoT processing and enterprise BI solutions, enabling you to create an integrated solution across data workloads.

1.1. *Technical* architecture *for Big Data*



1.2. *Common Data Flow for Big Data Project*





Ingest- The data is ingested using Azure Data Factory. The Azure Data Factory is a cloud based ETL tool for data integration and transformation. For streaming data ingestion, Azure

Events Hub service is used to build real-time data pipelines. It can stream millions of events per second from any source and seamlessly integrate with other Azure data services.

Store- For storage, Azure Data Lake Storage is the ideal candidate to provide a secure and massively scalable data store for all kinds of data sets with capabilities dedicated to big data analytics. Whereas the Data Catalog service is used to create a metadata catalog which stores information about business, technical and operational metadata and helps simplifying data asset discovery.

Prep & Train- For data preparation and training, both Azure HDInsight and Azure Databricks provide capabilities to not only clean, transform and enrich unstructured datasets but also apply machine learning techniques for building and training models and get deeper data insights. The users can create in-built notebooks with support of programming languages i.e. Scala, Python or R.

Model & Serve- The results of data processing are then stored in the data warehouse and analytics applications such as Azure Synapse Analytics to be used by Power BI reports and dashboards generated by the user. While the insights gained from machine-learning models are moved to Azure Cosmos DB for access through consumer applications.

2. Data Engineering Exercise

This section describes the DE exercise for Data solution on Azure public cloud.

2.1. Exercise 1

Write a python script to simulate the following three Tables:

Table 1: High frequency timeseries accelerometer (3-axis) and gyroscope (3-axis) data (50 Hz, duration of 5 min) for 200 subjects

Table 2: Demographic information for 200 subjects (Any 5 demographic variables)

Table 3: Programming information including program name, amplitude, frequency and pulse width for 200 subjects

Use random variable generators to create above tables and make sure the range of values are reasonable.

Type – Sample source data generation

Solution – Approach

Step 1 – Create Python virtual environment
Step 2 – Install required libs (Planning to use Faker and pandas-profiling). F*aker* is a Python package that generates fake data for any data project.
Step 3 – Build a python script to generate 200 Unique subject Data (Demographic and multiple Programs for Subjects).
Step 4 – Use previously generated 200 random subjects to generate timeseries : High frequency data.
Step 5 – Generate profile result for all three data sets
Step 6 – Save sample data sets in .csv format along with profile results(.HTML format)

Code Location

GitHub Link - https://github.com/vishu22314558/ABT_DE_Exercise_V2-main.git
Code repo Name - ABT_DE_Exercise_V2-main

Script to generate Sample Data -
- ABT_DE_Exercise_V2-main/PythonCode/Exercise-1.py

Script for generating Source data profile (jupyter notebook) -
ABT_DE_Exercise_V2-main/PythonCode/Source_Data_Profile.ipynb

Generated Sample data - ABT_DE_Exercise_V2-main/Sample_Data/
Data Set Name - Subject_Program.csv , Subject_Demographic.csv and Subject_Accelerometer.csv

HTML Profile result - ABT_DE_Exercise_V2-main/Profile_Result_Source_Data/
Profile Name - df_Subject_Program.html , df_Subject_Demographic.html and df_Subject_Accelerometer.html

2.2. Exercise 2

Write the three Tables (from step 1) to Azure SQL database (Secured with firewall)

Type – Data Ignition and Storage

Solution – Approach

We have three sample data files, So I will be trying two or three different approaches to load data into SQL Server.

Infrastructure Setup –

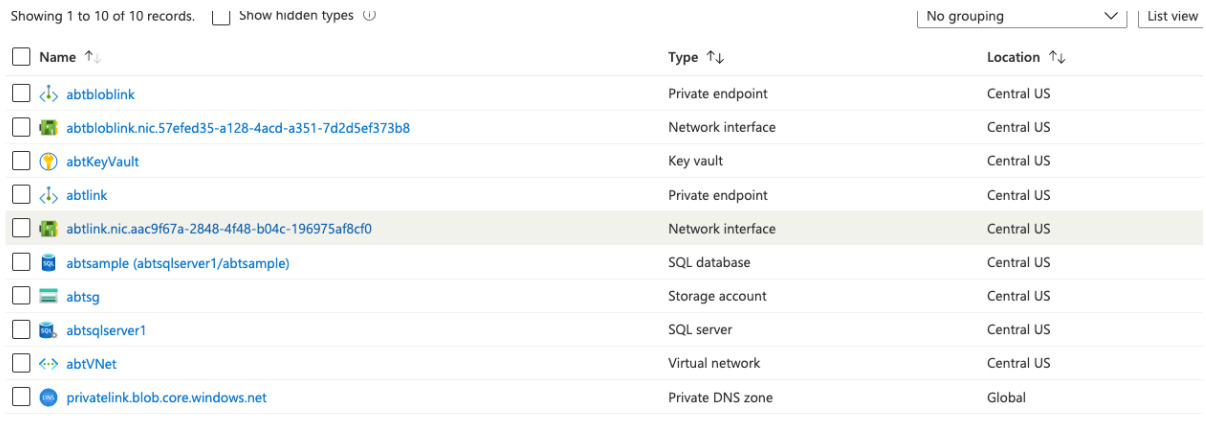
Create resource group “ABT-Exercise-rg” using PowerShell

Create AZURE SQL Serve instance (**abtsqlserver1**) and create database(**abtsample)** using PowerShell or UI and block all access except trusted IP.

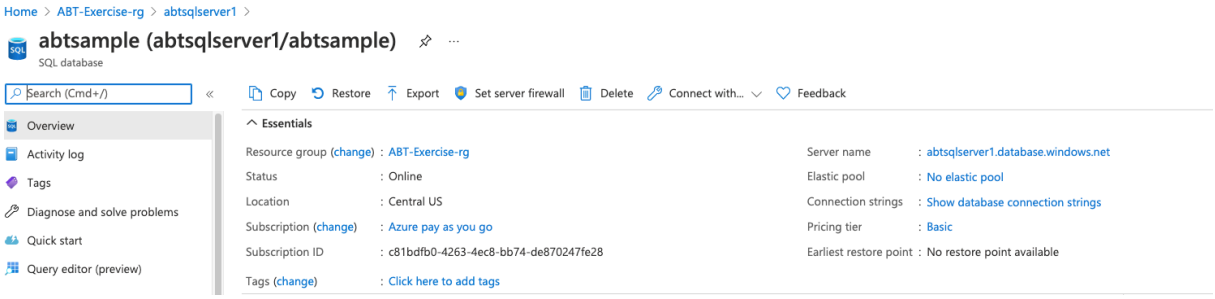
Databases in SQL Database are protected by firewalls in Azure. By default, all connections to the server and database are rejected.

Other Security Features
Transparent data encryption - The encryption feature automatically encrypts your data at rest and requires no changes to applications accessing the encrypted database. It is on by default.
Dynamic data masking - The data masking feature will automatically hide sensitive data in your database.
Auditing - The auditing feature tracks database events and writes events to an audit log in either Azure storage, Azure Monitor logs, or to an event hub. Auditing helps maintain regulatory compliance, understand database activity, and gain insight into discrepancies and anomalies that could indicate potential security violations.
Transport Layer Security (Encryption-in-transit) - SQL Database, SQL Managed Instance, enforce encryption (SSL/TLS) at all times for all connections. This ensures all data is encrypted "in transit" between the client and server irrespective of the setting of **Encrypt** or **TrustServerCertificate** in the connection string.

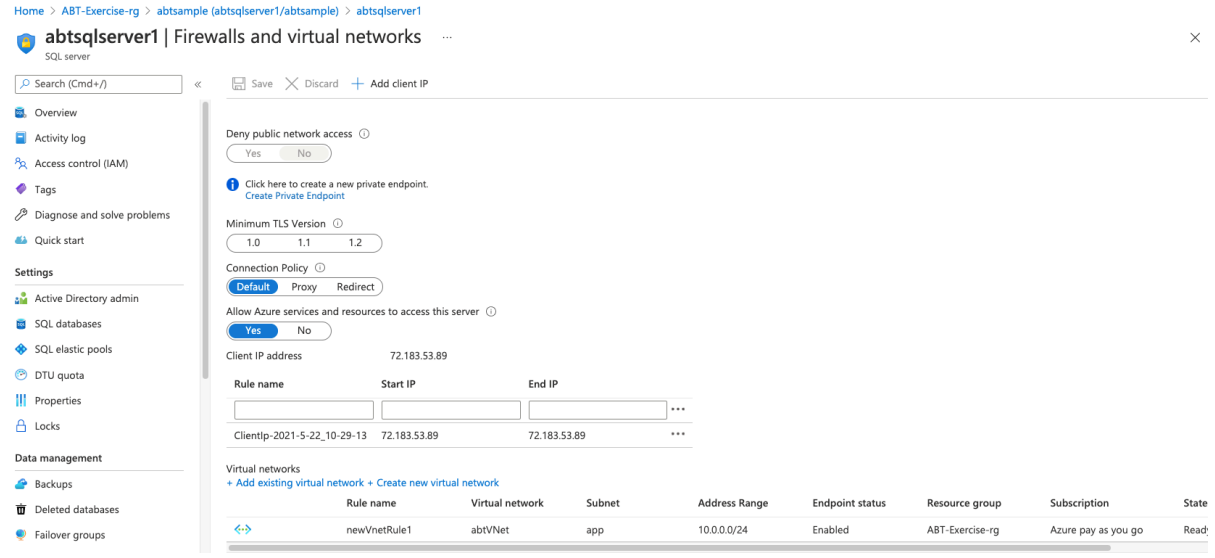
Sever Setup -
Azure SQL Sever - instance (**abtsqlserver1**) and database(**abtsample)**



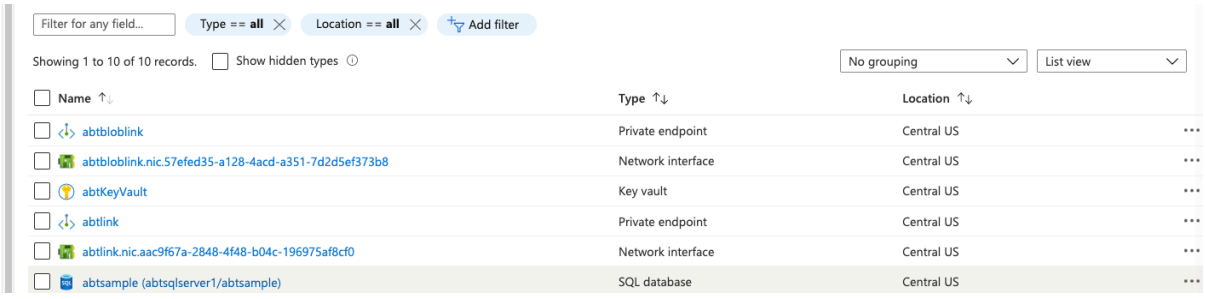
Database -



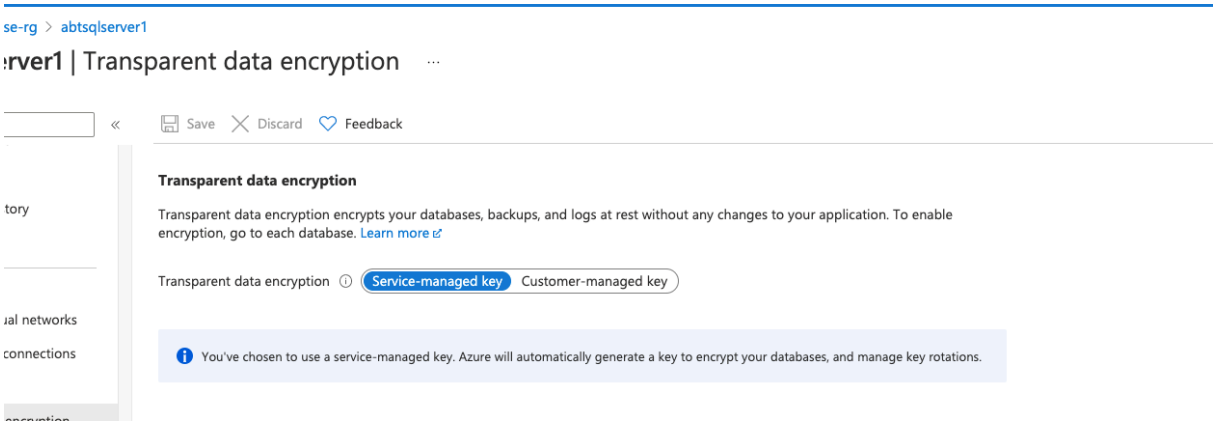
Network Connectivity - Firewall - No Public Access Connected to Vent - abtVNet Allowed Trusted IP



Private Link between Blob and SQL Sever



Transparent data encryption - Encryption at rest



File 1 (Subject_Demographic.csv) -> SQL Severe

Step 1 – Generate DDL for first table and execute it on SQL Server.
Step 2 - Build a python script to read .csv file and make connection to sql server and ingest the data.
Step 3 – login into SQL Server(Database) instance and verify the data in table.

Code Location

GitHub Link - https://github.com/vishu22314558/ABT_DE_Exercise_V2-main.git
Code repo Name - ABT_DE_Exercise_V2-main

DDL Location - ABT_DE_Exercise_V2-main/DataBaseObjects/
Script Name -SubjectDemographic.ddl

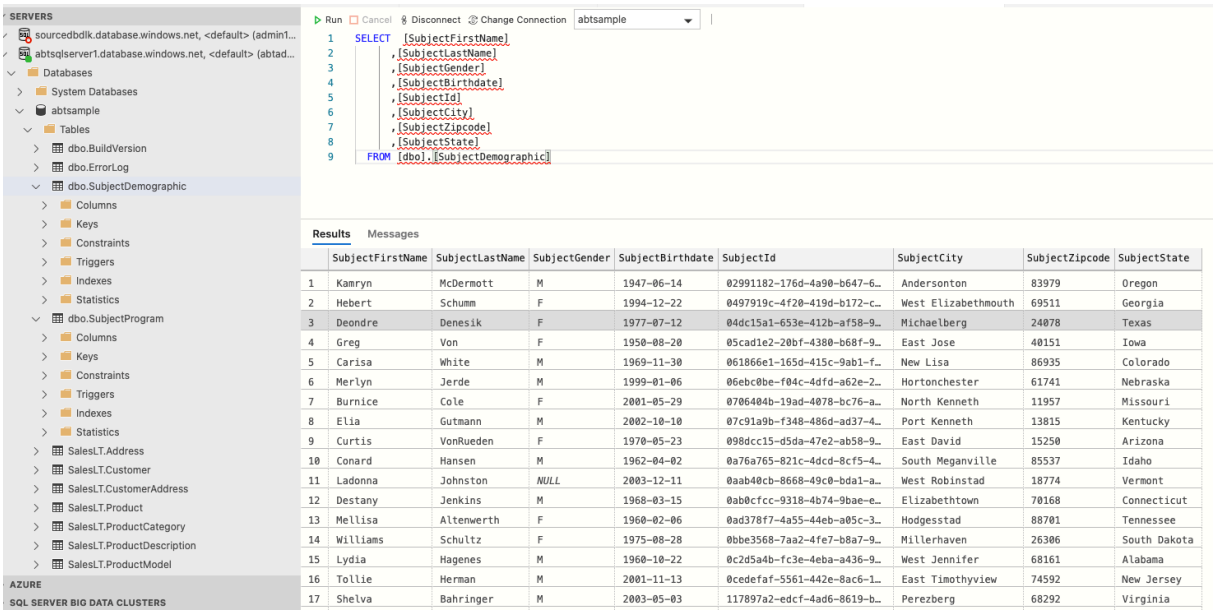
Python Script Location - ABT_DE_Exercise_V2-main/PythonCode/
Script Name - Exercise-2-File1.py

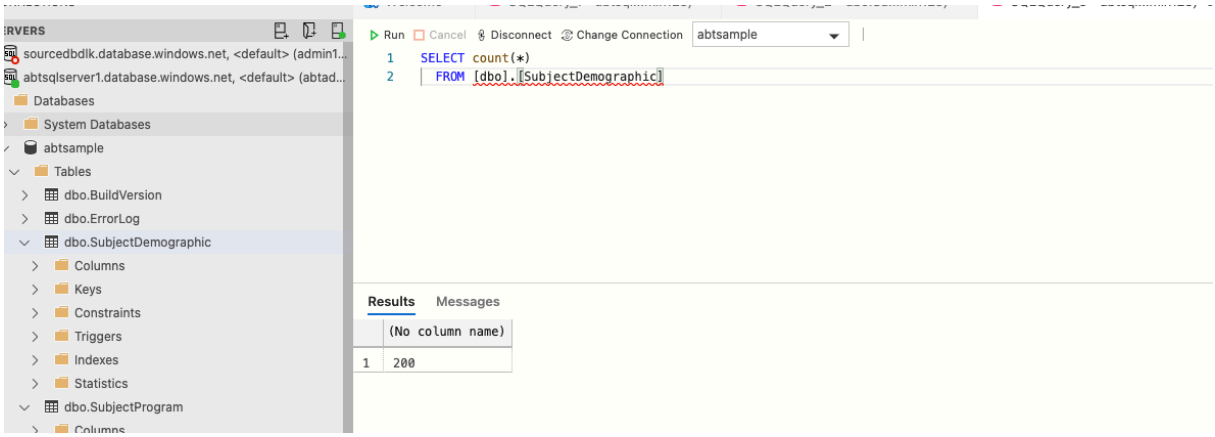
Note - Used System OS variable to secure the SQL Server credential

```
SQL_SERVER = os.environ.get("SQLSERVER")
logging.info("SQL Server Name :%s" , SQL_SERVER)
DATABASE_NAME = os.environ.get("DATABASE")
logging.info("SQL DATABASE Name :%s" , DATABASE_NAME)

USER_NAME = os.environ.get("SQLUSERNAME")
PASSWORD = os.environ.get("PASSWORD")
```

Note - Another Approach would be to create VM in Vnet and use private link to read data from Blob and ingest into SQL Server





File 2 (Subject_Accelerometer.csv) –> Blob -> Azure data factory ->SQL Server

Step 1 – Create Blob or ADLS gen2 storage account using power shell in ABT-Exercise-rg resource group
Step 2 – Use Apache Nifi to ingest data into Blob storage using Shared access signature or access keys.
Step 3 – Create Azure data Factory flow to load data into SQL Server. This flow will create the target table as part of data pipeline.
Step 4- login into SQL Server instance and verify the data in table.

Note – This data ingestion flow can be scheduled to run on a particular type of event or on schedule basis. We can also use ADLS Gen2 storage type to store the data if data volume is huge.

ADLS Gen2 (Azure Data Lake Storage) - Optimized storage for big data analytics workloads and can be used for Batch, interactive, streaming analytics and machine learning data such as log files, IoT data, click streams, large datasets.

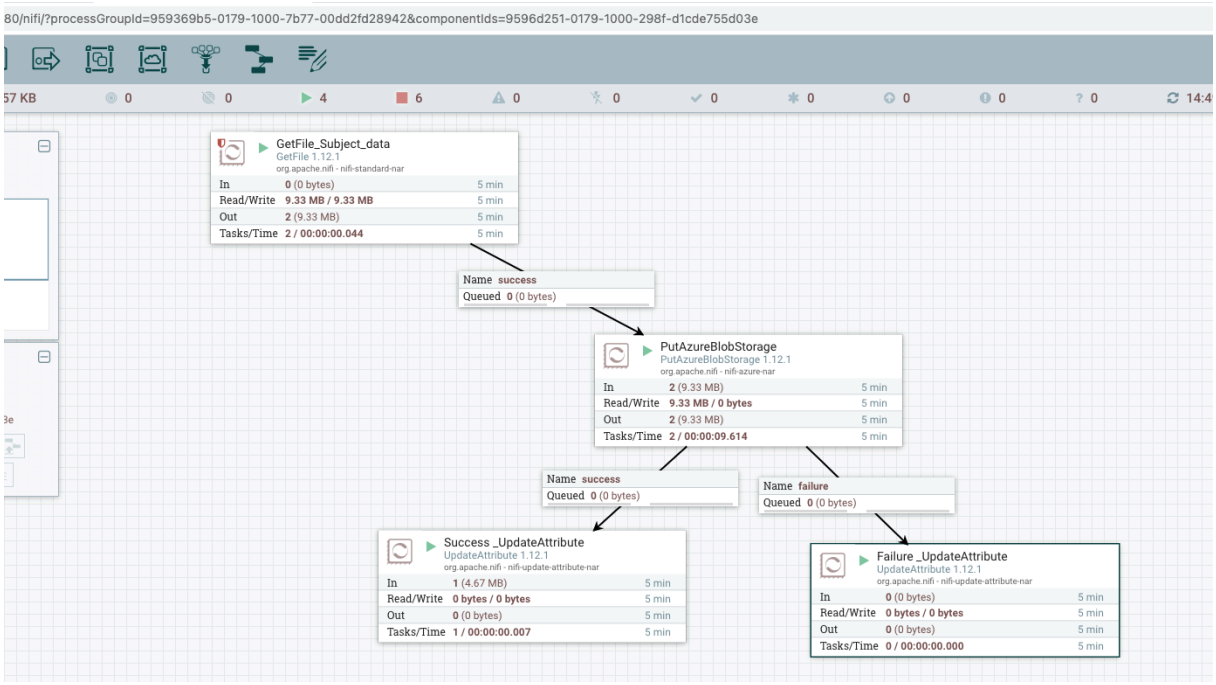
Code Location

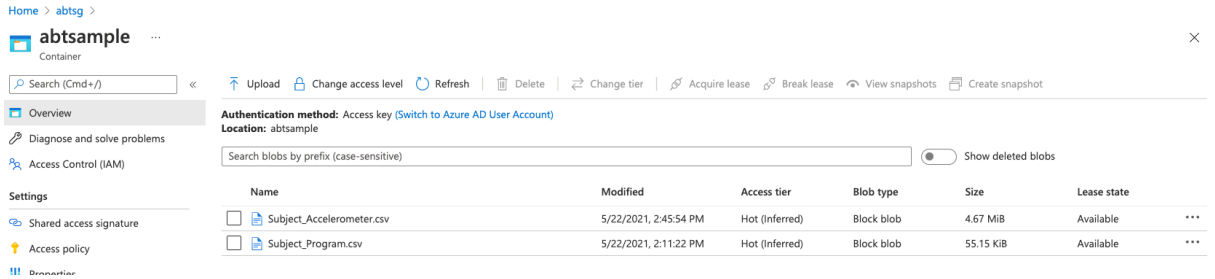
GitHub Link - https://github.com/vishu22314558/ABT_DE_Exercise_V2-main.git
Code repo Name - ABT_DE_Exercise_V2-main

Data Factory Code - ABT_DE_Exercise_V2-main/Azure_Code/CopyPipeline_DF/

Apache NiFi supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic.

Apache NiFi flow to ingest data files into Blob storage
PutAzureBlobStorage - utilizes SAS key to authenticate

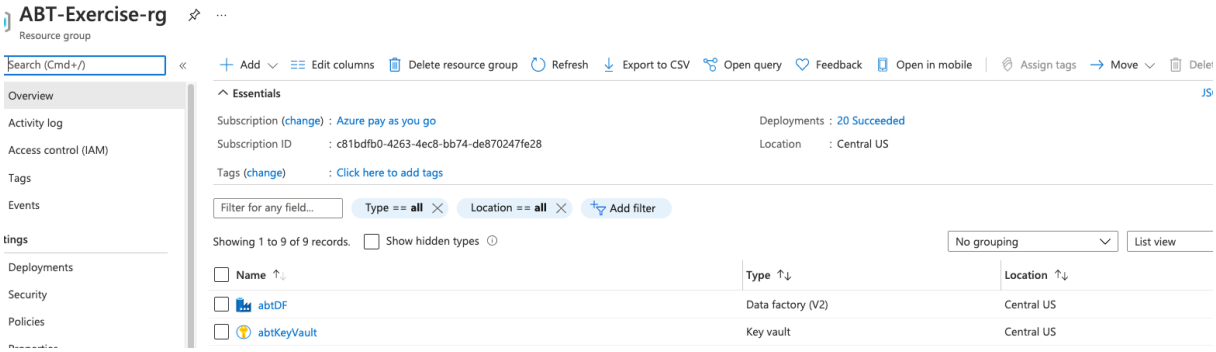


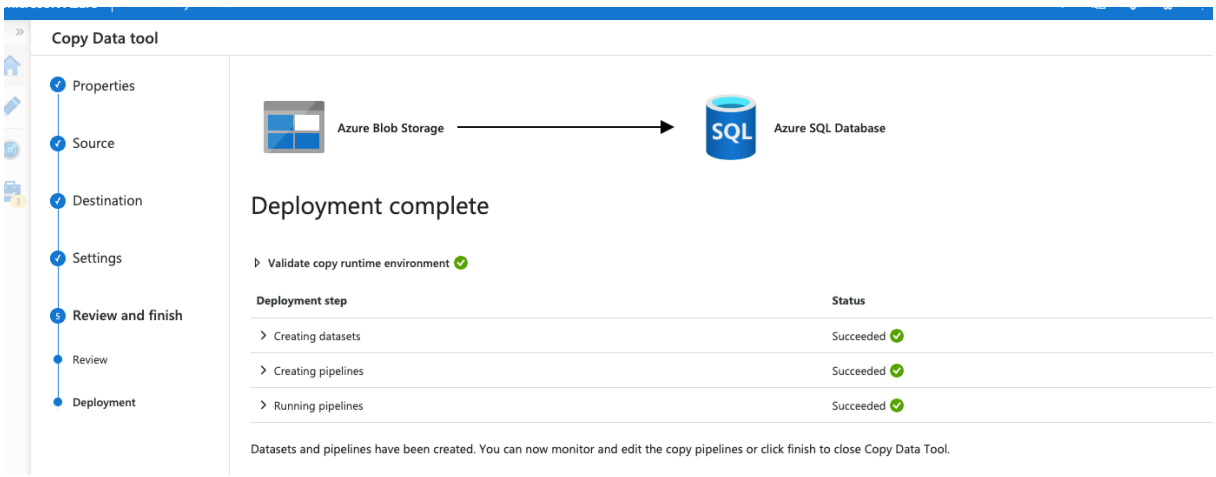


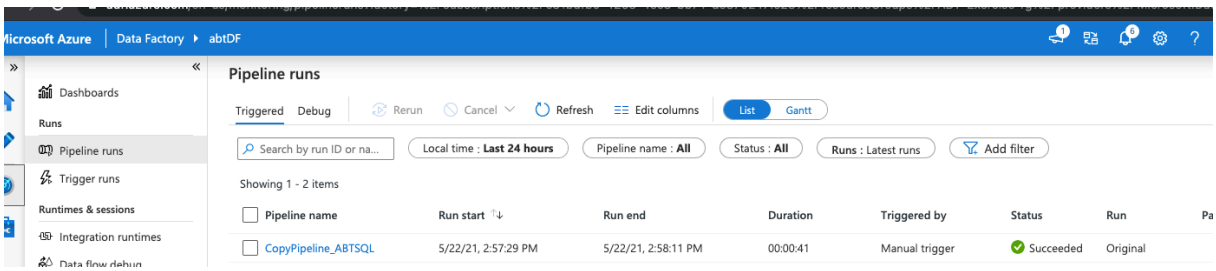
Create Copy data pipeline in DF

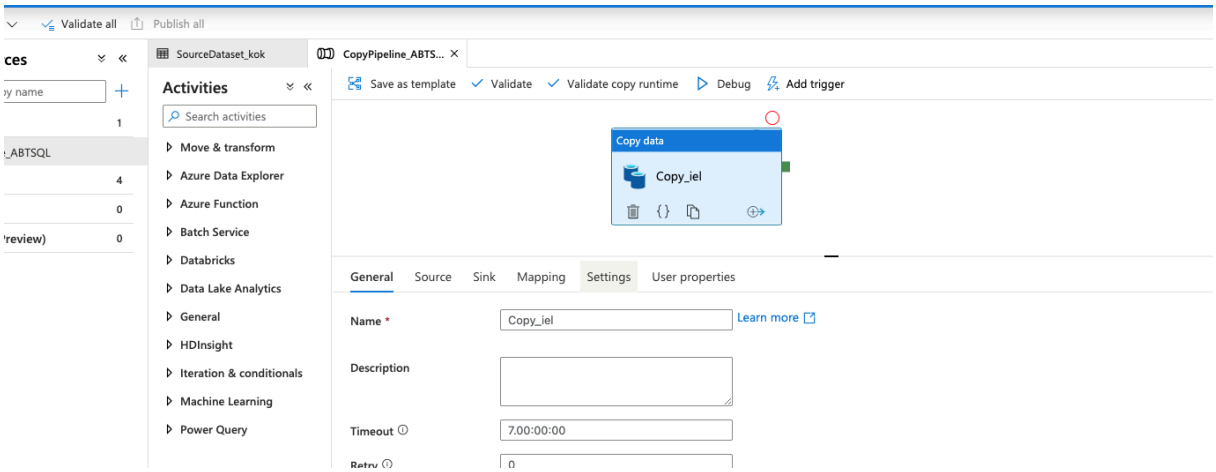
Target Table creation is part of Pipeline. This pipe line can be event/trigger driven or schedule basis.

Azure Data Factory can help us to complex spark ELT pipeline using transient **Databricks** or **Hdinsight** cluster.

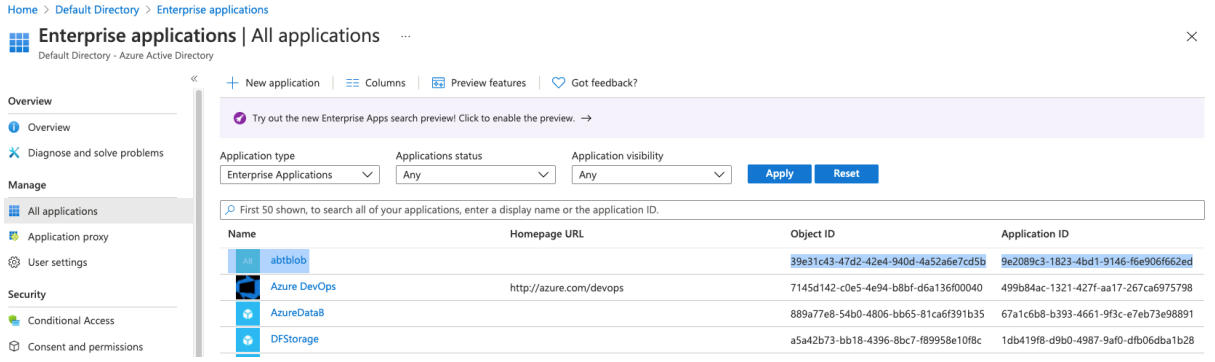


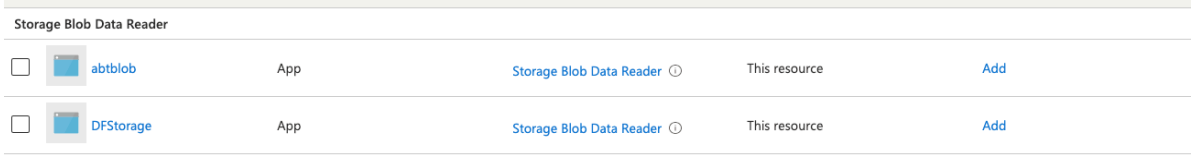




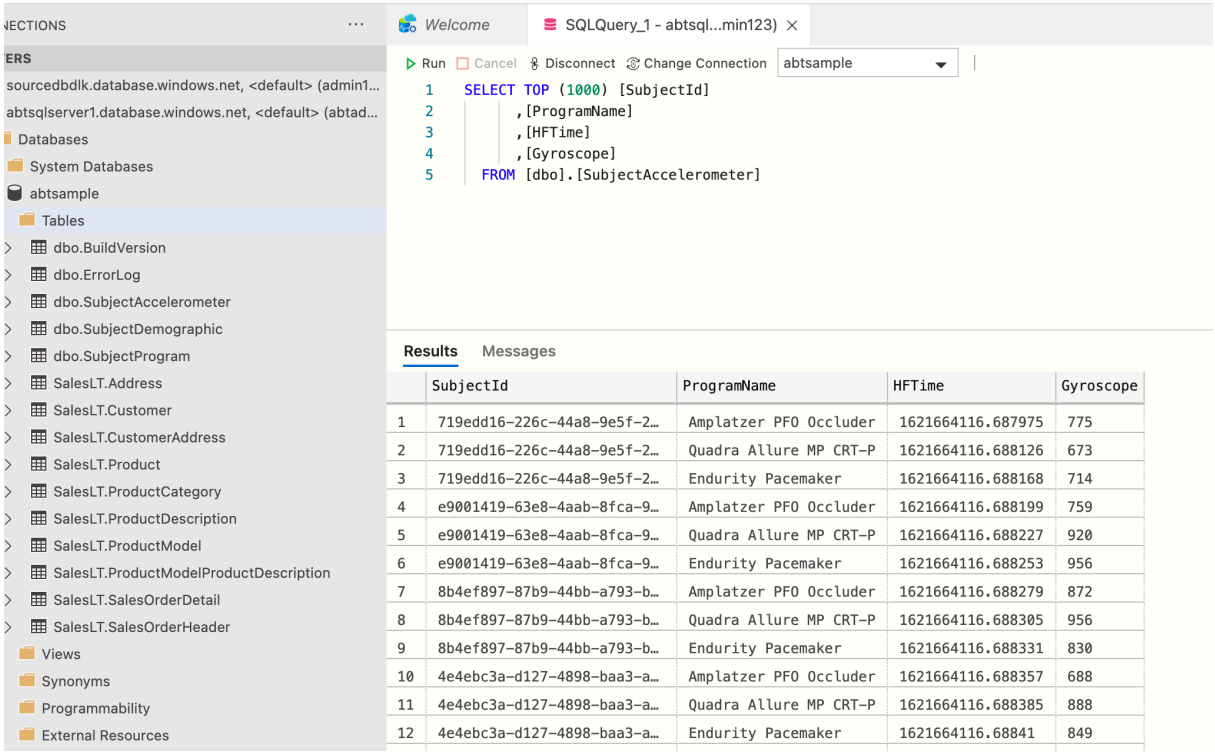


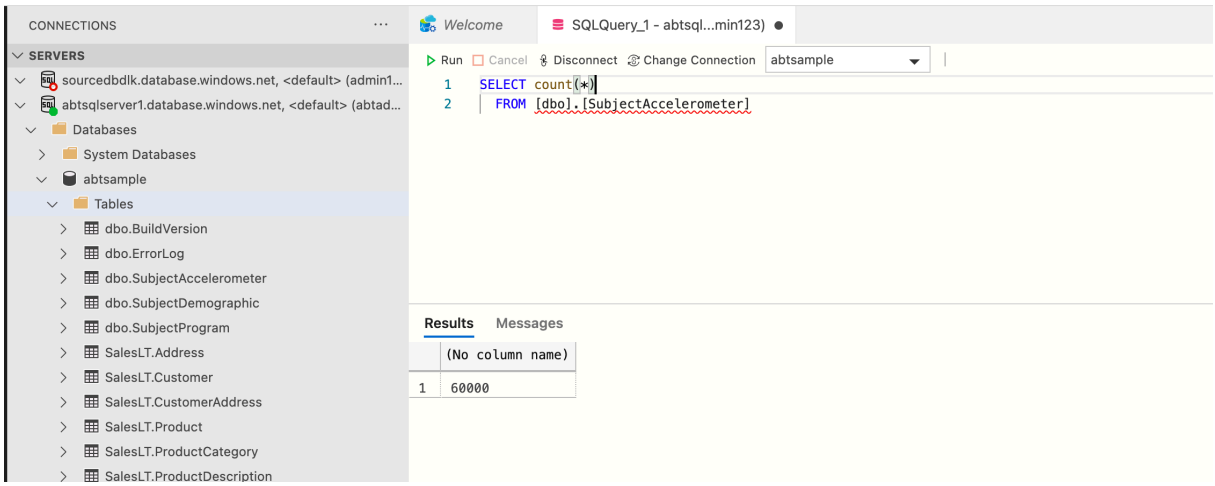
Security - Pipeline in encrypted End to End - Default
All Source Connection using Azure **Service Principle** to grant access - Read only
We can also use Azure Key vault to secure the credentials.





Target Table -





File 3 (Subject_Program.csv) –> Blob ->Databricks(Pyspark)->SQL Server

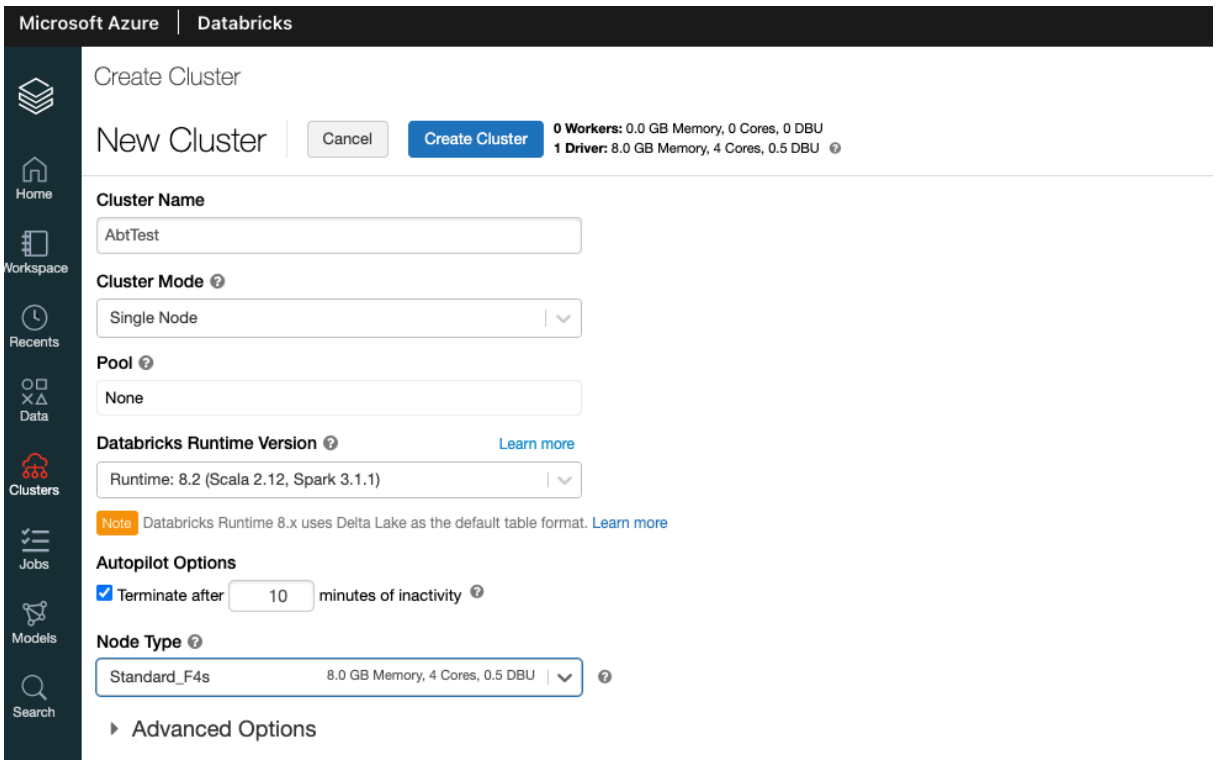
Step 1 – Create Blob or ADLS gen2 storage account using power shell in ABT-Exercise-rg resource group. - Completes in previous task
Step 2 – Use Apache Nifi to ingest data into Blob storage using Shared access signature or access keys. - Completes in previous task
Step 3 – Deploy Databricks in ABT-Exercise-rg resource group.
Step 4 - Create small one node cluster.
Step 4 - Create Python Notebook(abtSubjectProgram)
Step 5 - Build pyspark code to read data from Blob (using azure service principal) and ingest into SQL Sever.
Step 6- login into SQL Server instance and verify the data in table.
Step 7 - Terminate the cluster

Code Location

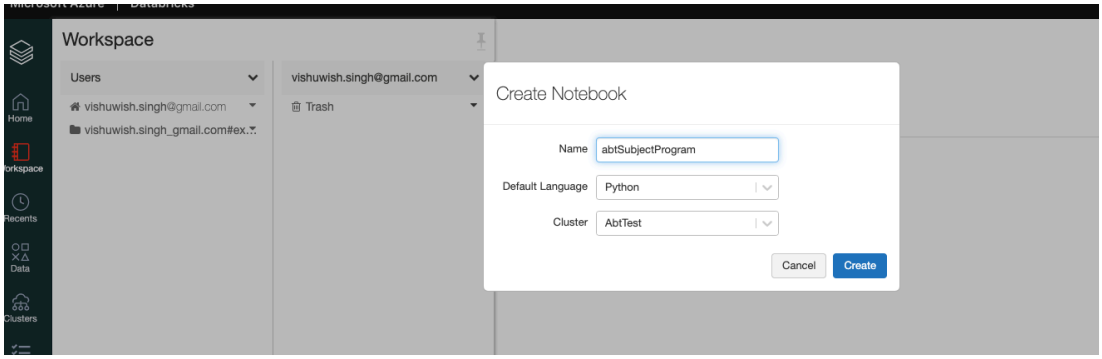
GitHub Link - https://github.com/vishu22314558/ABT_DE_Exercise_V2-main.git
Code repo Name - ABT_DE_Exercise_V2-main

Spark Code - Exercise-2-SubjetProgram_Spark.ipynb

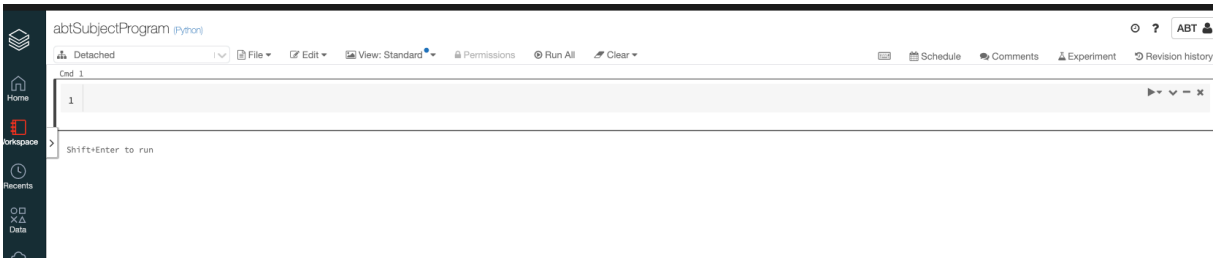
Spark Cluster Setup



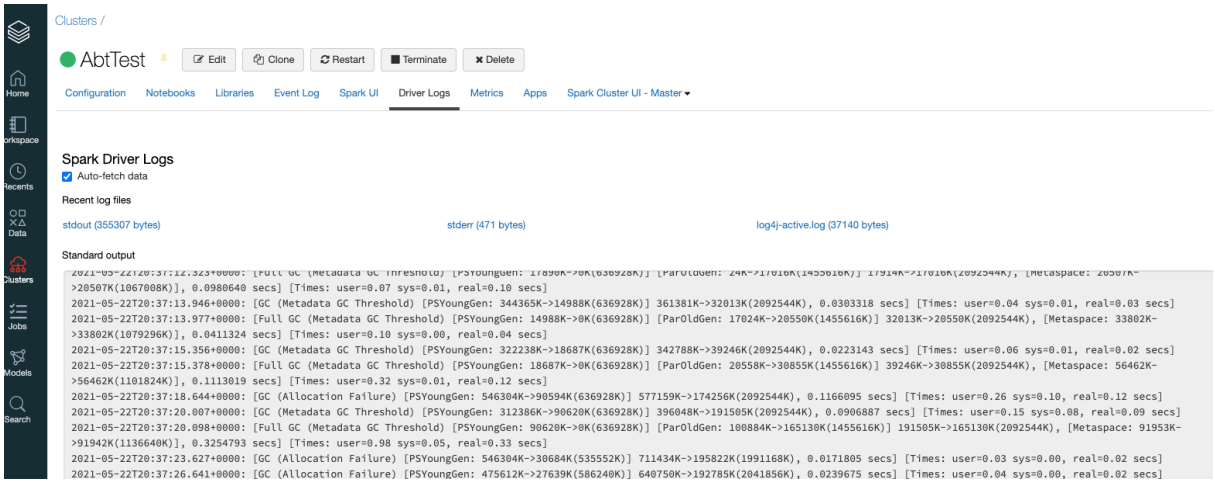
Spark Notebook Setup



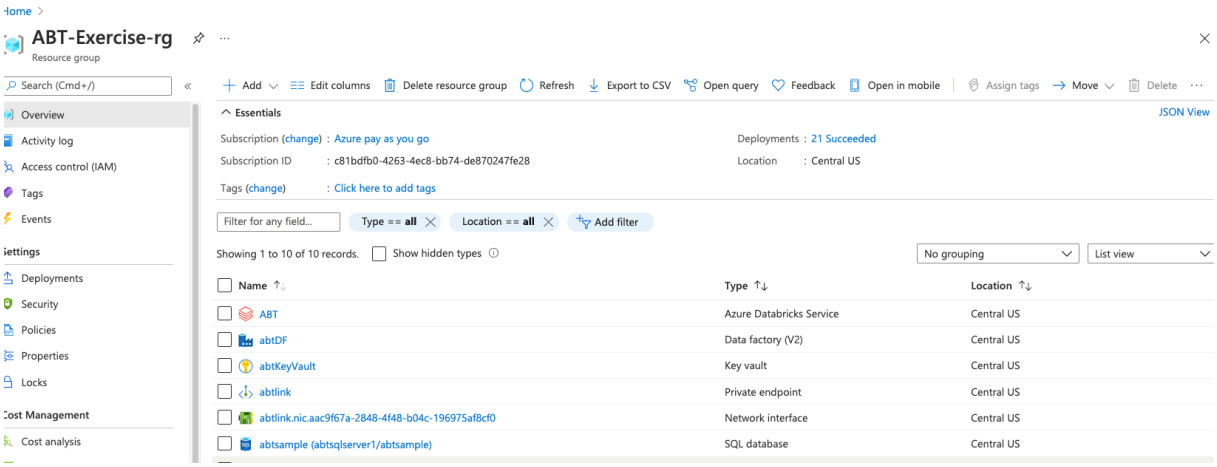
Notebook -



Cluster Driver Log -



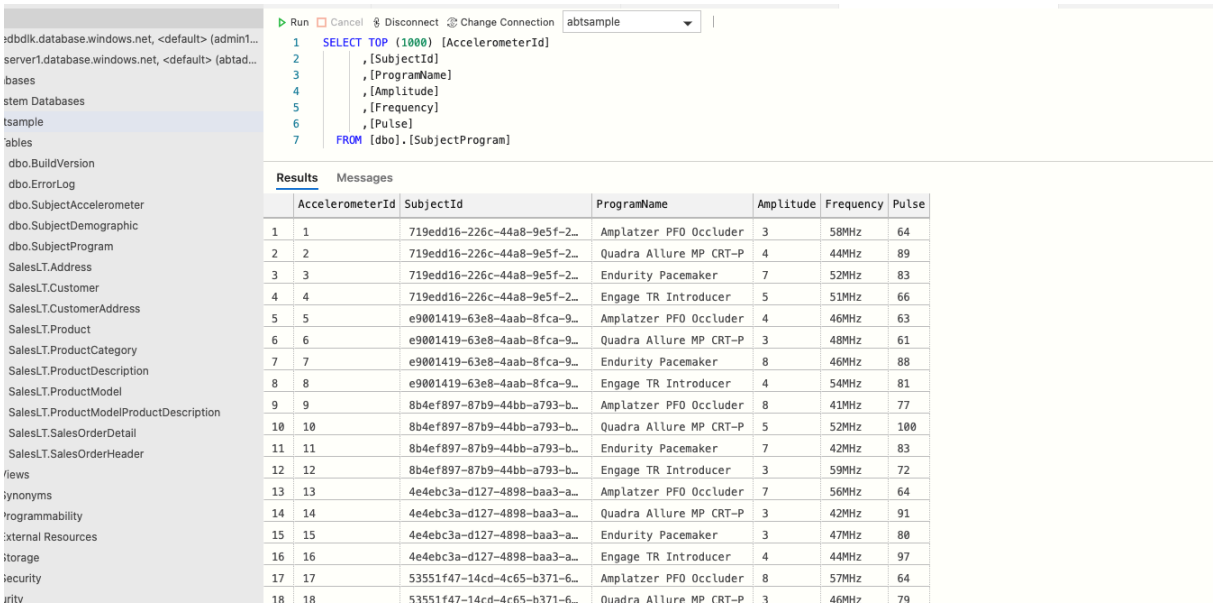
Resource Group -



PySpark code -

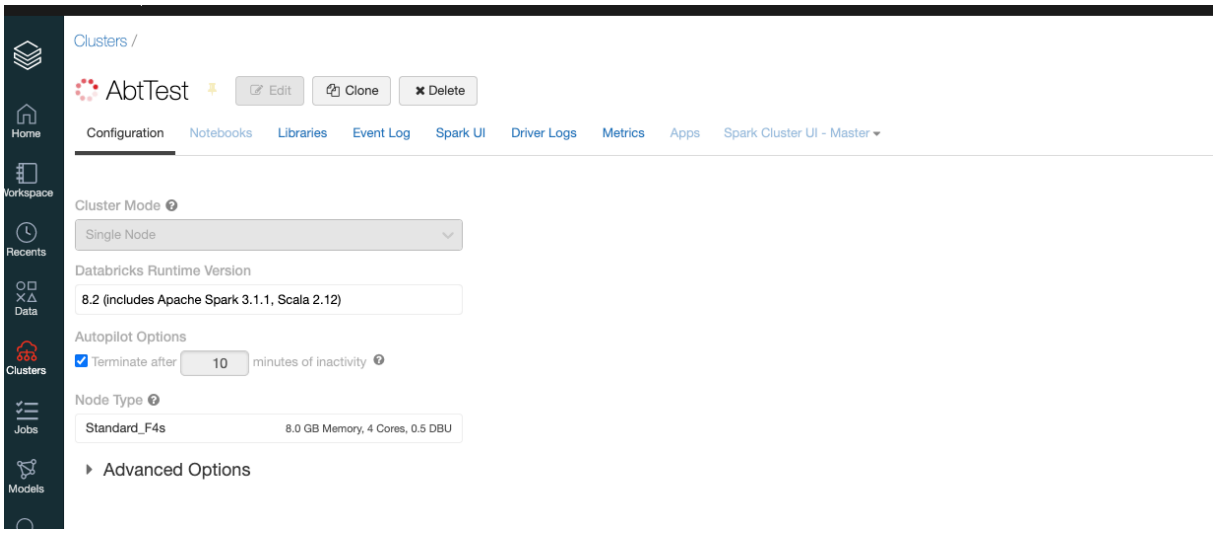
Insert data into SubjectProgram table -

Target Table -



	AccelerometerId	SubjectId	ProgramName	Amplitude	Frequency	Pulse
1	1	719edd16-226c-44a8-9e5f-2…	Amplatzer PFO Occluder	3	58MHz	64
2	2	719edd16-226c-44a8-9e5f-2…	Quadra Allure MP CRT-P	4	44MHz	89
3	3	719edd16-226c-44a8-9e5f-2…	Endurity Pacemaker	7	52MHz	83
4	4	719edd16-226c-44a8-9e5f-2…	Engage TR Introducer	5	51MHz	66
5	5	e9001419-63e8-4aab-8fca-9…	Amplatzer PFO Occluder	4	46MHz	63
6	6	e9001419-63e8-4aab-8fca-9…	Quadra Allure MP CRT-P	3	48MHz	61
7	7	e9001419-63e8-4aab-8fca-9…	Endurity Pacemaker	8	46MHz	88
8	8	e9001419-63e8-4aab-8fca-9…	Engage TR Introducer	4	54MHz	81
9	9	8b4ef897-87b9-44bb-a793-b…	Amplatzer PFO Occluder	8	41MHz	77
10	10	8b4ef897-87b9-44bb-a793-b…	Quadra Allure MP CRT-P	5	52MHz	100
11	11	8b4ef897-87b9-44bb-a793-b…	Endurity Pacemaker	7	42MHz	83
12	12	8b4ef897-87b9-44bb-a793-b…	Engage TR Introducer	3	59MHz	72
13	13	4e4ebc3a-d127-4898-baa3-a…	Amplatzer PFO Occluder	7	56MHz	64
14	14	4e4ebc3a-d127-4898-baa3-a…	Quadra Allure MP CRT-P	3	42MHz	91
15	15	4e4ebc3a-d127-4898-baa3-a…	Endurity Pacemaker	3	47MHz	80
16	16	4e4ebc3a-d127-4898-baa3-a…	Engage TR Introducer	4	44MHz	97
17	17	53551f47-14cd-4c65-b371-6…	Amplatzer PFO Occluder	8	57MHz	64
18	18	53551f47-14cd-4c65-b371-6…	Quadra Allure MP CRT-P	3	46MHz	79

Terminate cluster -



2.3. Exercise 3

Create an Azure Blob Storage and integrate it with VNet.

Type – Infrastructure and Security

Solution – Approach

Step 1 – Create Blob storage account (abtstg) using power shell in ABT-Exercise-rg resource group.
Step 2 – Create abtVNet VPC using power shell.
Step 3 – Integrate abtstg with abtVNet using service endpoint and/or private endpoint

Note - Service endpoints allow us to secure your critical Azure service resources to only your virtual networks. Traffic from your VNet to the Azure service always remains on the Microsoft Azure backbone network.
Private Endpoint is a network interface that connects you privately and securely to a service powered by Azure Private Link. Private Endpoint uses a private IP address from your VNet, effectively bringing the service into your VNet.

Both appear to allow a direct connection to the services but the key difference here appears to be in the wording. A service endpoint extends your VNet to the Azure service, whereas the private endpoints extend the service into your VNet (using an IP address in your subnet.

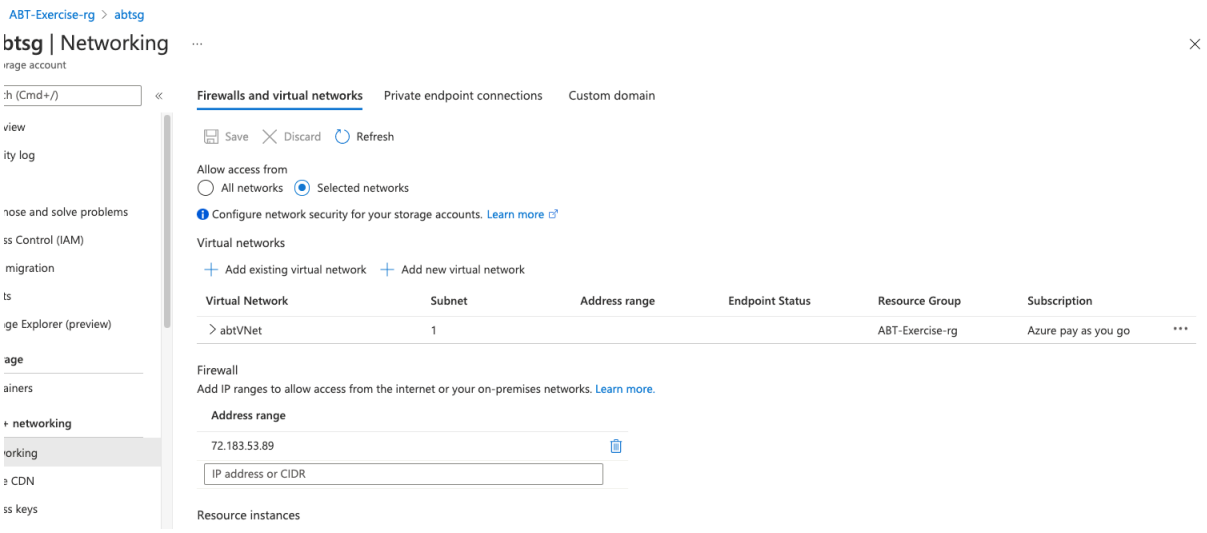
Code Location

GitHub Link - https://github.com/vishu22314558/ABT_DE_Exercise_V2-main.git
Code repo Name - ABT_DE_Exercise_V2-main

PowerShell Code - .ABT_DE_Exercise_V2-main/Azure_Code/Exercise-3.ps1

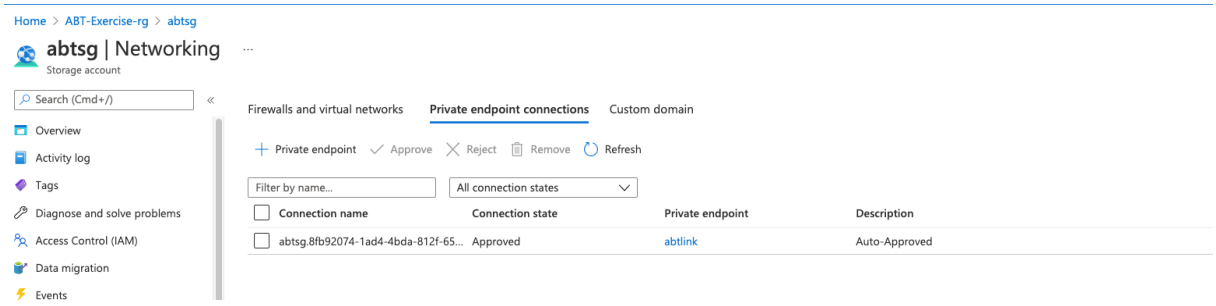
Blob Storage - Firewall and Virtual Network - Service Endpoint

Service Endpoint -
Below screenshot shows connectivity between BLOB and VNET using service endpoint and trusted Client IP.

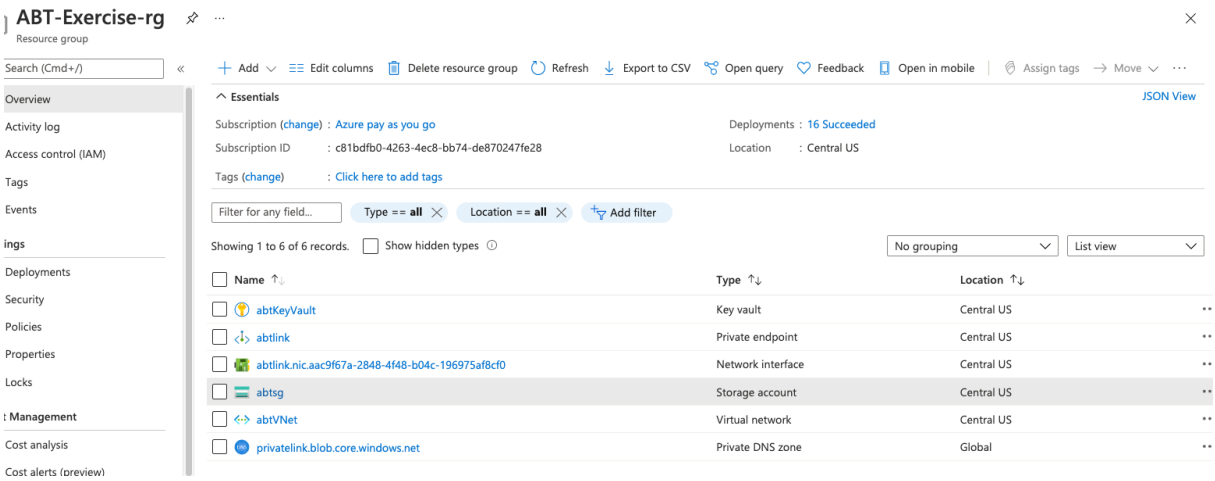


Blob Storage Private Endpoint with Vent - (**abtlink**)

Below screenshot shows connectivity between BLOB and VNET using private endpoint and .



Resource Group - ABT-Exercise-rg
Below screenshot shows resource group - Blob, Vnet , endpoint , Niccard and Private DNS zone for connectivity.



2.4. Exercise 4

Write an Event trigger function app (both python and .Net) to parse the Json files (attached example file) from the Blob Storage (step 3) and write it to the Azure SQL database.

Type – Serverless processing
Note - Due to time constrains, I have not created .net function.

Solution – Approach

Step 1 – Analyze JOSN file.
Step 2 – Create 3 normalize tables (main, locdata and hkdata) into SQL Server (**abtsample)**
Step 3 - Ingest JSON data into blog storage account using Nifi
Step 4 – Install all plugins needed for Azure Function development on VS code(local).
Step 5 – Create Function project (Blog Trigger).
Step 6 – Parse JOSN file and insert records in SQL tables (linking key - **guid**)
Step 7 – Login to SQL server and verify tables data

Note – This approach will work only if incoming data set is smaller in size. For big data set we should consider spark engine- **Databricks** or **Hdinsight** for compute requirement. Another option would be utilizing the **Azure Kubernetes Service (AKS).**

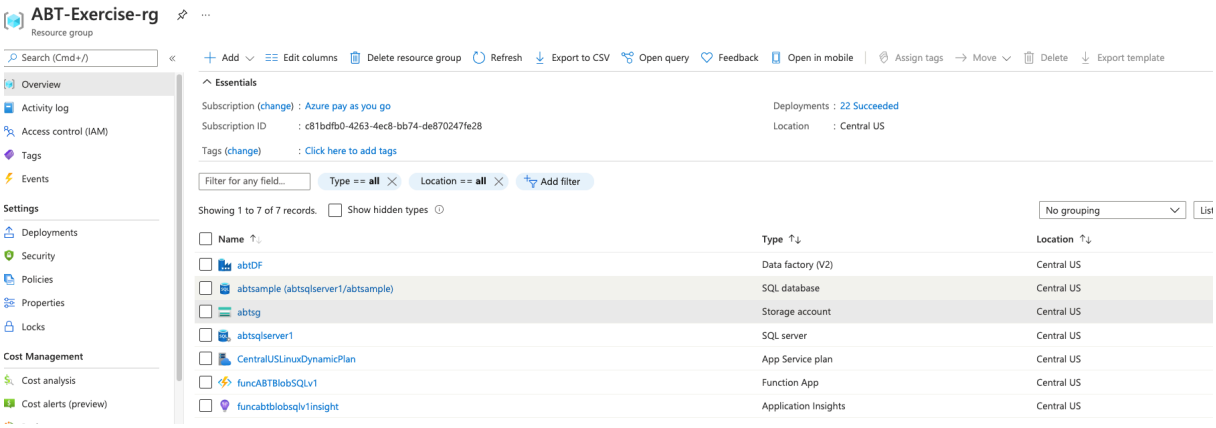
"Instead on Nifi loading data we can have another HHTP trigger Azure function to upload json data and link with second function**"**
[HTTP Trigger Function] -> [Blob Trigger Function] -> SQL Server

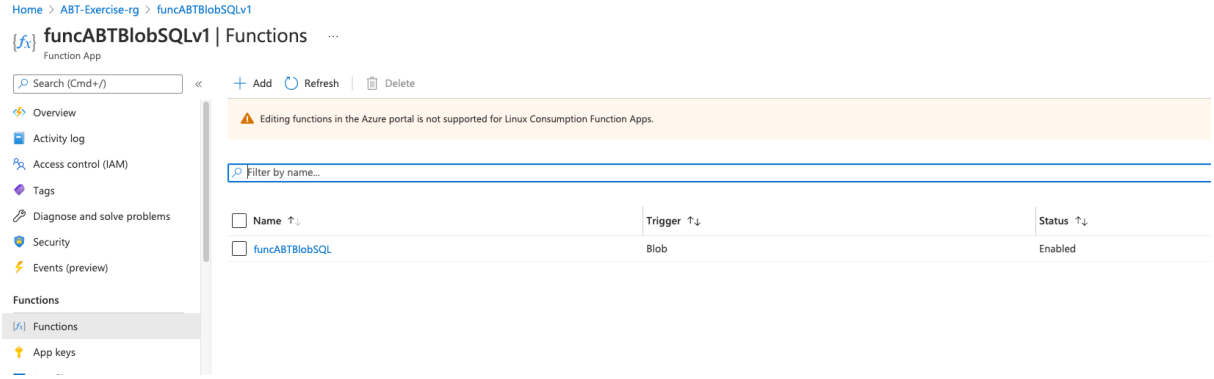
Code Location

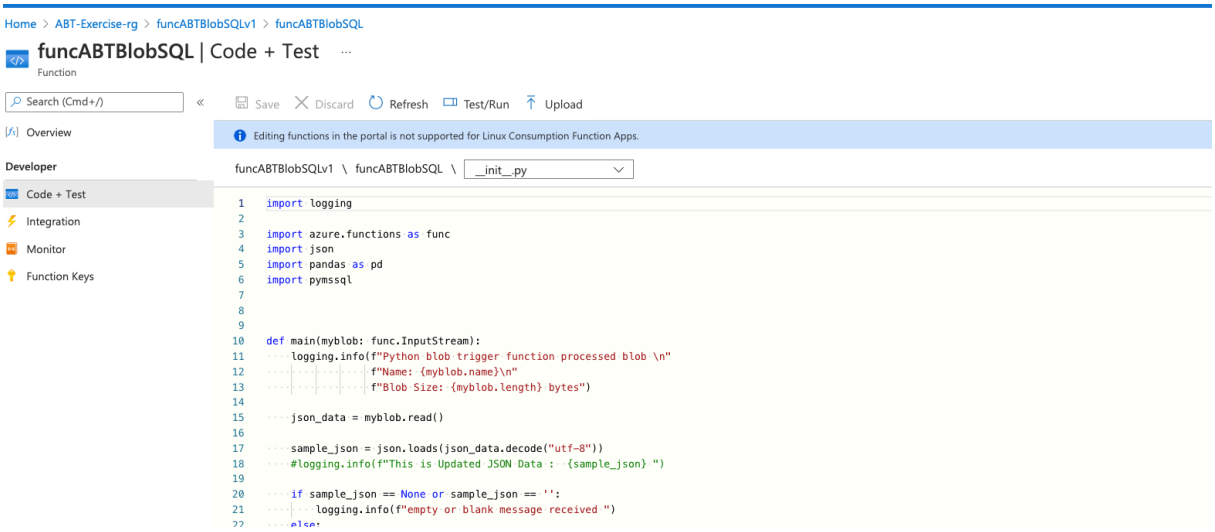
GitHub Link - https://github.com/vishu22314558/ABT_DE_Exercise_V2-main.git
Code repo Name - ABT_DE_Exercise_V2-main

Py Azure Function Code - Exercise-4-Azure-Function-Python

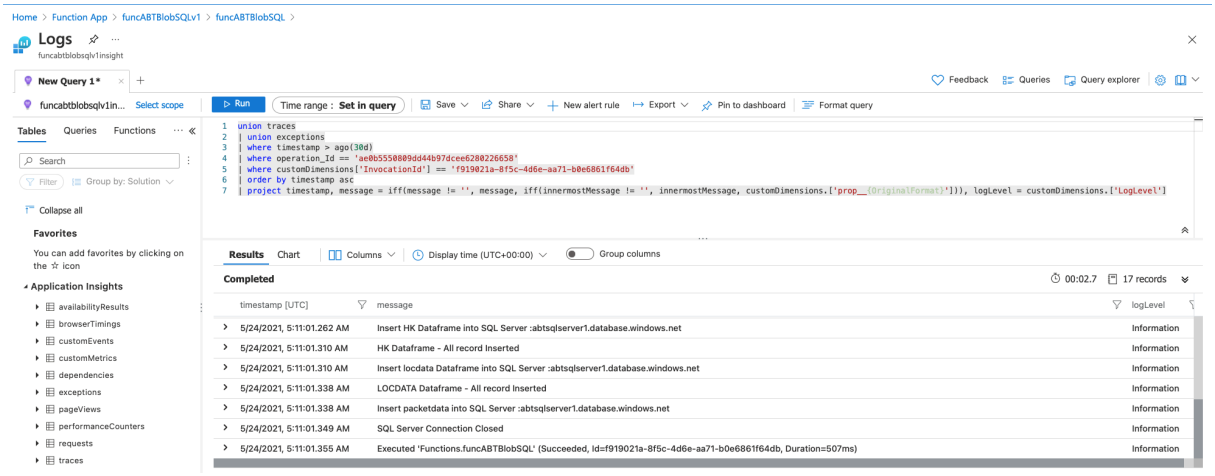
Azure Function -



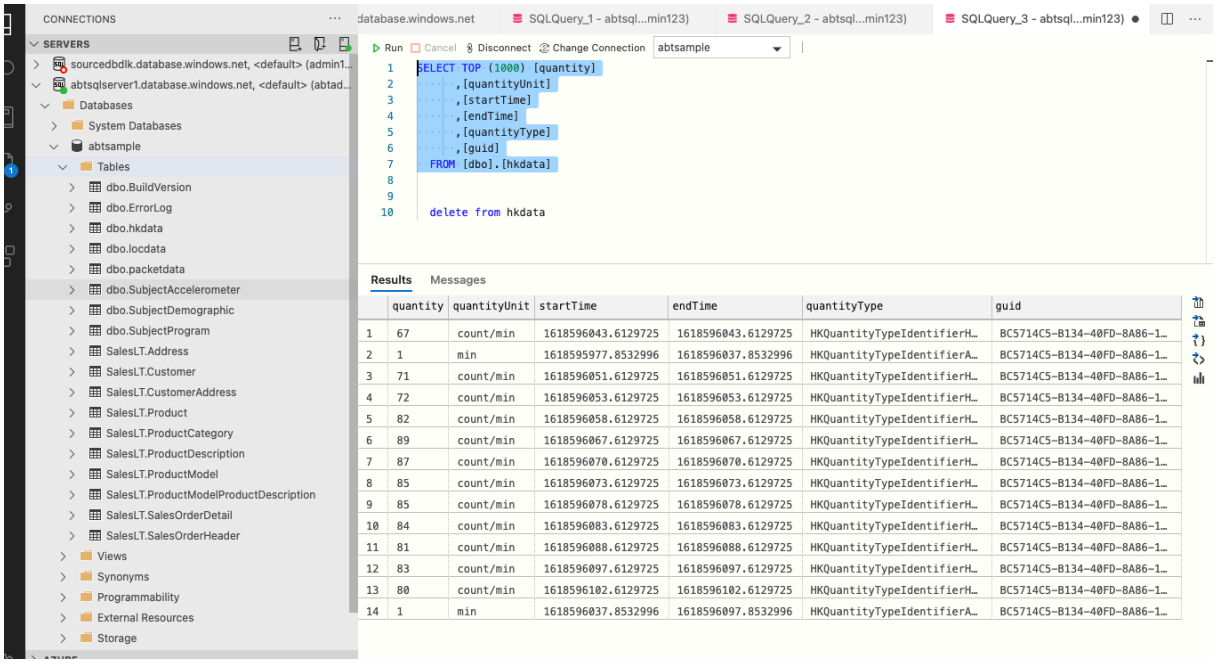


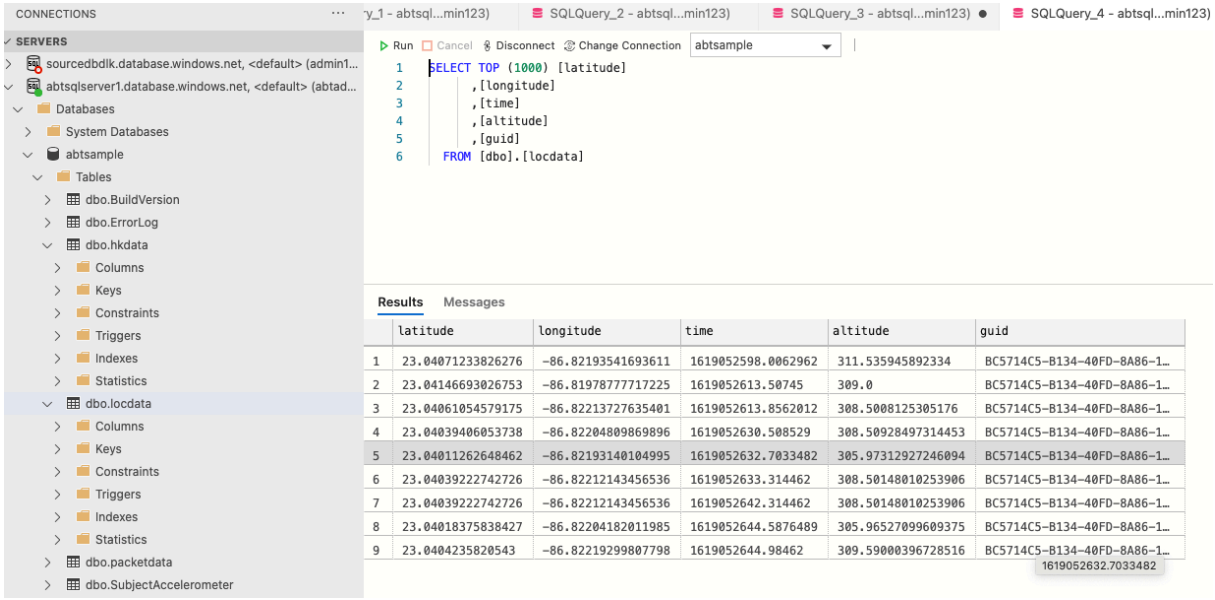


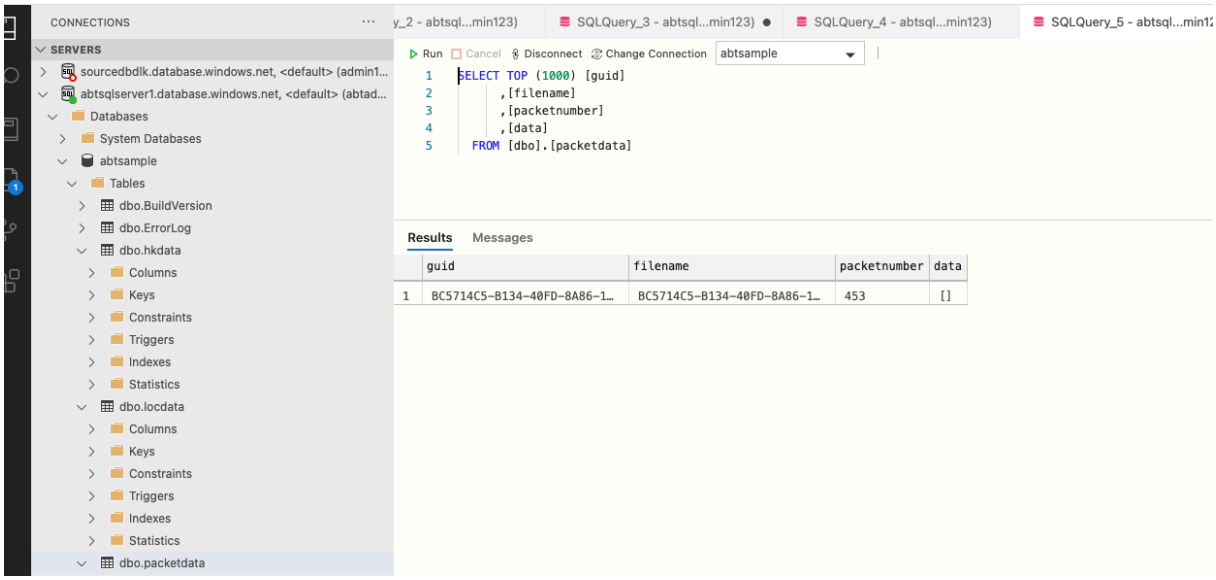
Trigger / Execution Log -



Target Tables -






CONNECTIONS
SERVERS
sourcedbdlk.database.windows.net, <default> (admin1...
abtsqlserver1.database.windows.net, <default> (abtad...
Databases
System Databases
abtsample
Tables
dbo.BuildVersion
dbo.ErrorLog
dbo.hkdata
Columns
Keys
Constraints
Triggers
Indexes
Statistics
dbo.locdata
Columns
Keys
Constraints
Triggers
Indexes
Statistics
dbo.packetdata
y_2 - abtsql...min123)
SQLQuery_3 - abtsql...min123)
SQLQuery_4 - abtsql...min123)
SQLQuery_5 - abtsql...min1
Run
Cancel
Disconnect
Change Connection
abtsample
SELECT TOP (1000) [guid]
,[filename]
,[packetnumber]
,[data]
FROM [dbo].[packetdata]
Results
Messages
guid
filename
packetnumber
data
1
BC5714C5-B134-40FD-8A86-1...
BC5714C5-B134-40FD-8A86-1...
453
[]

2.5. Exercise 5

The attached csv file contains IOT data from several sensors. "SubjectID" column is the unique identifier for each subject.
Write a python script to clean the data (missing and duplicate values), find the most important sensor and model the output variable. Try several techniques to handle the missing data and propose the optimized method.

Type – Machine learning

I will be utilizing Jupiter Notebook to build model

Solution – Approach

Setup Notebook Env (Python) and install python libs like sklearn, seaborn ..

Note – This will be a learning exercise for me and I will be trying RandomForestClassifier to model the output variable.

Step 1 - Define Problem Statement
Step 2 - Analyze Source data
Step 3 - Define data strategy to handle missing values
Step 4- Build RandomForestClassifier model
Step 5 - Implement different data strategy and capture accuracy
Step 6 - Implement Variable importance graph to find most important sensor
Step 7 - Plan for improvements and implementation

Code Location

GitHub Link - https://github.com/vishu22314558/ABT_DE_Exercise_V2-main.git
Code repo Name - ABT_DE_Exercise_V2-main

NoteBook - Exercise-5.ipynb

Quick Source Data Analysis -

```
[16]: df = pd.read_csv("IOT_data.csv")

[17]: df.head()
```

[17]:

	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5	Sensor 6	Sensor 7	Sensor 8	Time	Output	SubjectID
0	NaN	NaN	NaN	NaN	NaN	64.0	NaN	NaN	3/13/2021 5:54	2	8812EC1F-22DF-4729-A699-C7E639847E11
1	NaN	NaN	NaN	NaN	NaN	95.0	NaN	NaN	3/16/2021 13:44	2	8812EC1F-22DF-4729-A699-C7E639847E11
2	NaN	NaN	NaN	NaN	NaN	73.0	NaN	NaN	3/14/2021 11:29	2	8812EC1F-22DF-4729-A699-C7E639847E11
3	NaN	NaN	NaN	NaN	NaN	65.0	NaN	NaN	3/14/2021 7:50	2	8812EC1F-22DF-4729-A699-C7E639847E11
4	NaN	NaN	NaN	NaN	NaN	73.0	NaN	NaN	3/13/2021 1:31	2	8812EC1F-22DF-4729-A699-C7E639847E11

Data Description

"SubjectID" column is the unique identifier for each subject. Sensors 1 to 8 represent 8 values from 8 different sensors. Here is some statistics on missing value:

```
[18]: df.isna().mean().round(4)*100

[18]: Sensor 1      99.97
      Sensor 2      99.53
      Sensor 3      99.10
      Sensor 4     100.00
      Sensor 5      99.99
      Sensor 6       1.44
      Sensor 7      99.97
      Sensor 8      99.10
      Time           0.00
      Output         0.00
      SubjectID      0.00
      dtype: float64
```

Different Strategies -

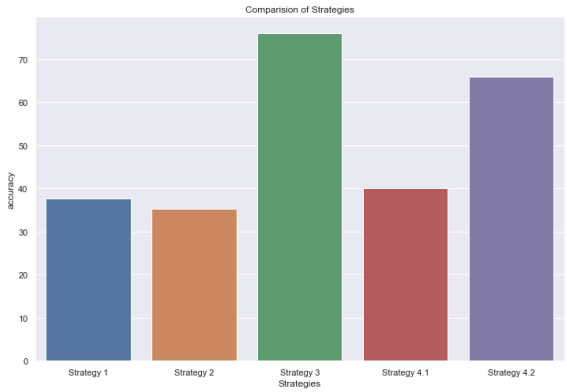
Strategy (Missing Value)

0. Drop missing values. Not acceptable here since it drops 100% of our data.
1. Replacing them with a distinctive number '-1' in our case.
2. Adding global mean, median with missing value indicator
3. Adding local mean, median as compared to each candidate respectively
4. Finding out missing value based on time series
5. Finding out missing value based on another model

Model accuracy based on data different Strategies -

```
ax.set(xlabel='Strategies', ylabel='accuracy', title='Comparision of Strategies')
```

```
[53]: [Text(0, 0.5, 'accuracy'),
       Text(0.5, 0, 'Strategies'),
       Text(0.5, 1.0, 'Comparision of Strategies')]
```



Best accuracy on the test datset we got is around 75% using the streategy # 3 of local median imputation. It fills the missing value based on the median for each candidate group. It reflected a jump of 40% over the baseline.

Most Important Sensor -

```
[51]: eli5.show_weights(perm, feature_names=X_columns)
```

[51]:

Weight	Feature
0.2818 ± 0.0067	Sensor 7
0.2412 ± 0.0074	Sensor 5
0.1536 ± 0.1275	Sensor 3
0.0597 ± 0.0070	Sensor 6
0.0559 ± 0.0039	Sensor 1
0.0294 ± 0.1166	Sensor 2
0.0004 ± 0.0006	Sensor 8

Sensor 7 and Sensor 5 are the most important

Future improvements and implementation

Chosen strategy however is difficult to implement realttime. Using spark streaming and Kafka, we can keep track of last known value of the sensor and use that value in case of any missing element. We can also keep track of certain statistics based on the time of the day or hour of the day if that is important for that sensor.

]: