The tools that are covered in this project are :

1. Azure Data Factory

2. Azure Data Lake Storage Gen2

3. Azure Databricks

4. Azure Synapse Analytics

5. Azure Key vault (For storing keys and vaults>>>)

6. Azure Active Directory (AAD) (Security and Governance)

7. Microsoft Power BI

Step 1 : On premise database in SQL server

Step 2 : Connect sql server to ADF and use ADF for data ingestion to move all the tables to cloud

Step 3 : Put all the data in Azure Data Lake Gen 2

Step 4 : Use databricks for big data analytics tool.

Step 5 : Load transformed data to Azure Synapse Analytics

Step 6: Visualize data using Power BI

Step 7: Automate pipeline and ensure end-to-end Data Flow

We will follow Lake House Architecture

* Bronze Layer: Bronze layer has an exact copy of what the Data looks in the on premise database.
* Silver Layer: Data from bronze layer to silver layer after implementing some transformation based on requirement.
* Gold Layer: Some more transformation is done from silver to gold layer, and this is the final data.

# Azure Synapse Analytics will be similar to on-premises SQL server

A screenshot of a computer

Description automatically generated

Agenda :

1: Environment setup

2: Data Ingestion

3: Data Transformation

4: Data Loading

5: Data Reporting

6: End to End Pipeline Testing

5 resources created under resource groupo on azure portal.

On sql server, backup restored and below queries run to create login for the databse

create Login ks with password = 'Migrationproject#login';

create user ks for login ks;

* Make sure SQL Server and Windows Authentication are enabled.
* Make sure port 1433 has access to inbound connections
* Data reader role assigned to the user

A screenshot of a computer

Description automatically generated

Got to key vault and Access the IAM service. Assign the user necessary access for creating secrets in Key Vault.

Azure AD user account is assigned the "Key Vault Administrator" or "Key Vault Secrets Operator" role. These roles grant the necessary permissions for managing secrets.

* Username and password secrets are created in Azure and assigned the same values as created for the user in SSMS
* This makes it safe as we are not going to type the passwords. It will be access through this.

**Move to Data factory.**

Install self-hosted integration runtime to connect to an on-premises database. This will be installed on the machine which is hosting the azure sql server database. In my case it will be installed in my laptop.

* Under data factory > Manage > Integration runtimes
* Create a new “Azure Self-Hosted” runtime.
* Name is SHIR : Self Hosted Integration Runtime

A screenshot of a computer

Description automatically generated

* Once installed and setup is complete. We can start the ingestion process.
* Head to Author tab and create a pipeline
* Create a copy data activity
* Configure the linked services with all the setup created so far.
  + There will be error in loading the secret name during configuration.
  + Go to key vault > access policies > create new > Select all for “secret permission” > next page > enter the data factory name under “principal” > review and create
* Head back to the configuration for the copy data activity and now configure the secret name for the password from azure key vault
* Configure the Sink to send the data to the Bronze Layer
* Checkout the result and once successful, it means we have all the configurations in place.
* Build a new pipeline in data factory and start with lookup activity. We need to get all the tables at once. We will be using a query to get all the relevant tables in one go.

A screenshot of a computer

Description automatically generated

* Start with foreach activity now. Configure it for all tables with dynamic content.

A screenshot of a computer

Description automatically generated

* Once configured. Trigger it for once. It will ingest all the data from on prem to cloud

A screenshot of a computer

Description automatically generated

* Configure the Service principle to mount the Azure gen 2 Data Lake
* Launch the data bricks workspace and create a compute and then create a databricks workspace.
* Create a notebook and configure the service principal configuration to connect databricks with the azure gen 2 data lake.

storage\_account\_name = "dataMigrationKS"

storage\_account\_access\_key = "E3f7ncUoKOMqvaRFcA5edobtsKSzENOs83jCKSuswDUltiJjaVXk858ofTEXYB/1SZcDAbsoRodj+AStndtRaQ=="

configs = {

"fs.azure.account.auth.type.migrationstorageks.dfs.core.windows.net": "OAuth",

"fs.azure.account.oauth.provider.type.migrationstorageks.dfs.core.windows.net": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",

"fs.azure.account.oauth2.client.id.migrationstorageks.dfs.core.windows.net": "d1f22564-f77c-47c4-badf-699bc5ba7724",

"fs.azure.account.oauth2.client.secret.migrationstorageks.dfs.core.windows.net": "P9e8Q~TMgejbBYZy8kdAWT5~EoVpF8PZgaaSNa1G",

"fs.azure.account.oauth2.client.endpoint.migrationstorageks.dfs.core.windows.net": "https://login.microsoftonline.com/18896149-8a2d-4354-bc19-cc06b5d568d0/oauth2/token"

}

spark.conf.set(

    f"fs.azure.account.key.{storage\_account\_name}.dfs.core.windows.net",

    storage\_account\_access\_key

)

* Run a query to check configuration.
* Running a query to check table access.

A screenshot of a computer

Description automatically generated

**Transformations :**

Bronze Layer :

1. Transforming the Modified Date Column.

2006-09-01T00:00:00.000+00:00 >>>>>> 2008-06-08

A screenshot of a computer

Description automatically generated

* Write the transformed data in the **SILVER CONTAINER** .

We will perform transformation on the data as well. For all the tables, we will run transformation on the columns containing “date” in the column name to effectively change all the rows in date column to UTC format.

**Query usred :**

from pyspark.sql.functions import from\_utc\_timestamp, date\_format

from pyspark.sql.types import TimestampType

for i in table\_name:

    path = bronzePath + i + '/' + i + '.parquet'

    df = spark.read.format('parquet').load(path)

    column = df.columns

    for col in column:

        if "Date" in col or "date" in col:

            df = df.withColumn(col, date\_format(from\_utc\_timestamp(df[col].cast(TimestampType()), "UTC"), "yyyy-MM-dd"))

    output\_path = silverPath +i + '/'

    df.write.format('delta').mode("overwrite").save(output\_path)

A screenshot of a computer program

Description automatically generated

* The above query completes the bronze to silver transformation

Below will display the last order(table in our case) in the list. In our case, it will be [SalesOrderHeader].

A screenshot of a computer

Description automatically generated

# The folder delta log is used for the version tracking.

* Files in the Silver Layer

A screenshot of a computer

Description automatically generated

* Column name format changed from ColumnName to Column\_Name

Silver to Gold transfer completed

Logic :

for name in table\_name:

    path = silverPath + name

    print(path)

    df = spark.read.format('delta').load(path)

    #Get the list of column names

    column\_names = df.columns

    for old\_col\_name in column\_names:

        # Convert column name from ColumnName to Column\_Name format

        new\_col\_name = "".join(["\_" + char if char.isupper() and not old\_col\_name[i-1].isupper() else char for i,char in enumerate(old\_col\_name)]).lstrip("\_")

        # Change the column name using withColumnREnamed and regexp\_replace

        df = df.withColumnRenamed(old\_col\_name, new\_col\_name)

    output\_path = goldPath + name + '/'

    df.write.format('delta').mode("overwrite").save(output\_path)

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

* Gold container has all the transformed files now

A screenshot of a computer

Description automatically generated

* **Move on to data factory to build pipeline for these.**
* We need to put the datafactory notebooks in databricks
* We need to create a link server for this.
  + Head to Manage> linked Service > New
  + databrikcs is cloud based resoruce. hence AutoresolveINtegrationruntime selected. INstead of SHIR.
  + Since we already have a cluster in Azure databricks, we will use the same cluster here as well. Hence we will makr “Existing Interactive cluster”

**A screenshot of a computer

Description automatically generated**

* Generate a personal access token from databricks under the user settings and create a secret and then configure it in the above screenshot

A screenshot of a computer

Description automatically generated

Configure all the settings as below and we are good to go.

A screenshot of a computer

Description automatically generated

Link Service Created

A screenshot of a computer

Description automatically generated

* Move to Author (left most pane) section of the ADF.
* Select “notebook” activity(databricks type)
* Generate personal access token and save it as secret in key vault. Then use it in configuration
* Once configured, trigger the pipeline once.

A screenshot of a computer

Description automatically generated

**DATA LOADING VIA AZURE SYNAPSE**

**#**Azure data Lake Gen2 is already linked to Azure Snapse when we create the synapse workspace.

#Azure Synapse is pretty much built on Azure Data Factory. Everything we can do on ADF can be pretty much done on Azure Synapse as well. It has extended features as well which enables us to make notebooks for transformations and pretty much do what we do on Azure databricks as well. It’s a combination of both.

* Process to Data Tab on the left plane and create a serverless SQL Database
* We are choosing serverless since the database is small and well structured and can be handled with small amount of computing power.
* Synapse is linked to Azure Data Lake Gen2 when we create the Synapse workspace.

Script to Query the data directly from the data lake. This can be used to create a view.

SELECT

    TOP 100 \*

FROM

    OPENROWSET(

        BULK 'https://migrationstorageks.dfs.core.windows.net/gold/SalesLT/Address/',

        FORMAT = 'DELTA'

    ) AS [result]

A screenshot of a computer

Description automatically generated

* VIEW can be created using the below Query. The data base has to be used as gold\_db.

CREATE VIEW address

AS

SELECT

    \*

FROM

    OPENROWSET(

        BULK 'https://migrationstorageks.dfs.core.windows.net/gold/SalesLT/Address/',

        FORMAT = 'DELTA'

    ) AS [result]

* In Azure Synapse, we can create a pipeline to create a view for all the tables. For this pipeline, we will create a stored procedure with parameters that can dynamically create view for all the tables in the gold container.

USE gold\_db

GO

CREATE OR ALTER PROC CreateSQLServerlessView\_gold

@ViewName nvarchar(100)

AS

BEGIN

DECLARE @statement VARCHAR(MAX)

    SET @statement = N'CREATE OR ALTER VIEW ' + @ViewName + ' AS

        SELECT \*

        FROM

            OPENROWSET(

            BULK ''https://migrationstorageks.dfs.core.windows.net/gold/SalesLT/' +@ViewName + '/'',

            FORMAT = ''DELTA''

        ) as [result]

    '

EXEC (@statement)

END

GO

A screenshot of a computer

Description automatically generated

Table name from the data lake will passed Dynamically to the stored procedure via the parameter and the same table name will be used to create views dynamically.

* Create the pipeline to use the stored Procedure to create the views
* We need new link service connection to connect to this serverless database.
* Go to manage tab > linked service >New > Select azure sql database > configure as below
* “AutoResolveIntegrationRuntime selected since this is a cloud based resource.
* “Fully Qualified Domain Name” is the same as “serverless SQL Endpoint”. Get it from Synapse Workspace properties.
* ”Authentication type” is selected to be “System assigned managed identity” . This will configure the linked service to use the access granted to “gmail account detail” used to create the Azure account, to connect to the database.

A screenshot of a computer

Description automatically generated

* Move on to “Integrate” from left most pane.
* Create New pipeline and select “Get Metadata” activity. This will get all the table name in the data lake form the gold container
* In configuration. Got to settings > New > data lake Gen2(as the dataset) > Binary (as the format)

A screenshot of a computer

Description automatically generated

* Under metadata activity settnings > fiels list > new > child items
* Child items will get all the tables under the folder specified in the “get metadata” activity.
* Select a “forEach” activity and connect the out output of “Get TableNames” activity with it. Configure the activity as below.
* Add dynamic content under settings :
  + @activity('Get TableNames').output.childItems
* The above will pass the whole output from the “Get TableNames” activity to the “forEach” activity as input. For each loop will iterate over all the table names one by one.
* Go to edit button in the “forEach” activity and enter it, then search for “Stored Procedure” activity in the activities. This stored procedure will get the table names from the outer loop as the parameter and it will create the view for all the table names dynamically. Configure it as below.

A screenshot of a computer

Description automatically generated

* Once all configured, trigger the pipeline and it will run and create views for all the tables in the gold database.

A screenshot of a computer

Description automatically generated

**Integrating Power BI**

* Select Azure synapse Analytics to get data in power Bi desktop.

A screenshot of a computer

Description automatically generated

* Under configuration
* Server: “Serverless SQL endpoint” from properties of Synapse WorkSpace.
* Database: gold\_db
* Select Microsoft Authentication and then sign in with the same credentials as used on the Azure platform.
* Load the data.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* Since Power BI is connected to the Synapse Workspace. Any changes in the synapse workspace will be reflected in Power Bi as well.
* Power BI dashboard as per the required KPIs

A screenshot of a computer

Description automatically generated

**End-to-End Testing**

* We need all the pipelines automated. We can achieve this by configuring a scheduled trigger in pipeline in Azure Data Factory.
* Configure the New Trigger as below.

A screenshot of a computer

Description automatically generated

* New Data entered in the Customer Table.

USE [AdventureWorksLT2017]

GO

SET IDENTITY\_INSERT [SalesLT].[Customer] ON;

INSERT INTO [SalesLT].[Customer]

([CustomerID]

,[NameStyle]

,[Title]

,[FirstName]

,[MiddleName]

,[LastName]

,[Suffix]

,[CompanyName]

,[SalesPerson]

,[EmailAddress]

,[Phone]

,[PasswordHash]

,[PasswordSalt]

,[rowguid]

,[ModifiedDate])

VALUES

(595959, 0, 'Mr. ', 'John', 'SR', 'Max', 'Jr.', 'xyz', 'adventure-works\John', 'john@gmail.com', '13456-095-0045', 'Trade', 'Analyst',

'1F54D149-9FDD-452E-BF85-BAE74CE87CDE','2005-08-01 00:00:00.000'),

(292929, 0, 'Mr. ', 'Stephen', 'BK', 'Ben', 'Jr.', 'abc', 'adventure-works\Stephen', 'stephen@gmail.com', '9876-675-8787', 'Inovate',

'Developer', '0C22A1E3-FAA3-4365-897B-C104BCD6ED6D','2005-08-01 00:00:00.000')

GO

A screenshot of a computer

Description automatically generated

Customer Count in Power BI Service before adding new data to Customer table : 847

A white rectangular sign with black text and numbers

Description automatically generated

* Trigger initiated the pipeline as scheduled

A screenshot of a computer

Description automatically generated

* Pipeline run completed :

A screenshot of a computer

Description automatically generated

* Refresh Initiated in Power BI

A screenshot of a computer

Description automatically generated

Customer Count in Power BI Service after adding new data to Customer table : 849

**A white rectangular card with black numbers and black text

Description automatically generated**

**This update confirms that the end to end pipeline has been successfully configured. All the components are in sync and performing as expected. Final update of data in Power Bi confirms testing concluded successfully.**