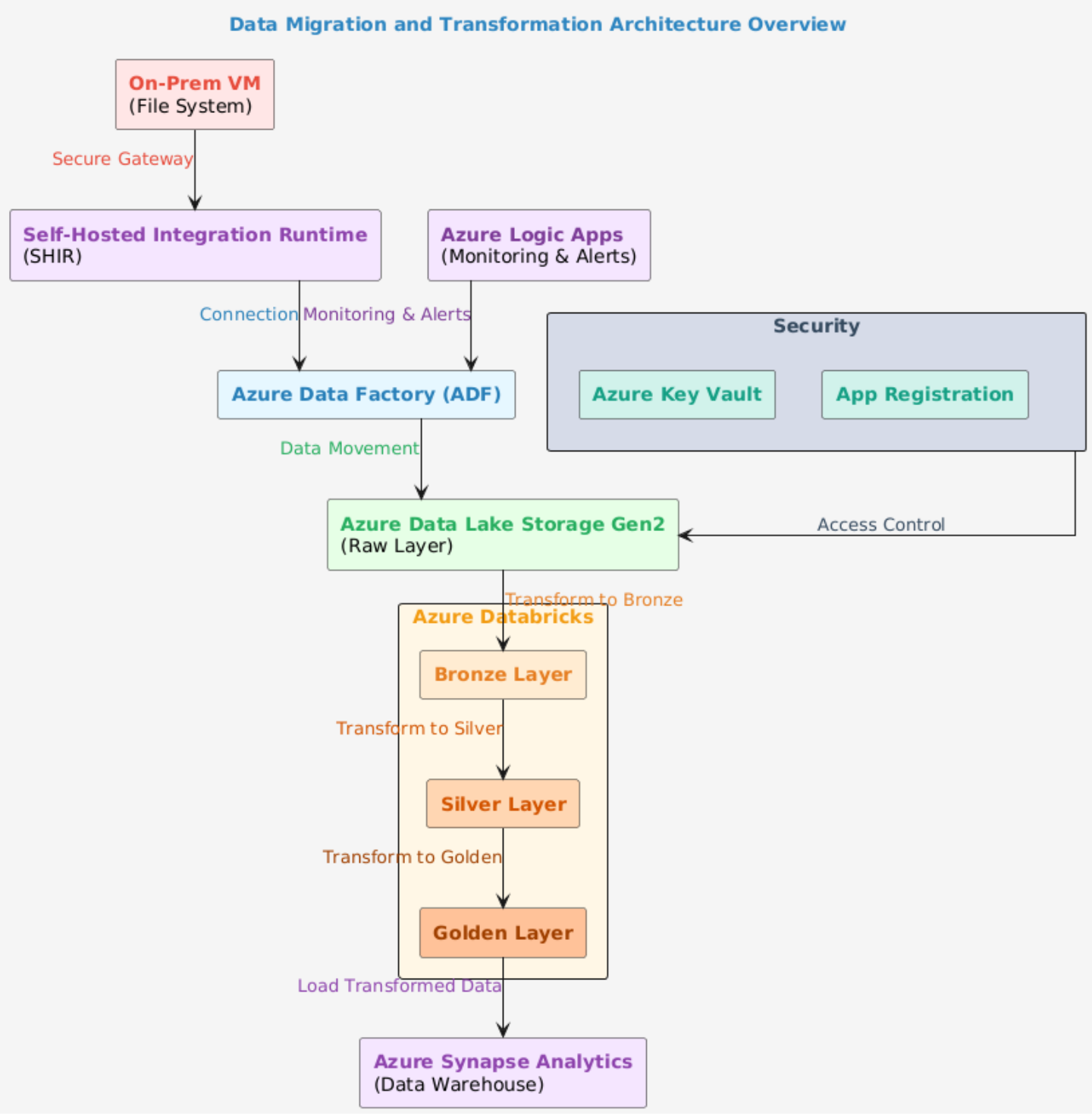
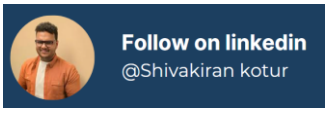
On-Prem to Azure Data Migration Architecture

This architecture outlines a comprehensive approach for migrating data from an on-premises environment to Azure using various Azure services. The required resources include:



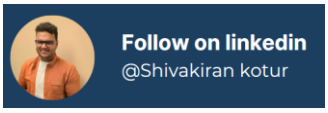


1. **On-Prem VM and File System**: The on-prem VM will host the file system with data in formats such as TXT, CSV, and Parquet.
2. **Azure Data Factory (ADF)**: ADF will serve as the primary tool for orchestrating and automating the data migration process.
3. **Azure Data Lake Storage Gen2 (ADLS Gen2)**: Used to store raw, preprocessed, and processed data. The raw data will be stored in the landing layer, while cleaned and transformed data will be moved to the preprocessed and processed layers.
4. **Azure Synapse Analytics**: The destination data warehouse where the final processed data will be loaded for analytics and reporting.
5. **Azure Databricks with PySpark**: Used to create the bronze, silver, and golden layers. This involves cleaning the data by removing nulls and duplicates, transforming data by joining tables, and applying business logic to create structured and ready-for-reporting datasets.
6. **Azure App Registration**: Facilitates secure connections to ADLS Gen2 via mount points for data access.
7. **Azure Logic Apps**: Used to set up alert mechanisms for monitoring the migration and data processing workflows.
8. **Azure Key Vault**: Stores and manages secrets, keys, and credentials for secure access to Azure resources.

Follow on linkedin
@Shivakiran kotur

Architecture Overview:

1. **On-Prem Data Storage**: We start by establishing an on-prem VM with a file system containing data in formats such as TXT, CSV, and Parquet.
2. **Connecting On-Prem to Azure**: Once Azure Data Factory (ADF) is set up, we use the Self-Hosted Integration Runtime (SHIR) to create a secure gateway between the on-prem VM and Azure. The SHIR allows ADF to connect to on-prem data sources.
3. **Data Movement to ADLS Gen2**: Using ADF, we perform various activities like Lookup, Metadata, Copy, and Stored Procedure activities to move the data from on-premises to Azure Data Lake Storage Gen2 (ADLS Gen2), specifically into the raw or landing layer.
4. **Data Transformation in Azure Databricks**: In Azure Databricks, we create the bronze, silver, and golden layers:



- **Bronze Layer**: Raw data is cleaned by removing nulls, duplicates, and irrelevant records.
- **Silver Layer**: Preprocessed data is transformed with additional logic, such as applying joins or other business rules.
- **Golden Layer**: The final, cleaned, and transformed data is ready for reporting and analytics.

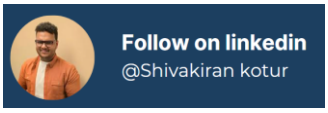
5. **Data Loading into Synapse**: The processed data is then loaded into **Azure Synapse Analytics** as a data warehouse, where it will be used for analytics and reporting.
6. **Security and Connectivity**:
   - **App Registration**: Provides secure access to ADLS Gen2 by using mount points.
   - **Azure Key Vault**: Ensures the safe storage of secrets and credentials required for accessing resources.
7. **Monitoring and Alerts**: **Azure Logic Apps** are configured to send alerts on various activities, such as task failures or successful migrations, ensuring smooth monitoring of the pipeline.

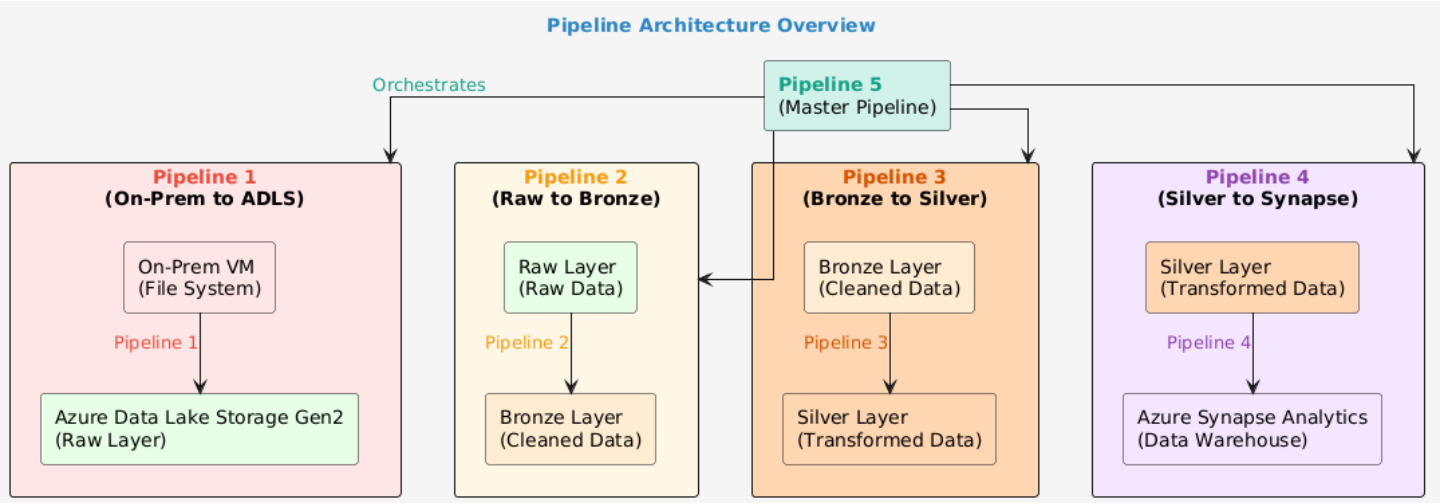
Follow on linkedin
@Shivakiran kotur

This architecture ensures a secure, automated, and scalable solution for migrating on-prem data to Azure, transforming it for analytics, and storing it in a centralized data warehouse.

Here are the pipeline headings:

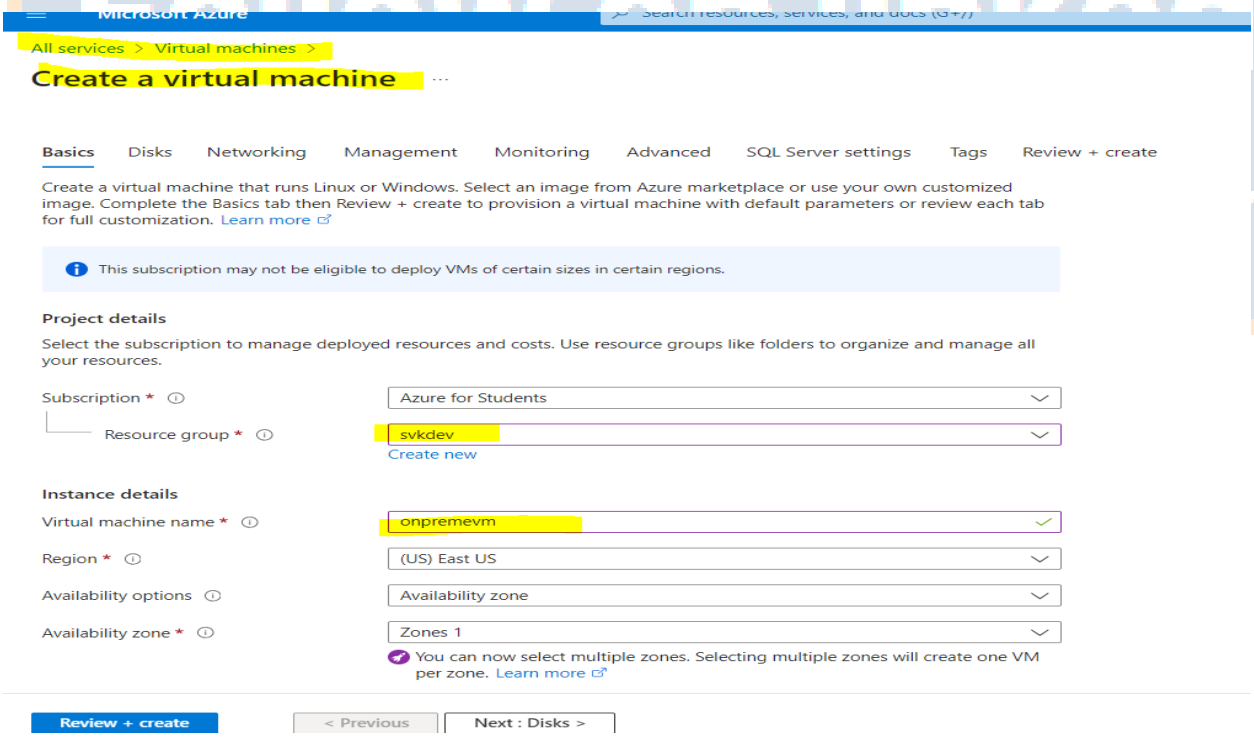
1. **Pipeline 1: On-Prem to ADLS**
2. **Pipeline 2: Raw ADLS to Bronze ADLS**
3. **Pipeline 3: Bronze ADLS to Silver ADLS**
4. **Pipeline 4: Silver ADLS to SQL Data Warehouse (Synapse)**
5. **Pipeline 5: Master Pipeline**



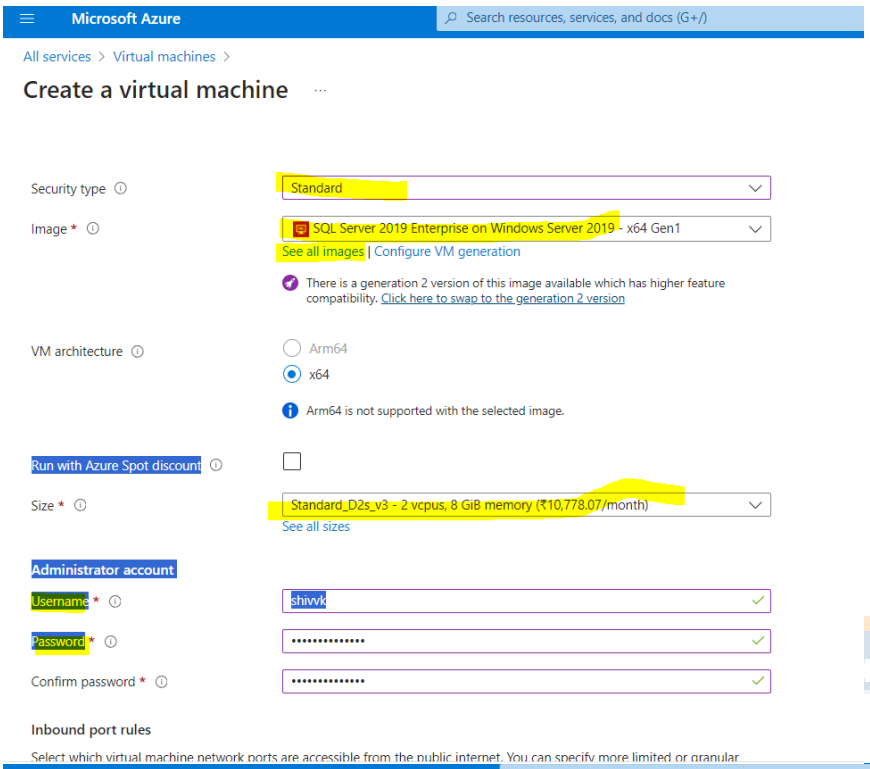


Step → create on-prem VM

Follow on linkedin

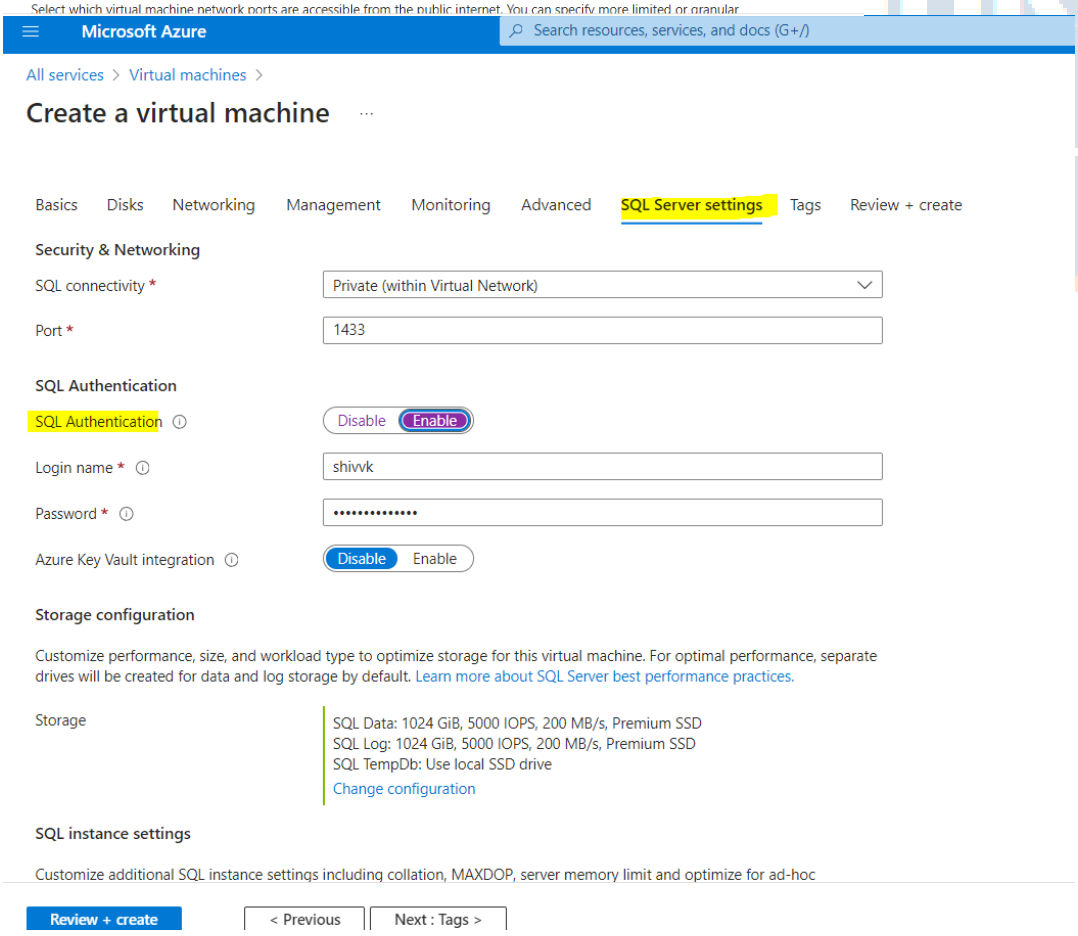


Follow on linkedin
@Shivakiran kotur

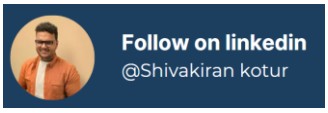


Under imagine select your sql server

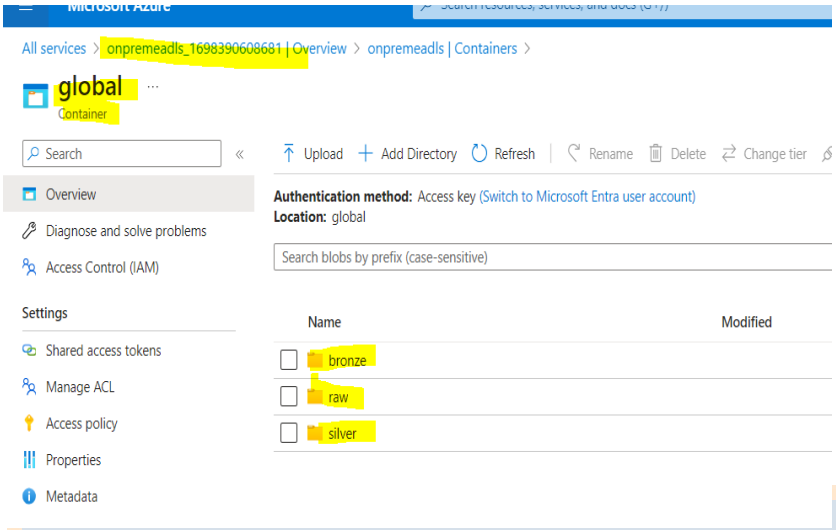
Next directly go to sql server by giving next→ next → enable the sql auth



inkedin
tur



Step2→ ceate adf



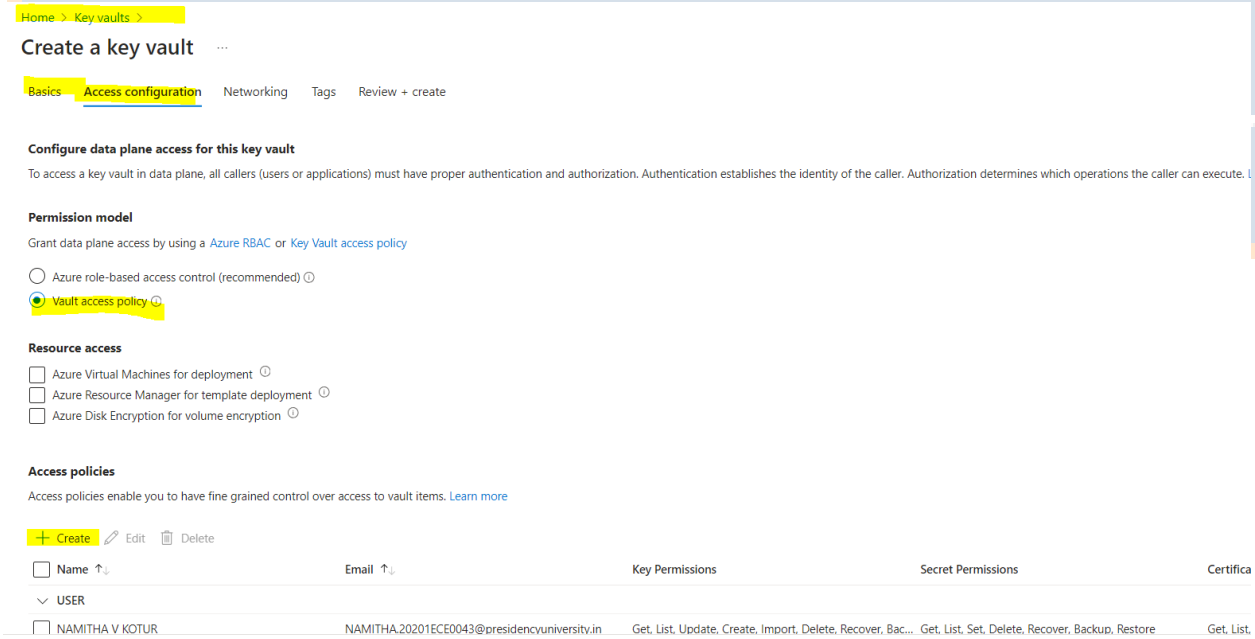
Step 3 → create adls gen 2 storage account

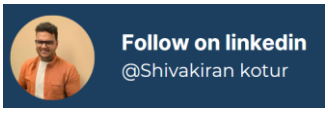
create a container → global

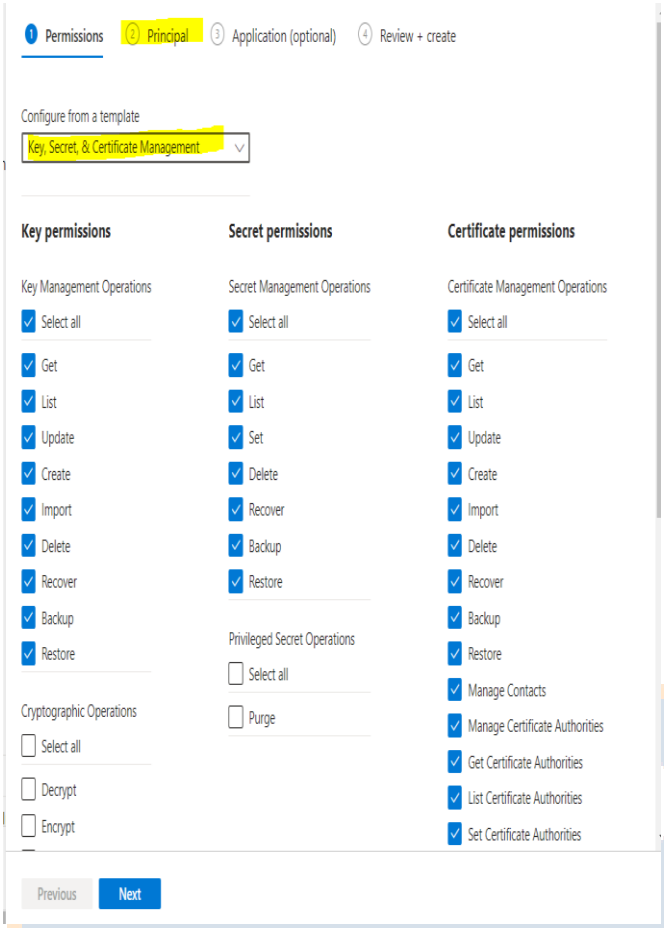
under global→ create 3 directory as raw, bronze, silver

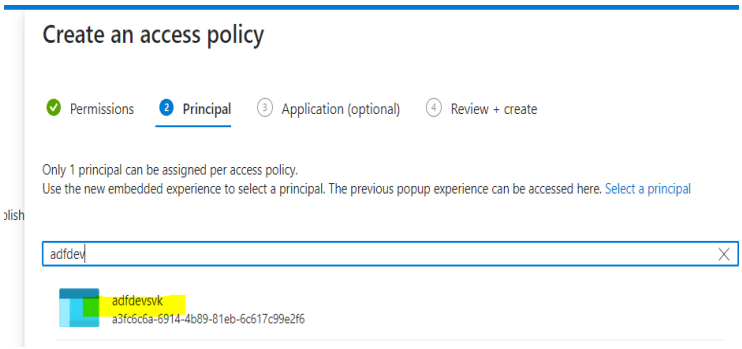
Follow on linkedin

step 4→ create key vault





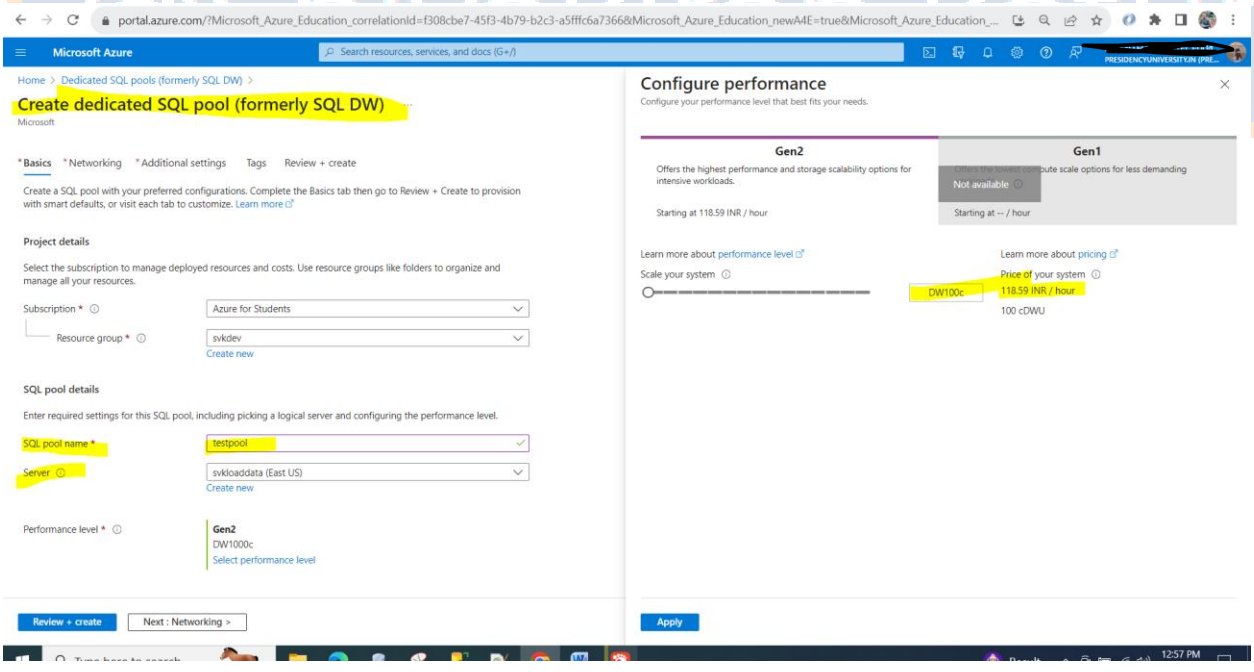


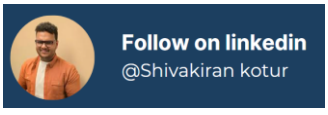


Under principal give acess to your ADF so that adf can access the key vault

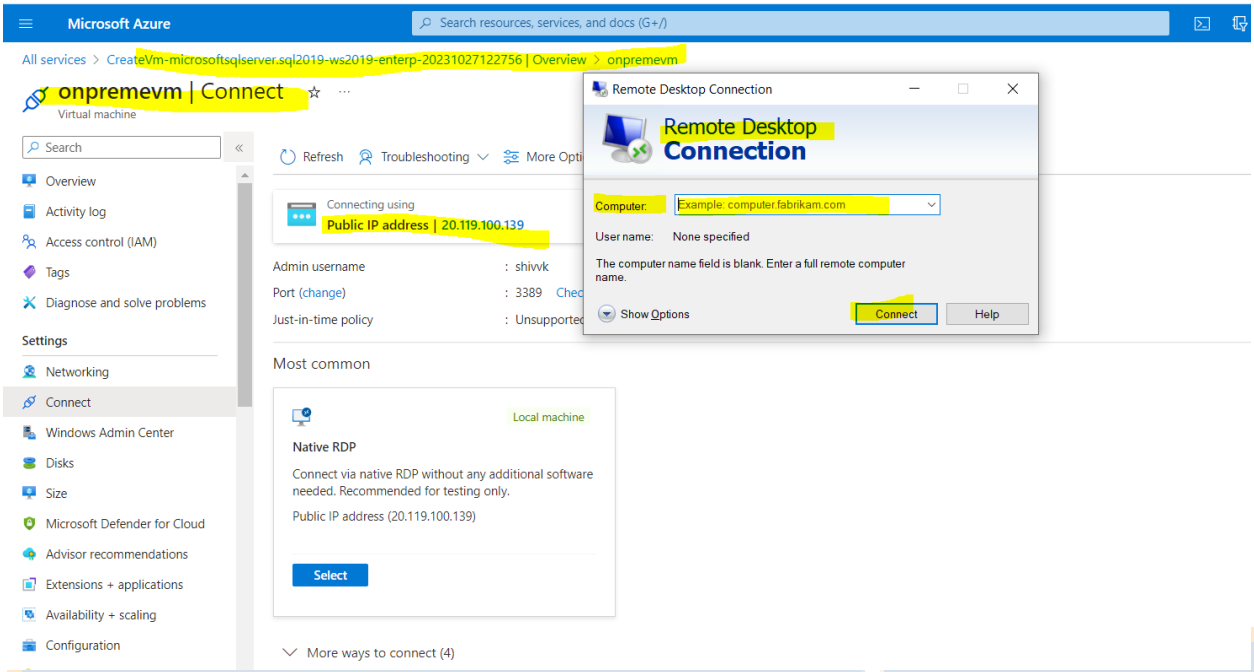
After giving access review and create key vault.

step 5→ create a dedicated sql pool(i.e sql dw)





Step 6→ go to VM and connect to VM



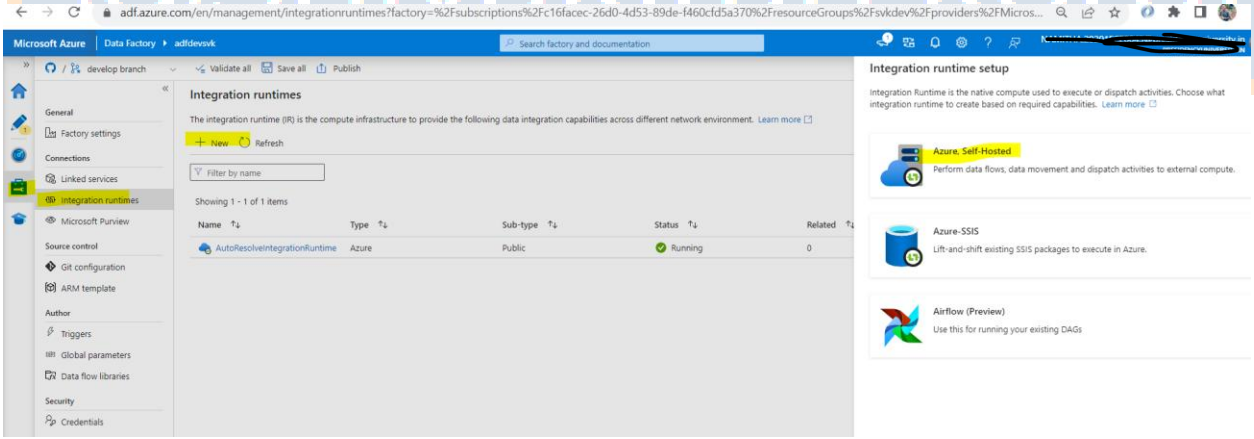
Follow on linkedin

In your local open RDP and paste the ip address and give credentials and login in.

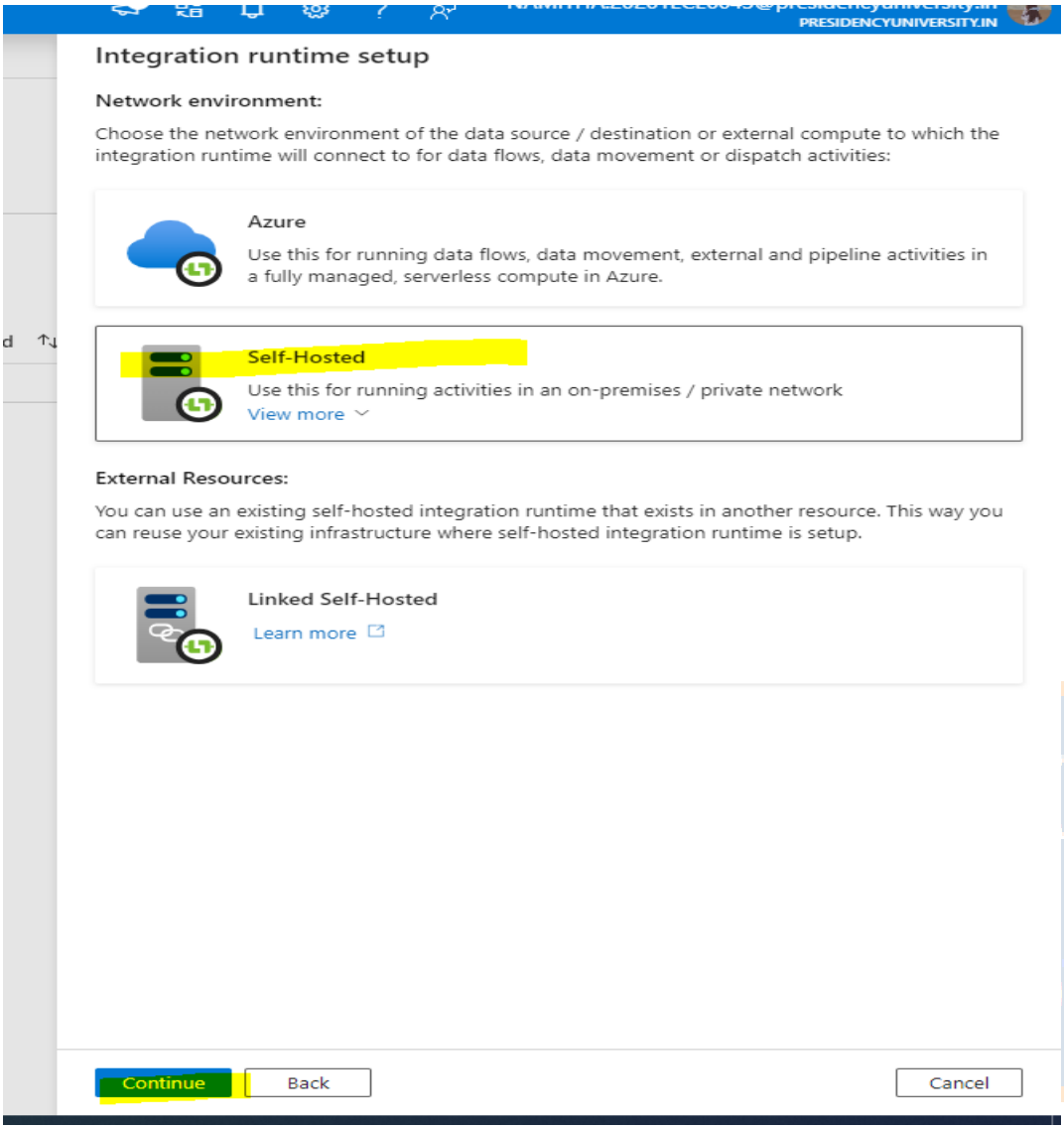
Minimize the RDP

@Shivakiran kotur

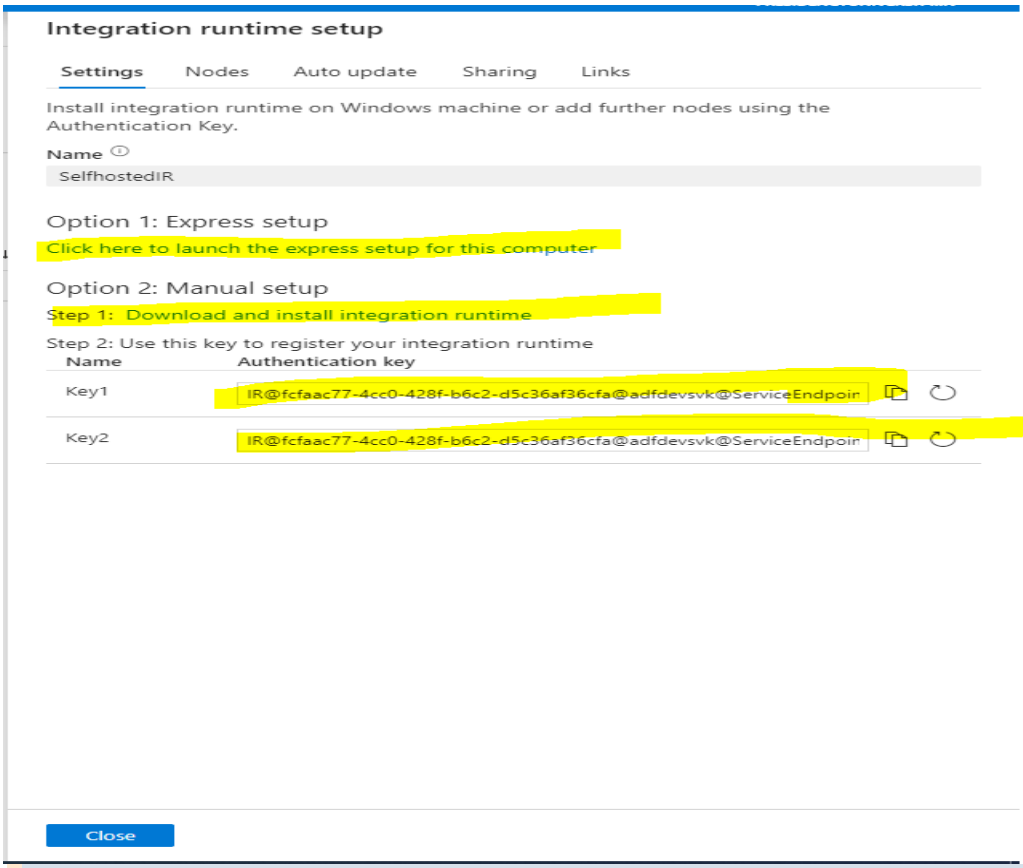
Step 7→ go to ADF and create Self hosted run time



Follow on linkedin
@Shivakiran kotur



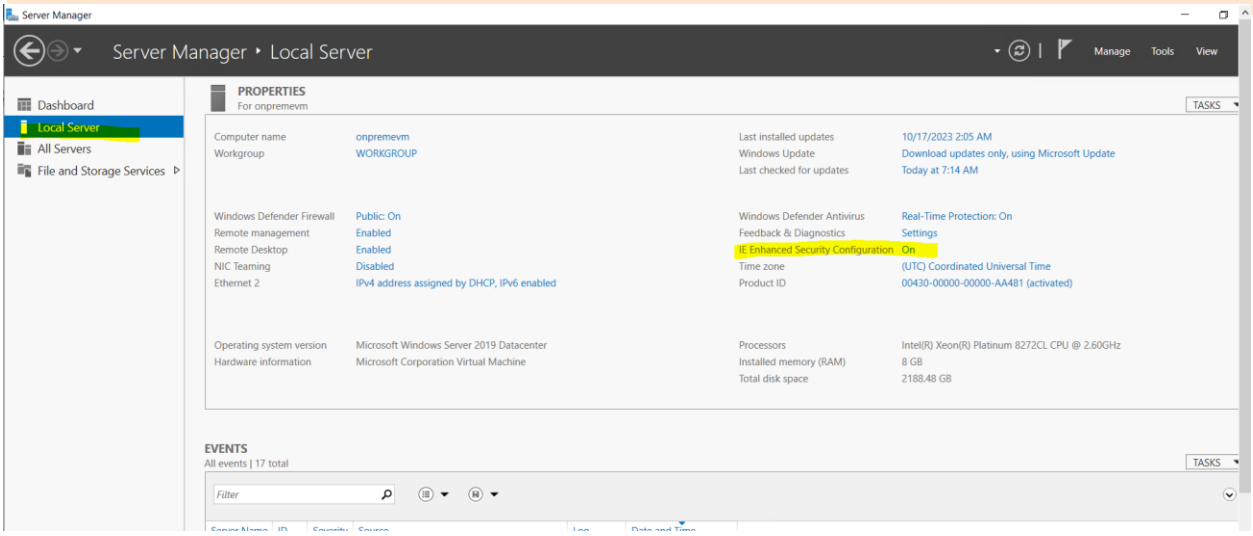
Follow on linkedin
@Shivakiran kotur



@Shivakiran kotur

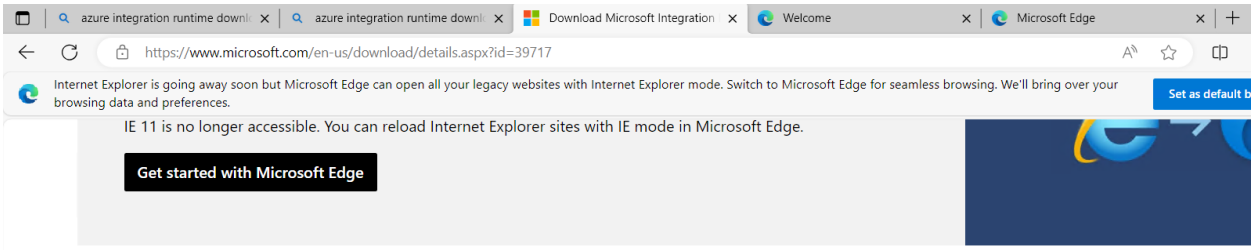
Here download the integration runtime in your VM (remember to copy the keys)

Step→ go to VM → local server→ off the enchanced sercuity



Follow on linkedin
@Shivakiran kotur

Open the explorer→ download the integration run time in your RDP



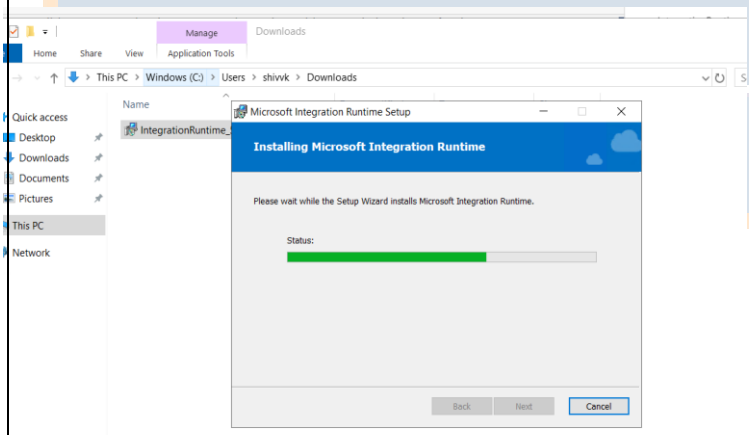
Microsoft Integration Runtime

The Microsoft Integration Runtime is a customer managed data integration infrastructure used by Azure Data Factory and Azure Synapse Analytics to provide data integration capabilities across different network environments.

Important! Selecting a language below will dynamically change the complete page content to that language.

Select language English Download

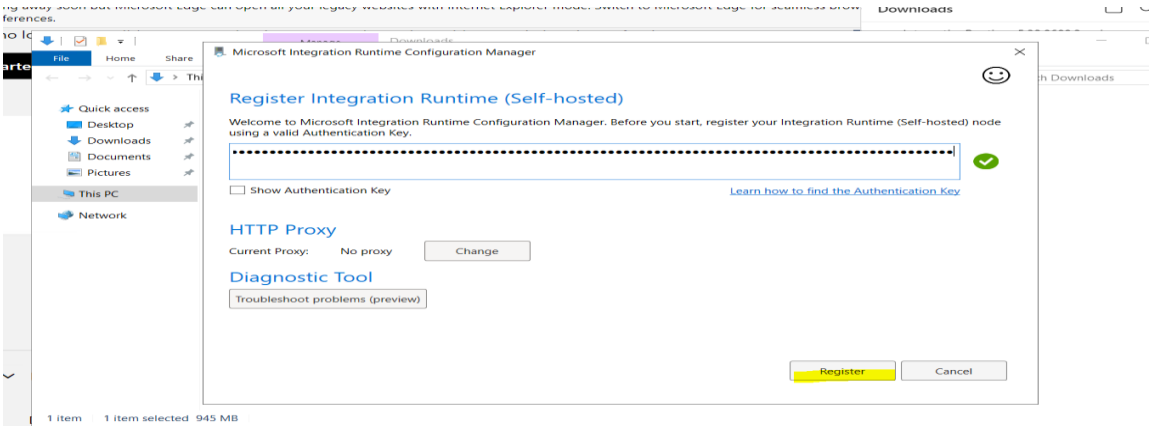
Follow on linkedin



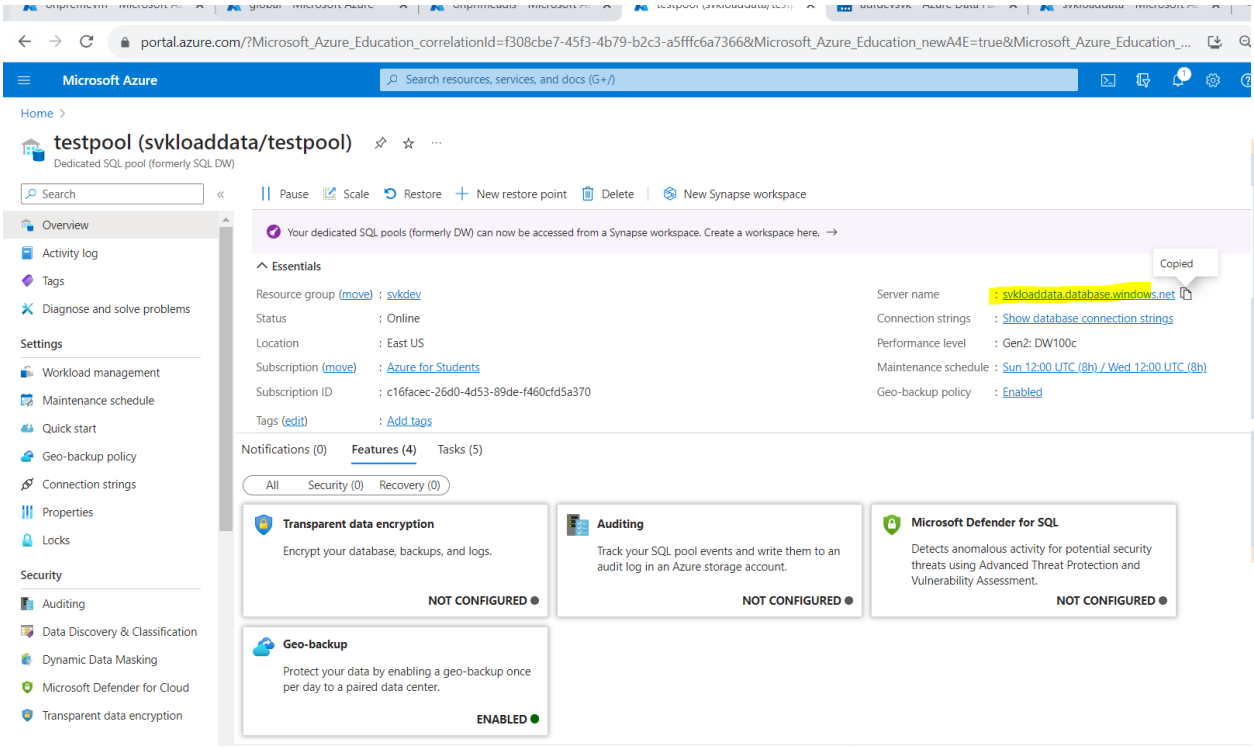
once downloaded install in RDP

once downloaded copy the key from ADF and paste it and register in RPD

Follow on linkedin
@Shivakiran kotur

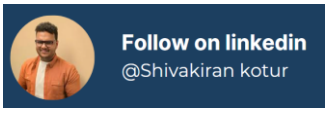


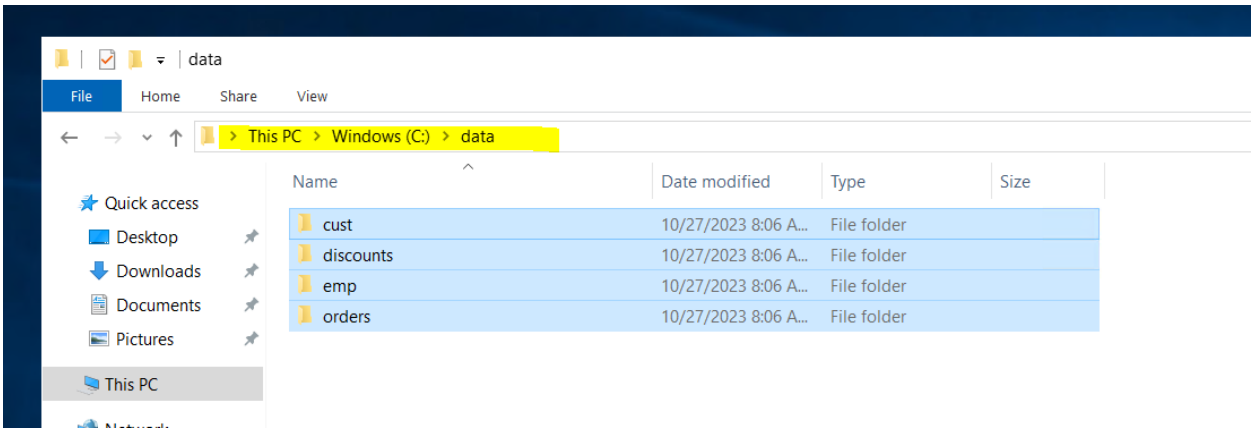
Step 8→ mean while connect your Sql pool to your SSMS



Step 9→ put the csv files in c folder of VM to create the file system data

(make sure they are in txt format as in VM there is no Excel to support)





Step 10→ we need to disable dmgcmd.exe in integration runtime folder in VM

which Is c folder

C:\Program Files\Microsoft Integration Runtime\5.0\Shared

Open power shell in RDP and disable the dmgcmd.exe file from integration runtime folder

Open PowerShell.Navigate to the directory where dmgcmd.exe is located using the cd (Change Directory) command. For example:

powershell

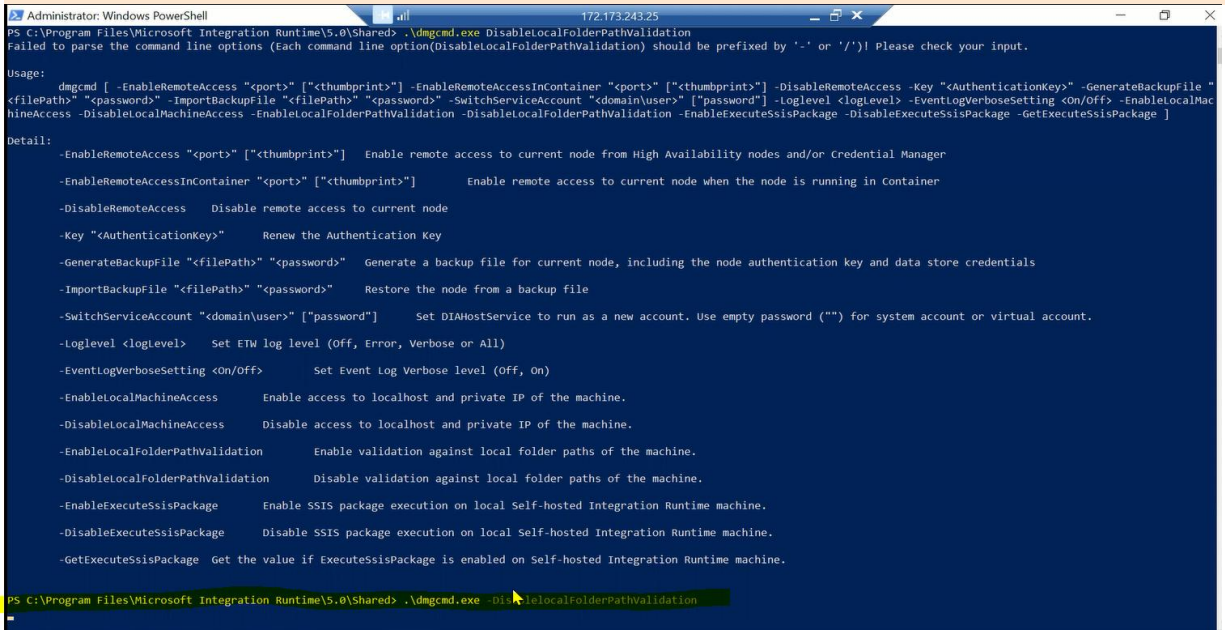
```
cd "C:\Program Files\Microsoft Integration Runtime\5.0\Shared"
```

After you have navigated to the correct directory, you can run the executable as follows:

powershell

```
.\dmgcmd.exe -DisableLocalFolderPathValidation
```

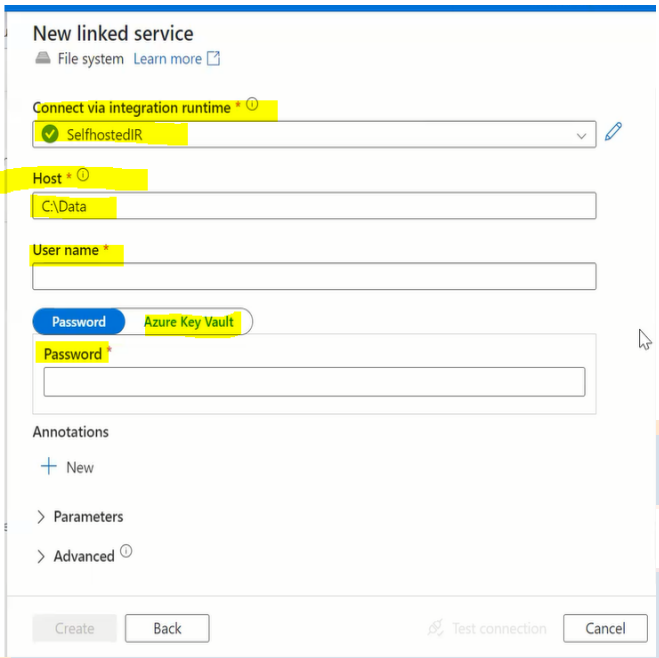
Follow on linkedin
@Shivakiran kotur



Follow on linkedin
@Shivakiran kotur

Step 11→ create the linked services in ADF for

Linked service→ File system → self hosted



in Host → give path where we have coped the file system data in VM

username → VM / RDP username

password→ place the password in key vault and use it as the password

later check for test connection→ connection should be established.(if dgmcdm is disabled it will get connected)

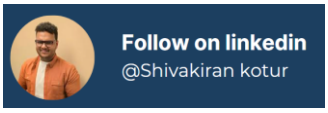
on linkedin
@Shivakiran kotur

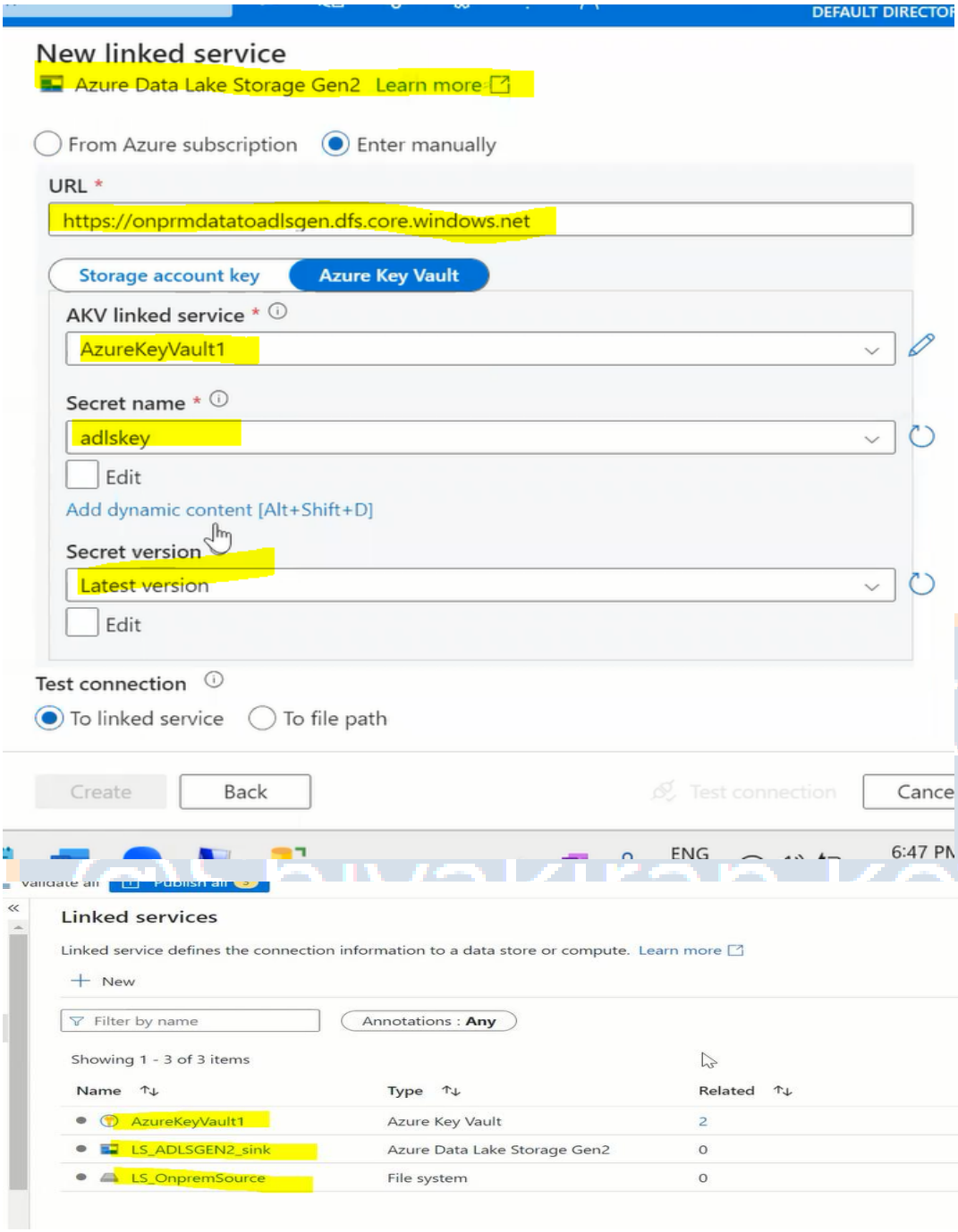
Similarly create for ADLS gen 2 using key vault →

url→ hhtps:\\<adlscontainername>.dfs.core.windows.net

copy the connection string of the container from adls--> storage →adls container→access key→connnection string copy→ key vault→ secret→ create the key

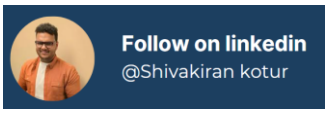
create the linked service for key vault if not created to access the secret

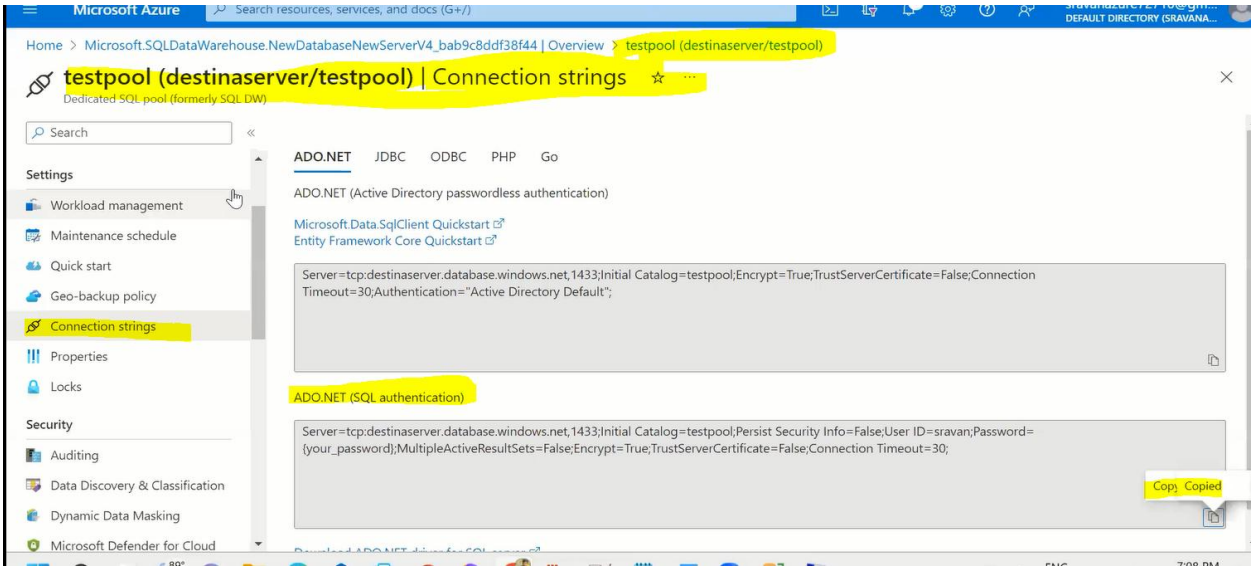




Similary create the linked service for Sql pool (linked service → snapse analytics)

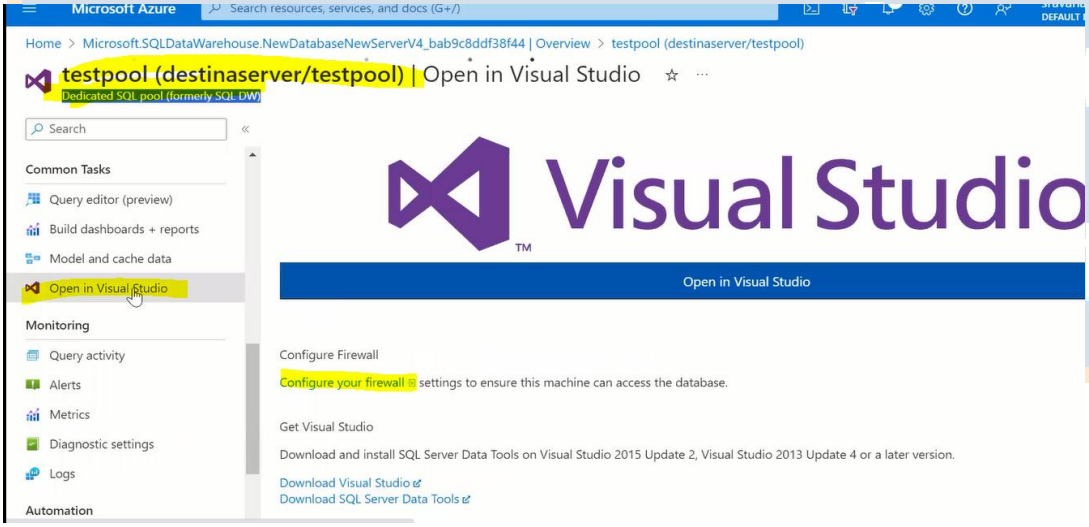
Here we use the key vault secret connection using dot.net connection string

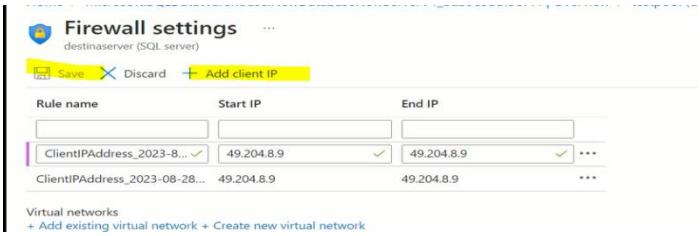




replace the password → create the key vault secret

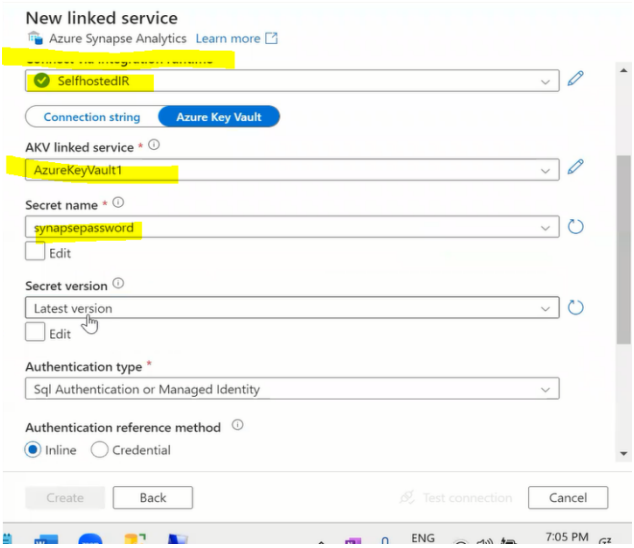
➔ Configure the firewall for sql pool





there a option to enable the azure services, have to enable it

Follow on linkedin
@Shivakiran kotur



use selfhosted IR which is created for runtime

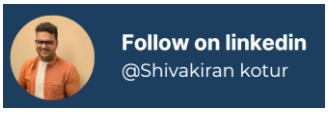
Step 12→ execute the following scripts in testpool database in SSMS

Table syntax:

Follow on linkedin
@Shivakiran kotur

```
-- Create the table 'metadata'
CREATE TABLE metadata (
    sourcefoldername VARCHAR(50),
    storagepath VARCHAR(50),
    isactive INT,
    status VARCHAR(50)
);
```

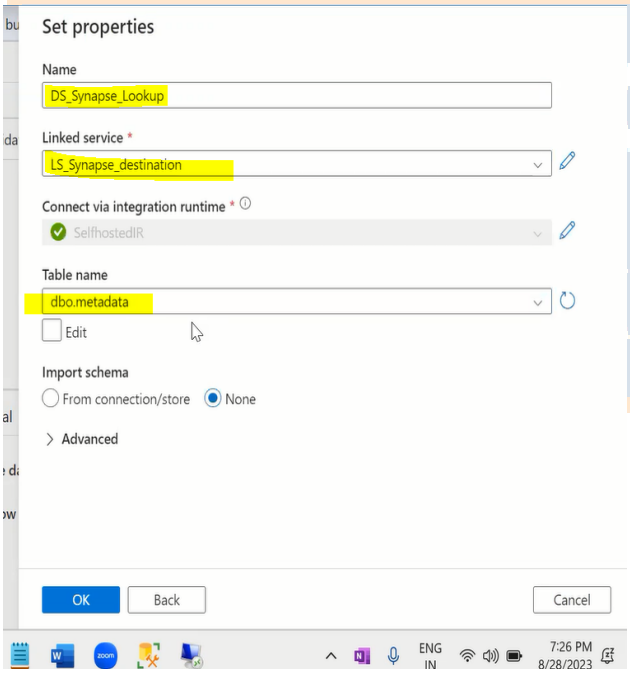
```
-- Insert data into the 'metadata' table
INSERT INTO metadata (sourcefoldername, storagepath, isactive, status)
VALUES
    ('cust', 'cust', 0, 'ready'),
    ('orders', 'orders', 0, 'ready'),
    ('emp', 'emp', 0, 'ready'),
    ('discounts', 'discounts', 0, 'ready');

-- Create the 'metadata_usp' stored procedure
CREATE PROCEDURE metadata_usp (@status VARCHAR(50), @sourcefoldername
VARCHAR(50)
AS
BEGIN
    UPDATE metadata
```



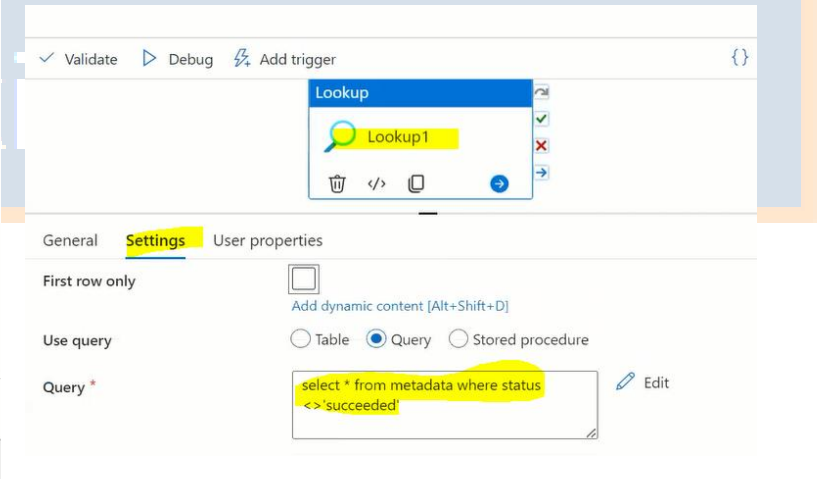
```
    SET status = @status
    WHERE sourcefoldername = @sourcefoldername;
END;

-- Create the 'reset_status_usp' stored procedure
CREATE PROCEDURE reset_status_usp
AS
BEGIN
    UPDATE metadata
    SET status = 'ready';
END;
```

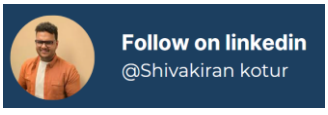
Step 13→ Create the pipeline in ADF

First activity is lookup→ to lookup metadata table from azure synapse



Take lookup activity→ setting→create dataset→ create dataset as below





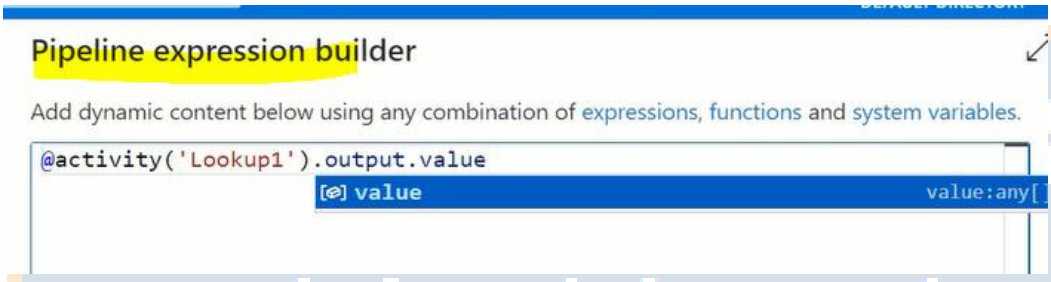
Preview data

Linked service: LS_Synapse_destination

Object:

	sourcefoldername	storagepath	isactive	status
1	orders	orders	0	ready
2	cust	cust	0	ready
3	discounts	discounts	0	ready
4	emp	emp	0	ready

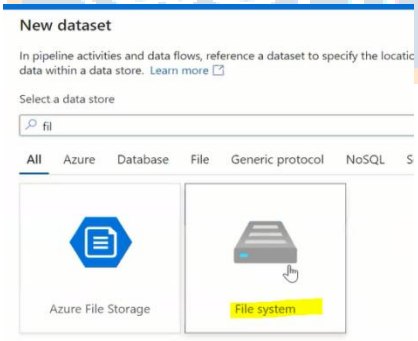
Step → take for each activity next to lookup, take the output of lookup as input to for each loop

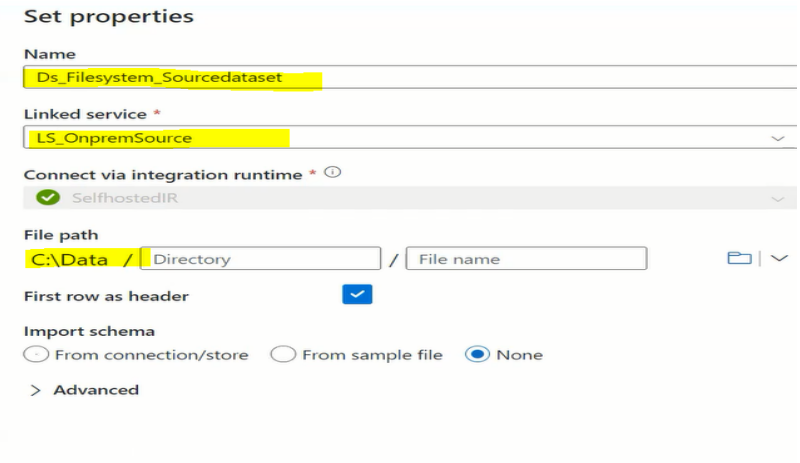


@Shivakiran kotur

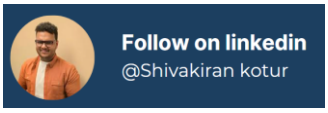
Inside for each → take copy activity → copy activity

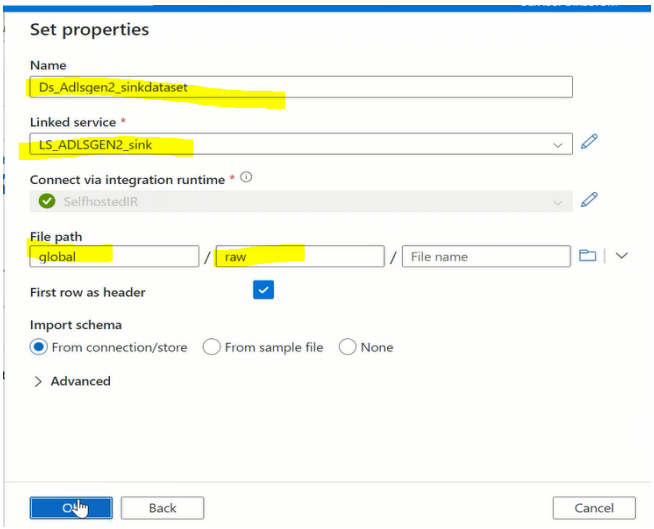
for copy activity source is file system → create the dataset → file system→ csv file





for sink use adls gen 2→ create dataset for the same

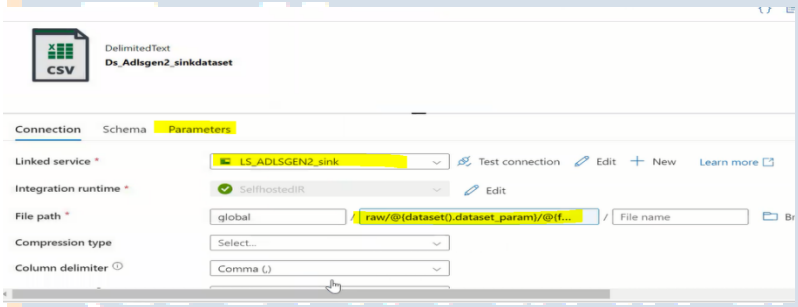




Here file path should be in sink raw, as we are going to put raw data in raw folder

Parameterize the directory → for file system dataset source

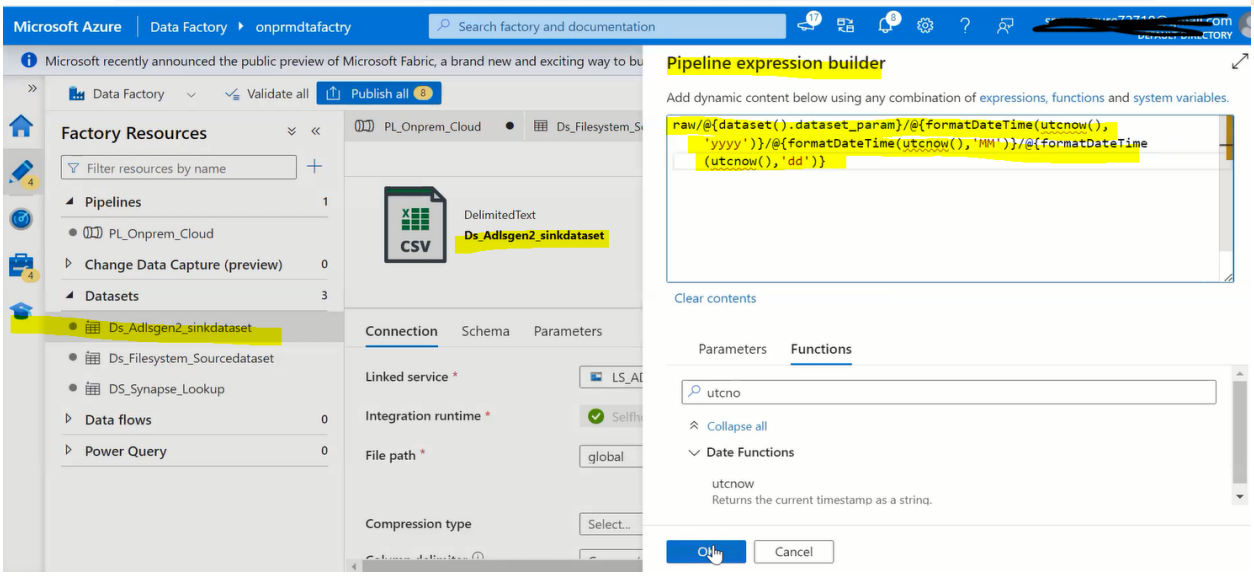
And in sink dataset also parameterize

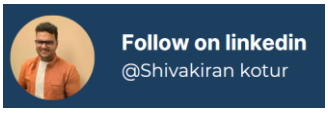


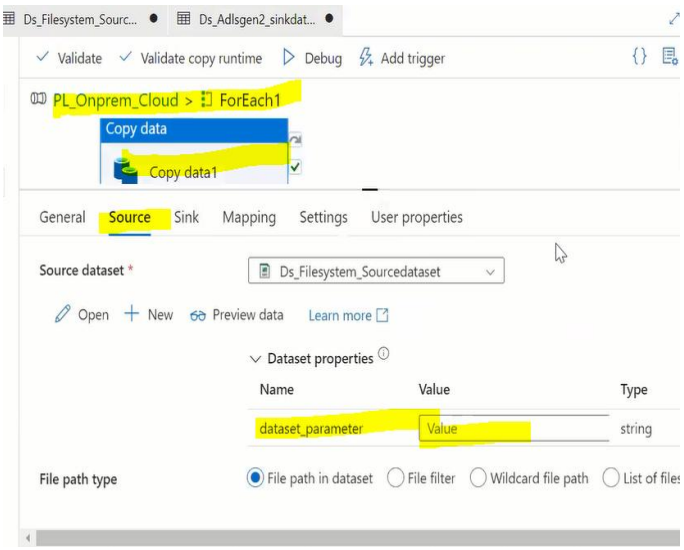
Add→ directory in form of

Raw/filename/yyyy/mm/dd

It should be created as folder.



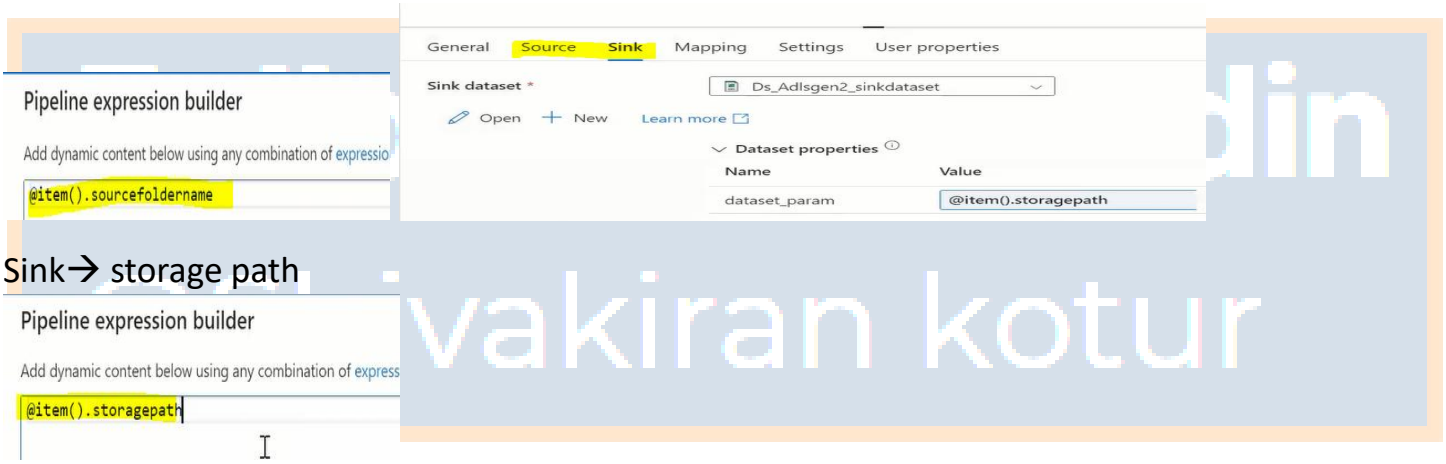




Take output from lookup activity


```
"count": 4,
"value": [
    {
        "sourcefoldername": "orders",
        "storagepath": "orders",
        "isactive": 0,
        "status": "ready"
    },
    {
        "sourcefoldername": "cust",
        "storagepath": "cust",
        "isactive": 0,
        "status": "ready"
    },
```

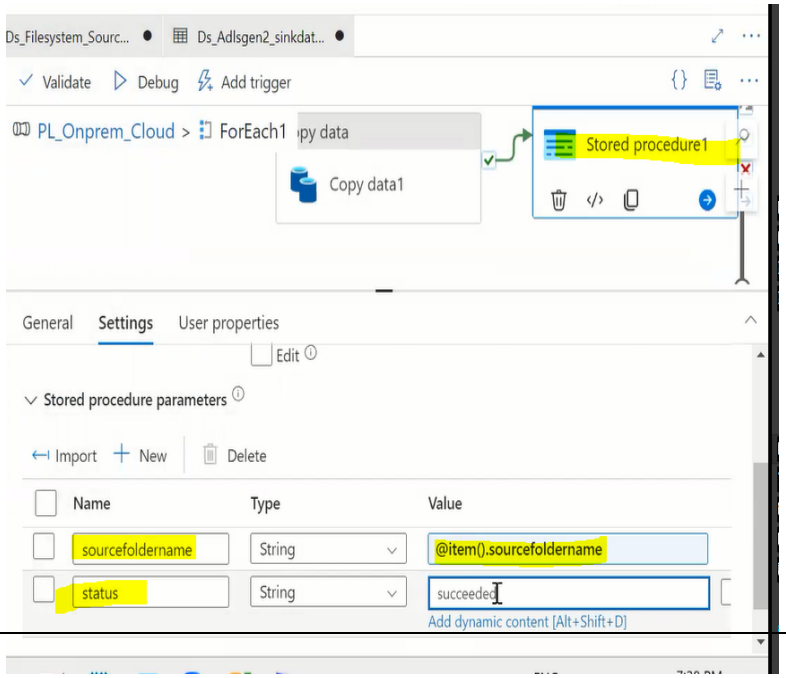

Source→ sourcefoldername



Sink→ storage path

vakiran kotur

Following the copy activity take store proc activity

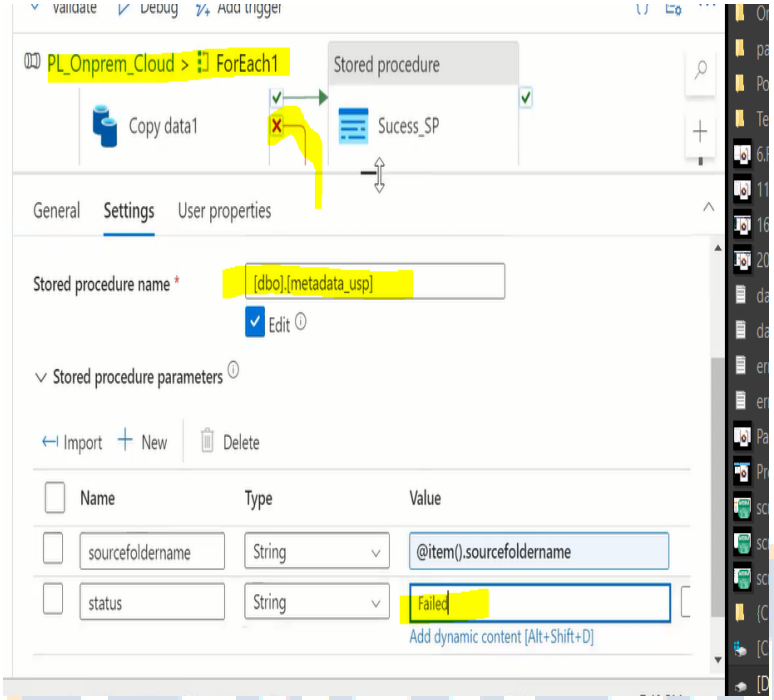


Use→metadat_usp as store proc,

And import parameter,

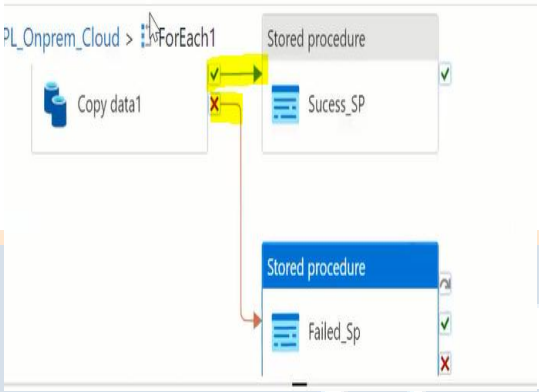
Add sourcefoldername as dynamically.

Status hard code succeeded indicating the copy activity finished



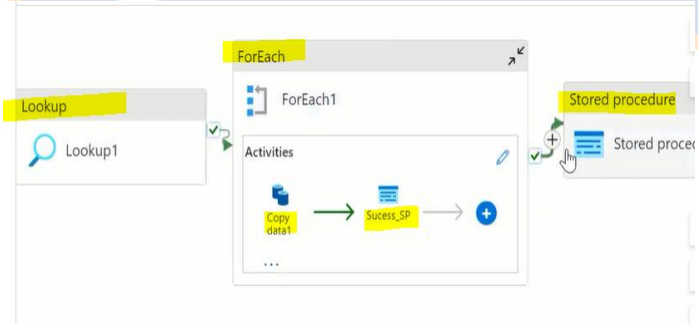
Take another store procedure activity for failure

Use the same store procedure and hardcode status as failed



@Shivakiran kotur

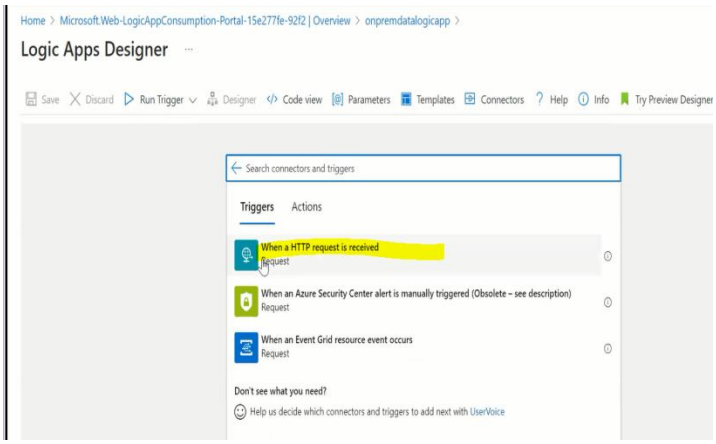
Next take the store procedure outside the for each activity



Here we reset the sp on success

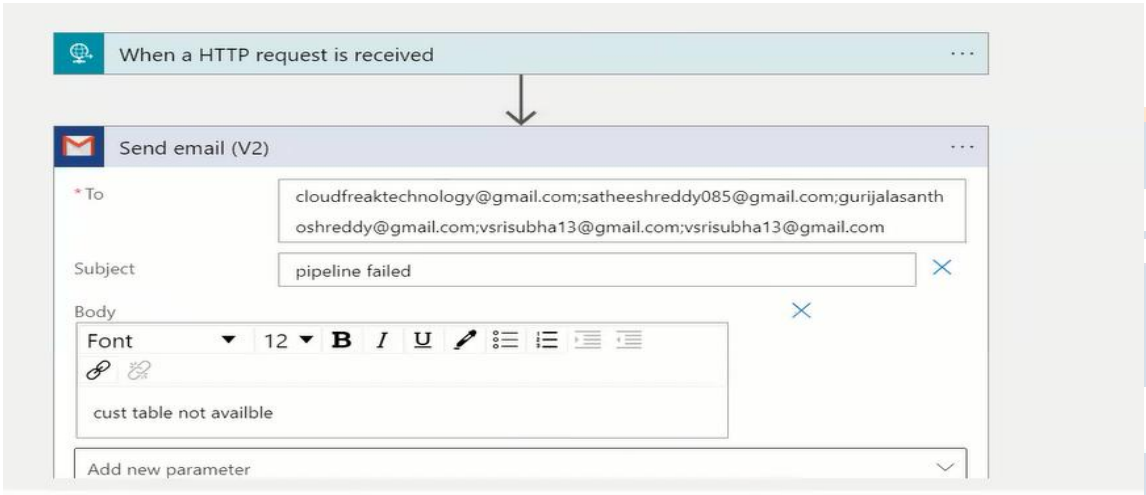
```
CREATE PROCEDURE reset_status_usp
AS
BEGIN
    UPDATE metadata
    SET status = 'ready';
END;
```

Follow on linkedin
@Shivakiran kotur

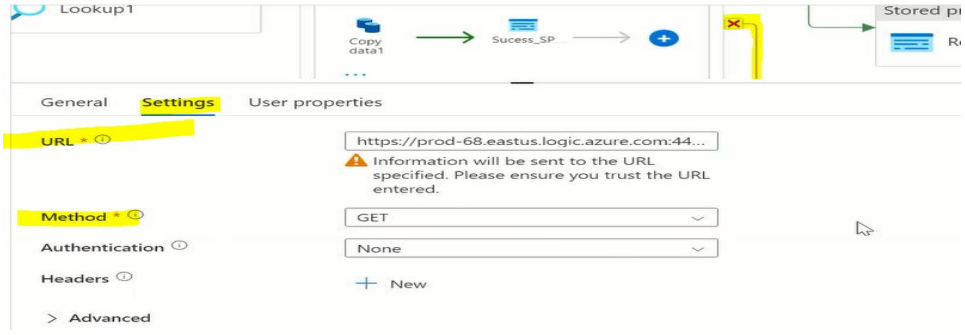


Step → Create a logic app

Create logic app→go to resource→ create blank new→ search for hhtp→select request→ add new parameter→method →GET→ add next step → gmail→send email→name = gmail →sign in → add the to email→ subject → save → url will be generated in http→ copy url

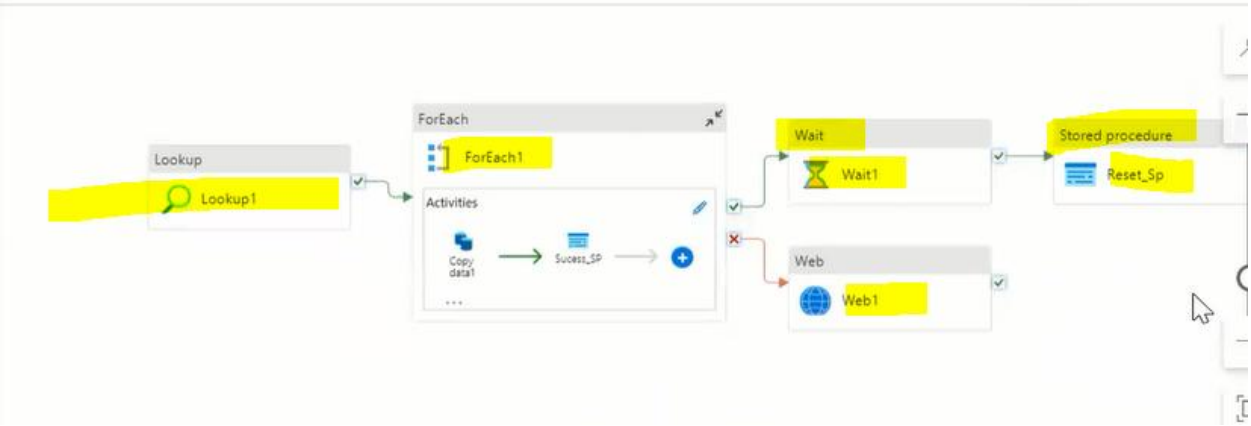


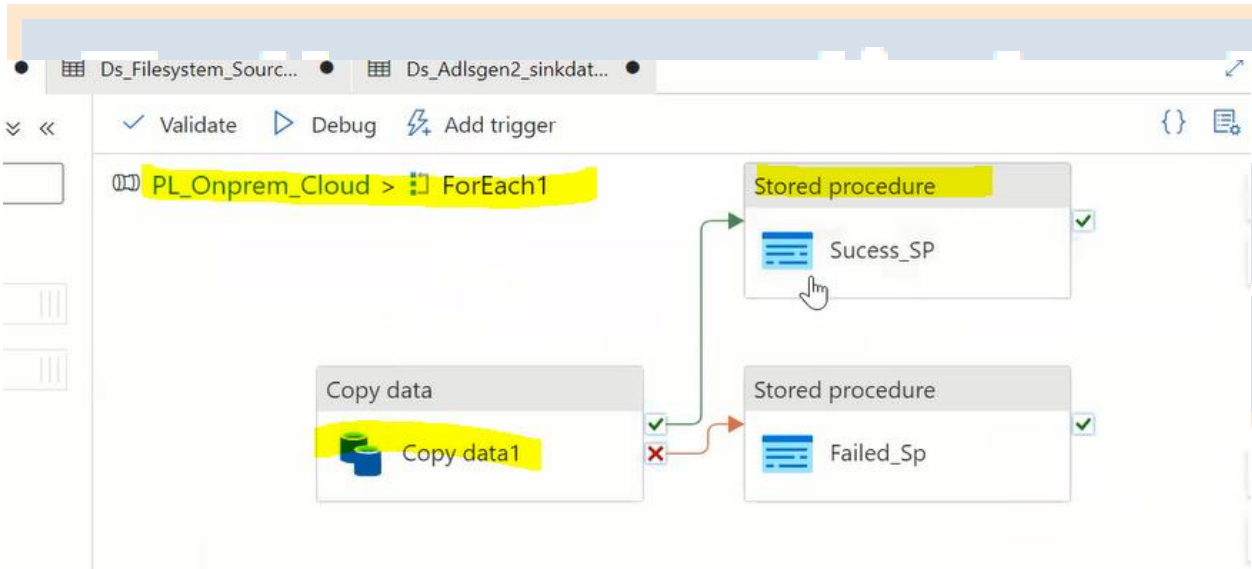
Step→ go to ADF→ on failed of for each take a web activity



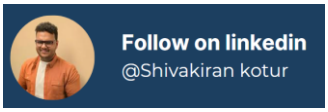
Follow on linkedin
@Shivakiran kotur

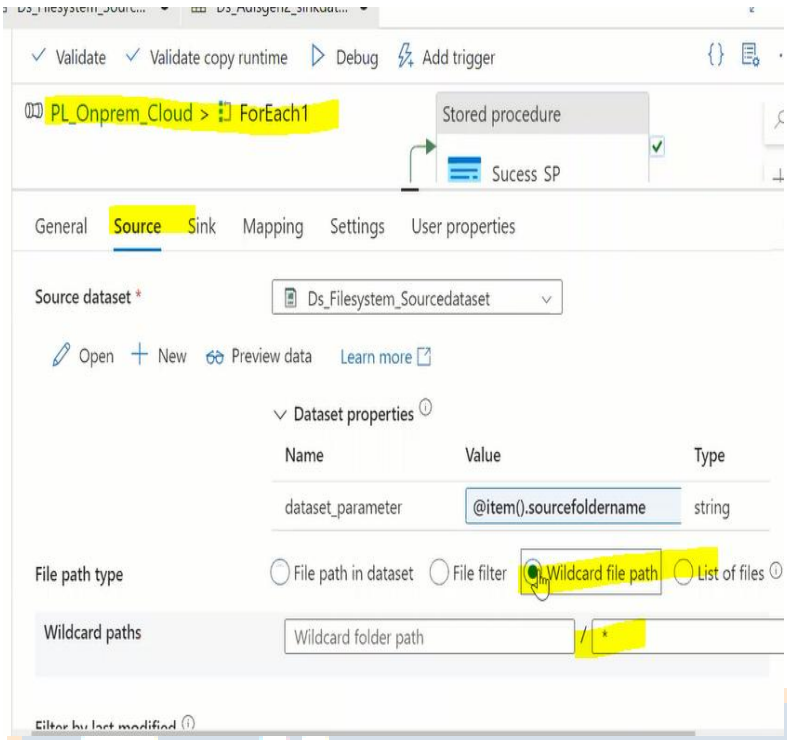
Small change→ add the wait activity after for each loop → 30 secs





Here add the wild card and * as there are folder in source in RDP where the files are added





If any failure like the change in file name or meatadata occurs email is triggered.

Follow on linkedin
@Shivakiran kotur

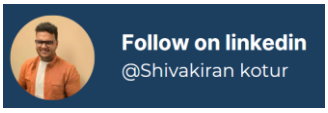
Step 14→ perform data quality checks and move cleansed data from Raw layer to bronze layer.

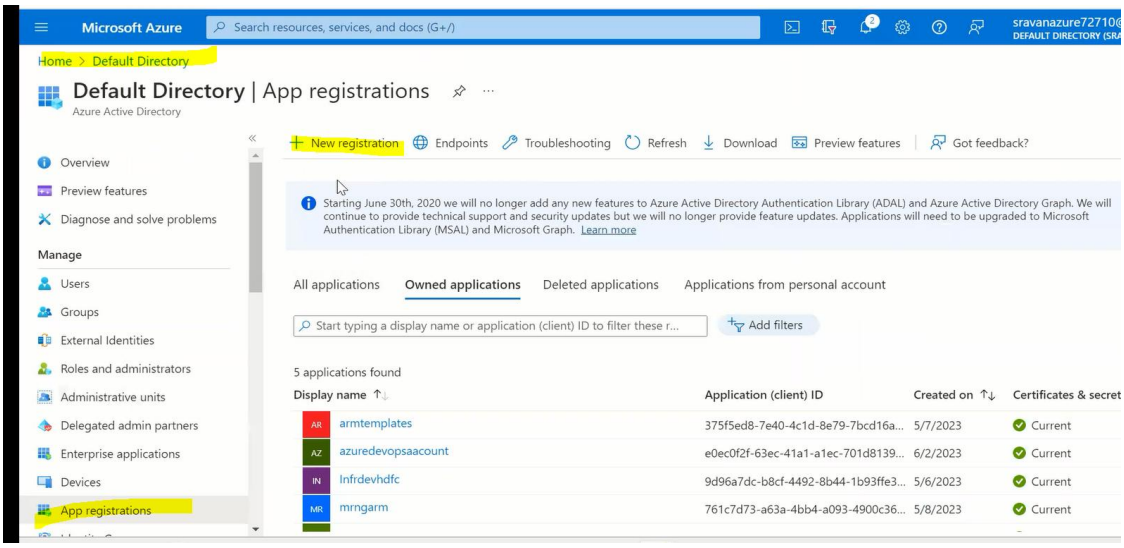
Data quality checks are performed in data bricks →Create Azure databricks service in Azure.

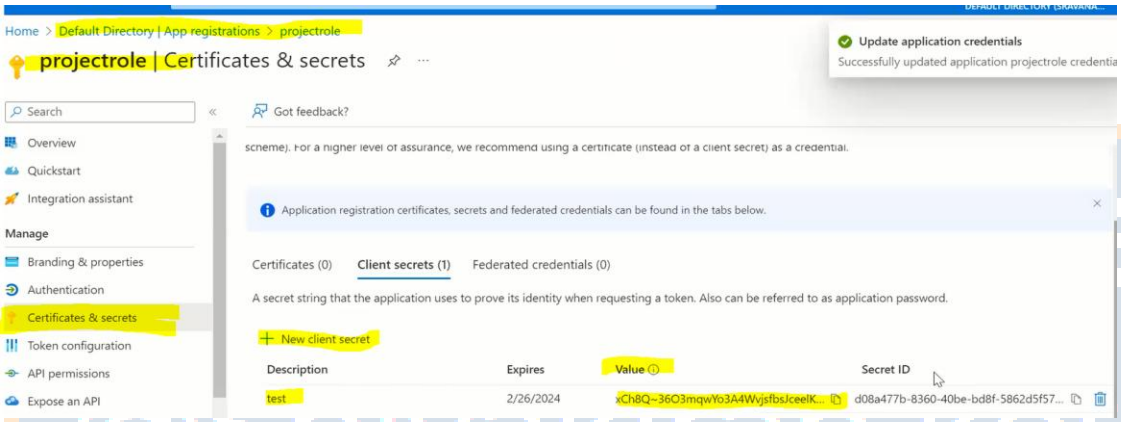
Data source is ADLS container → create a mount point to connect to container (since we are using only one container and inside that container we are moving data from raw folder to bronze folder only 1 mount point is enough)

For mount ADLS → required service→ ADB, Azure key vault and SPN→ i.e App registration(where we create a new client network and from there extract client secret value, client ID, tenant ID and create the key vault secrets for the same)

Create app registration → go to app directory →app registration→once created→ go to secrets and certificate→new client secret → copy the secret value and keep as it will be encrypted once web is closed

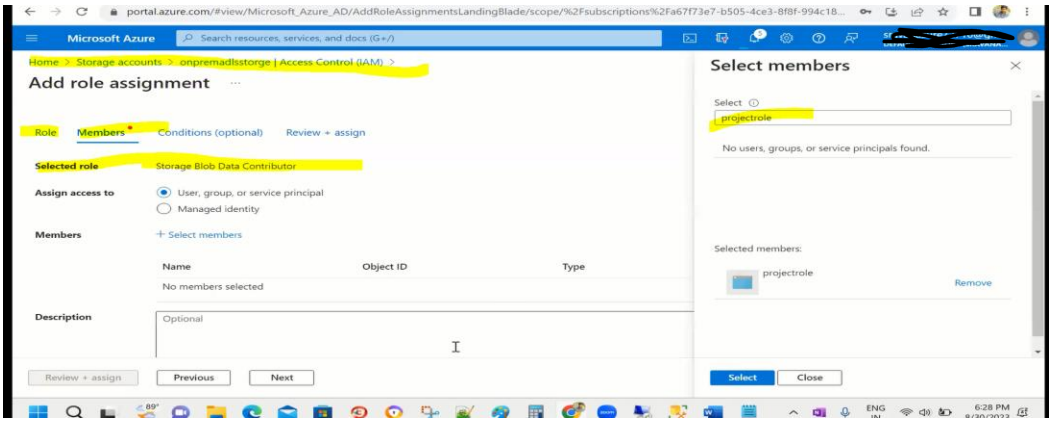






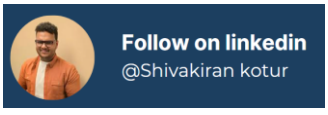
@Shivakiran kotur

Step→ go to your Adls storage account→I am role→ create a role for storage Blob data contributor →assign that to app regist that i.e created above → review and assign



Step 15→ go to ADB →create notebook→

Create dbutilities widgets (once executes the widgets box appears at top through which we can filter the particular data and run the entire notebook)



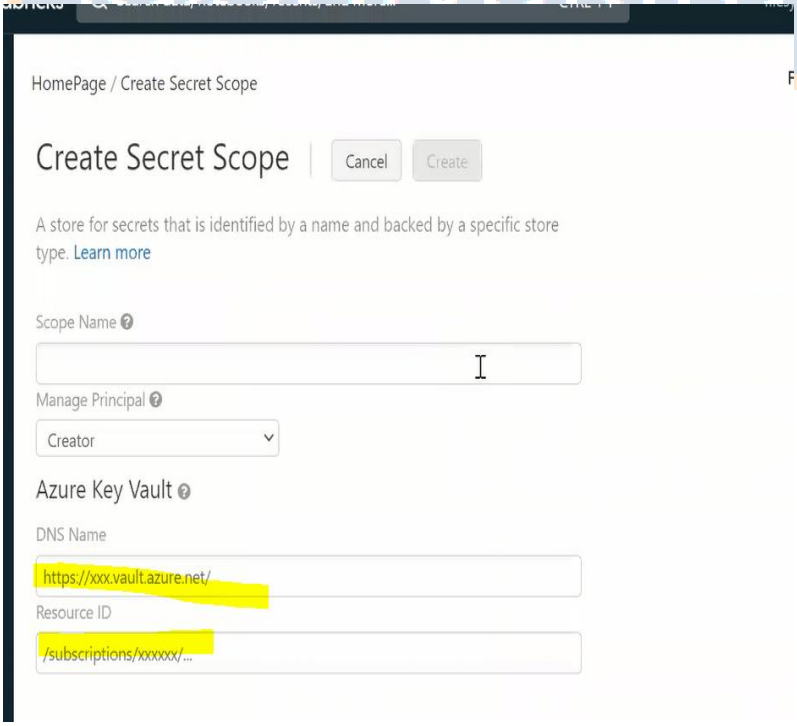
```
dbutils.widgets.text(processeddate,'')
dbutils.widgets.text(foldername,'')
```

step→ create the ADLS mount point

```
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": dbutils.secrets.get(scope="adlsgenkey",key="appid"),
      "fs.azure.account.oauth2.client.secret":
dbutils.secrets.get(scope="adlsgenkey",key="apppwd"),
      "fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/f5ea40f2-
c7b8-4658-8d25-0aac8535e48c/oauth2/v2.0/token",
      "fs.azure.createRemoteFileSystemDuringInitialization": "true"}

dbutils.fs.mount(
source = "abfss://global@adlsgenstorageaccountny.dfs.core.windows.net/",
mount_point = "/mnt/global",
extra_configs = configs)
```

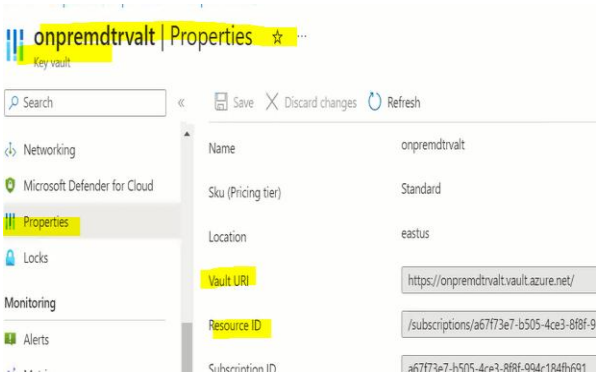
@Shivakiran kotur

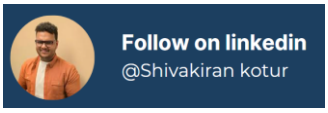
create the scope by adding attend of url as →#secrets/createScope

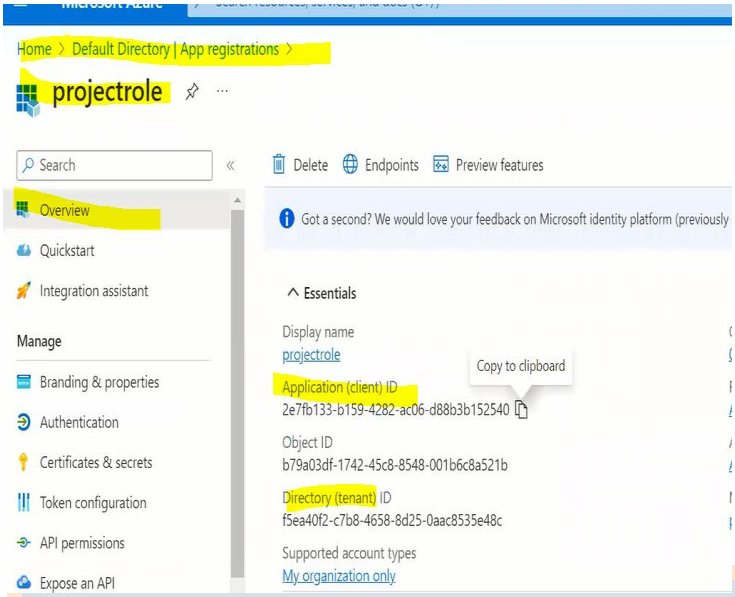


Give the scope name → adlsgenkey

DNS name and rescource ID → from azure key vault







in mount point→

client.secret":→ i.e apppwd the secret value copied from app registration

client id→ application id

tenant id copy and paste in

fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/<tenantID > /oauth2/v2.0/token

source = "abfss://<containername>@<storageaccountname>.dfs.core.windows.net/",

Follow on linkedin

once execute comment all (select all in text and Ctrl+/ shortcut to comment all together)

Step 16→Aim is to move data from raw to bronze

@Shivakiran kotur

```
src_path="/mnt/global/raw"
```

Cmd 3

```
dbutils.widgets.text('processeddate','') # 2023/08/30
dbutils.widgets.text('foldername','')    # cust
```

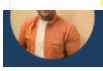
Command took 0.07 seconds -- by sravanazure72710@gmail.com at 8/30/2023, 6:45:22

Cmd 4

```
foldername=dbutils.widgets.get('foldername','')
pdate=dbutils.widgets.get('processeddate','')
```

Cmd 5

```
src_final_path =src_path+pdate+"/"+pdate
```

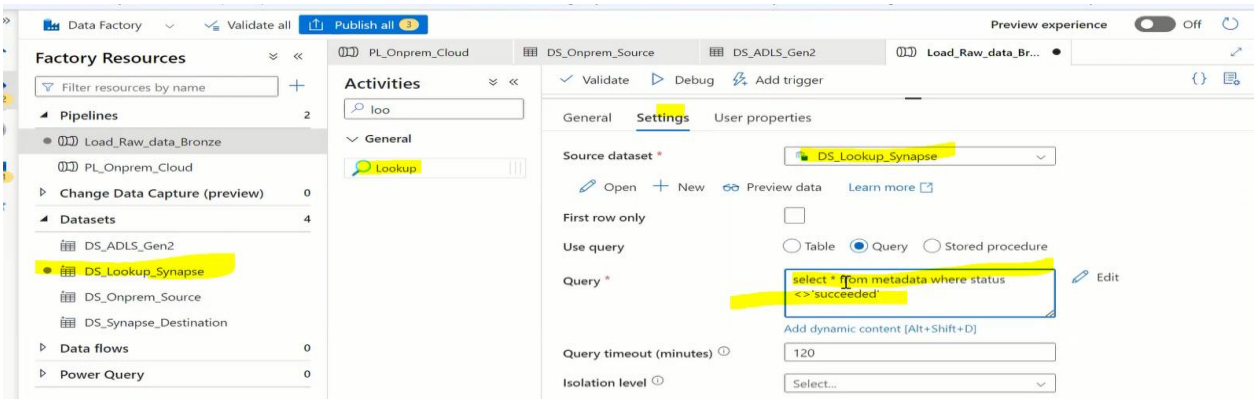


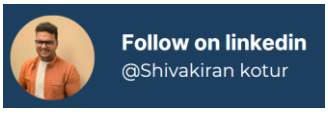

```
src_path="/mnt/global/raw/"
dest_path="/mnt/global/bronze/"

dbutils.widgets.text(processeddate,'')
dbutils.widgets.text(foldername,'')

foldername = dbutils.widgets.get(' foldername')
pdate = dbutils.widgets.get(' processeddate')

print(foldername )
print(cdate)
src_final_path=src_path+foldername+"/"+pdate
print(src_final_path)
dest_final_path=dest_path+foldername+"/"+pdate
```

```
print(dest_final_path)
#following is the code for cleaning the data
try:
    # Read data from source path
    df = spark.read.format("csv").option("header", True).load(src_final_path)

    # Count the number of rows in the source DataFrame
    src_count = df.count()
    print("Source count:", src_count)

    # Remove duplicates
    df1 = df.dropDuplicates()

    # Count the number of rows in the destination DataFrame
    dest_count = df1.count()
    print("Destination count:", dest_count)

    # Write the cleaned data to the destination path
    df1.write.mode("overwrite").format("csv").option("header", True).save(dest_final_path)

    # Exit the notebook with success message and counts
    print("Success: Source count = " + str(src_count) + ", Destination count = " + str(dest_count))
except Exception as e:
    # Handle exceptions and exit with an error message
    dbutils.notebook.exit("Error: " + str(e))
```

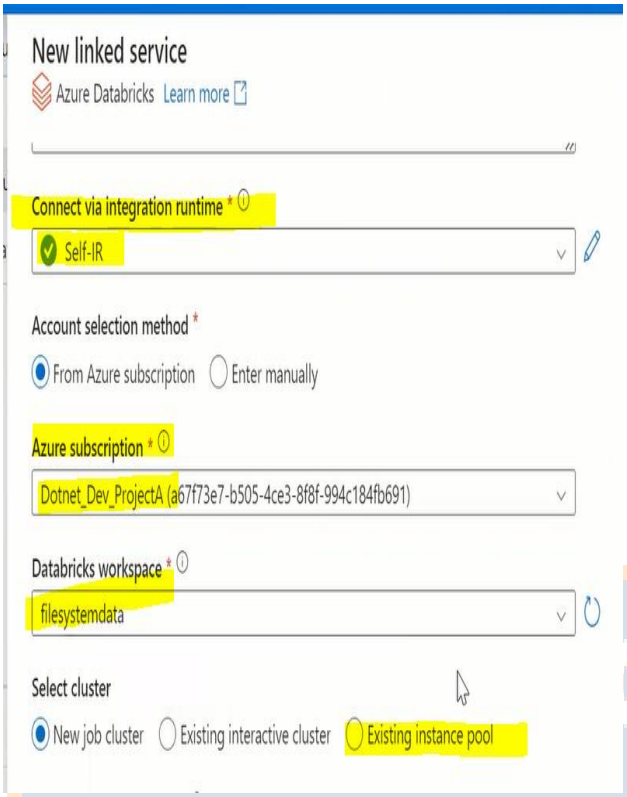
Step 17→ go to ADF to create the pipeline for the movement of data from raw to bronze

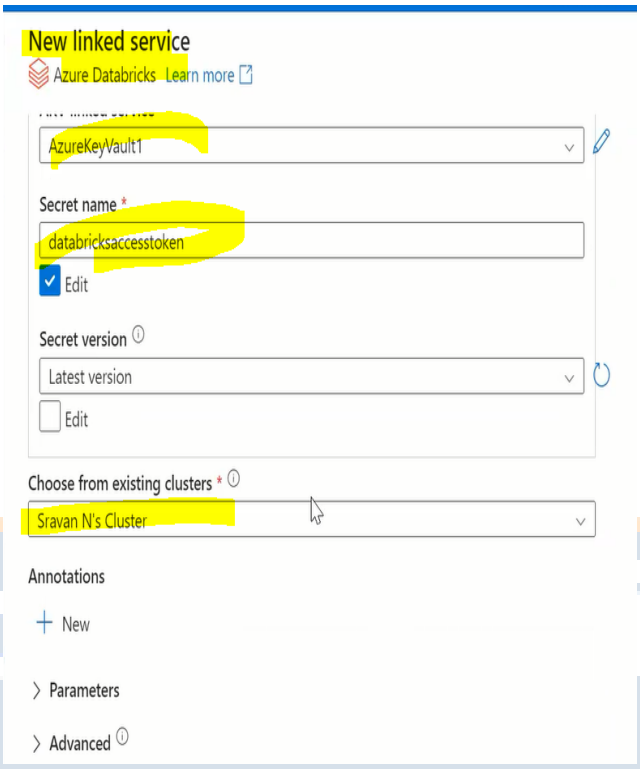
Take the lookup activity →create the source dataset for synapse →

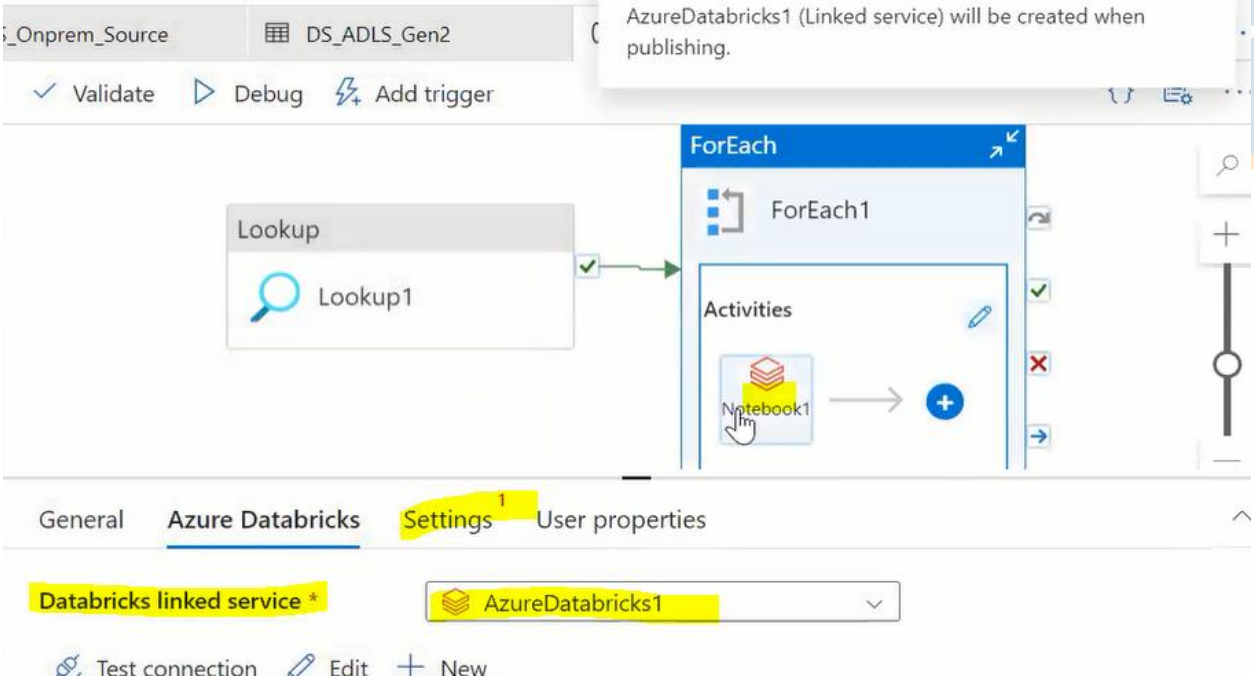


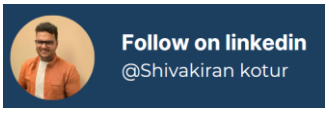


Next take→ foreach activity→ take output of lookup as input→ go inside for each → take notebook activity → create linked service→ dataset for notebook

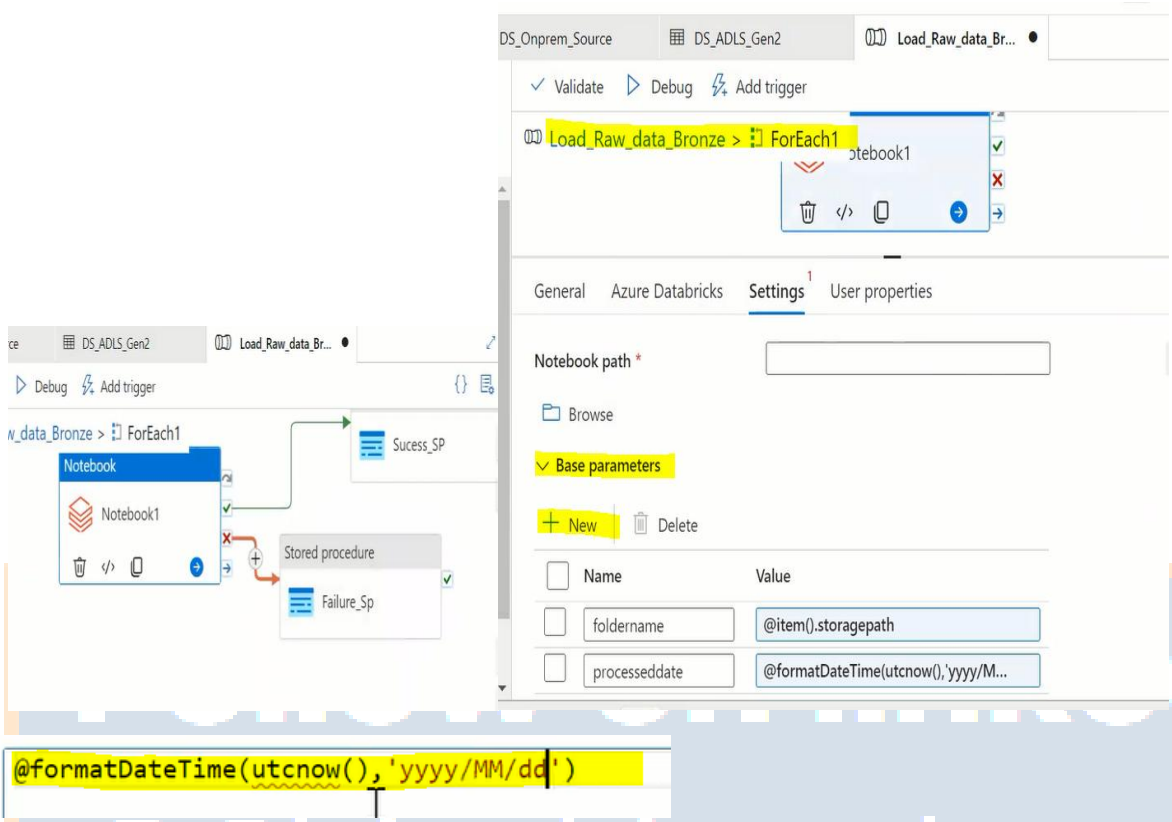








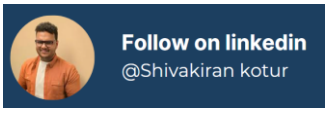
Under setting→ we have to pass 2 base parameters (as we have take processeddate and foldername as widgets in notebook)

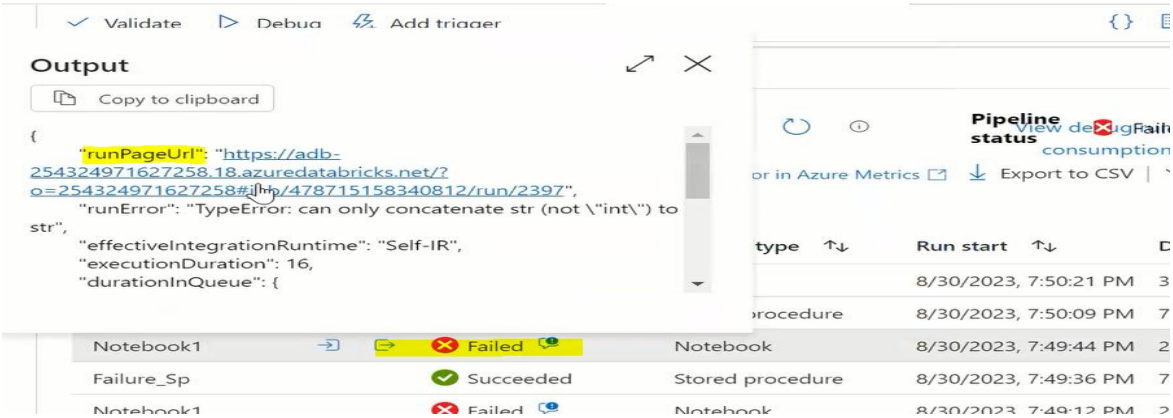


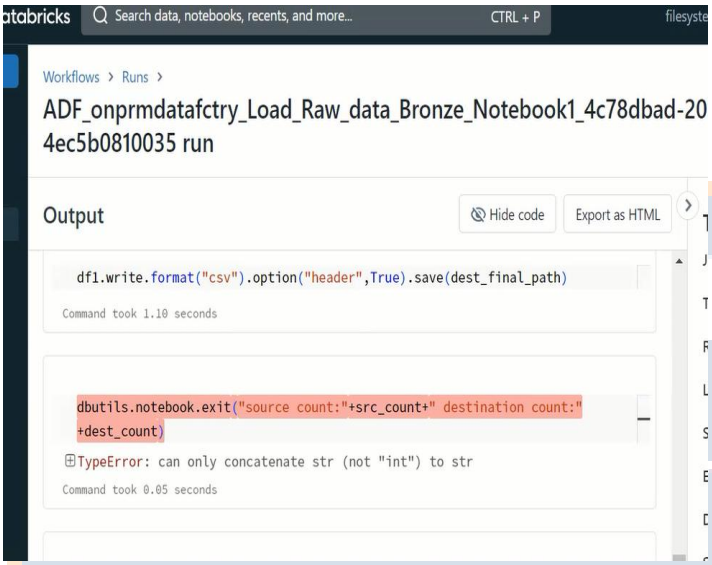
Simialry to last pipeline→following the notebook activity take 2 store proc , 1 for success and other for failure add the parameter

Outside the for each take reset store proc and web activity for failure where we call the logic app

If any errors in notebook, pipeline fails and in output of the activity we get the notebook url, click on that and it directly goes to ADB, and one with highlighted one is having error fix it and run the pipeline again.







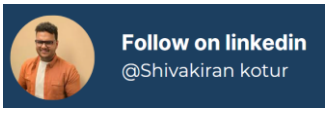
cannot edit here directly→ so go to the main development branch and edit the code.

n linkedin
an kotur

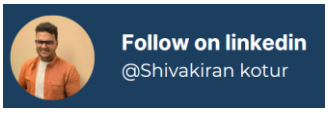
Step 18→ to do transformation and move data from bronze layer to Silver layer

Below is the scripts to perform transformation in notebook from Bronze to silver, here only join transfprmtion for cust table is performed

```
# Set source and destination paths
src_path = "/mnt/global/bronze/"
dest_path = "/mnt/global/silver/"

# Input widgets for folder name and processing date
dbutils.widgets.text('foldername', '')
dbutils.widgets.text('pdate', '')

try:
   # Get user input for folder name and processing date
```



```
foldername = dbutils.widgets.get('foldername')
pdate = dbutils.widgets.get('pdate')

print("Folder Name:", foldername)
print("Processing Date:", pdate)

# Create source and destination paths based on user input
src_final_path = src_path + foldername + "/" + pdate
print("Source Path:", src_final_path)

# Destination path for writing processed data
dest_final_path = dest_path + 'dim' + foldername
print("Destination Path:", dest_final_path)

# Load data from the source path
df = spark.read.format("csv").option("header", True).load(src_final_path)
src_count = df.count()
print("Source Count:", src_count)

# Display the DataFrame
df.show()

# Create a sample DataFrame (df11) - replace this with your actual data
df11 = spark.createDataFrame([(2, '78654345'), (3, '67865467')], ['cid', 'cphone'])
df11.show()

# Join dataframes if foldername is 'cust', otherwise use df as is
from pyspark.sql.functions import col

if foldername == 'cust':
    df1 = df.alias('a').join(df11.alias('b'), col('a.cid') == col('b.cid'), "inner").select('a.*',
'b.cphone')
    df1.show()
else:
    df1 = df

# Count rows in the destination DataFrame
dest_count = df1.count()
```

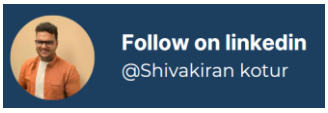


```
    # Write processed data to the destination path
    df1.coalesce(1).write.mode("overwrite").format("csv").option("header",
True).save(dest_final_path)

    print("Processing completed successfully.")
    print("Source Count:", src_count)
    print("Destination Count:", dest_count)
    dbutils.notebook.exit("Processing completed successfully.")
except Exception as e:
    print("Error:", str(e))
    dbutils.notebook.exit("Error: " + str(e))
```

Create a pipeline similar to above raw to Bronze, change the notebook and provide the base parameters properly

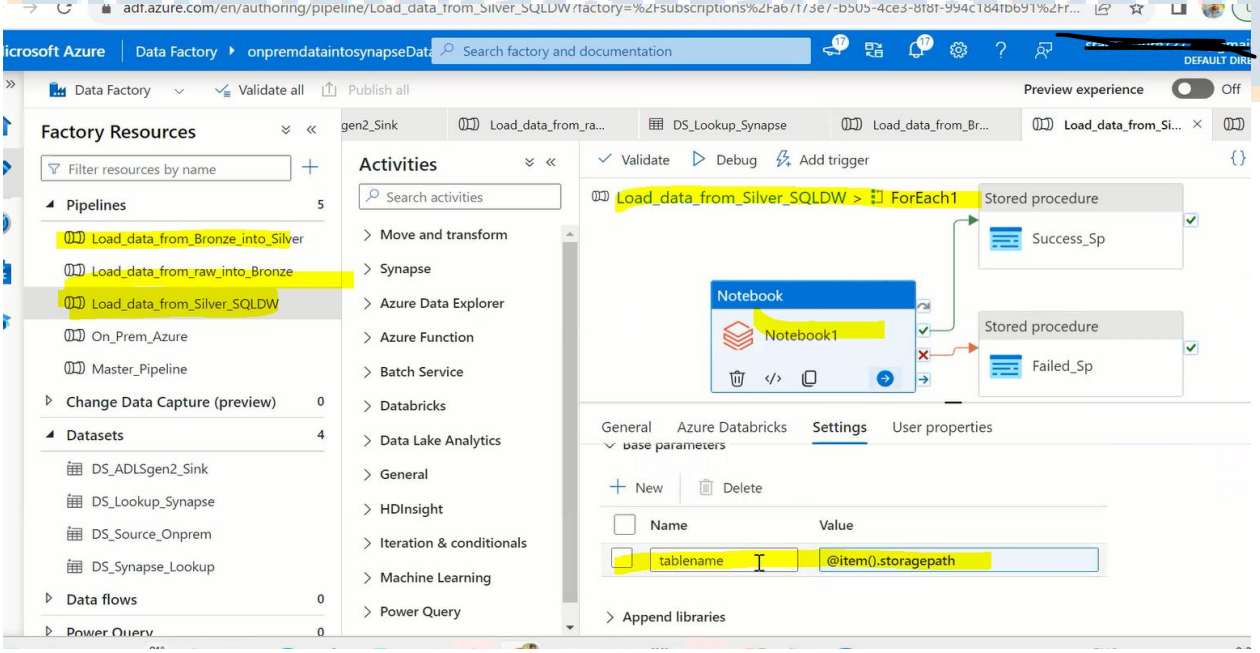
Step 19→ move data from Silver layer to Sql DW

```
print("Source Count:", src_count)
print("Destination Count:", dest_count)

# Load SQL data into the data warehouse
dbutils.widgets.text('foldername', '')

foldername = dbutils.widgets.get('foldername')
print("Folder Name:", foldername)

# Set source and destination paths for SQL data
src_path = "/mnt/global/silver/" + 'dim' + foldername
dest_path = "dim" + foldername
print("Source Path:", src_path)
print("Destination Path:", dest_path)

# Read data from the source path
df = spark.read.format("csv").option("header", True).load(src_path)
src_count = df.count()
print("Source Count:", src_count)

# Set Azure Storage account key
```



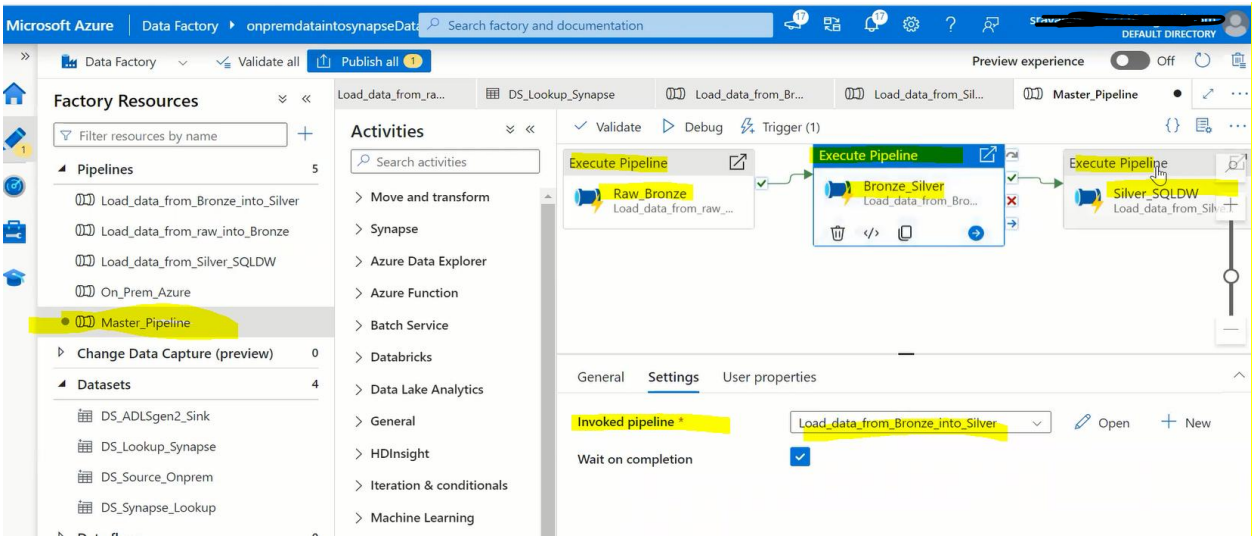
```
spark.conf.set("fs.azure.account.key.onpremdatasynasegen.dfs.core.windows.net",
"o82RdY56QpidiJOBzA0+c0xBYomGajKVXZ8oZKRr+TtVSjYOTI5+i6IVTmOFL5E73Ha5wJHe7aQ1+
AStdIFwNA==")

# Write data to SQL Data Warehouse (using JDBC connection from key vault)
df.write \
   .mode("overwrite") \
   .format("com.databricks.spark.sqldw") \
   .option("url", dbutils.secrets.get(scope="adlsgenkey", key="sqljdbcpwd")) \
   .option("dbtable", dest_path) \
   .option("tempDir",
"abfss://global@onpremdatasynasegen.dfs.core.windows.net/tmp/synapse") \
   .option("forwardSparkAzureStorageCredentials", "true") \
   .save()
# Display source count
print("Source Count:", src_count)
dbutils.notebook.exit("Source Count: " + str(src_count) + ", Destination Count: " +
str(dest_count)
```

Create a pipeline similar to above here we give only 1 base parameter



Follow on linkedin
@Shivakiran kotur

Step 20→Create a master pipeline to execute all the pipeline using execute pipeline activity



Follow on linkedin
@Shivakiran kotur