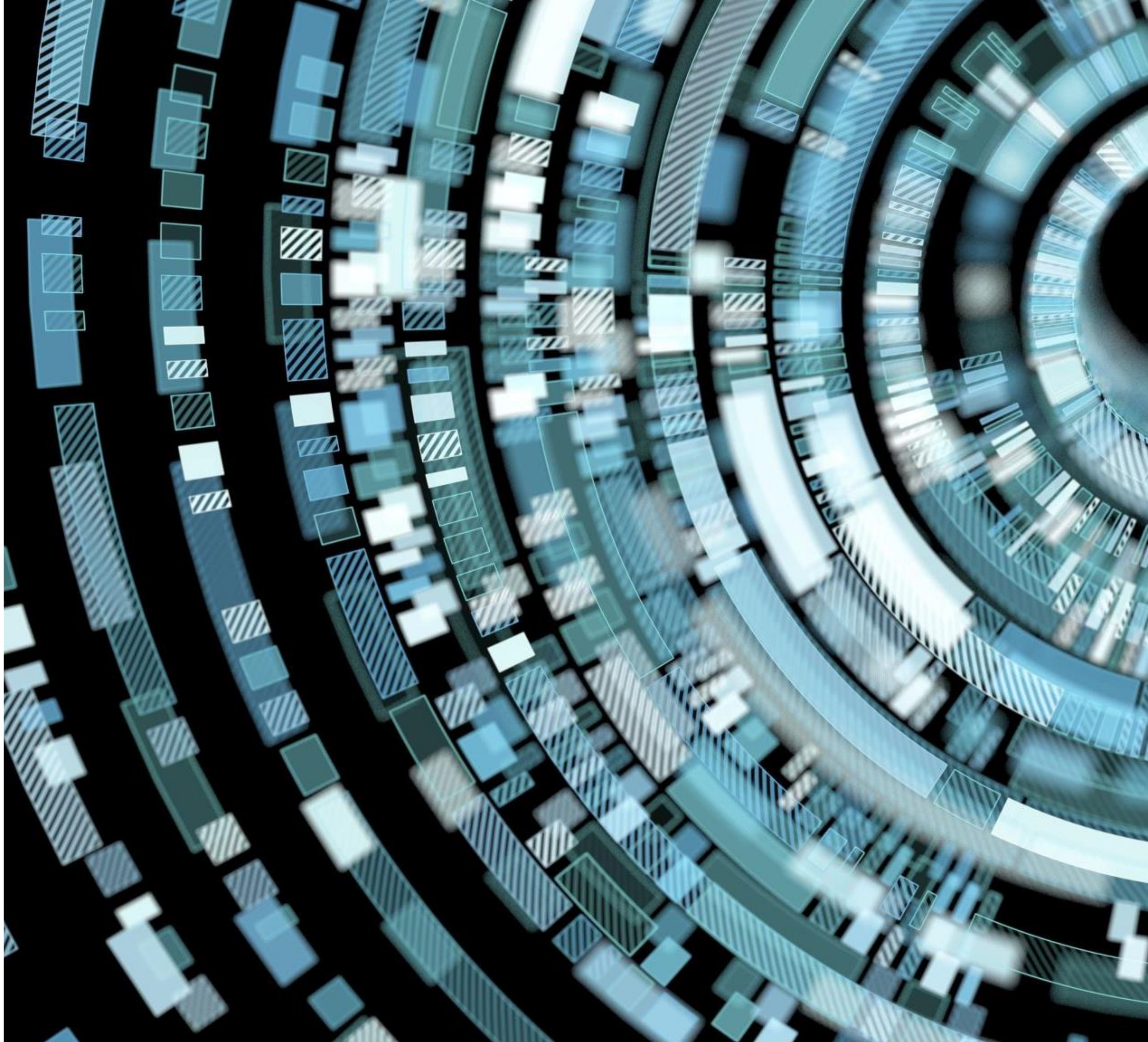
DATABRICKS SPARK HIVE DEMO

https://advancedsqlpuzzles.com

Scott Peters



OBJECTIVE

For this demo, we will perform the following…

- Connect Databricks to an Azure Key Vault
- Connect Databricks to an Azure Data Lake
- Create an ETL process to import csv files from an Azure Data Lake
- Merge the data into Hive tables
- Insert the data into a SQL Server database
- Automate the ETL via an Azure Data Factory pipeline

OBJECTIVE

The Git repository for this demo is located here:
https://Github.com/smpetersGithub/AdvancedSQLPuzzles/tree/main/Suppliers%20and%20Parts%20Hive%20Demo

Included in the repository are the following:

- A PDF version of this presentation (Databricks Spark Hive Demo.pdf)
- Databricks Workspace (SuppliersAndParts.dbc)
- CSV Files to Import
- SQL Server Database DDL Script

If you are viewing on YouTube, each slide is set to 20 seconds.
You can pause using the spacebar, and the keys “J” and “L” will rewind and fast forward 10 seconds.

OBJECTIVE

For this demo we will be using the “Suppliers and Parts” database which consists of three csv files.

1) Suppliers.csv
2) Parts.csv
3) Shipments.csv

It’s a popular database that has its own Wikipedia page.

The Suppliers and Parts Wikipedia article is located here:
https://en.wikipedia.org/wiki/Suppliers_and_Parts_database

AZURE RESOURCES

First, we will need to provision the following resources:

- Azure Key Vault
- Azure Data Lake
- Azure Data Factory
- SQL Server
- SQL Server Database
- Databricks Workspace

AZURE RESOURCES

Few reminders when provisioning your resources.

- When provisioning Databricks, select the premium version as this allows for creating secret scopes.

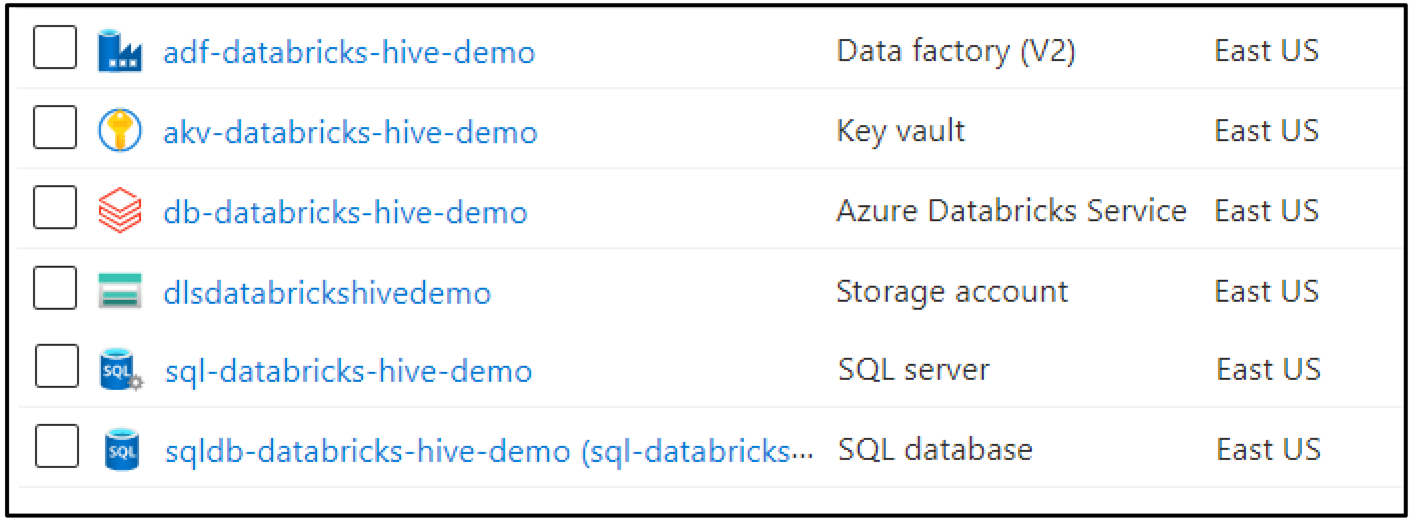
- If using an Azure free trial version, you will need to upgrade to a subscription plan to access Databricks.

- The Azure Storage Account needs to be provisioned as a data lake.

- A basic SQL Server database is will be sufficient for this tutorial. You will need to enable the database server to connect to other Azure resources.

AZURE RESOURCES

For this demo I created a resource group "rg-databricks-hive-demo" and provisioned the following resources:



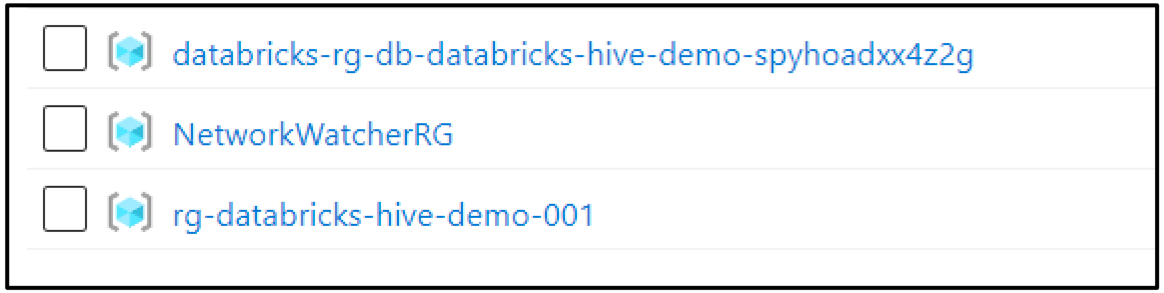
□	adf-databricks-hive-demo	Data factory (V2)	East US
□	akv-databricks-hive-demo	Key vault	East US
□	db-databricks-hive-demo	Azure Databricks Service	East US
□	dlsdatabrickshivedemo	Storage account	East US
□	sql-databricks-hive-demo	SQL server	East US
□	sqldb-databricks-hive-demo (sql-databricks...	SQL database	East US

The documentation for Azure naming conventions is located here: https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-best-practices/resource-abbreviations

AZURE RESOURCES

When you provision the resources, additional resource groups and resources are created that you do not have control of.

When you have finished provisioning the resources, you will see an additional resource group for Databricks, and another for the Network Watcher.



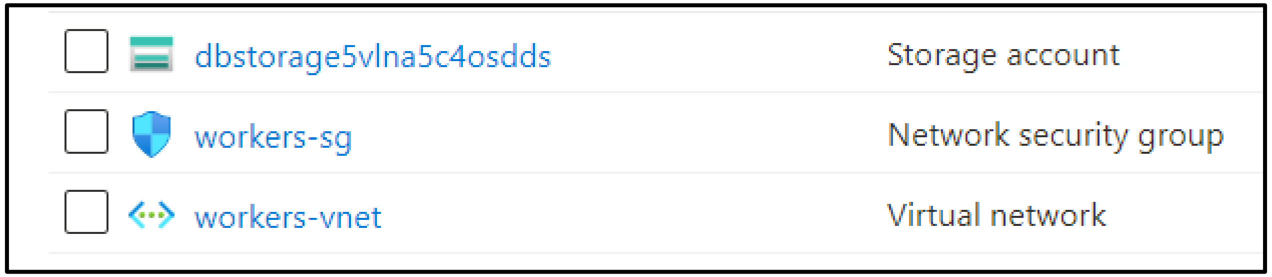
□ databricks-rg-db-databricks-hive-demo-spyhoadxx4z2g

□ NetworkWatcherRG

□ rg-databricks-hive-demo-001

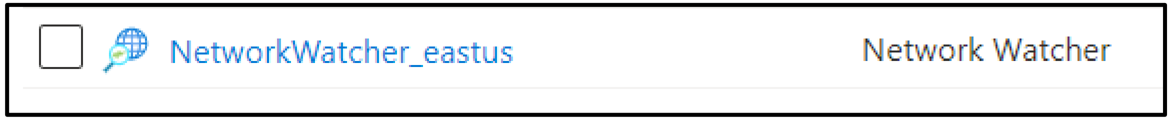
AZURE RESOURCES

The “databricks-rg-db-databricks-hive-demo-spyhoadxx4z2g” resource group has the following resources:



□	dbstorage5vlna5c4osdds	Storage account
□	workers-sg	Network security group
□	workers-vnet	Virtual network

The “NetworkWatcherRG” resource group has the following resources:



□	NetworkWatcher_eastus	Network Watcher

AZURE RESOURCES

Here are the Azure resource names if you want to copy and paste.

- db-databricks-hive-demo (Databricks)
- dlsdatabrickshivedemo (Azure Data Lake)
- akv-databricks-hive-demo (Azure Key Vault)
- sqldb-databricks-hive-demo (SQL Database)
- sql-databricks-hive-demo (SQL Server)
- adf-databricks-hive-demo (Azure Data Factory)

DATABRICKS

Next, we will connect our resources together.

In Databricks, we will need to understand two concepts:

1) Secret Scopes
2) Databricks Tokens

First, we will begin with secret scopes.

Secret scopes allow you to securely connect to your Azure Key Vault service, where we will store our database and storage account credentials.

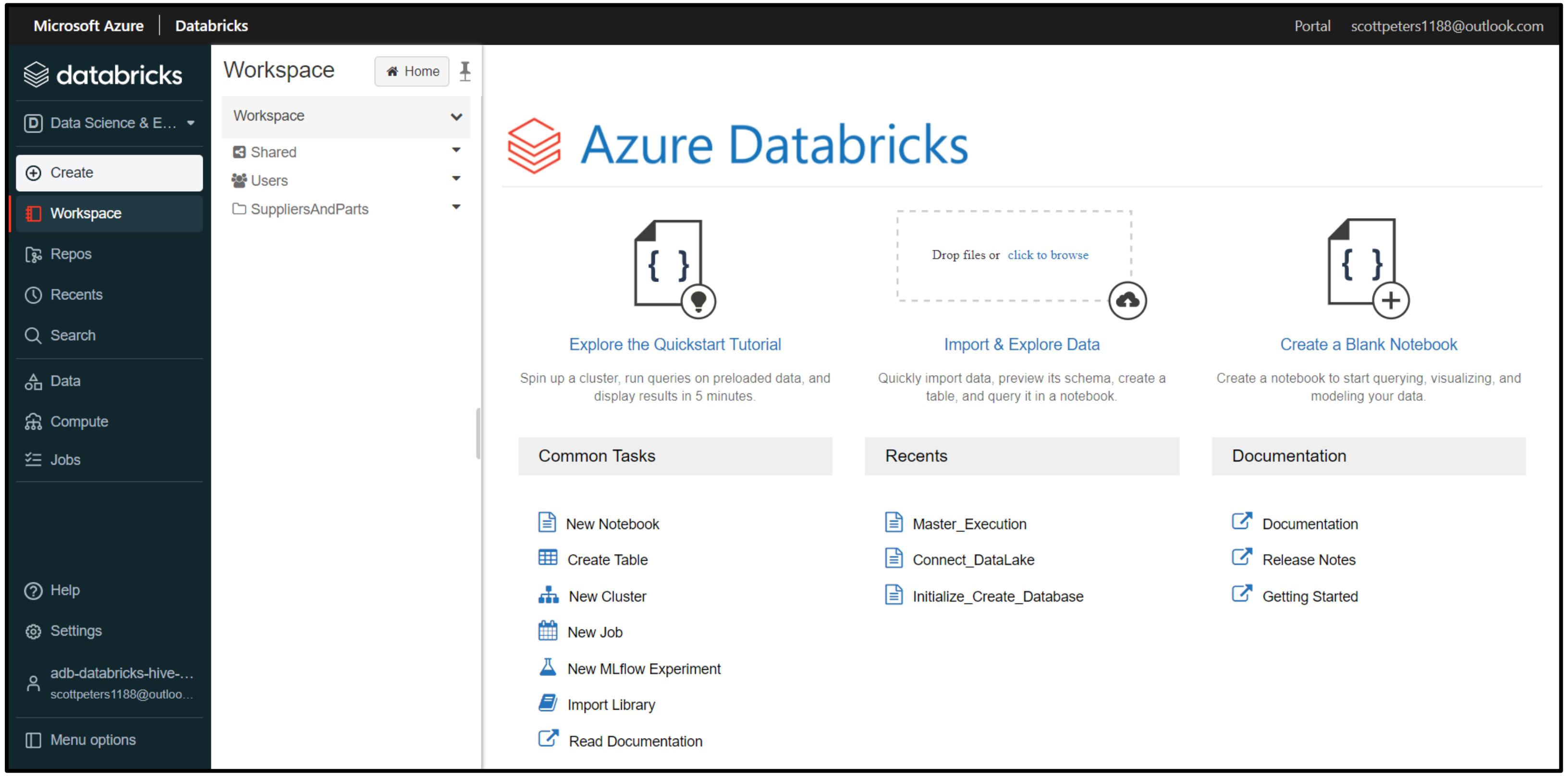
There are two different ways of creating a scope:

1) Azure Key Vault backed
2) Databricks backed

The Databricks secret scope documentation is located here:
https://docs.microsoft.com/en-us/azure/databricks/security/secrets/secret-scopes

DATABRICKS

Open the Databricks Workspace.



Spin up a cluster, run queries on preloaded data, and display results in 5 minutes.

Quickly import data, preview its schema, create a table, and query it in a notebook.

Create a notebook to start querying, visualizing, and modeling your data.

DATABRICKS

Next, in the Databricks workspace, access the secret scope window.

To access the secret scope window in Databricks, attach the string "#secrets/createScope" after your Databrick's instance.

A Databrick's instance will have the following URL:
https://<databricks-instance>#secrets/createScope

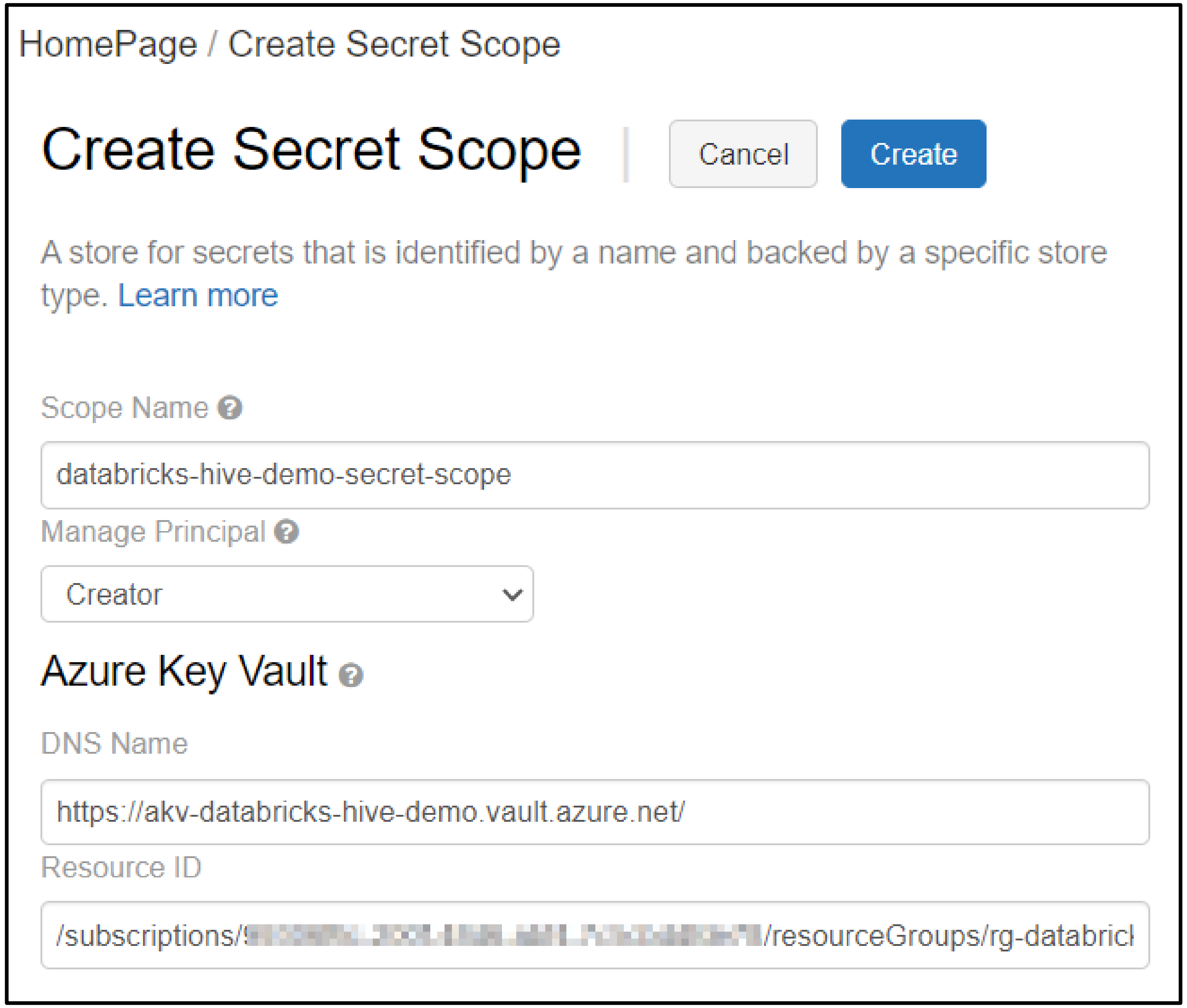
This URL is case sensitive; scope in createScope must be uppercase.

The documentation for Databricks secret scopes is located here:
https://docs.microsoft.com/en-us/azure/databricks/security/secrets/secret-scopes

DATABRICKS SECRET SCOPE

The “Create Secret Scope” window will look like the following.

We will cover the “DNS Name” and “Resource ID” in next slides.



HomePage / Create Secret Scope

Create Secret Scope

A store for secrets that is identified by a name and backed by a specific store type. Learn more

Scope Name: databricks-hive-demo-secret-scope

Manage Principal: Creator

Azure Key Vault

DNS Name: https://akv-databricks-hive-demo.vault.azure.net/

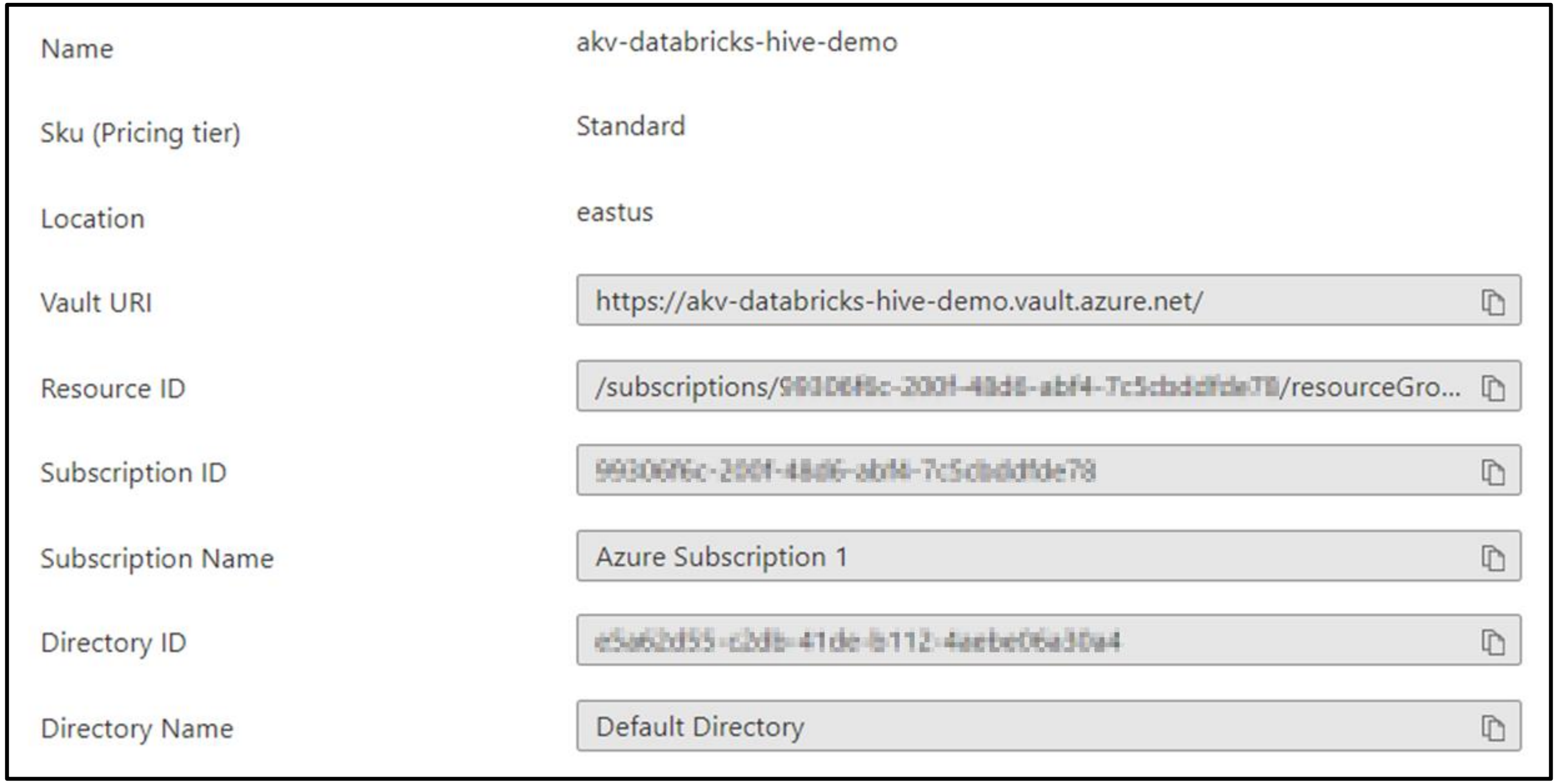
Resource ID: /subscriptions/[redacted]/resourceGroups/rg-databricl

DATABRICKS SECRET SCOPE

The DNS Name (Vault URI) and the Resource ID are in your Azure Key Vault properties page.

Copy these values from your Azure Key Vault properties page and insert the values into your “Create Secret Scope” window.

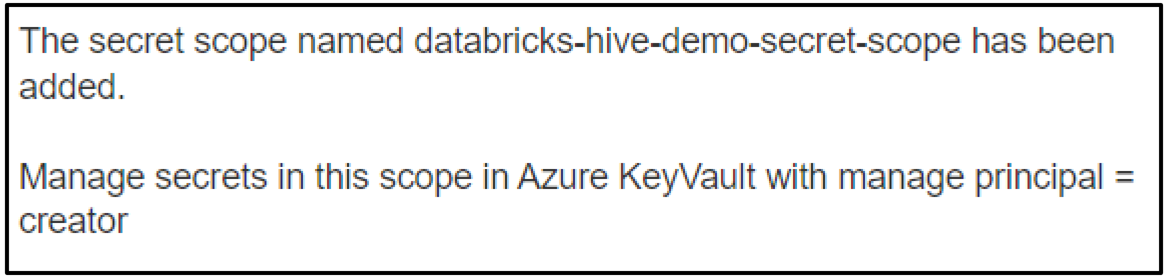
Azure Key Vault Properties Page



Name	akv-databricks-hive-demo
Sku (Pricing tier)	Standard
Location	eastus
Vault URI	https://akv-databricks-hive-demo.vault.azure.net/
Resource ID	/subscriptions/[redacted]/resourceGro...
Subscription ID	[redacted]
Subscription Name	Azure Subscription 1
Directory ID	[redacted]
Directory Name	Default Directory

DATABRICKS SECRET SCOPE

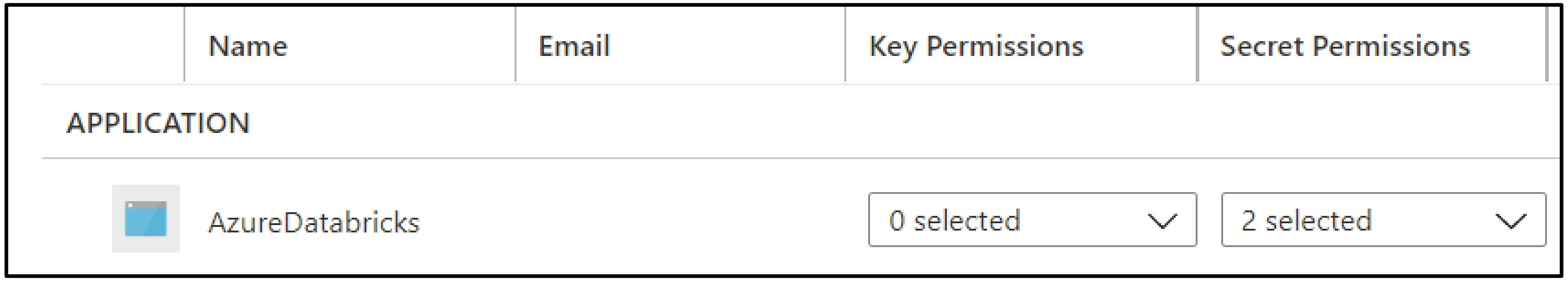
Once completed, you will get the following confirmation.



The secret scope named databricks-hive-demo-secret-scope has been added.

Manage secrets in this scope in Azure KeyVault with manage principal = creator

And you will see the following access policy added to the key vault.



	Name	Email	Key Permissions	Secret Permissions
APPLICATION				
	AzureDatabricks		0 selected	2 selected

The documentation for the Azure Key Vault access policies is located here: https://docs.microsoft.com/en-us/azure/key-vault/general/assign-access-policy?tabs=azure-portal

DATABRICKS TOKEN

Next, we will create a Databricks token.

This token will allow the Azure Data Factory to access the Databricks notebook. Tokens replace passwords in an authentication flow and should be protected like passwords.

We will use this token when we setup a linked service in our Azure Data Factory.

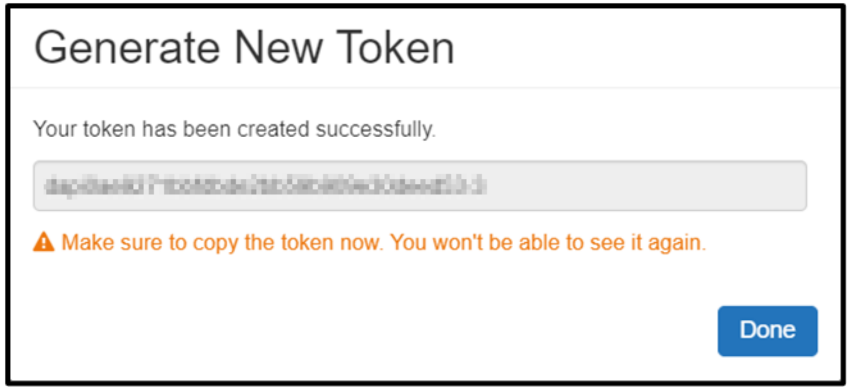
The documentation for Databricks tokens is located here:
https://docs.databricks.com/dev-tools/api/latest/authentication.html

This token will be stored in the Azure Key Vault, which we will setup later.

DATABRICKS TOKEN

Create a token in Azure Databricks by navigating to the “User Settings” in your Databricks workspace and selecting “Generate New Token”.

Save the token, as directed by the yellow warning message.



Generate New Token

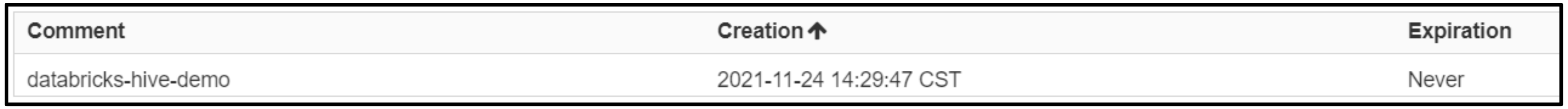
Your token has been created successfully.

[redacted]

⚠ Make sure to copy the token now. You won't be able to see it again.

I created the token “databricks-hive-demo”, but it is the token’s value that is important, not the name. This token will be stored in the Azure Key Vault, which we will setup later.

Once created, you will see the following in your Databricks “User Settings”.



Comment	Creation 🡑	Expiration
databricks-hive-demo	2021-11-24 14:29:47 CST	Never

AZURE DATA LAKE

Next, we will setup the Azure Data Lake.

Navigate to the storage account you provisioned and create two containers:

1) source
2) hive

Within the "hive" container create a directory called "database".

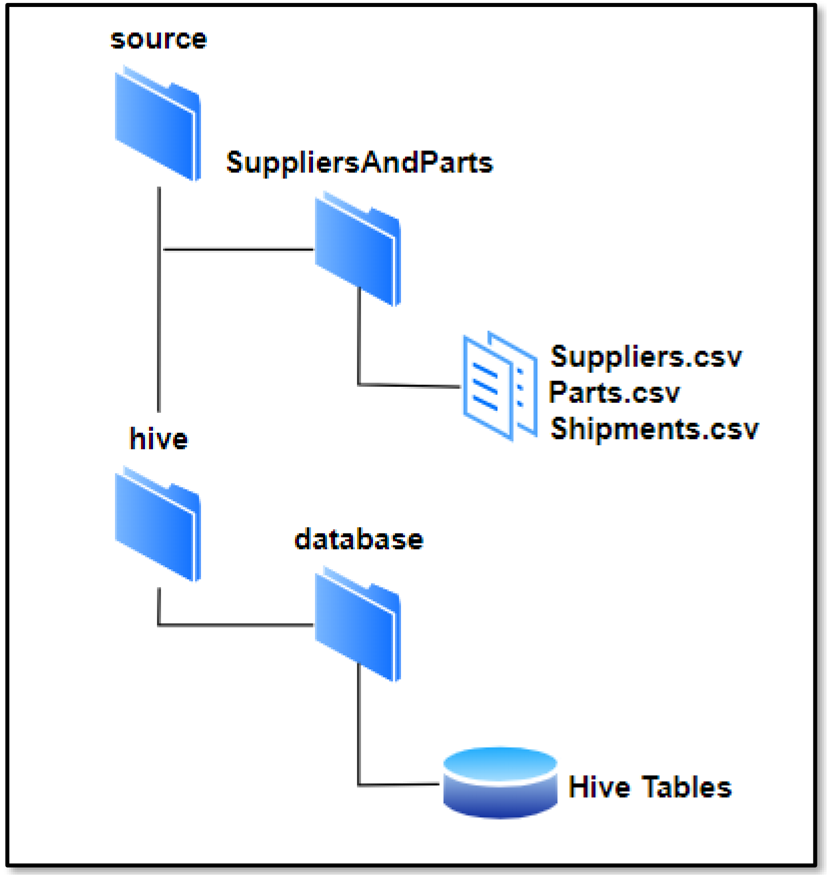
Within the "source" container, create a directory called "SuppliersAndParts".

The directory "source > SuppliersAndParts" is where we will store the csv files to import, and "hive > Database" will contain the Hive tables (which we will create later).

Upload the text files from the Git repository into the "source > SuppliersAndParts" container.

AZURE DATA LAKE

Your file structure in the storage account will look like the following:



Copy the files "Suppliers.csv", "Parts.csv", and "Shipments.csv" files into the "SuppliersAndParts" folder.

The "database" folder will contain the Hive tables, which we will setup later in this demo.

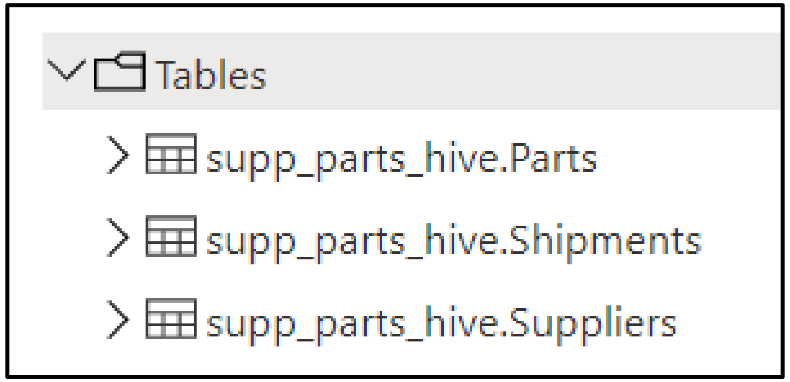
The Git repository for this demo is located here: https://Github.com/smpetersGithub/AdvancedSQLPuzzles/tree/main/Suppliers%20and%20Parts%20Hive%20Demo

SQL SERVER DATABASE

Next, we will create the schema and tables in the SQL Server database.

The script to create the schema and tables is provided in the Git directory.

We will create a schema named “supp_parts_hive” and the following tables:



Tables
- supp_parts_hive.Parts
- supp_parts_hive.Shipments
- supp_parts_hive.Suppliers

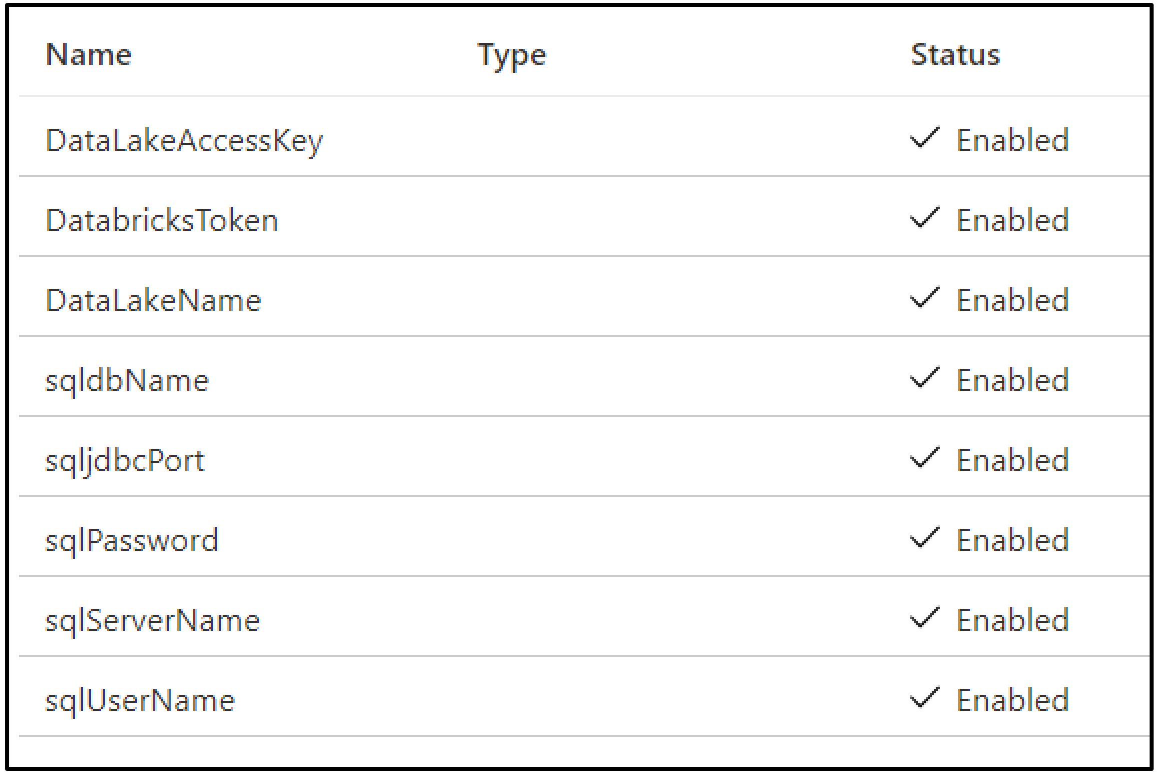
The Git repository for this demo is located here:
https://Github.com/smpetersGithub/AdvancedSQLPuzzles/tree/main/Suppliers%20and%20Parts%20Hive%20Demo

AZURE KEY VAULT

Next, we will create the secrets inside of the Azure Key Vault.

Here are the secrets you will need to setup in the Azure Key Vault.

Databricks will be able to access these values via the secret scope we created earlier.



Name	Type	Status
DataLakeAccessKey		✓ Enabled
DatabricksToken		✓ Enabled
DataLakeName		✓ Enabled
sqldbName		✓ Enabled
sqljdbcPort		✓ Enabled
sqlPassword		✓ Enabled
sqlServerName		✓ Enabled
sqlUserName		✓ Enabled

These secrets will be used within the Databricks notebooks, except for “DatabricksToken”, which will be used by the Azure Data Factory linked service.

AZURE KEY VAULT

Secret Name	Value	Description
sqljdbcPort	1433	This should always be 1433
sqldbName	sqldb-databricks-hive-demo	The name of the database
sqlPassword	MyPassword	The password of the database
sqlUserName	MyUserName	The admin user of the database
sqlServerName	sql-databricks-hive-demo.database.windows.net	The database server connection string
DataLakeAccessKey	Review the next few slides….	Copied from the Azure Storage Account properties
DataLakeName	dlsdatabrickshivedemo	The name of the storage account
DatabricksToken	Created in previous slides…	Used by Data Factory linked service

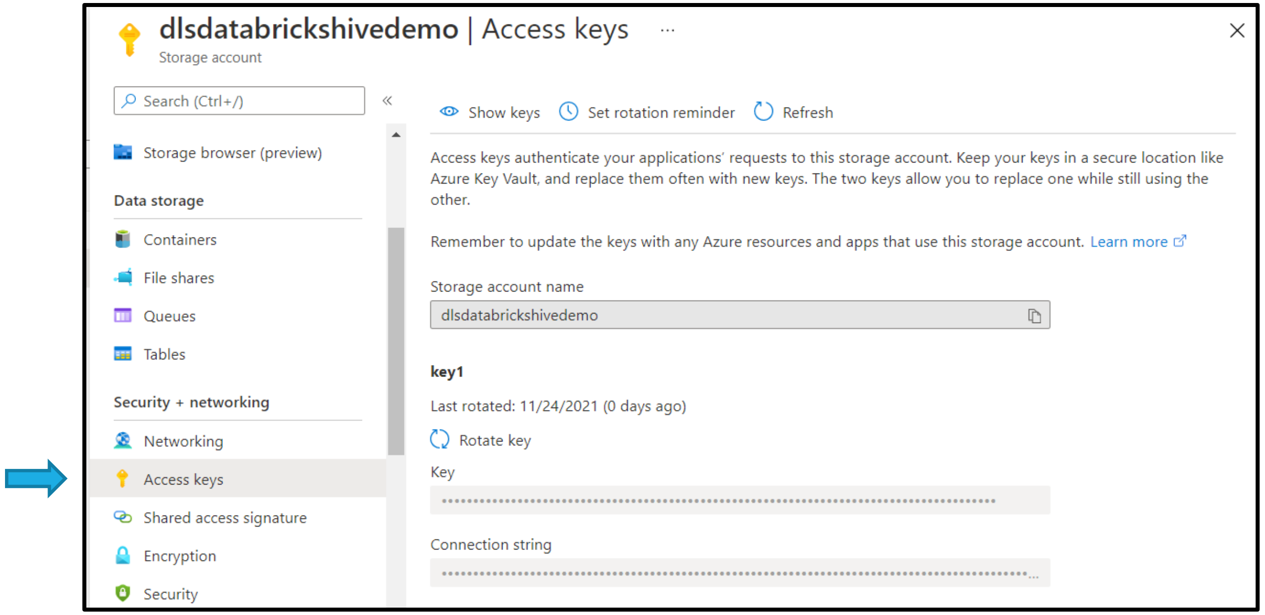
The two services we need Databricks to authenticate to are:

1) SQL Server Database
2) Azure Storage Account

The secret “DatabricksToken” will be used by the Data Factory linked service.
We will create the Azure Storage Account access key in the next slides.

AZURE KEY VAULT

The secret “DataLakeAccessKey” is created by navigating to your storage account and copying the access key.



Access keys authenticate your applications' requests to this storage account. Keep your keys in a secure location like Azure Key Vault, and replace them often with new keys. The two keys allow you to replace one while still using the other.

Remember to update the keys with any Azure resources and apps that use this storage account. Learn more

Storage account name: dlsdatabrickshivedemo

key1

Last rotated: 11/24/2021 (0 days ago)

Rotate key

Key: ••

Connection string: ••...

The documentation for Azure Storage Account access keys is located here: https://docs.microsoft.com/en-us/azure/storage/common/storage-account-keys-manage?tabs=azure-portal

RECAP

Let’s do a quick recap of everything we have accomplished so far:

- We have provisioned our Azure Resources
- We created a Databricks secret scope
- We created a Databricks token
- We setup our Azure Storage Account as a data lake with the needed directories and imported the csv files
- We created the schema and tables in our SQL Server Database
- We created the secret keys in our Azure Key Vault

RECAP

Next, we will perform the following:

- Import a Databricks workspace
- Create a Databricks cluster
- Run the database setup scripts
- Review the code in the workspace
- Test our Databricks workspace
- Create an Azure Data Factory pipeline
- Execute the Azure Data Factory pipeline

DATABRICKS

The following documentation gives a good overview of what we will be accomplishing for the remainder of these slides. Take a few moments to review the following documentation.

A Microsoft tutorial for incorporating Databricks into Data Factory is located here: https://docs.microsoft.com/en-us/azure/data-factory/transform-data-using-databricks-notebook

If you are unfamiliar with creating clusters, running notebooks, or navigating the workspace, I recommend working through a basic tutorial of Databricks before proceeding.

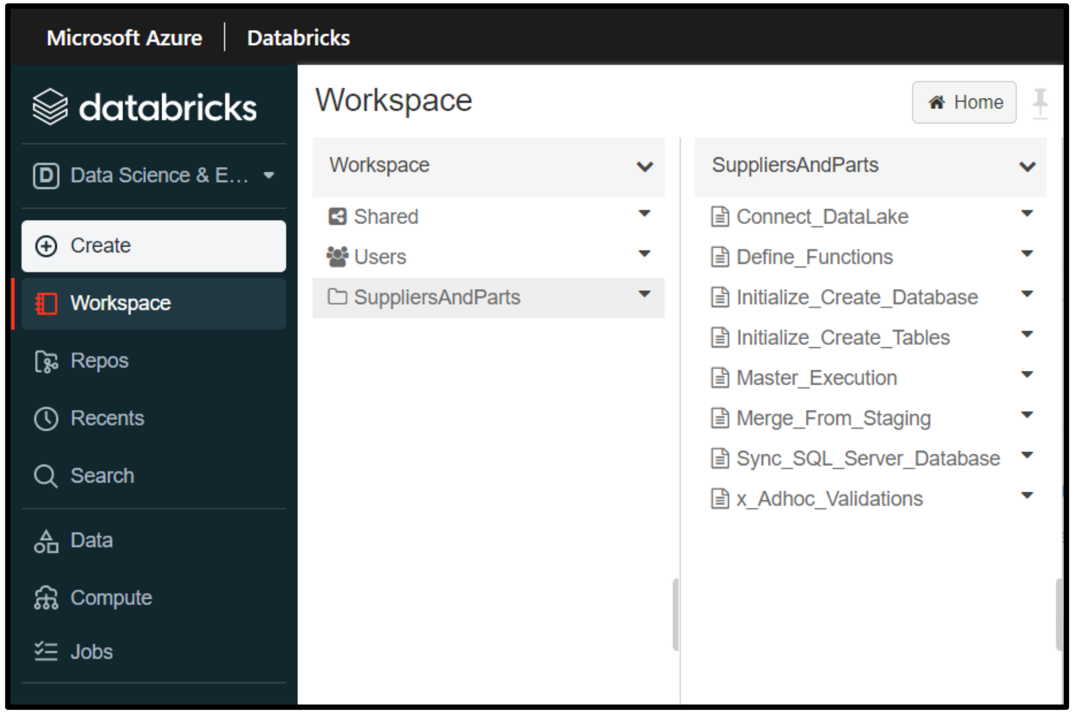
DATABRICKS

Begin by importing the workspace from the Git repository.

First, we will begin by importing the “SuppliersAndParts.dbc” workspace from the Git repository.

This workspace contains the notebooks needed to create our ETL.

Once you import the dbc file, you will see the following “SuppliersAndParts” folder in your workspace.



DATABRICKS

The workbook consists of the following notebooks:

Included Notebooks

- Connect_DataLake
- Create_Database
- Create_Tables
- Insert_Hive_Staging_Functions
- Insert_SQL_Server_Database
- Master_Execution
- Merge_Hive_Production
- Validation

DATABRICKS

Here is a brief description of each notebook sorted by their purpose.

We will cover each of these notebooks in more detail.

Notebook	Description
Create_Database	Initial script to create the Hive database.
Create_Tables	Initial script to create the Hive tables.
Connect_DataLake	Reads variables from the Azure Key Vault and connects to the Azure Data Lake. This notebook is used by all other notebooks.
Insert_Hive_Staging_Functions	Recreates the Hive staging tables and then inserts the data from the csv file into the staging table. This is the first notebook to execute in the pipeline.
Merge_Hive_Production	Creates the functions needed to import the csv files into the staging tables. This is the first notebook to run in the pipeline.
Insert_SQL_Server_Database	Truncates and then inserts the data from the Hive production tables into the SQL Server database tables. This is the third notebook to execute in the pipeline.
Master_Execution	The master notebook which calls all other notebooks.
Validation	General script used for auditing

DATABRICKS

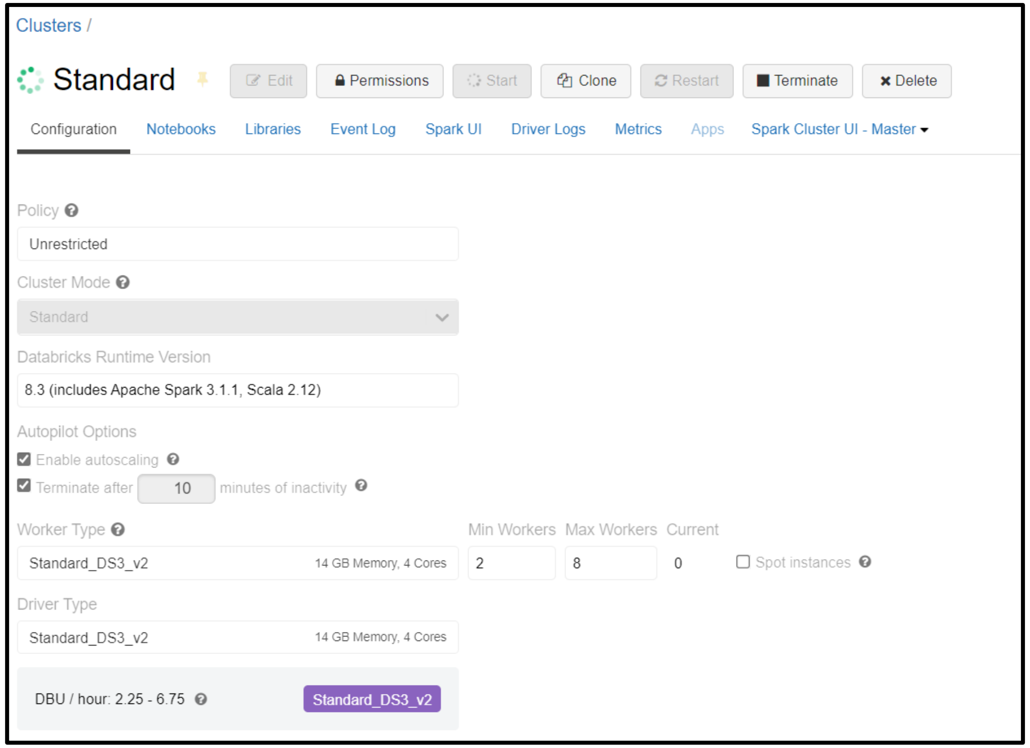
Before we move on, we need to create a Databricks cluster to run our notebooks.

The documentation for configuring Databricks clusters is located here: https://docs.microsoft.com/en-us/azure/databricks/clusters/configure

Remember to put a termination activity time on the cluster!

Here are the details of the cluster I have created for this demo.

A simple standard cluster will suffice. Feel free to experiment with different cluster sizes.



Standard

Policy: Unrestricted

Cluster Mode: Standard

Databricks Runtime Version: 8.3 (includes Apache Spark 3.1.1, Scala 2.12)

Autopilot Options

☑ Enable autoscaling

☑ Terminate after 10 minutes of inactivity

Worker Type: Standard_DS3_v2 — 14 GB Memory, 4 Cores

Min Workers: 2 Max Workers: 8 Current: 0 ☐ Spot instances

Driver Type: Standard_DS3_v2 — 14 GB Memory, 4 Cores

DBU / hour: 2.25 - 6.75 Standard_DS3_v2

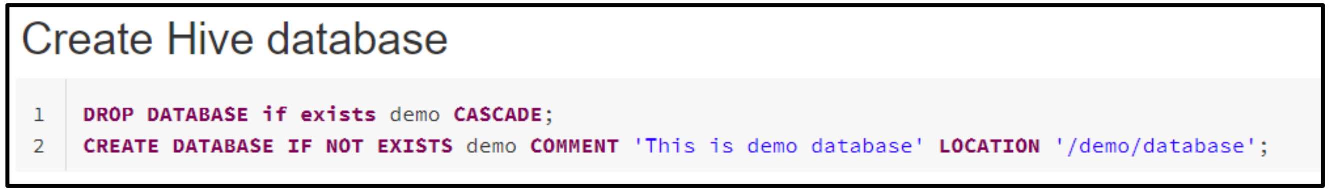
DATABRICKS

Let's first run the notebooks that create the Hive database and tables.

Run the following notebook first:

Create_Database

The following SQL statement creates a Hive database named "demo".



Create Hive database

```sql
DROP DATABASE if exists demo CASCADE;
CREATE DATABASE IF NOT EXISTS demo COMMENT 'This is demo database' LOCATION '/demo/database';
```

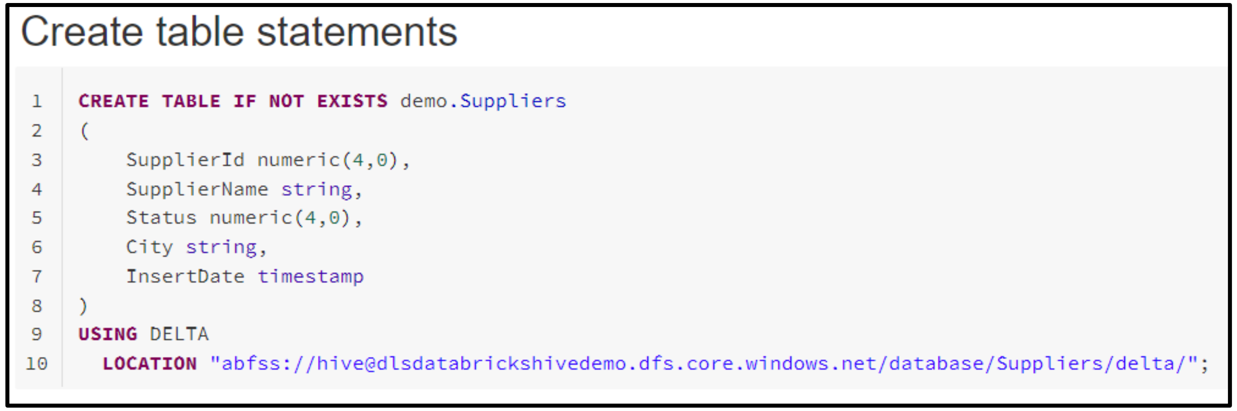
DATABRICKS

Second, create the tables for the Suppliers, Parts and Shipments data.

Run the following notebook next:

Create_Tables

This will create the three production tables for 1) Suppliers, 2) Parts and 3) Shipments.



Create table statements

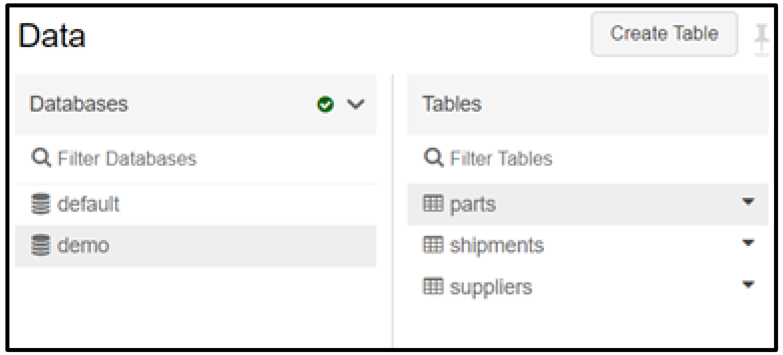
```
CREATE TABLE IF NOT EXISTS demo.Suppliers
(
    SupplierId numeric(4,0),
    SupplierName string,
    Status numeric(4,0),
    City string,
    InsertDate timestamp
)
USING DELTA
  LOCATION "abfss://hive@dlsdatabrickshivedemo.dfs.core.windows.net/database/Suppliers/delta/";
```

You will need to change the location only if you named the Azure Storage Account directory differently than this demo.

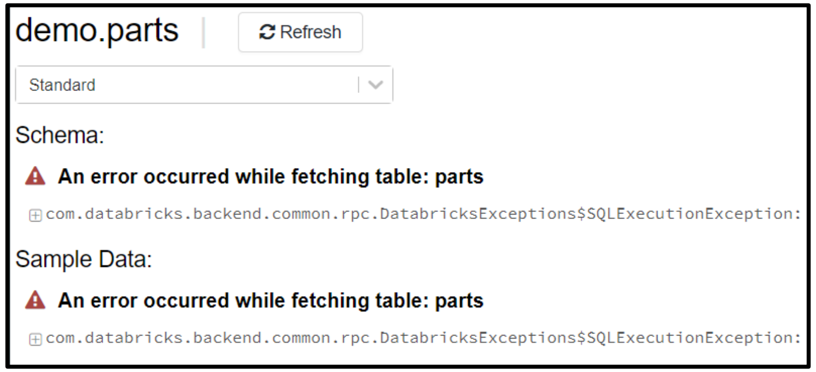
DATABRICKS

Second, create the tables for the Suppliers, Parts and Shipments data.

After creating the tables, you will see the production tables in the "Data" pane of the workspace. The staging tables will appear here after we run the pipeline for the first time.



You will get the following error when you select a table. This is perfectly normal.



Schema:

An error occurred while fetching table: parts

```
com.databricks.backend.common.rpc.DatabricksExceptions$SQLExecutionException:
```

Sample Data:

An error occurred while fetching table: parts

```
com.databricks.backend.common.rpc.DatabricksExceptions$SQLExecutionException:
```

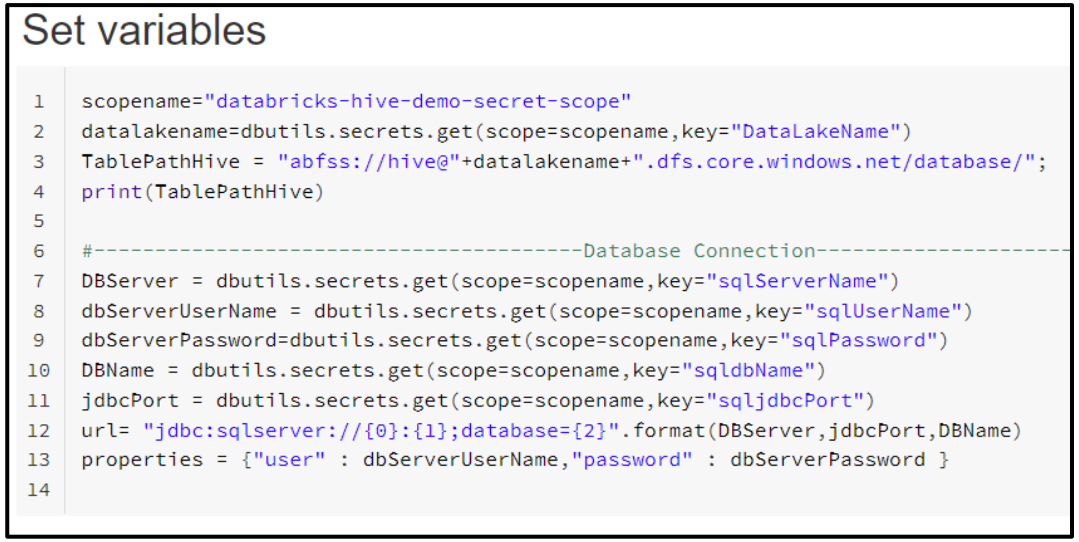
DATABRICKS

Next, we will test the connection to the Azure Key Vault and the Azure Storage Account.

Run the following notebook to test the connection to the Azure Key Vault and the Azure Storage Account services.

Connect_DataLake

The following code connects to the Azure Key Vault.



Set variables

```
scopename="databricks-hive-demo-secret-scope"
datalakename=dbutils.secrets.get(scope=scopename,key="DataLakeName")
TablePathHive = "abfss://hive@"+datalakename+".dfs.core.windows.net/database/";
print(TablePathHive)

#----------------------------------------Database Connection--------------------
DBServer = dbutils.secrets.get(scope=scopename,key="sqlServerName")
dbServerUserName = dbutils.secrets.get(scope=scopename,key="sqlUserName")
dbServerPassword=dbutils.secrets.get(scope=scopename,key="sqlPassword")
DBName = dbutils.secrets.get(scope=scopename,key="sqldbName")
jdbcPort = dbutils.secrets.get(scope=scopename,key="sqljdbcPort")
url= "jdbc:sqlserver://{0}:{1};database={2}".format(DBServer,jdbcPort,DBName)
properties = {"user" : dbServerUserName,"password" : dbServerPassword }

```

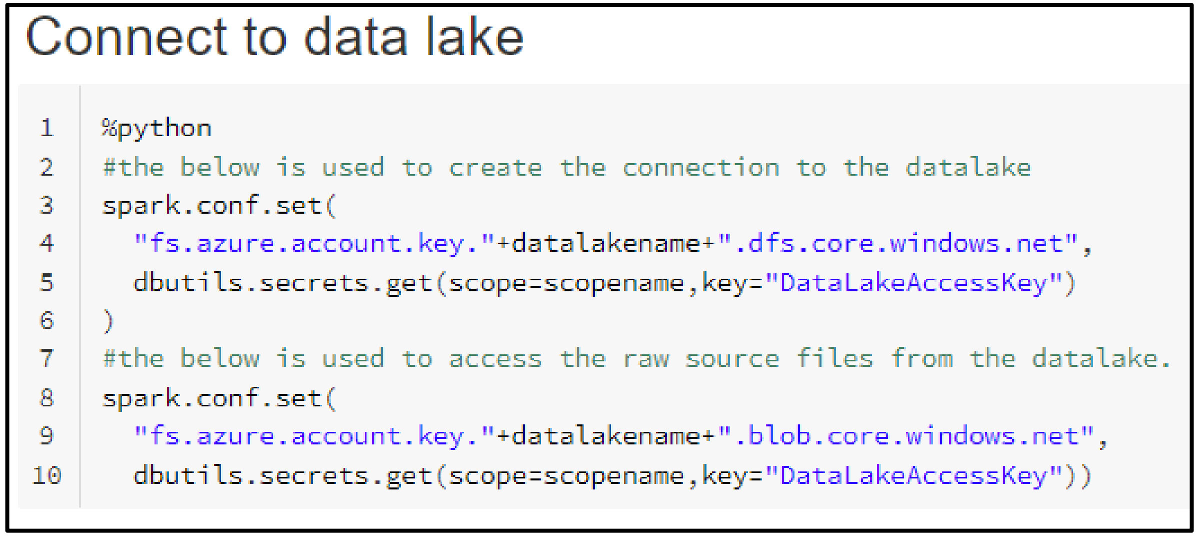
If the Azure Key Vault secret names are different than this demo, you will need to modify these values. Note the only variable hardcoded is the secret scope that we created earlier.

DATABRICKS

Next, we will test the connection to the Azure Key Vault and the Azure Storage Account.

Connect_DataLake

The following code creates the connection to the data lake.



Connect to data lake

```
%python
#the below is used to create the connection to the datalake
spark.conf.set(
  "fs.azure.account.key."+datalakename+".dfs.core.windows.net",
  dbutils.secrets.get(scope=scopename,key="DataLakeAccessKey")
)
#the below is used to access the raw source files from the datalake.
spark.conf.set(
  "fs.azure.account.key."+datalakename+".blob.core.windows.net",
  dbutils.secrets.get(scope=scopename,key="DataLakeAccessKey"))
```

It connects once with the “dfs” syntax, and then again with the “blob” syntax.

The documentation for Spark configuration settings is located here: https://docs.microsoft.com/en-us/azure/databricks/kb/data/get-and-set-spark-config

DATABRICKS

Next, let’s test if we can run the master notebook.

Master_Execution

The “Master_Execution” notebook will run once for each of the three csv files:

1) Suppliers.csv
2) Parts.csv
3) Shipments.csv

This notebook calls the following notebooks in order:

1) Insert_Hive_Staging_Functions
2) Merge_Hive_Production
3) Insert_SQL_Server_Database

DATABRICKS

Next, let's test if we can run the master notebook.

Master_Execution

This notebook performs the following steps:

Step 1

It first reads the functions from the "Insert_Hive_Staging_Functions", which also reads the "Connect_DataLake" notebook.

Step 2

It then determines the variables "FileName" and the "ProcessName".

Step 3

Next, it runs the function "ProcessHiveStagingData", which is located in the the "Insert_Hive_Staging_Functions" notebook.

Step 4

It then runs the "Merge_Hive_Production" notebook, and then the "Insert_SQL_Server_Database".

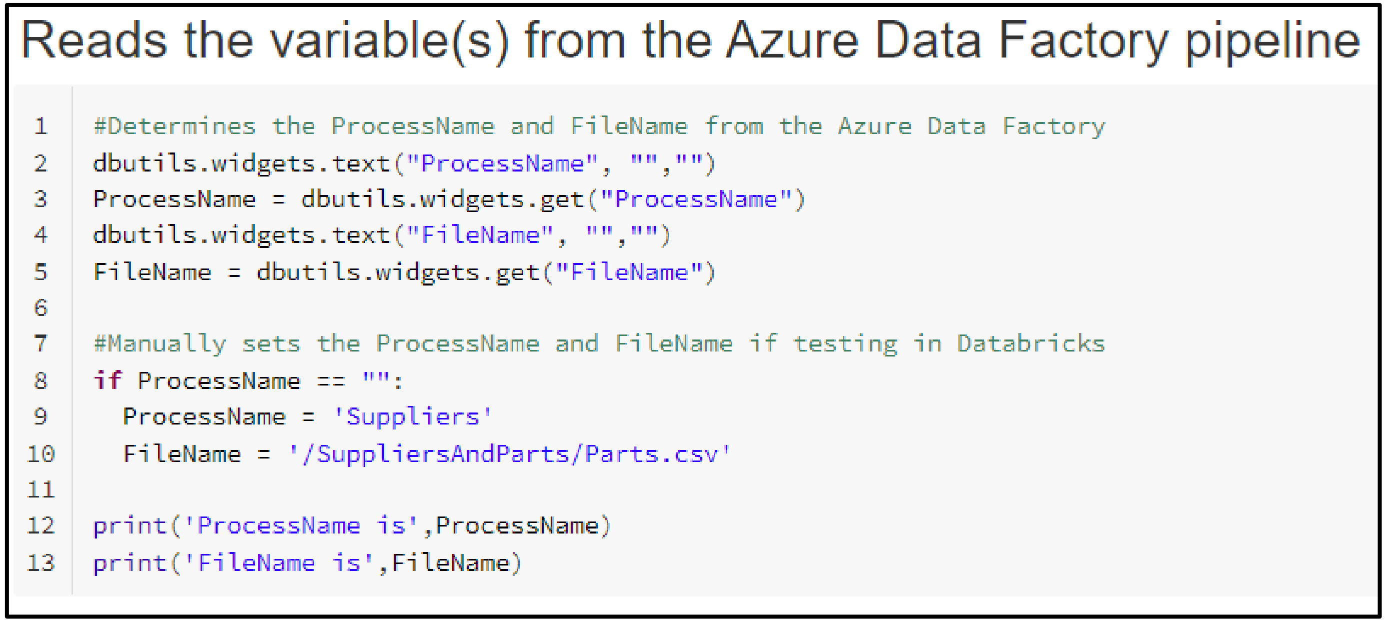
DATABRICKS

Next, let’s test if we can run the master notebook.

DBUTILS.WIDGETS

The “dbutils.widgets” code reads the variable “ProcessName” and “FileName” from the Azure Data Factory.

To test the code in Databricks, manually set the value of “ProcessName” and “FileName” where applicable.



Reads the variable(s) from the Azure Data Factory pipeline

```
#Determines the ProcessName and FileName from the Azure Data Factory
dbutils.widgets.text("ProcessName", "","")
ProcessName = dbutils.widgets.get("ProcessName")
dbutils.widgets.text("FileName", "","")
FileName = dbutils.widgets.get("FileName")

#Manually sets the ProcessName and FileName if testing in Databricks
if ProcessName == "":
  ProcessName = 'Suppliers'
  FileName = '/SuppliersAndParts/Parts.csv'

print('ProcessName is',ProcessName)
print('FileName is',FileName)
```

DATABRICKS

Insert_Hive_Staging_Functions

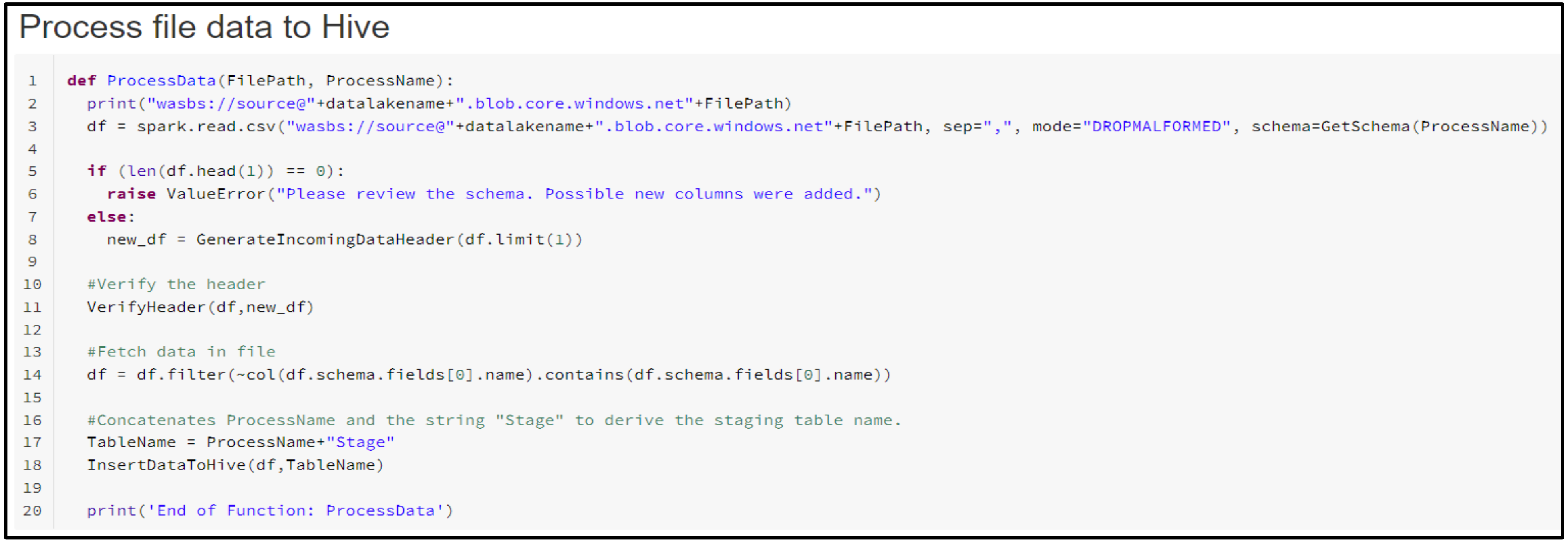
This notebook defines the following functions needed to import and verify the csv file and insert it into the staging table.

- GetSchema (Define the schema details for the file)
- GenerateIncomingDataHeader (Create a data frame of the file header)
- VerifyHeader (Validate incoming file header to schema definition)
- ProcessHiveStagingData (Reads the csv file and executes the InsertDataToHive function)
- InsertDataToHive (Inserts the data into the Hive tables)

DATABRICKS

Insert_Hive_Staging

In the “ProcessData” function, the variable “TableName” is the concatenation of the variable “ProcessName” and the string “Stage”.



Process file data to Hive

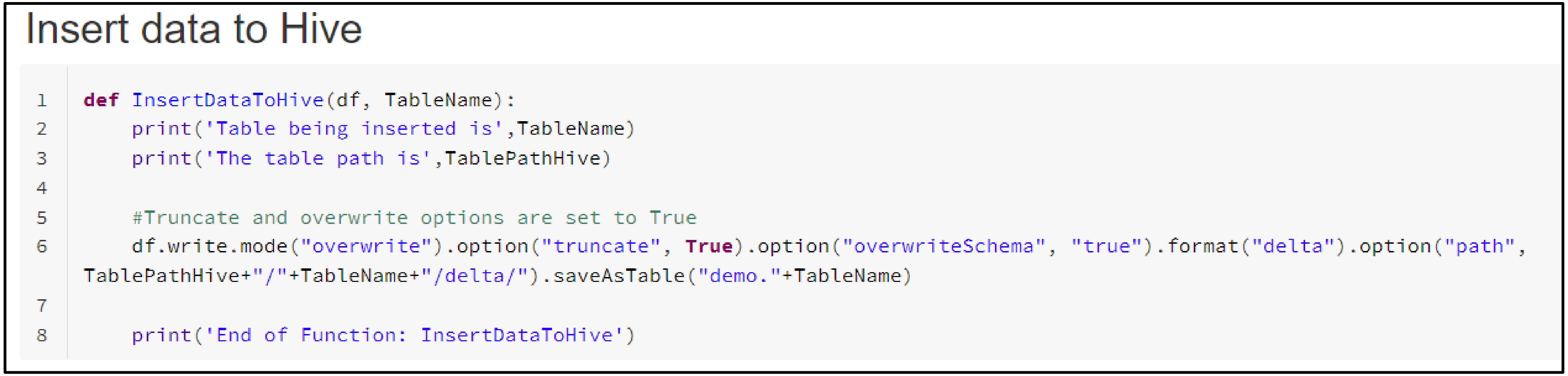
```python
def ProcessData(FilePath, ProcessName):
  print("wasbs://source@"+datalakename+".blob.core.windows.net"+FilePath)
  df = spark.read.csv("wasbs://source@"+datalakename+".blob.core.windows.net"+FilePath, sep=",", mode="DROPMALFORMED", schema=GetSchema(ProcessName))

  if (len(df.head(1)) == 0):
    raise ValueError("Please review the schema. Possible new columns were added.")
  else:
    new_df = GenerateIncomingDataHeader(df.limit(1))

  #Verify the header
  VerifyHeader(df,new_df)

  #Fetch data in file
  df = df.filter(~col(df.schema.fields[0].name).contains(df.schema.fields[0].name))

  #Concatenates ProcessName and the string "Stage" to derive the staging table name.
  TableName = ProcessName+"Stage"
  InsertDataToHive(df,TableName)

  print('End of Function: ProcessData')
```

DATABRICKS

Insert_Hive_Staging

The insert into the staging table has the options of “overwrite” and “truncate” and “overwriteSchema”.

On each execution the staging table is recreated and then inserted into.



Insert data to Hive

```
def InsertDataToHive(df, TableName):
    print('Table being inserted is',TableName)
    print('The table path is',TablePathHive)

    #Truncate and overwrite options are set to True
    df.write.mode("overwrite").option("truncate", True).option("overwriteSchema", "true").format("delta").option("path", TablePathHive+"/"+TableName+"/delta/").saveAsTable("demo."+TableName)

    print('End of Function: InsertDataToHive')
```

DATABRICKS

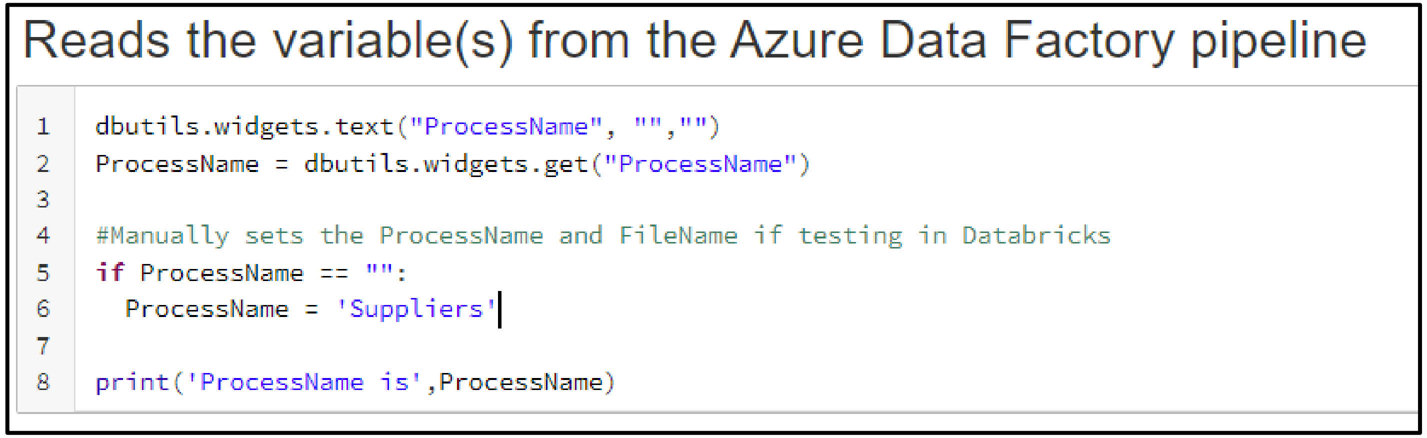
Now we will review some of the specifics in the code.

Merge_Hive_Production

This notebook merges the data from the staging to the production tables.

This notebook also uses the “dbutils.widgets”, which we have discussed in previous slides.

To run this notebook in Databricks for the “Suppliers.csv” file, manually set the variable “ProcessName” to “Suppliers”.



Reads the variable(s) from the Azure Data Factory pipeline

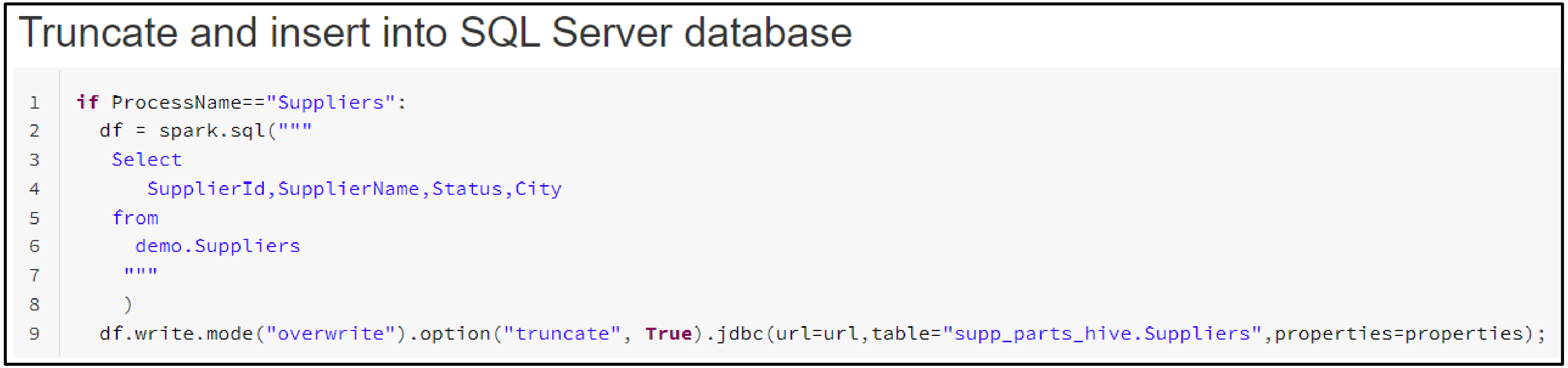
```
dbutils.widgets.text("ProcessName", "","")
ProcessName = dbutils.widgets.get("ProcessName")

#Manually sets the ProcessName and FileName if testing in Databricks
if ProcessName == "":
  ProcessName = 'Suppliers'

print('ProcessName is',ProcessName)
```

DATABRICKS

Insert_SQL_Server_Database

The insert into to the SQL Server database has the same overwrite and truncate options as the inserts into the Hive staging tables.

If you receive connection errors to the database, check that the SQL Server has the option set to connect to other Azure services.



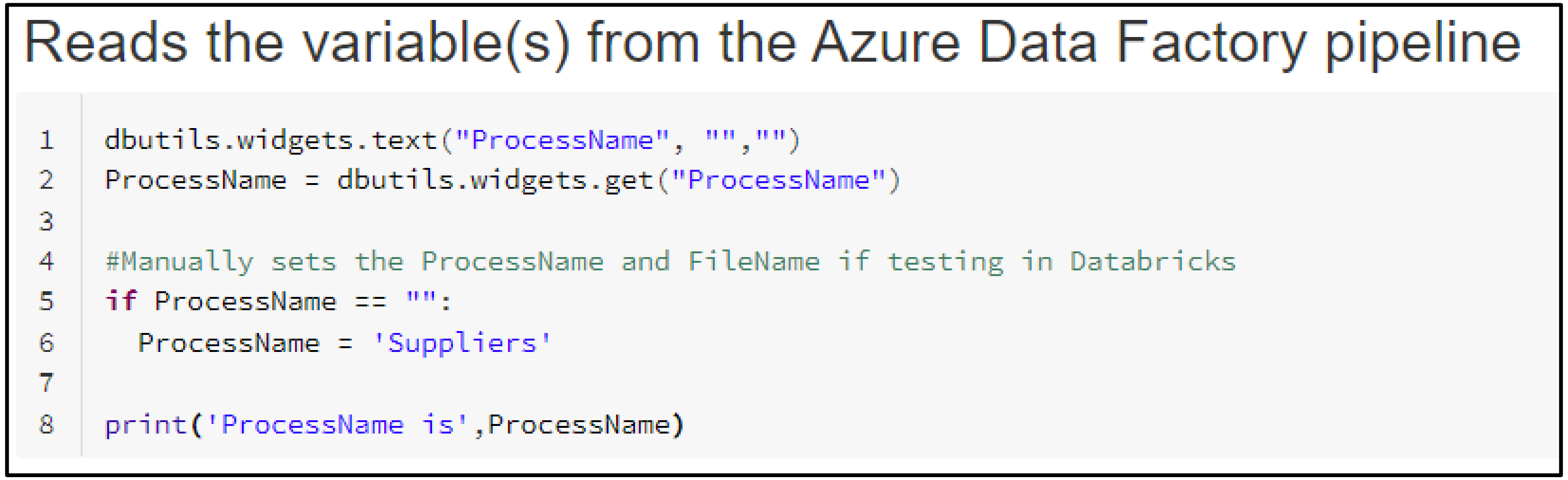
Truncate and insert into SQL Server database

```
if ProcessName=="Suppliers":
  df = spark.sql("""
   Select
      SupplierId,SupplierName,Status,City
   from
     demo.Suppliers
    """
    )
  df.write.mode("overwrite").option("truncate", True).jdbc(url=url,table="supp_parts_hive.Suppliers",properties=properties);
```

DATABRICKS

Insert_SQL_Server_Database

This notebook also uses the “dbutils.widgets”.



Reads the variable(s) from the Azure Data Factory pipeline

```
dbutils.widgets.text("ProcessName", "","")
ProcessName = dbutils.widgets.get("ProcessName")

#Manually sets the ProcessName and FileName if testing in Databricks
if ProcessName == "":
  ProcessName = 'Suppliers'

print('ProcessName is',ProcessName)
```

DATABRICKS

Now we will review some of the specifics in the code.

Validations

This notebook is for creating and saving any ad-hoc statements.

After you have ran an ETL, I recommend using this notebook to explore the data.

Remember you may have only tested on your Suppliers data. The Parts and Shipments tables may be blank if you have not tested these processes.

DATA FACTORY

Next, we will setup the Azure Data Factory.

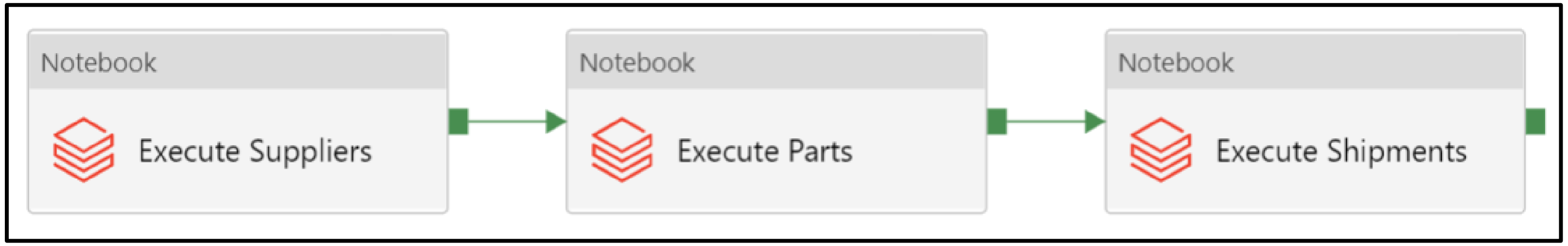
We will need to create two Azure Linked Services.

One to link to our 1) Azure Key Vault, and the second to link to the 2) Databricks Workspace.

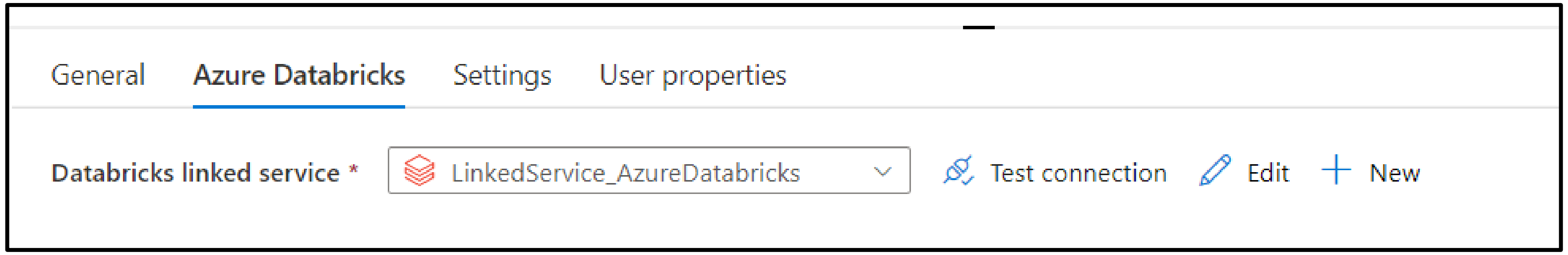
The Databricks Linked Service will utilize the Azure Key Vault Linked Service and access the Database token we setup earlier in the demo.

DATA FACTORY

Before we create the linked services, create a pipeline and add three Databricks notebooks activities. Chain them together to avoid getting "unexpected failure while waiting for the cluster" errors.



Each of these activities will utilize the Databricks linked service we will create in the next slides.

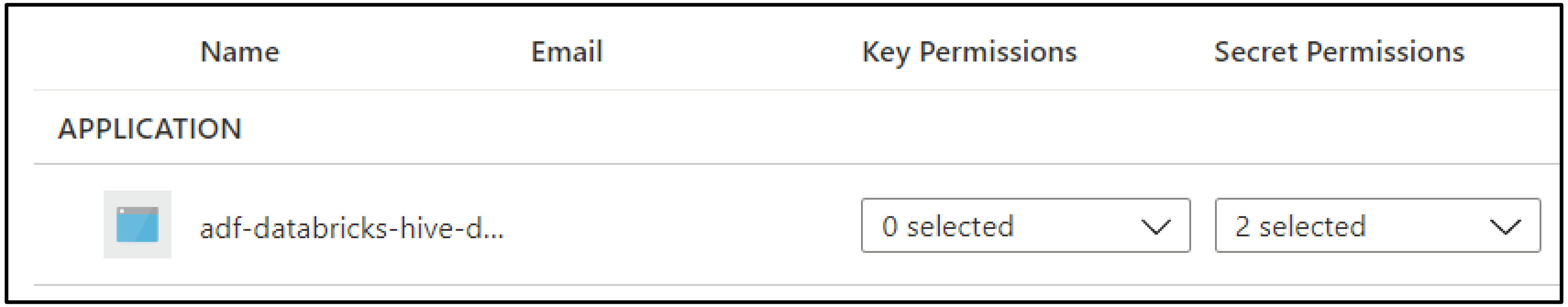


Databricks linked service * LinkedService_AzureDatabricks

DATA FACTORY

First, the Data Factory will need access to the Azure Key Vault. Navigate to your Azure Key Vault and add an access policy that allows the Data Factory secret permissions.

The Azure Key Vault you provisioned will now have an access policy for your Databricks and the Data Factory.



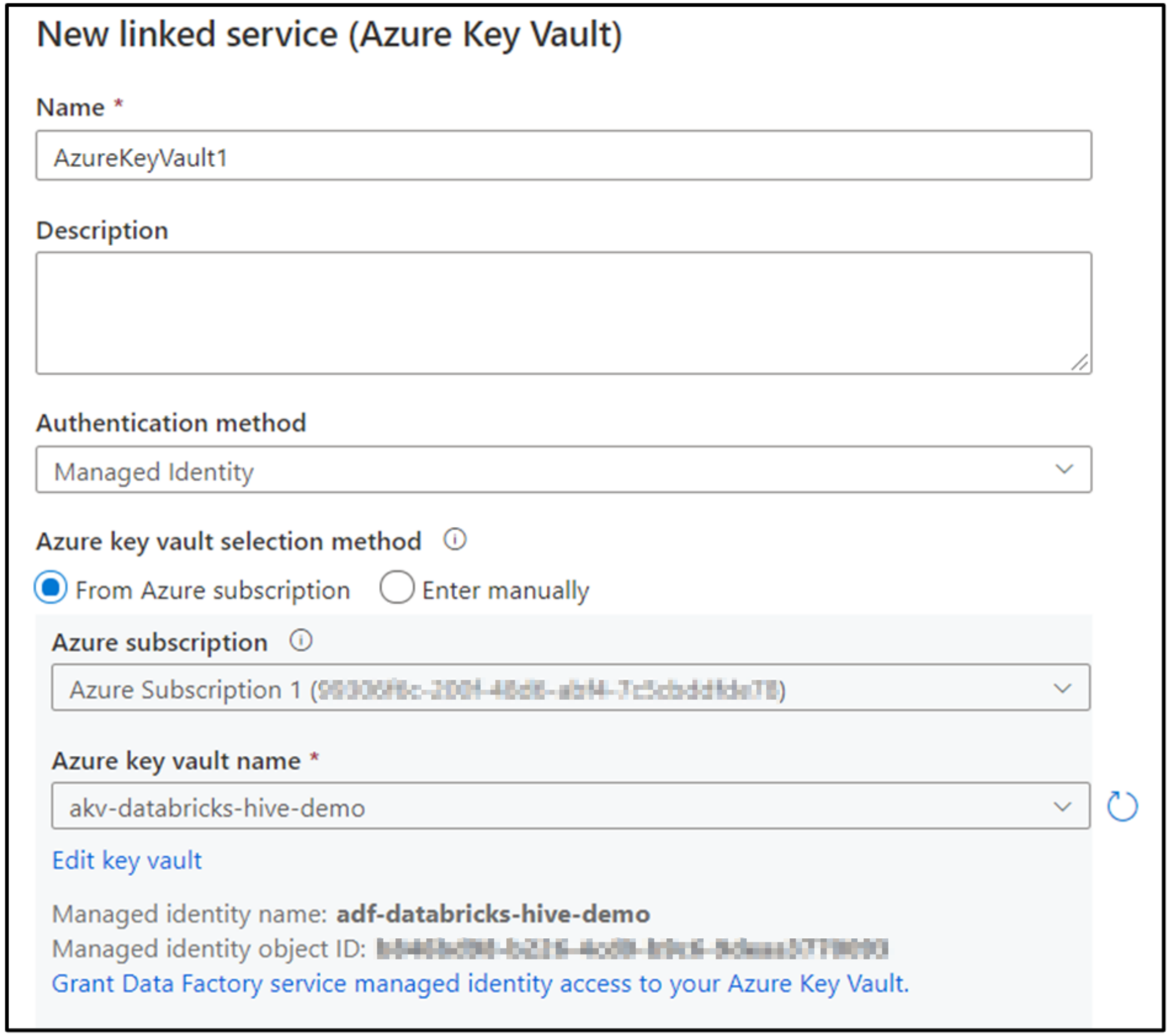
Name	Email	Key Permissions	Secret Permissions
APPLICATION			
adf-databricks-hive-d...		0 selected	2 selected

DATA FACTORY

AKV Linked Service

To link to the Azure Key Vault, create the AKV linked service using your Azure subscription.

You can also enter the AKV manually by providing the URL of the linked service, which is located in the properties page of the AKV.



New linked service (Azure Key Vault)

Name *

AzureKeyVault1

Description

Authentication method

Managed Identity

Azure key vault selection method ⓘ

● From Azure subscription ○ Enter manually

Azure subscription ⓘ

Azure Subscription 1 ([redacted])

Azure key vault name *

akv-databricks-hive-demo

Edit key vault

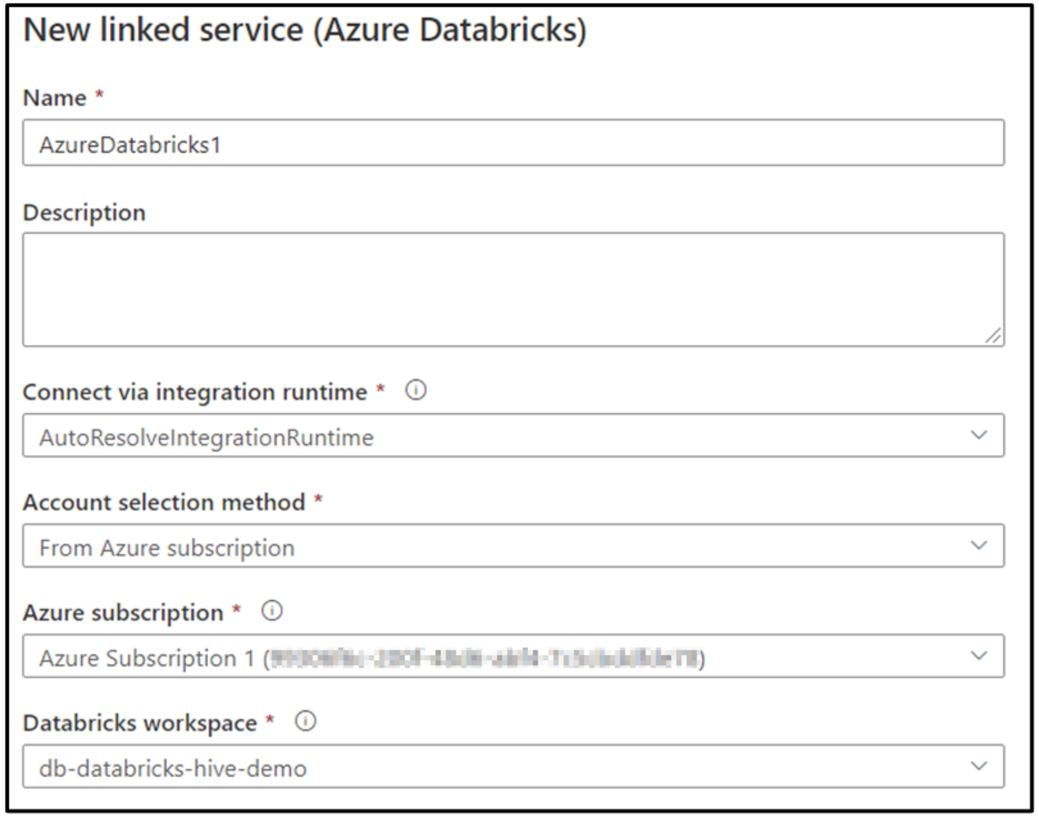
Managed identity name: **adf-databricks-hive-demo**
Managed identity object ID: [redacted]
Grant Data Factory service managed identity access to your Azure Key Vault.

DATA FACTORY

Databricks Linked Service

To link to the Databricks workspace, fill in the following dropdowns with your subscription and workspace information.

You can also enter the information in manually by selecting “Enter Manually” from the dropdown box.



New linked service (Azure Databricks)

Name *

AzureDatabricks1

Description

Connect via integration runtime *

AutoResolveIntegrationRuntime

Account selection method *

From Azure subscription

Azure subscription *

Azure Subscription 1 ([redacted])

Databricks workspace *

db-databricks-hive-demo

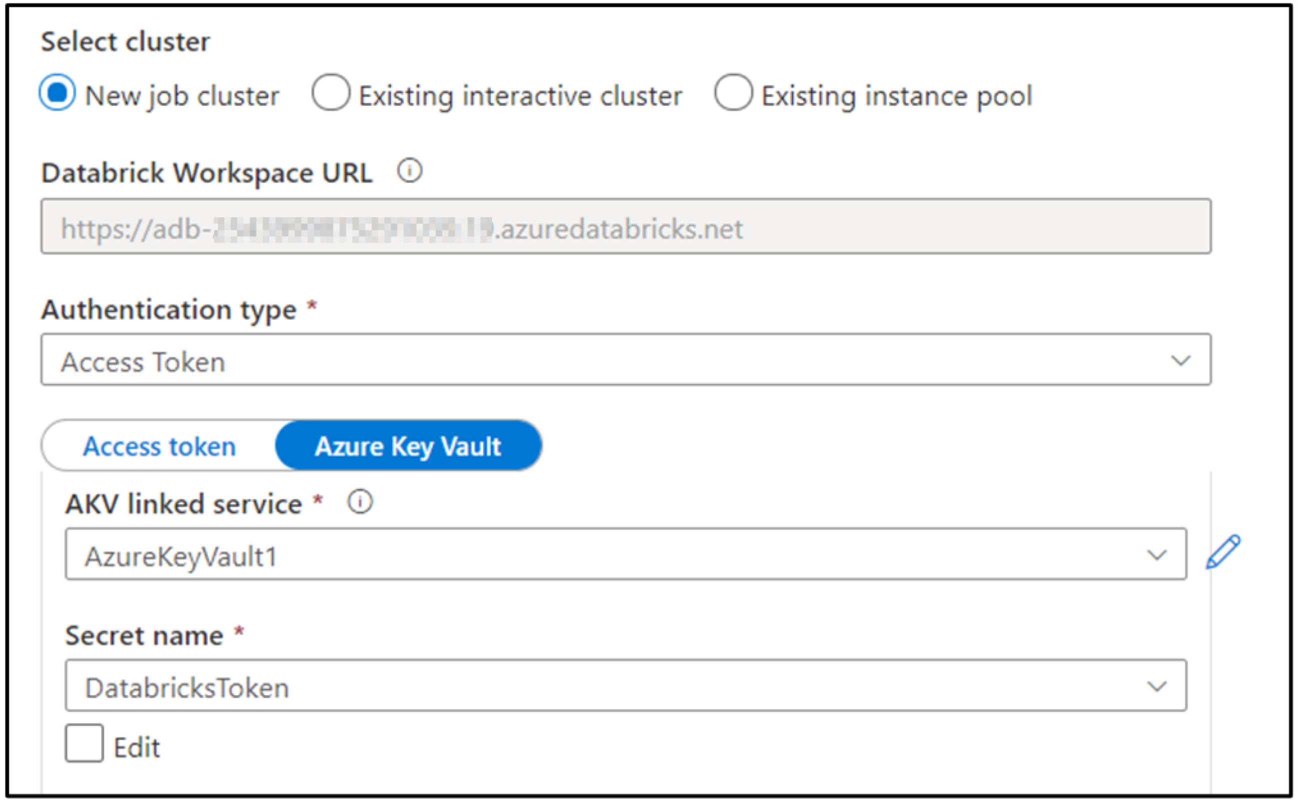
DATA FACTORY

Databricks Linked Service

Next, select “Access Token” as the authentication type.

Link to the AKV by using the linked service we created in a prior slide.

Then select the secret name you used to store the Databricks token.



Select cluster

● New job cluster ○ Existing interactive cluster ○ Existing instance pool

Databrick Workspace URL ⓘ

https://adb-[redacted].azuredatabricks.net

Authentication type *

Access Token

AKV linked service * ⓘ

AzureKeyVault1

Secret name *

DatabricksToken

□ Edit

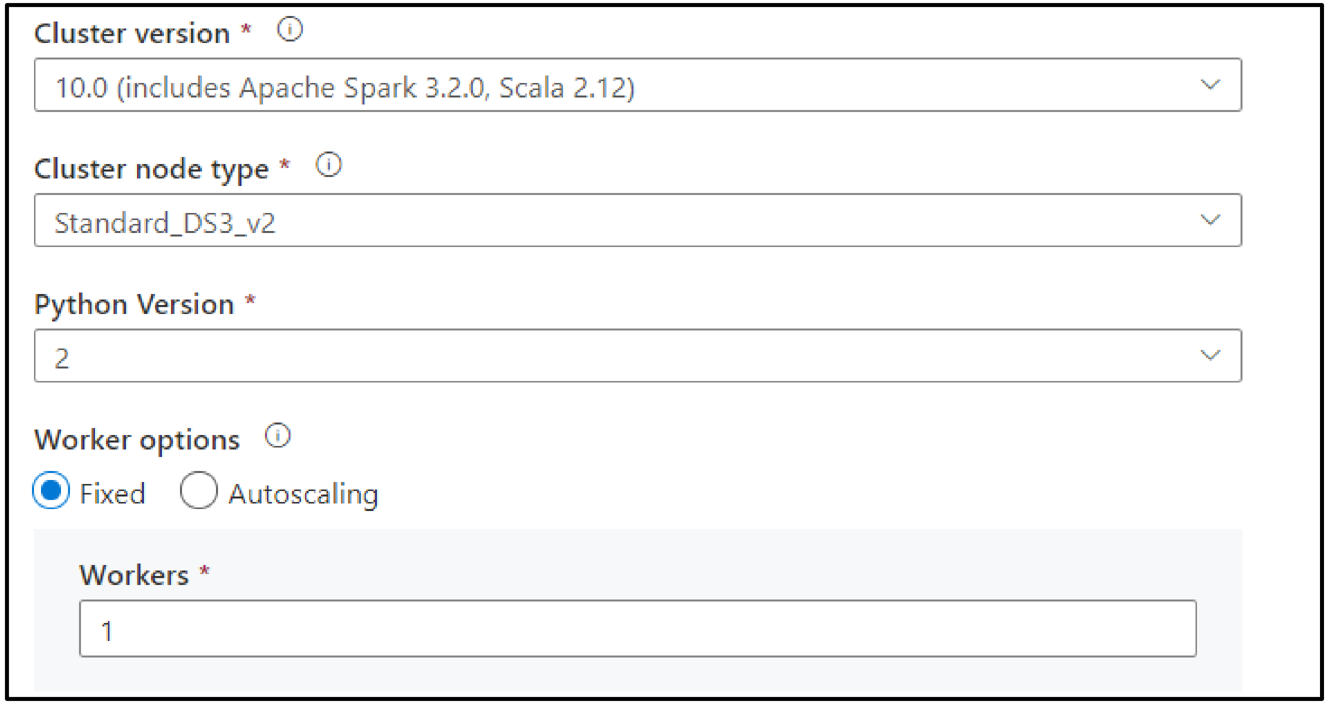
DATA FACTORY

Databricks Linked Service

Finish the linked service by filling in the rest of the fields.

Here you can see the options that I chose for cluster version, node type, etc.….

Selecting a new job cluster is the best option for cost savings, but it does add a few minutes of processing time to the pipeline to allow for the cluster to start up.



Cluster version *

10.0 (includes Apache Spark 3.2.0, Scala 2.12)

Cluster node type *

Standard_DS3_v2

Python Version *

2

Worker options

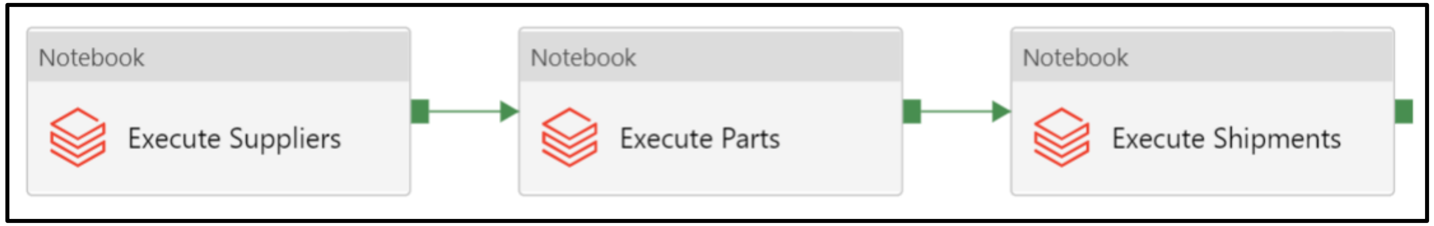
● Fixed ○ Autoscaling

Workers *

1

DATA FACTORY

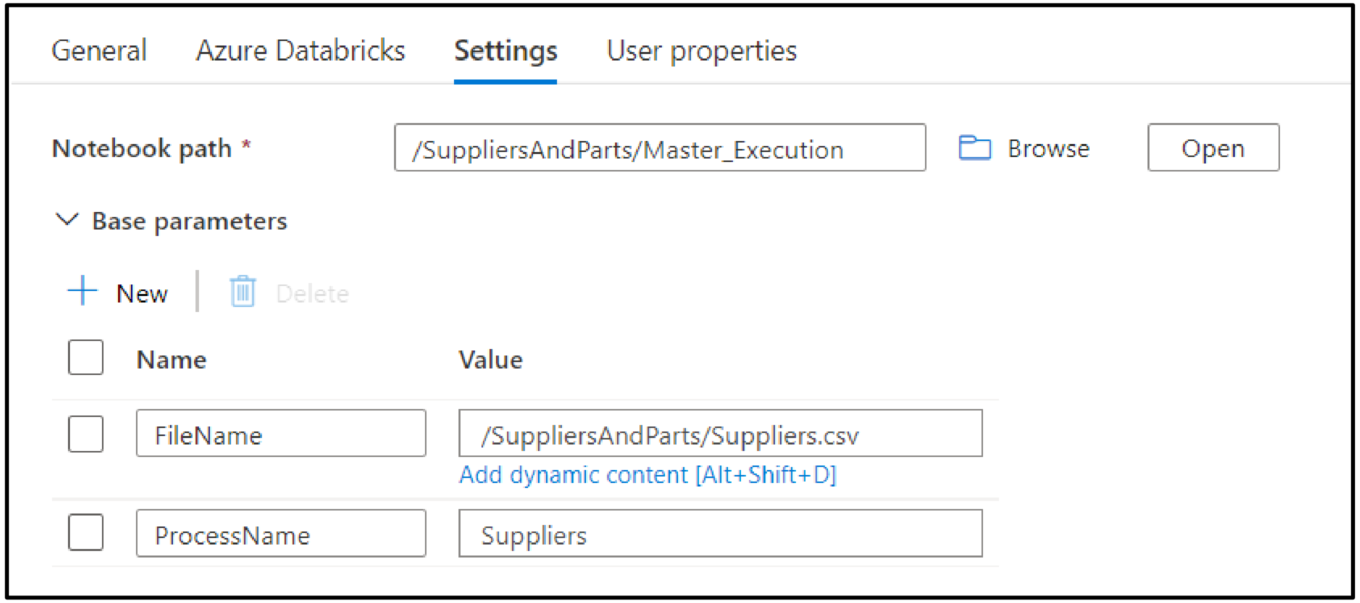
Now let's move back to the following pipeline activities and update the "Settings" tab for each activity.



Select the notebook path and create two parameters 1) "FileName" and 2) "ProcessName".

Fill in the appropriate information as shown in the screenshot.

You will need to do this for the 1) Suppliers, 2) Parts and 3) Shipments activities.



Notebook path * /SuppliersAndParts/Master_Execution

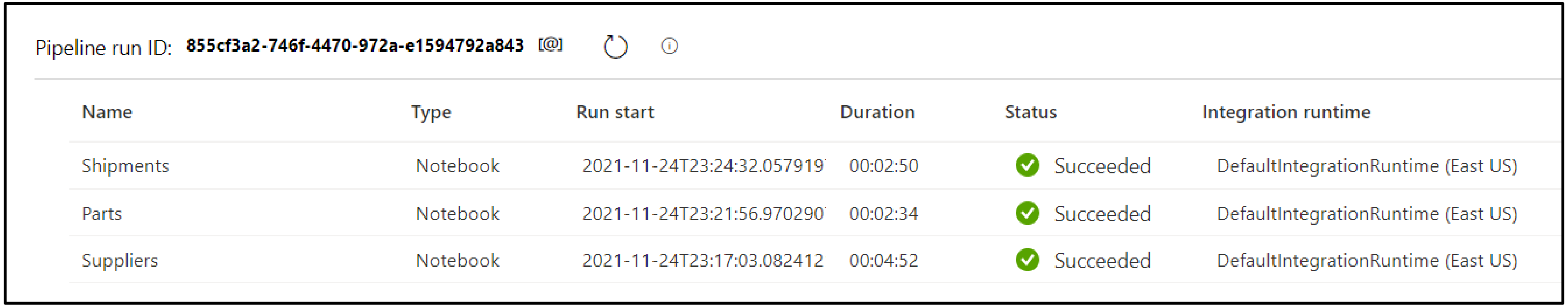
	Name	Value
□	FileName	/SuppliersAndParts/Suppliers.csv
□	ProcessName	Suppliers

DATAFACTORY

Execute the pipeline by selecting “Debug”.

Ensure there is no running cluster in your Databricks workspace before executing, else you will get “unexpected failure while waiting for the cluster” error.

The total execution time for my pipeline is around 11 minutes.



Pipeline run ID: **855cf3a2-746f-4470-972a-e1594792a843**

Name	Type	Run start	Duration	Status	Integration runtime
Shipments	Notebook	2021-11-24T23:24:32.057919	00:02:50	✓ Succeeded	DefaultIntegrationRuntime (East US)
Parts	Notebook	2021-11-24T23:21:56.970290	00:02:34	✓ Succeeded	DefaultIntegrationRuntime (East US)
Suppliers	Notebook	2021-11-24T23:17:03.082412	00:04:52	✓ Succeeded	DefaultIntegrationRuntime (East US)

TESTING

To test your ETL, review the data in your Hive tables with the statements in the “Validations” notebook. Also, review the data in the SQL Server tables as well.

Then alter the CSV files by adding new records and modifing the current records. Run the pipeline and verify the output.

To best learn the specifics of the Databricks notebooks, I recommend changing names of your AKV, storage account, secret names, etc… determine where the notebooks break and fix accordingly.

CONGRATULATIONS

You now have a template for using Databricks as a transformation service!

Check out my Git repository and SQL blog for all sort of puzzles, tips and tricks.

The Git repository for this demo is located here:
https://Github.com/smpetersGithub/AdvancedSQLPuzzles/tree/main/Suppliers%20and%20Parts%20Hive%20Demo