AZURE DATABRICKS PYSPARK 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 GitHub repository for this demo is located here:
https://github.com/smpetersgithub/AdvancedSQLPuzzles

Included in the repository are the following:

- A PDF version of this presentation
- Databricks workspace
- CSV files to import
- DDL scripts

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

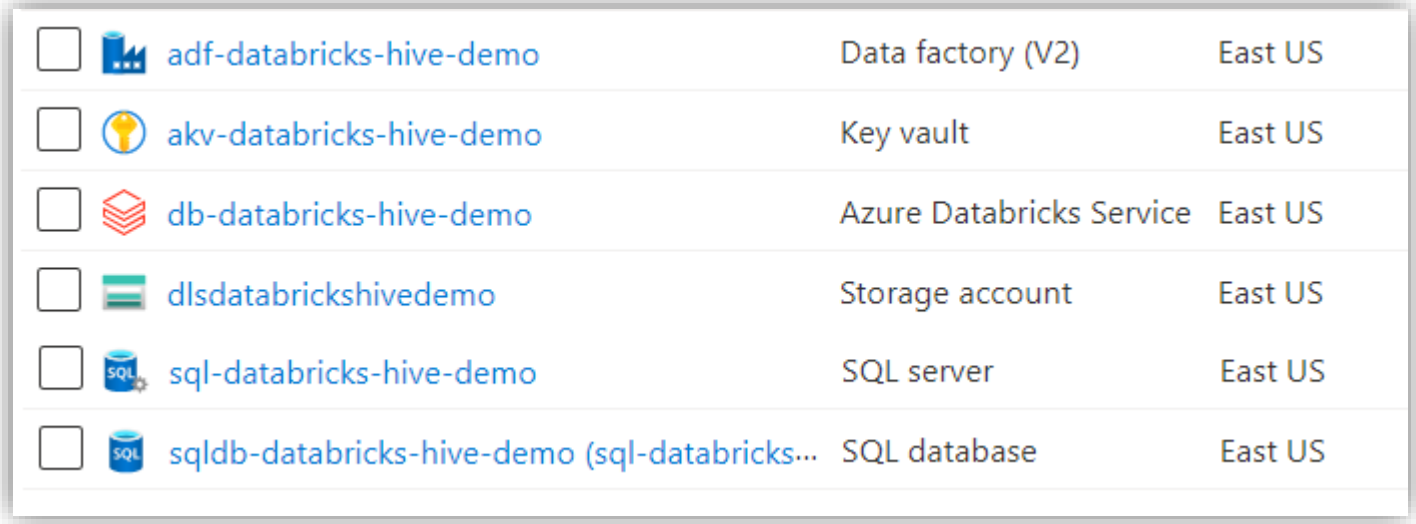
AZURE RESOURCES

Few reminders when provisioning the 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

The documentation for Azure naming rules is located here: https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/resource-name-rules

AZURE RESOURCES

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

Name	Type
db-databricks-hive-demo	Databricks
dlsdatabrickshivedemo	Azure Storage Account
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

Certain resources need to be globally unique, which we will cover in the next slide….

AZURE RESOURCES

Naming rules and restrictions:

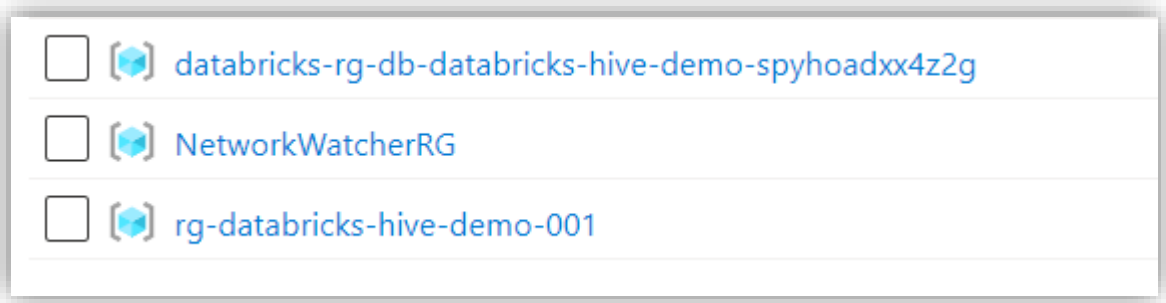
Resource	Scope	Length	Valid Characters
Databricks	Resource Group	3-64	Alphanumeric, underscores, and hyphens.
Storage Account	Global	3-24	Lowercase letters and numbers.
Key Vault	Global	3-24	Alphanumeric and hyphens. Starts with letter. End with letter or digit. Can't contain consecutive hyphens.
SQL Database	Server	1-128	Can't use: <>*%&:\/? or control characters. Can't end with period or space.
SQL Server	Global	1-63	Lowercase letters, numbers, and hyphens. Can't start or end with hyphen.
Azure Data Factory	Global	3-63	Alphanumeric and hyphens. Start and end with alphanumeric.

The documentation for Azure naming rules is located here:
https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/resource-name-rules

AZURE RESOURCES

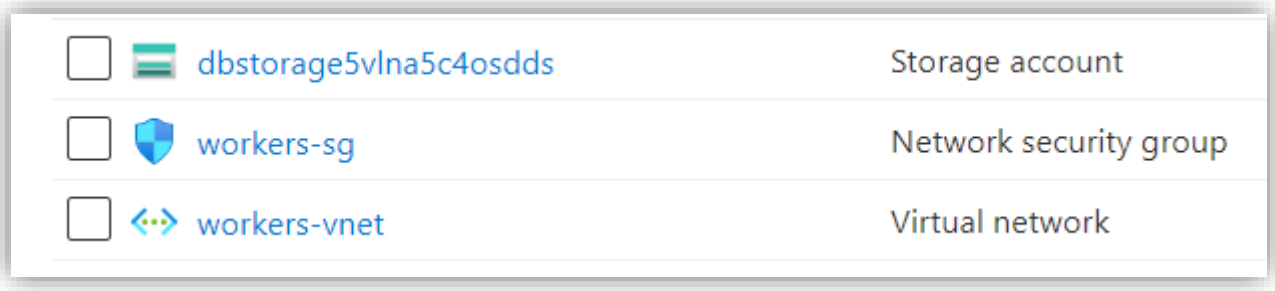
When you provision the resources, additional resource groups and resources are created.

You will see an additional resource group for Databricks, and another for the Network Watcher, as shown below.

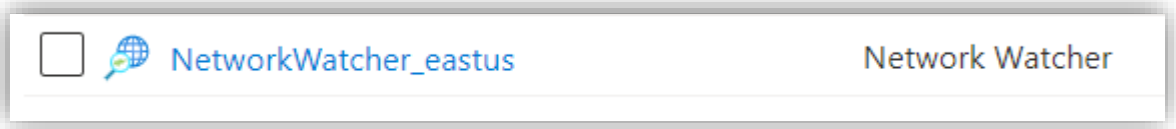


AZURE RESOURCES

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



The NetworkWatcherRG resource group has the following resources:



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.

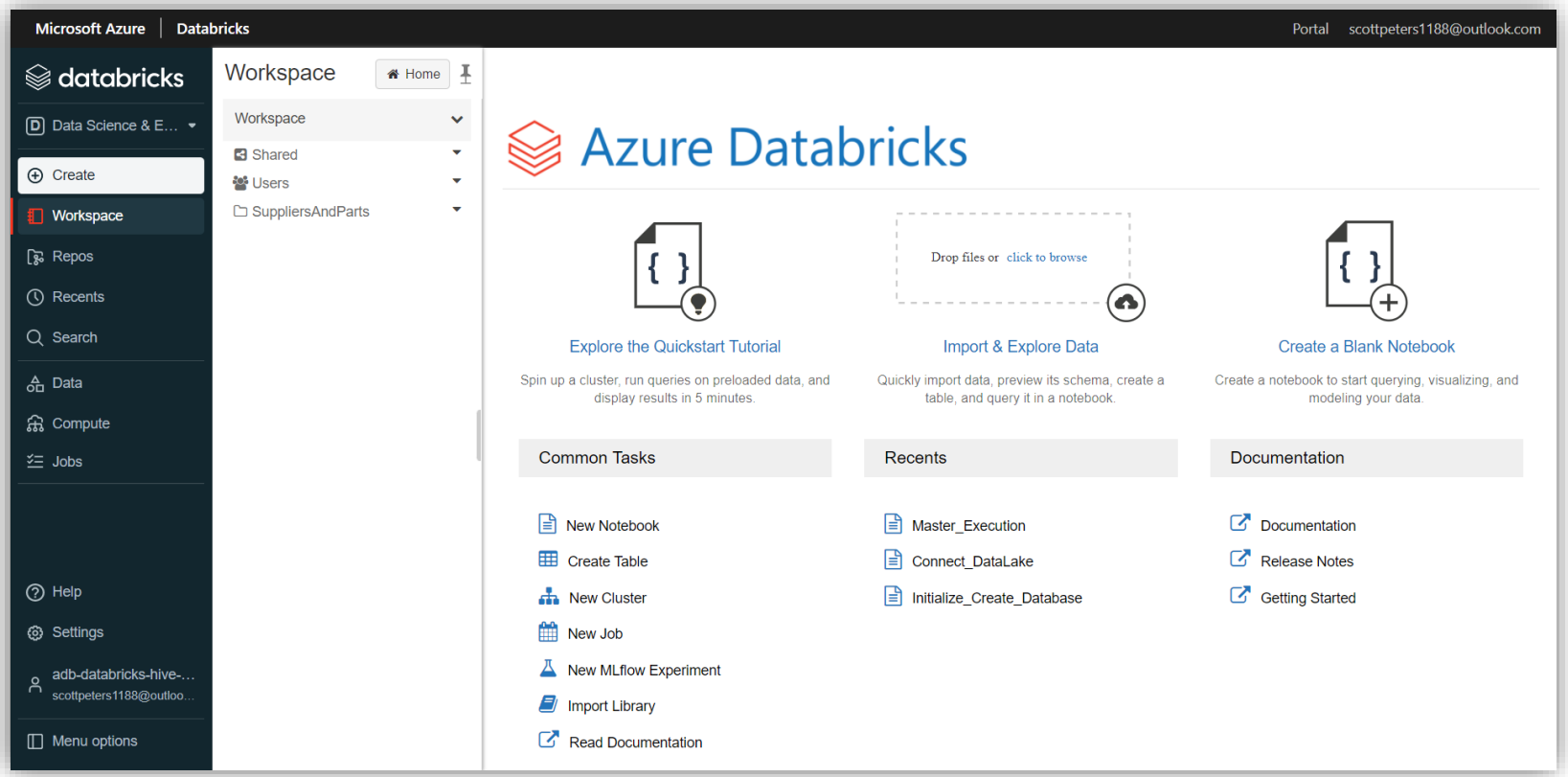
There are two different ways of creating a scope:

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.



DATABRICKS

Next, in the Databricks workspace, access the Create Secret Scope window.

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

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

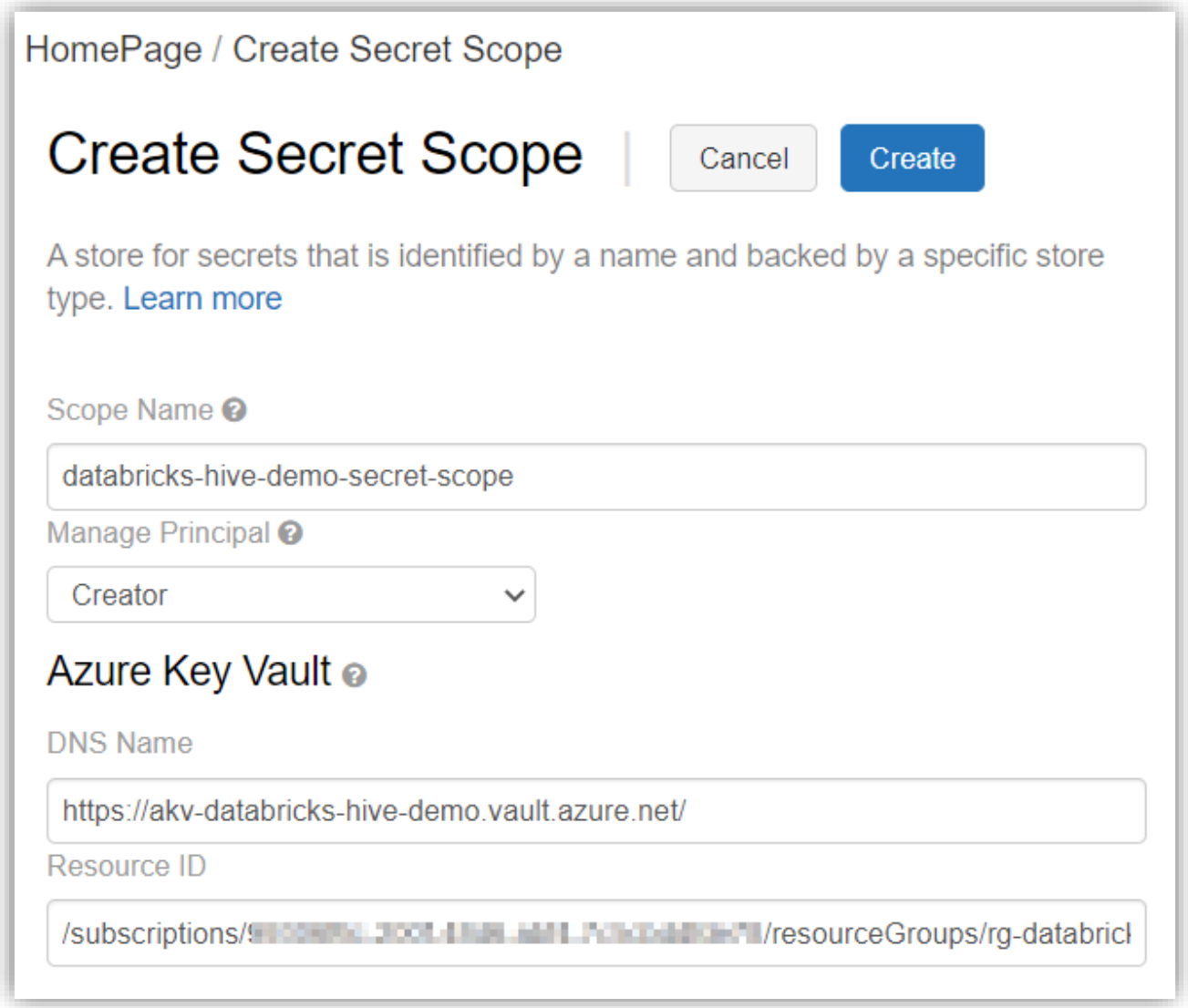
This URL is case sensitive; scope in createScope must be uppercase.

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.

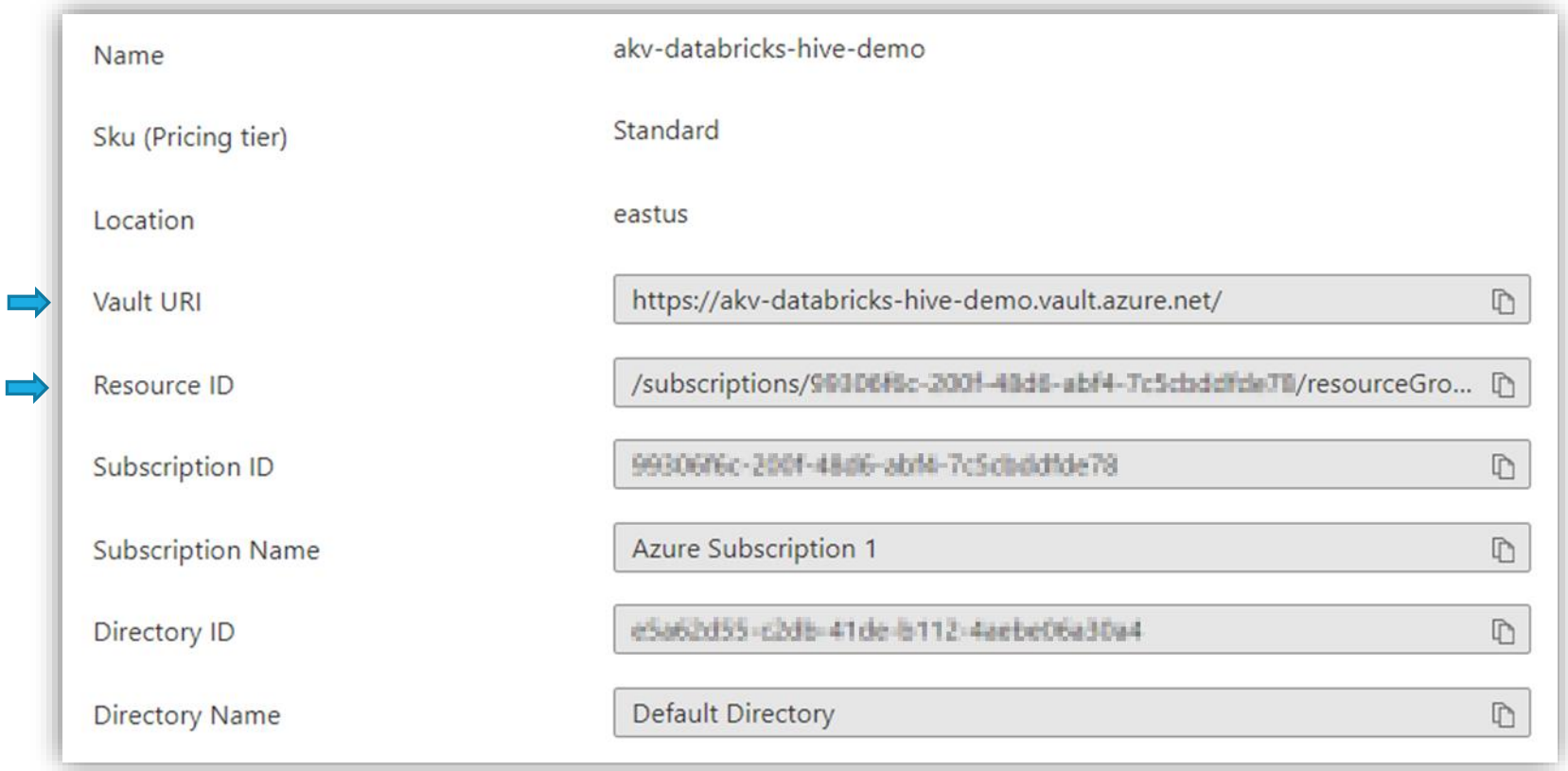


DATABRICKS SECRET SCOPE

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

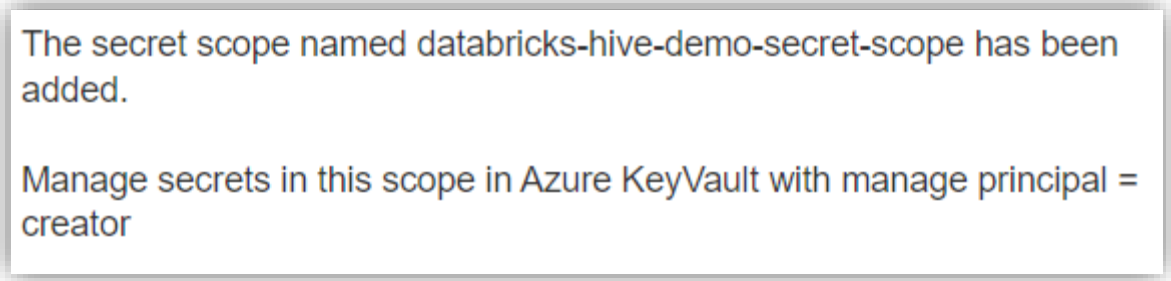
Copy these values from your Azure Key Vault properties page and insert the values into your Create Secret Scope window.

Azure Key Vault Properties Page

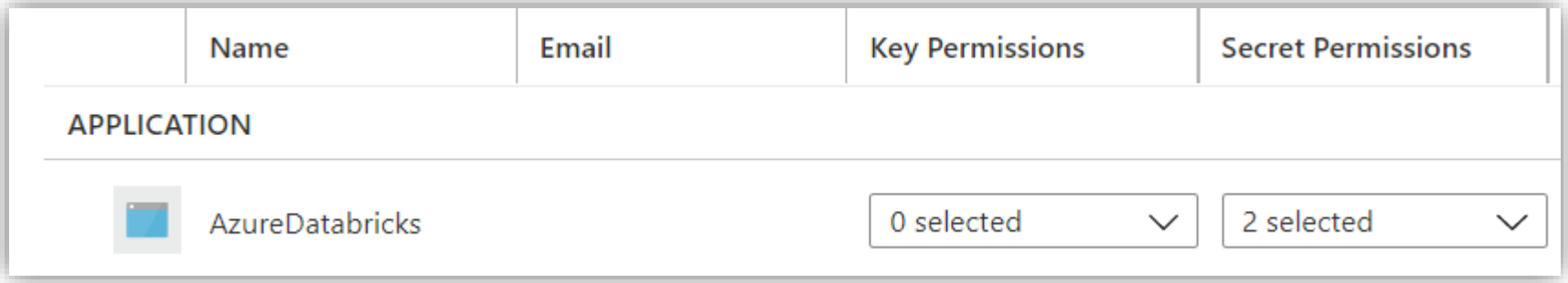


DATABRICKS SECRET SCOPE

Once completed, the following confirmation will appear.



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



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.

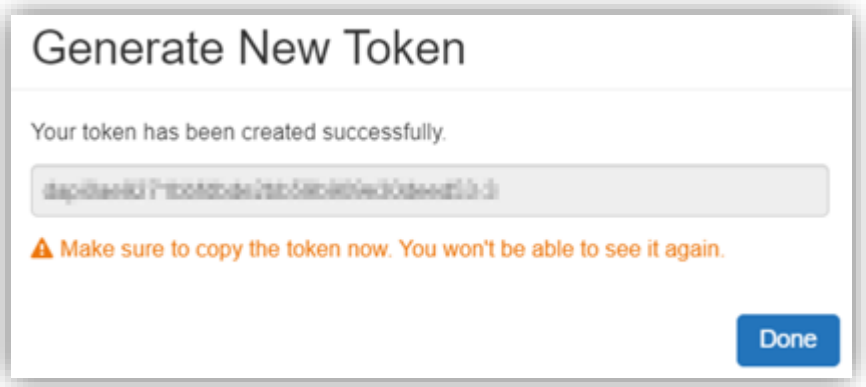
The documentation for Databricks tokens is located here:
https://docs.databricks.com/dev-tools/api/latest/authentication.html

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.



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.

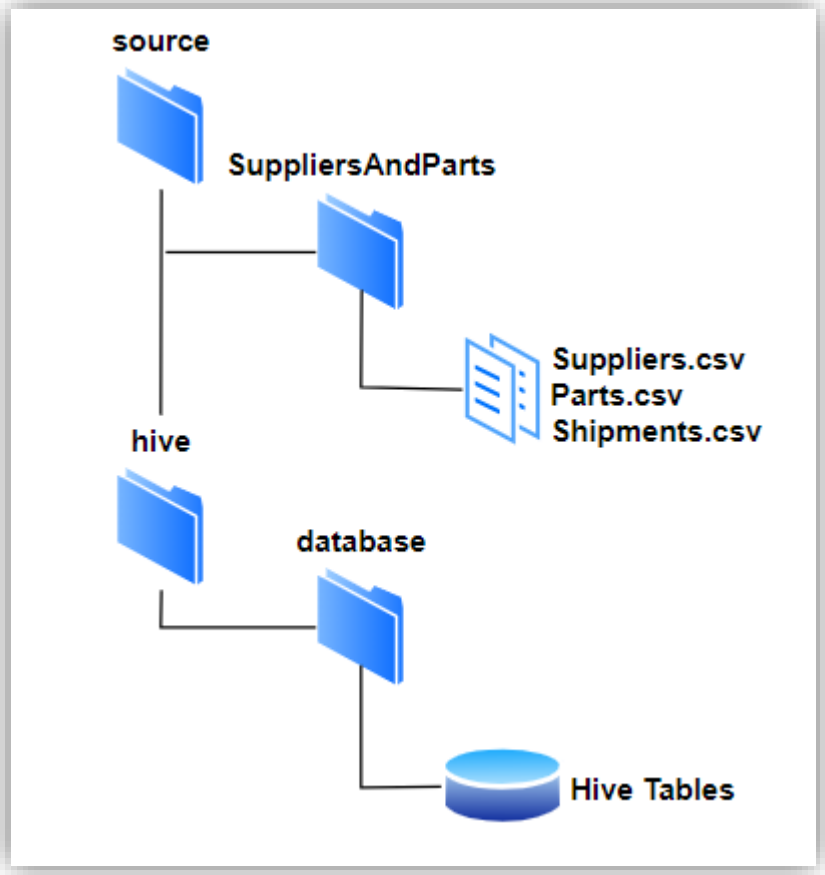
Within the source container, create a directory called SuppliersAndParts.

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 GitHub repository into the source > SuppliersAndParts container.

AZURE DATA LAKE

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



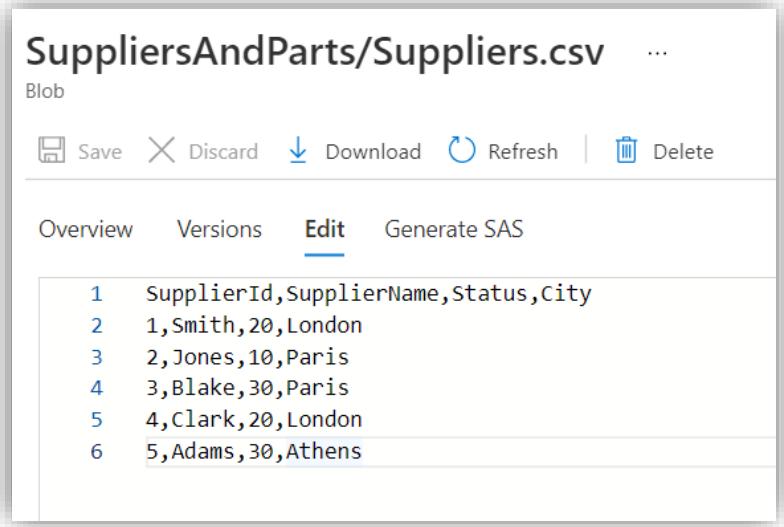
Copy the files Suppliers.csv, Parts.csv, and Shipments.csv files into the SuppliersAndParts folder.

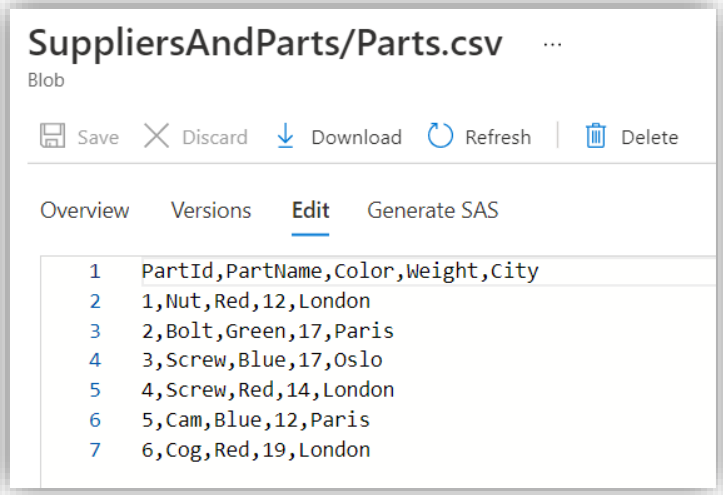
The database folder will contain the Hive tables, which we will setup later in this demo.

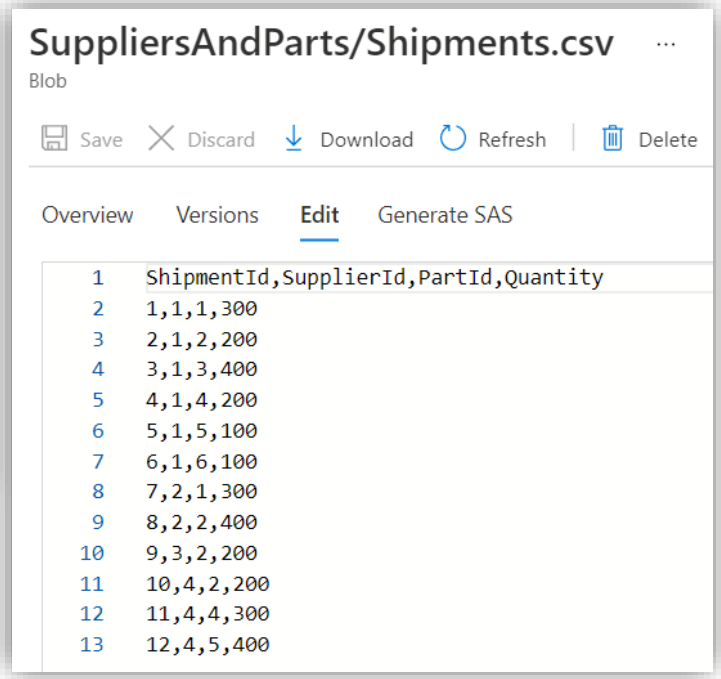
The GitHub repository for this demo is located here:
https://github.com/smpetersgithub/AdvancedSQLPuzzles

AZURE DATA LAKE

A quick peek at the three files reveals the following contents:





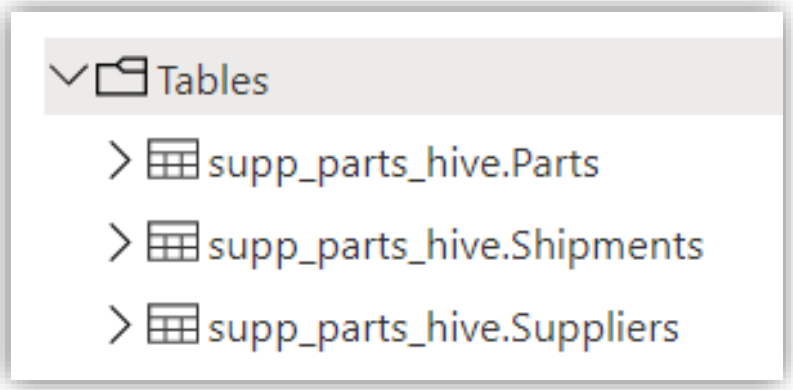


SQL SERVER DATABASE

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 GitHub directory.

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



The GitHub repository for this demo is located here:
https://github.com/smpetersgithub/AdvancedSQLPuzzles

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 slides….	Copied from the storage account properties
DataLakeName	dlsdatabrickshivedemo	The name of the storage account
DatabricksToken	Created in previous slides…	Used by Data Factory linked service

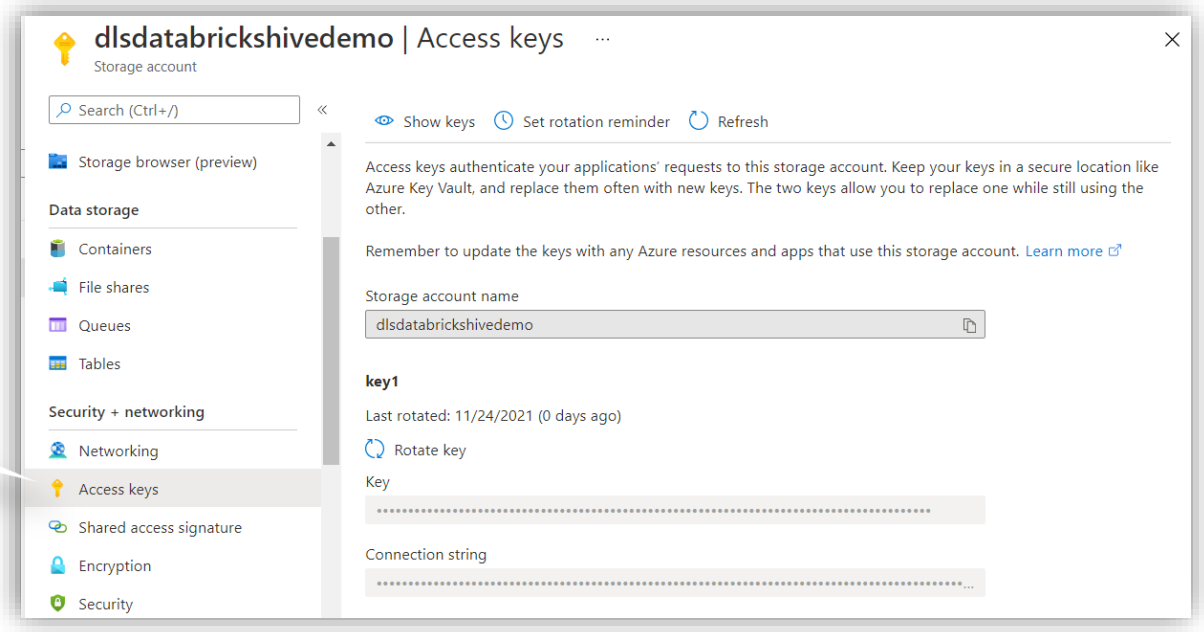
The two services we need Databricks to authenticate to are:

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.



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 the 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 the SQL Server Database
- We created the secret keys in the 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 the 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.

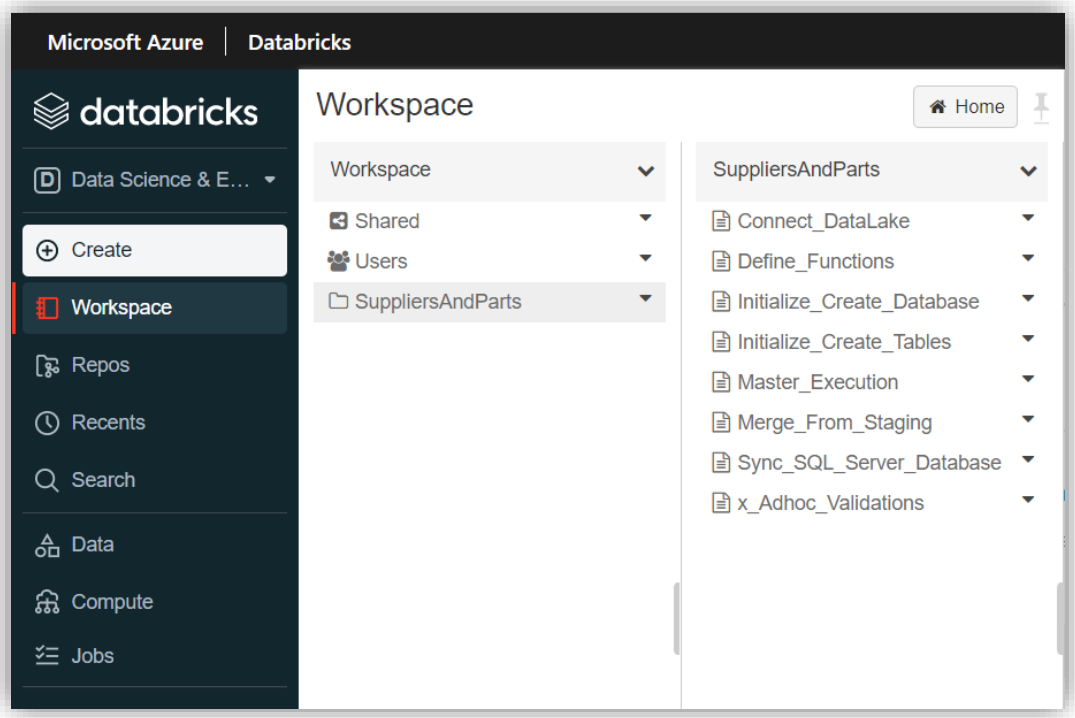
DATABRICKS

Begin by importing the workspace from the GitHub repository.

First, we will begin by importing the SuppliersAndParts.dbc workspace from the GitHub 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.

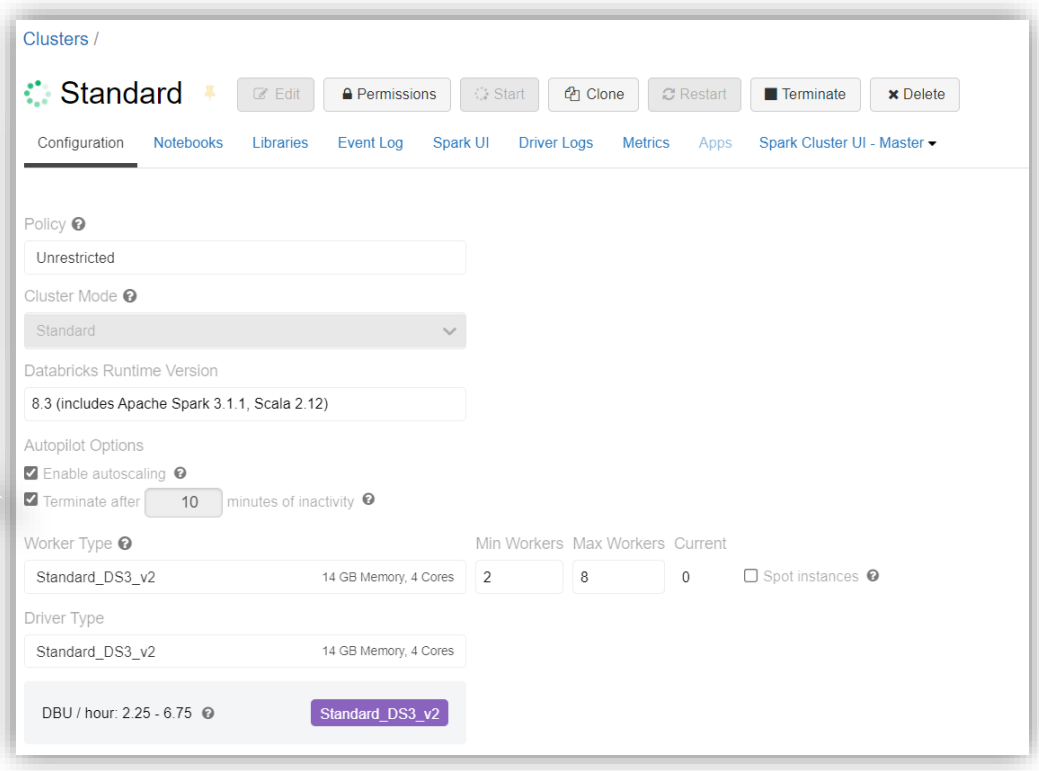
Notebook	Description
Create_Database	Initial script to create the Hive database.
Create_Tables	Initial script to create the Hive tables.
Connect_DataLake	Reads variables from the Azure Key Vault and connects to the Azure Data Lake.
Insert_Hive_Staging_Functions	Recreates the Hive staging tables and then inserts the data from the csv file into the staging table.
Merge_Hive_Production	Creates the functions needed to import the csv files into the staging tables.
Insert_SQL_Server_Database	Truncates and then inserts the data from the Hive production tables into the SQL Server tables.
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 inactivity time on the cluster!



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

DATABRICKS

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

```
1  DROP DATABASE if exists demo CASCADE;
2  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

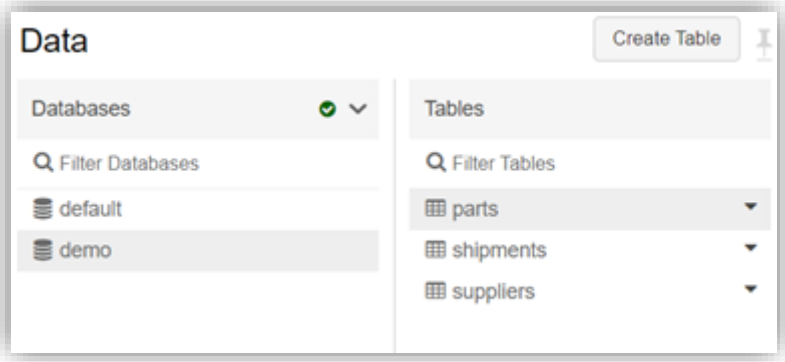
```
CREATE TABLE IF NOT EXISTS demo.Suppliers
(
    SupplierId numeric(4,0),
    SupplierName string,
    Status numeric(4,0),
    City string,
    InsertDate timestamp
)
USING DELTA
  LOCATION "abfss://hive@dlsdatabrickshivedemo.dfs.core.windows.net/database/Suppliers/delta/";
```

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

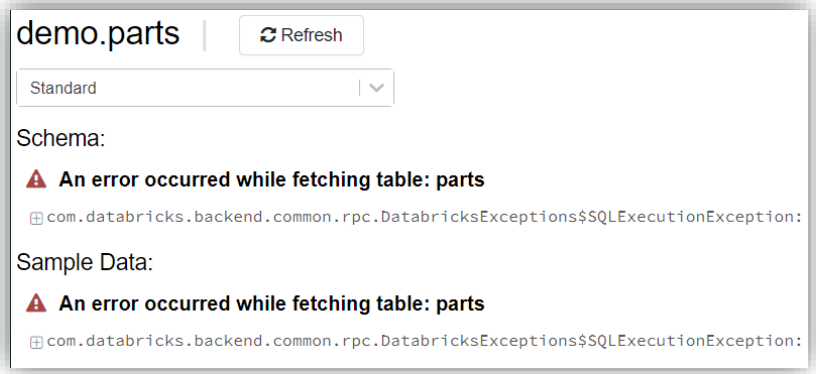
DATABRICKS

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 normal.



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 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

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

DBUTILS.WIDGETS

The documentation for Databricks widgets is located here:
https://docs.databricks.com/notebooks/widgets.html

Widgets

November 16, 2021

Input widgets allow you to add parameters to your notebooks and dashboards. The widget API consists of calls to create various types of input widgets, remove them, and get bound values.

Widgets are best for:

- Building a notebook or dashboard that is re-executed with different parameters
- Quickly exploring results of a single query with different parameters

Widget types

There are 4 types of widgets:

- `text`: Input a value in a text box.
- `dropdown`: Select a value from a list of provided values.
- `combobox`: Combination of text and dropdown. Select a value from a provided list or input one in the text box.
- `multiselect`: Select one or more values from a list of provided values.

DATABRICKS

Now we will look at some of the specifics in the notebooks.

You most probably encountered a few errors in running the master notebook.

In the next slides we will look at some of the specifics in the notebooks to help troubleshoot and understand the logic.

We will not cover all the different features of this code, but enough to be sufficient to troubleshoot, create test runs, and understand the overall process steps.

We will first start with the notebook, Insert_Hive_Staging_Functions and its various functions, before reviewing the other notebooks in this workspace.

DATABRICKS

Insert_Hive_Staging_Functions

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

The function ProcessHiveStagingData calls all other functions.

Order	Function Name	Description
Master	ProcessHiveStagingData	Reads the csv file and executes InsertDataToHiveFunction
1	GetSchema	Defines the schema for the file
2	GenerateIncomingDataHeader	Creates a data frame of the file header
3	VerifyHeader	Validates incoming file header to schema definition
4	InsertDataToHive	Inserts the data into the Hive tables

We will cover each of these functions in the next few slides.

DATABRICKS

Insert_Hive_Staging_Functions

The ProcessHiveStagingData is the master function that calls the other four functions.
I have highlighted the function calls below.

Process data to Hive

```
1  def ProcessHiveStagingData(FilePath, ProcessName):
2    print("wasbs://source@"+datalakename+".blob.core.windows.net"+FilePath)
3    df = spark.read.csv("wasbs://source@"+datalakename+".blob.core.windows.net"+FilePath, sep=",", mode="DROPMALFORMED", schema=GetSchema(ProcessName))
4
5    if (len(df.head(1)) == 0):
6      raise ValueError("Please review the schema. Possible new columns were added.")
7    else:
8      new_df = GenerateIncomingDataHeader(df.limit(1))
9
10   #Verify the header
11   VerifyHeader(df,new_df)
12
13   #Fetch data in file
14   df = df.filter(~col(df.schema.fields[0].name).contains(df.schema.fields[0].name))
15
16   #Concatenates ProcessName and the string "Stage" to derive the staging table name.
17   TableName = ProcessName+"Stage"
18   InsertDataToHive(df,TableName)
19
20   print('End of Function: ProcessData')
```

Get Schema

Generate Incoming Header

Verify Header

Insert Data To Hive

DATABRICKS

Insert_Hive_Staging_Functions

The variable TableName is the concatenation of the variable ProcessName and the string "Stage". Also, the field separator is set when we create the data frame.

Process data to Hive

```
1  def ProcessHiveStagingData(FilePath, ProcessName):
2    print("wasbs://source@"+datalakename+".blob.core.windows.net"+FilePath)
3    df = spark.read.csv("wasbs://source@"+datalakename+".blob.core.windows.net"+FilePath, sep=",", mode="DROPMALFORMED", schema=GetSchema(ProcessName))
4
5    if (len(df.head(1)) == 0):
6      raise ValueError("Please review the schema. Possible new columns were added.")
7    else:
8      new_df = GenerateIncomingDataHeader(df.limit(1))
9
10   #Verify the header
11   VerifyHeader(df,new_df)
12
13   #Fetch data in file
14   df = df.filter(~col(df.schema.fields[0].name).contains(df.schema.fields[0].name))
15
16   #Concatenates ProcessName and the string "Stage" to derive the staging table name.
17   TableName = ProcessName+"Stage"
18   InsertDataToHive(df,TableName)
19
20   print('End of Function: ProcessData')
```

The column separator is set here

TableName is set here

DATABRICKS

Insert_Hive_Staging_Functions

A Dataframe named df is also created in this function.

Process data to Hive

```
1  def ProcessHiveStagingData(FilePath, ProcessName):
2    print("wasbs://source@"+datalakename+".blob.core.windows.net"+FilePath)
3    df = spark.read.csv("wasbs://source@"+datalakename+".blob.core.windows.net"+FilePath, sep=",", mode="DROPMALFORMED", schema=GetSchema(ProcessName))
4
5    if (len(df.head(1)) == 0):
6      raise ValueError("Please review the schema. Possible new columns were added.")
7    else:
8      new_df = GenerateIncomingDataHeader(df.limit(1))
9
10   #Verify the header
11   VerifyHeader(df,new_df)
12
13   #Fetch data in file
14   df = df.filter(~col(df.schema.fields[0].name).contains(df.schema.fields[0].name))
15
16   #Concatenates ProcessName and the string "Stage" to derive the staging table name.
17   TableName = ProcessName+"Stage"
18   InsertDataToHive(df,TableName)
19
20   print('End of Function: ProcessData')
```

The csv file is imported here and a Dataframe is created via the spark.read.csv call

DATABRICKS

Insert_Hive_Staging_Functions

The GetSchema function in the first function called from the master function ProcessHiveStagingData.

The variable Name is passed to the function.

This defines the blueprint, known as a schema, that defines the name and data type of each column. We use StringType to define each column.

Define the schema details for the file

```
1  def GetSchema(Name):
2    if Name == "Suppliers":
3      schema = StructType(
4      [
5          StructField('SupplierId', StringType()),
6          StructField('SupplierName', StringType()),
7          StructField('Status', StringType()),
8          StructField('City', StringType())
9
10     ]
11     )
```

The documentation for Spark StructType class is located here:
https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.types.StructType.html

DATABRICKS

Insert_Hive_Staging_Functions

The GenerateIncomingDataHeader function creates the Dataframe new_df of the imported file's header.

We will perform a validation in the next function, VerifyHeader.

Create a Dataframe of the imported file's header

```
1  def GenerateIncomingDataHeader(df):
2    list_of_new_column_names = []
3
4    for i in df.limit(1).collect()[0]:
5      list_of_new_column_names.append(i)
6    for i,k in enumerate(df.schema.fields):
7      k.name = list_of_new_column_names[i]
8
9    new_df = spark.createDataFrame(df.rdd, df.schema)
10   return new_df
11
12   print('End of Function: GenerateIncomingDataHeader')
```

DATABRICKS

Insert_Hive_Staging_Functions

The function VerifyHeader compares the file header to the schema definition.

If they are not the same, an error is raised.

Validate incoming file header to schema definition

```
1  def VerifyHeader(df,new_df):
2    for idx1,el1 in enumerate(df.dtypes):
3      for idx2,el2 in enumerate(new_df.dtypes):
4          if idx1 == idx2:
5              if el1[0] == el2[0]:
6                  continue
7              else:
8                  raise ValueError("Schema does not match for Col {0} and {1}".format(el1[0],el2[0]))
9
10   print('End of Function: VerifyHeader')
```

DATABRICKS

Insert_Hive_Staging_Functions

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

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

Insert data to Hive

```
1  def InsertDataToHive(df, TableName):
2      print('Table being inserted is',TableName)
3      print('The table path is',TablePathHive)
4
5      #Truncate and overwrite options are set to True
6      df.write.mode("overwrite").option("truncate", True).option("overwriteSchema", "true").format("delta").option("path",
   TablePathHive+"/"+TableName+"/delta/").saveAsTable("demo."+TableName)
7
8      print('End of Function: InsertDataToHive')
```

DATABRICKS

Merge_Hive_Production

Moving on to the next notebooks…

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

```
1  dbutils.widgets.text("ProcessName", "","")
2  ProcessName = dbutils.widgets.get("ProcessName")
3
4  #Manually sets the ProcessName and FileName if testing in Databricks
5  if ProcessName == "":
6    ProcessName = 'Suppliers'
7
8  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

```
1  dbutils.widgets.text("ProcessName", "","")
2  ProcessName = dbutils.widgets.get("ProcessName")
3
4  #Manually sets the ProcessName and FileName if testing in Databricks
5  if ProcessName == "":
6    ProcessName = 'Suppliers'
7
8  print('ProcessName is',ProcessName)
```

DATABRICKS

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.

Also available in this notebook is how to view the history of the tables.

DATA FACTORY

Next, we will setup the Azure Data Factory.

We will need to create two Azure Linked Services.

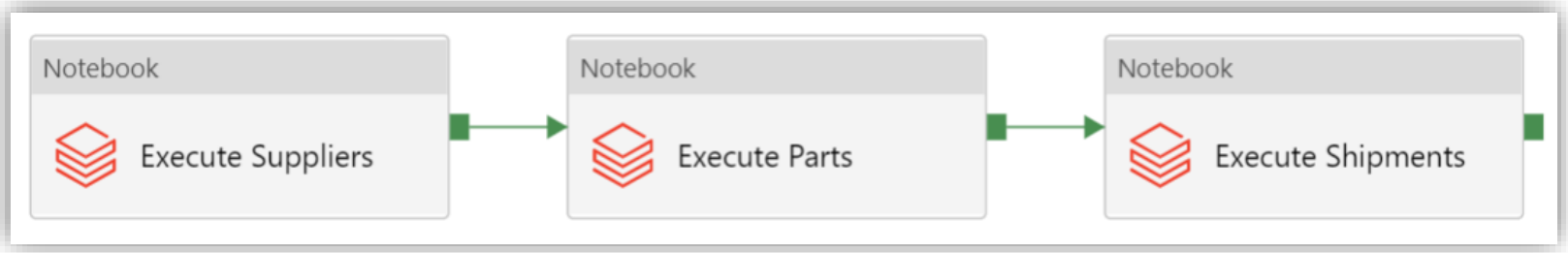
One to link to our 1) Azure Key Vault, and the second to link to the 2) Databricks Workspace.

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

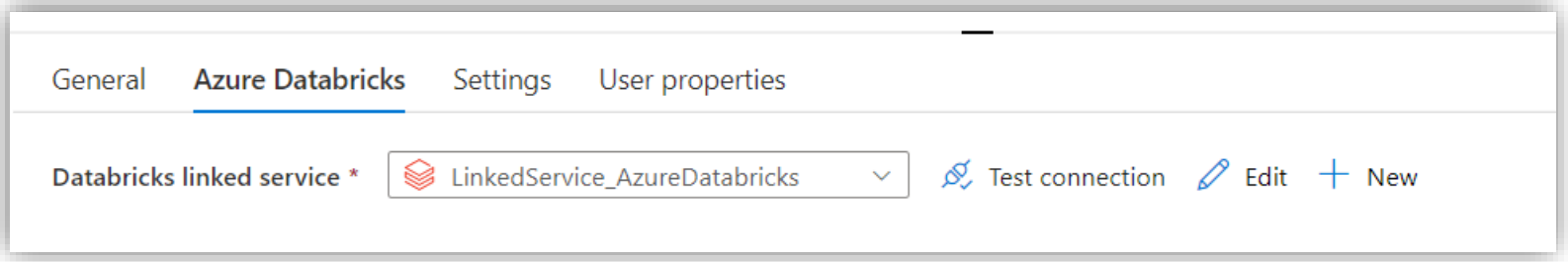
We will cover these steps in the next few slides.

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.



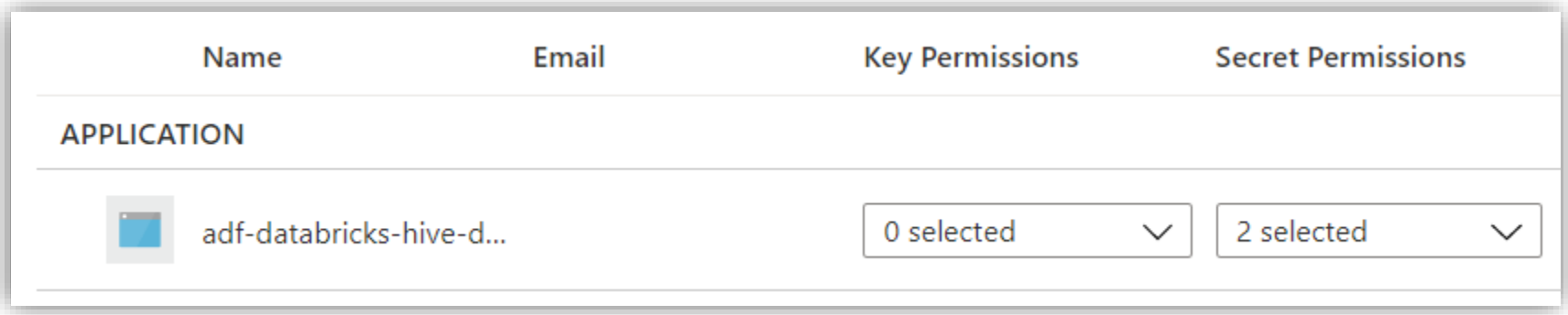
Each of these activities will utilize the Databricks linked service we will create in the next slides.



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.

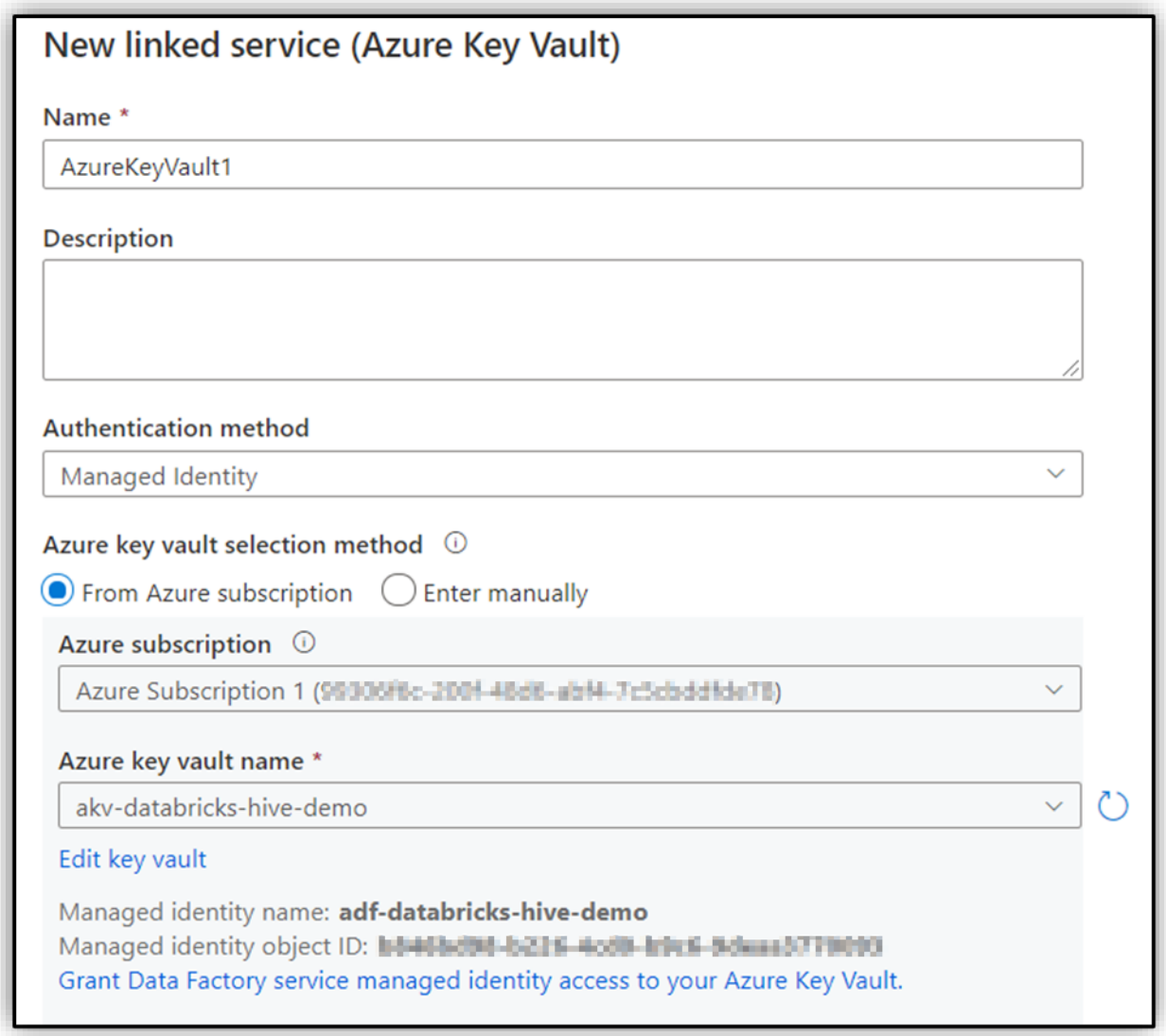


DATA FACTORY

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.

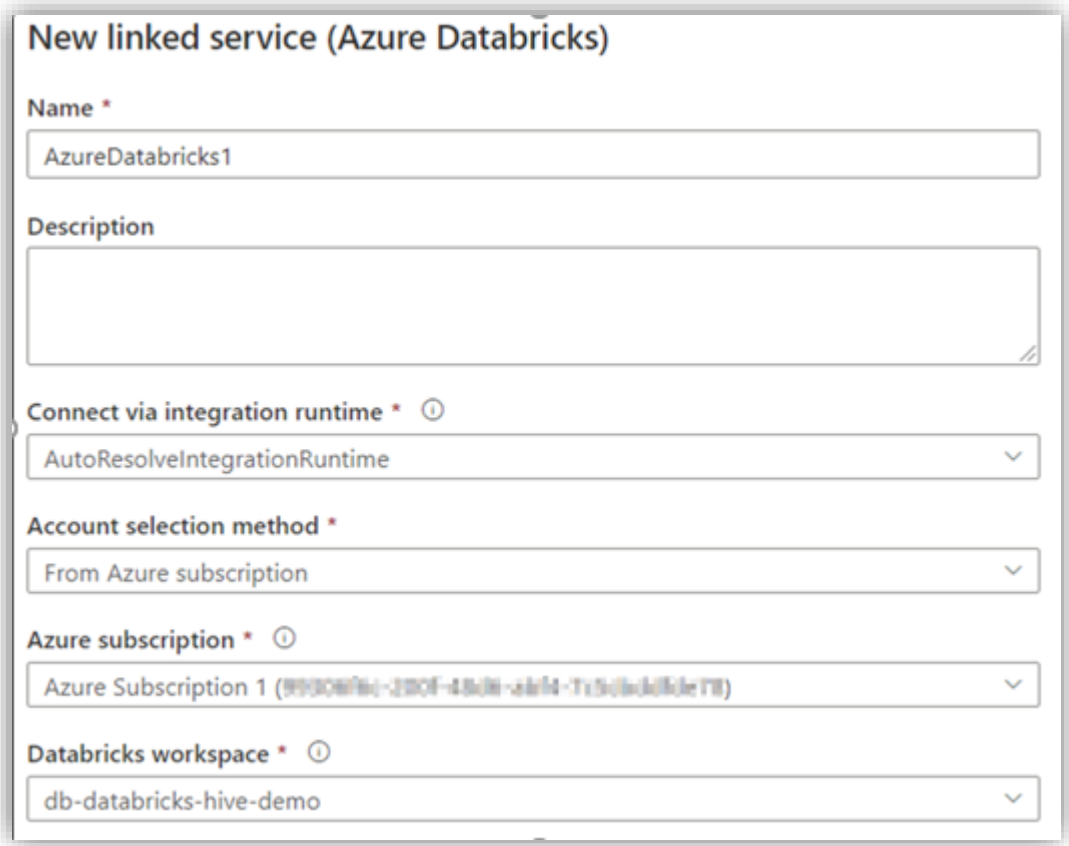


DATA FACTORY

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.



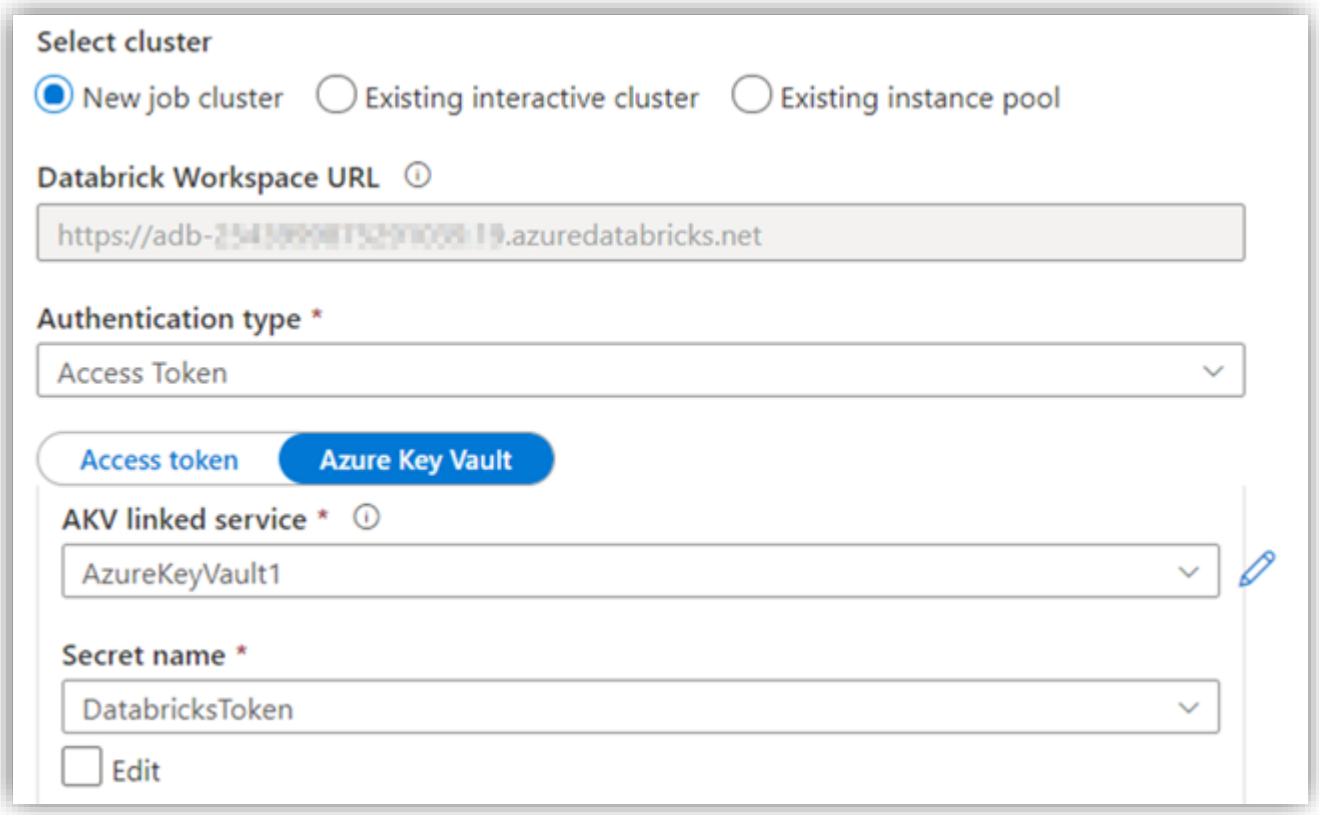
DATA FACTORY

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.



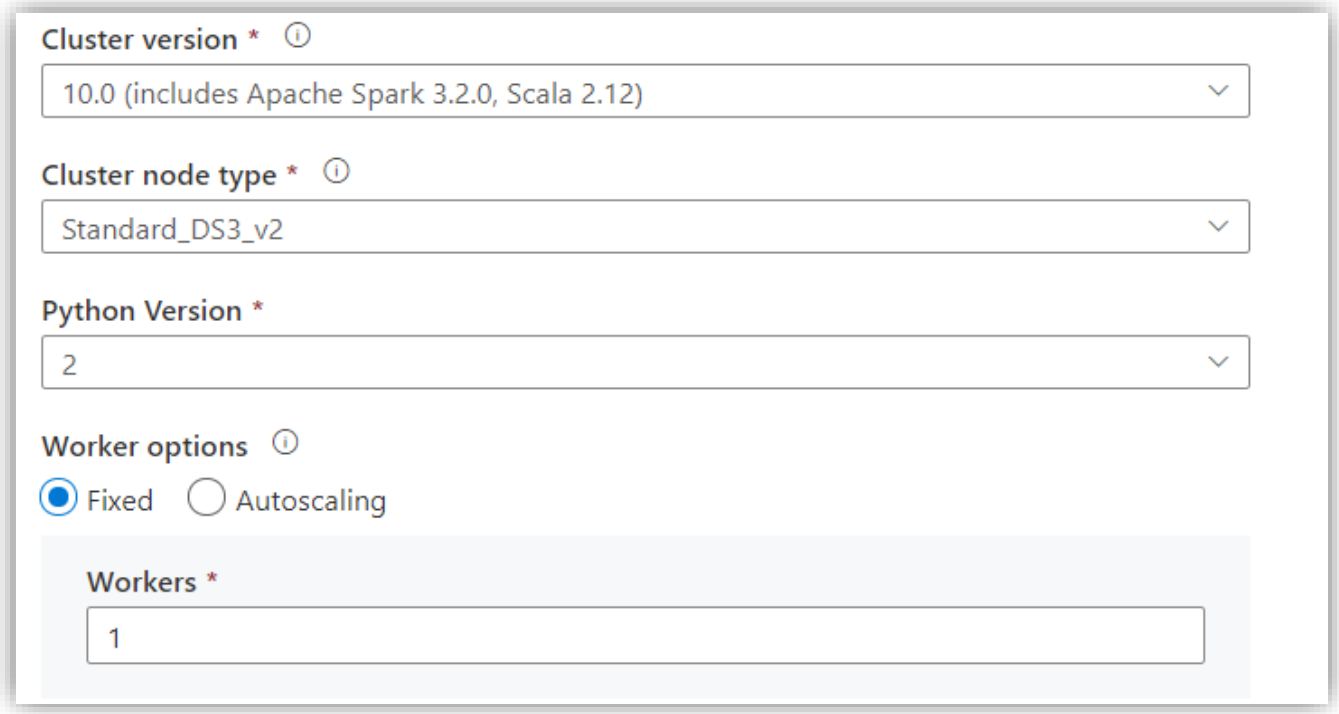
DATA FACTORY

Databricks Linked Service

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

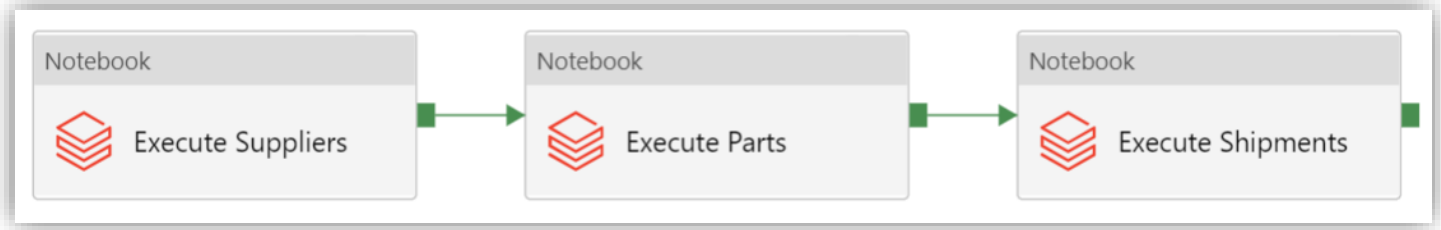
Here you can see the options that I chose for cluster version, node type, etc.….

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.



DATA FACTORY

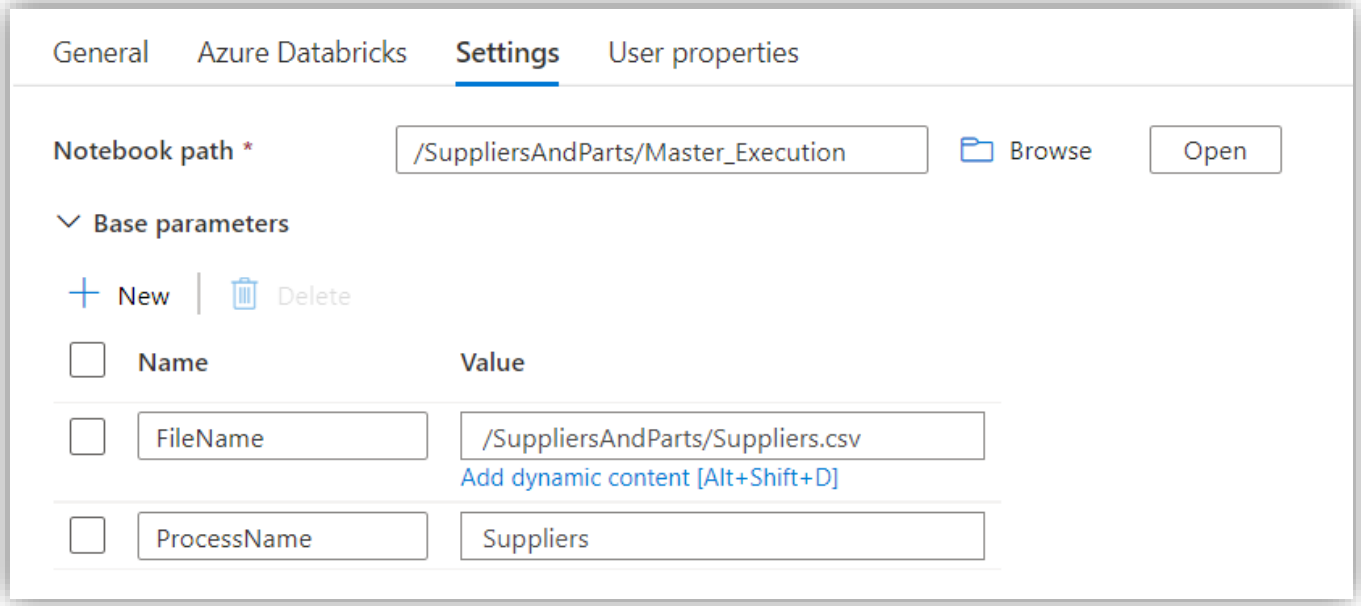
Now let us move back to the following pipeline activities and update the Settings tab for each activity.



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.

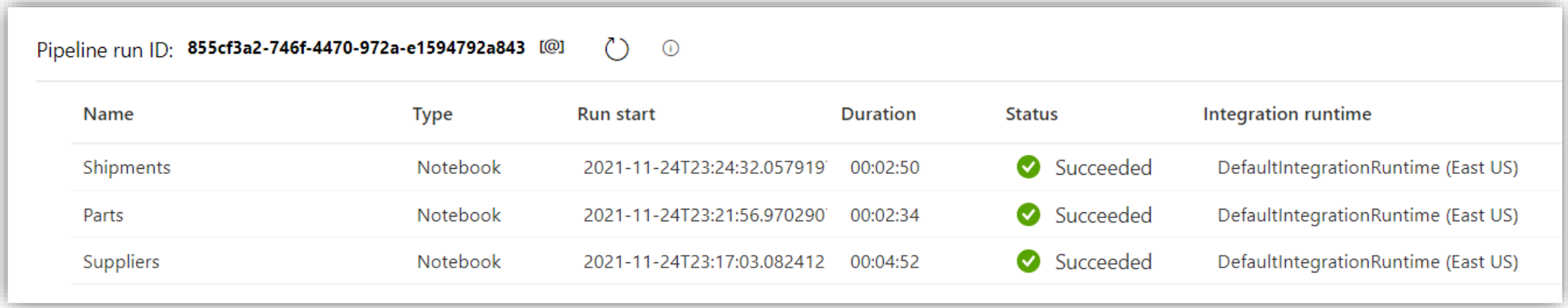


DATAFACTORY

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 modifying 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 GitHub repository and SQL blog for all sort of puzzles, tips and tricks.

The GitHub repository for this demo is located here:
https://github.com/smpetersgithub/AdvancedSQLPuzzles