You can prepare these source files by using the Adventure Works LT2019 database which is available here: github.com/microsoft/sql-server-samples/tree/master/samples/databases.

Here is the SQL query which is used for Customer1.json in Figure 17-2.

SELECT  
       (  
              SELECT firstname,  
                     lastname,  
                     middlename,  
                     title,  
                     customerid FOR json path,  
                     without\_array\_wrapper)  
FROM   saleslt.customer

Similar to the previous query, here is the SQL query which is used for Customer2.json.

SELECT  
       (  
              SELECT firstname,  
                     lastname,  
                     middlename,  
                     title,  
                     customerid,  
                     companyname,  
                     emailaddress,  
                     salesperson,  
                     namestyle FOR json path,  
                     without\_array\_wrapper)  
FROM   saleslt.customer

Finally, here is the SQL query which is used for Customer3.json.

SELECT  
       (  
              SELECT firstname,  
                     lastname,  
                     middlename,  
                     title,  
                     customerid,  
                     companyname,  
                     emailaddress,  
                     salesperson,  
                     namestyle,  
                     modifieddate,  
                     phone,  
                     rowguid FOR json path,  
                     without\_array\_wrapper)  
FROM   saleslt.customer

In Databricks, create a new secret scope by navigating to https://<DATABRICKS-INSTANCE>#secrets/createScope and replace <DATABRICKS-INSTANCE> with your own Databricks URL instance. This URL will take you to the UI where you can create your secret scope. Paste the Key Vault URI and Resource ID from your Key Vault into the respective DNS Name and Resource ID section.

Here is the code shown in the figure above. Note that you could use either the numBytes or numFiles properties.

#spark.conf.set("spark.databricks.cloudfiles.schemaInference.sampleSize.numBytes",10000000000)

spark.conf.set("spark.databricks.cloudfiles.schemaInference.sampleSize.numFiles",10)

Here is the code that is used in the figure above. Since you have given Databricks access to the Key Vault secret scope, there should be no errors when your run this code.

subscriptionId = dbutils.secrets.get("akv-0011","subscriptionId")

tenantId = dbutils.secrets.get("akv-0011","tenantId")

clientId = dbutils.secrets.get("akv-0011","clientId")

clientSecret = dbutils.secrets.get("akv-0011","clientSecret")

resourceGroup = dbutils.secrets.get("akv-0011","resourceGroup")

queueconnectionString = dbutils.secrets.get("akv-0011","queueconnectionString")

SASKey = dbutils.secrets.get("akv-0011","SASKey")

Here is the code which is shown in the figure above. Since cloudFiles will automatically create EventGrid topics and subscriptions it will need the credentials in this code to get access to the relevant Azure resources. The \_checkpoint folder will store the schema meta-data and will also keep track of multiple versions of the evolved schemas. The partitionColumns config provides the option to read Hive style partition folder structures. The schema evolution mode of ‘failOnNewColumns’ will simply fail the job when new columns are detected and will require manual intervention to define and update and new schema. We will not be exploring this option.

cloudfile = {

"cloudFiles.subscriptionID": subscriptionId,

"cloudFiles.connectionString": queueconnectionString,

"cloudFiles.format": "json",

"cloudFiles.tenantId": tenantId,

"cloudFiles.clientId": clientId,

"cloudFiles.clientSecret": clientSecret,

"cloudFiles.resourceGroup": resourceGroup,

"cloudFiles.useNotifications": "true",

"cloudFiles.schemaLocation": "/mnt/raw/Customer\_stream/\_checkpoint/",

"cloudFiles.schemaEvolutionMode": "rescue"

#"cloudFiles.inferColumnTypes": "true"

#"cloudFiles.schemaEvolutionMode": "failOnNewColumns"

#"cloudFiles.schemaEvolutionMode": "addNewColumns"

#"cloudFiles.partitionColumns": ""

}

With the AdditionalOptions properties, you can define schema hints, rescued data columns and more. In this code block, you are specifying which column to add the rescued data into.

AdditionalOptions = {"rescueDataColumn":"\_rescued\_data"}

In this next code block, set the ADLS gen2 config by adding the ADLS gen2 account and SAS Key.

spark.conf.set("fs.azure.account.key.adlsg2v001.dfs.core.windows.net","SASKey")

Here is the code that is used in the figure above.

df = (spark.readStream.format("cloudFiles")

.options(\*\*cloudfile)

.options(\*\*AdditionalOptions)

.load("abfss://data@adlsg2v001.dfs.core.windows.net/raw/Customer\_stream/"))

Upon opening the file, notice how it captures the JSON data frame schema structure as expected.

{"dataSchemaJson":"{\"type\":\"struct\",\"fields\":[{\"name\":\"FirstName\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"LastName\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"MiddleName\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"Title\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"customerid\",\"type\":\"long\",\"nullable\":true,\"metadata\":{}}]}","partitionSchemaJson":"{\"type\":\"struct\",\"fields\":[]}"}

Here is the code. Remember to delete your \_schemas folder so that the process can infer the schema from scratch once again.

AdditionalOptions = {

"cloudFiles.schemaHints":"customerid int",

"rescueDataColumn":"\_rescued\_data"}

Here is the code that is used in the figure above.

df = (spark.readStream.format("cloudFiles")

.options(\*\*cloudfile)

.options(\*\*AdditionalOptions)

.load("abfss://data@adlsg2v001.dfs.core.windows.net/raw/Customer\_stream/"))

Upon opening this file, notice that it contains the additional columns that were added from Customer2.json.

v1

{"dataSchemaJson":"{\"type\":\"struct\",\"fields\":[{\"name\":\"FirstName\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"LastName\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"MiddleName\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"Title\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"customerid\",\"type\":\"long\",\"nullable\":true,\"metadata\":{}},{\"name\":\"CompanyName\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"EmailAddress\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"SalesPerson\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"namestyle\",\"type\":\"boolean\",\"nullable\":true,\"metadata\":{}}]}","partitionSchemaJson":"{\"type\":\"struct\",\"fields\":[]}"}

Upon opening the version 2 schema struct file, notice that it includes the new columns from Customer3.json.

v1

{"dataSchemaJson":"{\"type\":\"struct\",\"fields\":[{\"name\":\"FirstName\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"LastName\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"MiddleName\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"Title\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"customerid\",\"type\":\"long\",\"nullable\":true,\"metadata\":{}},{\"name\":\"CompanyName\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"EmailAddress\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"SalesPerson\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"namestyle\",\"type\":\"boolean\",\"nullable\":true,\"metadata\":{}},{\"name\":\"ModifiedDate\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"Phone\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"rowguid\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}}]}","partitionSchemaJson":"{\"type\":\"struct\",\"fields\":[]}"}

Here is the readStream code that you will need to run by pre-defining the cloudFiles options and storage location mount point.

read\_df = (spark.readStream.format("cloudFiles")

.options(\*\*cloudfile)

.options(\*\*AdditionalOptions)

.load("/mnt/raw/Customer\_stream/"))

Here is the code that you will need to run to append the data stream in Delta format to the defined location. Once you read and write the streaming data into Delta format in ADLS gen2, you can begin to view and manage the Auto Loader resources programmatically.

write\_df = (read\_df.writeStream

.format("delta")

.trigger(once=True)

.outputMode("append")

.option("checkpointLocation", "/mnt/raw/Customer\_stream/\_checkpoint/")

.start("/mnt/raw/Customer\_stream/data/")

Here is the code which will import the CloudFilesAzureResourceManager.

import com.databricks.sql.CloudFilesAzureResourceManager

Here is the Scala code that will get the necessary secrets from your Azure Key Vault.

val subscriptionId = dbutils.secrets.get("akv-0011","subscriptionId")

val tenantId = dbutils.secrets.get("akv-0011","tenantId")

val clientId = dbutils.secrets.get("akv-0011","clientId")

val clientSecret = dbutils.secrets.get("akv-0011","clientSecret")

val resourceGroup = dbutils.secrets.get("akv-0011","resourceGroup")

val queueconnectionString = dbutils.secrets.get("akv-0011","queueconnectionString")

val SASKey = dbutils.secrets.get("akv-0011","SASKey")

Here is the code that you will need to run to create the necessary Auto Loader Resource Manager.

val manager = CloudFilesAzureResourceManager

.newManager

.option("cloudFiles.connectionString", queueconnectionString)

.option("cloudFiles.resourceGroup", resourceGroup)

.option("cloudFiles.subscriptionId", subscriptionId)

.option("cloudFiles.tenantId", tenantId)

.option("cloudFiles.clientId", clientId)

.option("cloudFiles.clientSecret", clientSecret)

.create()

Alternatively, you have the option of manually setting up an Event Grid subscription and storage queue by specifying the associated path in the code below. This option demonstrates the flexibility of either setting up the Auto Loader Resource manager manually or automatically, as needed.

// Set up an AQS queue and an event grid subscription associated with the path used in the manager. Available in Databricks Runtime 7.4 and above.

manager.setUpNotificationServices()

Here is the code that you will need to run to list notification services created by Auto Loader.

// List notification services created by Auto Loader

val NotificationSvc = manager.listNotificationServices()

Here is the code which will delete the specified ‘streamID’ storage queue and Event Grid subscription of the registered Auto Loader resource.

// Tear down the notification services created for a specific stream ID.

// Stream ID is a GUID string that you can find in the list result above.

val streamId = "4d42351a-fd96-4668-a3f1-f3bab3df0223"

manager.tearDownNotificationServices(streamId)

Run the following code to first identify this filtered list of queries within the Auto Loader resources.

val FilterNotificationSvc = NotificationSvc.filter("path like '%abfss%'")

Here is the code that you would run to collect and delete the list of filtered Auto Loader resources.

for (row <- FilterNotificationSvc.collect())

{

val streamId = row.get(0).toString()

manager.tearDownNotificationServices(streamId)

}