References:

<https://hevodata.com/learn/databricks-autoloader/#t6>

<https://learn.microsoft.com/en-us/azure/databricks/ingestion/auto-loader/directory-listing-mode>

<https://docs.databricks.com/ingestion/auto-loader/file-notification-mode.html#permissions-s3>

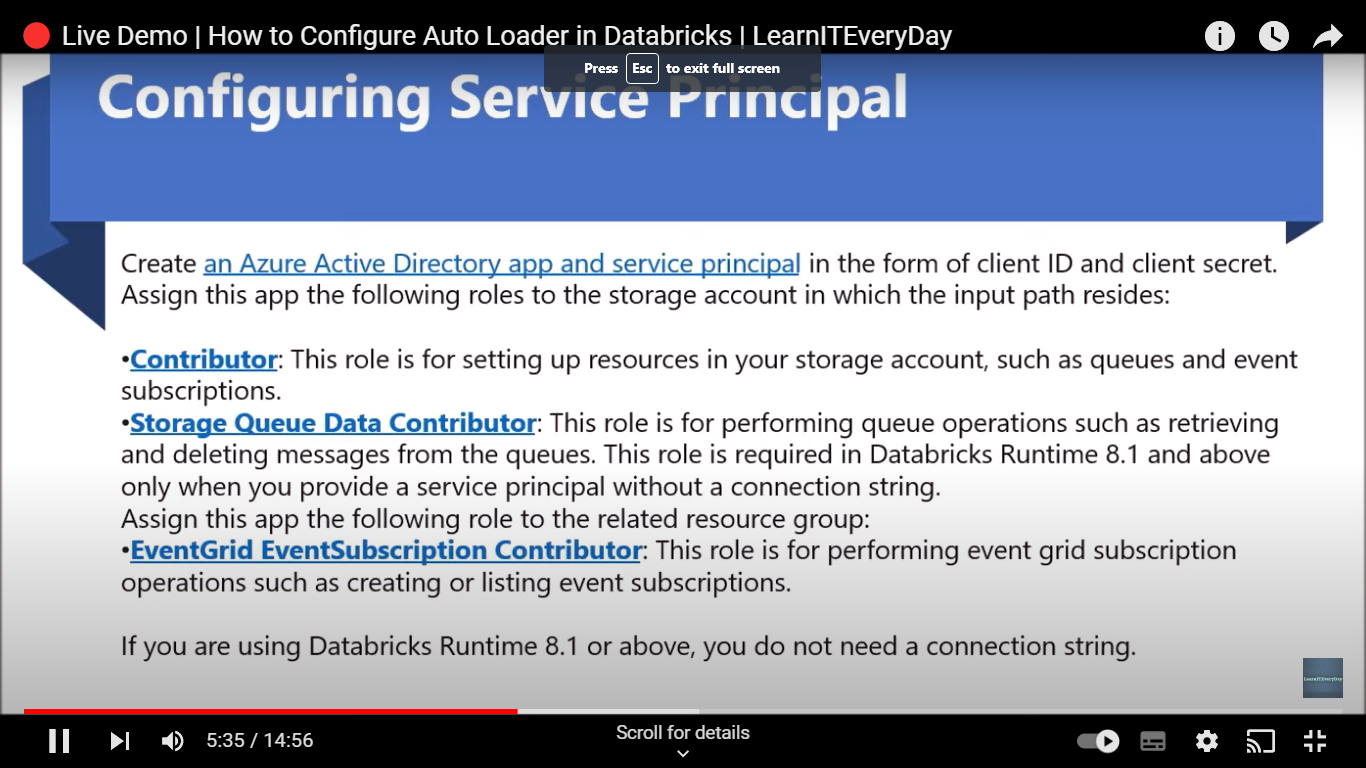
<https://learn.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#eventgrid-eventsubscription-contributor>

Auto Loader incrementally and efficiently processes new data files as they arrive in cloud storage. Auto Loader can load data files from AWS S3 (s3://), Azure Data Lake Storage Gen2 (ADLS Gen2, abfss://), Google Cloud Storage (GCS, gs://), Azure Blob Storage (wasbs://), ADLS Gen1 (adl://), and Databricks File System (DBFS, dbfs:/). Auto Loader can ingest JSON, CSV, PARQUET, AVRO, ORC, TEXT, and BINARYFILE file formats.

Auto Loader provides a Structured Streaming source called cloudFiles. Given an input directory path on the cloud file storage, the cloudFiles source automatically processes new files as they arrive, with the option of also processing existing files in that directory. Auto Loader has support for both Python and SQL in Delta Live Tables.

You can use Auto Loader to process billions of files to migrate or backfill a table. Auto Loader scales to support near real-time ingestion of millions of files per hour.

In order to configure Auto Loader, the following roles should be assigned at storage account level:



Auto Loader supports two file detection modes.

1. Directory listing mode
2. File notification mode

**Directory listing mode:**

Auto Loader uses directory listing mode by default. In directory listing mode, Auto Loader identifies new files by listing the input directory. Directory listing mode allows you to quickly start Auto Loader streams without any permission configurations other than access to your data on cloud storage.

For best performance with directory listing mode, use Databricks Runtime 9.1 or above.

Azure Databricks has optimized directory listing mode for Auto Loader to discover files in cloud storage more efficiently than other Apache Spark options.

For example, if you have files being uploaded every 5 minutes as **/some/path/YYYY/MM/DD/HH/fileName**, to find all the files in these directories, the Apache Spark file source lists all subdirectories in parallel.

**File notification mode:**

In file notification mode, Auto Loader automatically sets up a notification service and queue service that subscribes to file events from the input directory. You can use file notifications to scale Auto Loader to ingest millions of files an hour. When compared to directory listing mode, file notification mode is more performant and scalable for large input directories or a high volume of files but requires additional cloud permissions.

You can switch between file notifications and directory listing at any time and still maintain exactly-once data processing guarantees.

**Required permissions for configuring file notification for ADLS Gen2 and Azure Blob Storage**

You must have read permissions for the input directory.To use file notification mode, you must provide authentication credentials for setting up and accessing the event notification services. In Databricks Runtime 8.1 and above, you only need a service principal for authentication. For Databricks Runtime 8.0 and below, you must provide both a service principal and a connection string.

Service principal - using Azure built-in roles

Create an Azure Active Directory app and service principal in the form of client ID and client secret.

**Assign this app the following roles to the storage account in which the input path resides:**

Contributor: This role is for setting up resources in your storage account, such as queues and event subscriptions.

Storage Queue Data Contributor: This role is for performing queue operations such as retrieving and deleting messages from the queues. This role is required in Databricks Runtime 8.1 and above only when you provide a service principal without a connection string.

**Assign this app the following role to the related resource group:**

EventGrid EventSubscription Contributor: This role is for performing event grid subscription operations such as creating or listing event subscriptions.

For more information, see Assign Azure roles using the Azure portal.

Service principal - using custom role

**How to use Databricks Autoloader:**

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

.option(<cloudFiles-option>, <option-value>)

.schema(<schema>)

.load(<input-path>)

df.writeStream.format("delta")

.option("checkpointLocation", <checkpoint-path>)

.trigger(<trigger>)

.start(<output-path>)

Here,

* **cloudFiles-option:** Autoloader Configuration option.
* **Schema:** The data schema of the file you provide.
* **Input-path & output-path:** The input path to the storage where the new files arrive and the output stream path respectively.
* **checkpointLocation:** Stream Checkpoint Location.
* **Trigger:** An optional parameter to trigger your stream.

**Configure your Databricks Autoloader:**

Common for both file detection methods i.e. the Directory Listing, and File Notification mode, you can configure your Databricks Autoloader using the following parameter:

**cloudFiles.allowOverwrites:** With the default value set as true, this decides whether to permit input directory file changes to overwrite existing data.

**cloudFiles.format:** It specifies the data coming from the source path. For example, it takes .json for JSON files, .csv for CSV Files, etc.

**cloudFiles.includeExistingFiles**: Set to true by default, this checks whether to include existing files in the Stream Processing Input Path or to only handle the new files arriving after initial setup.

**cloudFiles.inferColumnTypes**: With the initial value set as false, this checks whether to infer exact column types when leveraging schema inference.

**cloudFiles.maxBytesPerTrigger**: This sets the maximum number of bytes the Autoloader will process in every trigger.

**cloudFiles.maxFileAge:** Used to set the duration for which the event is tracked for deduplication purposes. This is commonly used when rapidly ingesting millions of files per hour.

**cloudFiles.resourceTags:** Key-value pairs to help identify the right resources.

**cloudFiles.schemaEvolutionMode:** Sets various modes for schema evolution i.e when new columns are detected in the data.

**cloudFiles.schemaHints**: This is the schema information of your data provided by you to the Autoloader.

**cloudFiles.schemaLocation**: This describes the location for storing the inferred schema along with associated changes.

**cloudFiles.validateOptions:** This checks if the Autoloader options discussed above so far are having a valid value or not.

**cloudFiles.backfillInterval:** 100% delivery of the file that has been uploaded is not guaranteed via the file notification mode. You can use the Backfills to ensure all the files get processed. This parameter decides the interval for triggering the Backfills.

**Ingest CSV, JSON, or Image Data with Databricks Autoloader:**

Dataloader allows efficient data ingestion for various file formats such as JSON, CSV, PARQUET, AVRO, TEXT, BINARYFILE, and ORC files.

To ingest the CSV data via the Autoloader, you can use the following code:

spark.readStream.format("cloudFiles")

.option("cloudFiles.format", "csv")

# The schema location directory keeps track of your data schema over time

.option("cloudFiles.schemaLocation", "<path-to-checkpoint>")

.load("<path-to-source-data>")

.writeStream

.option("mergeSchema", "true")

.option("checkpointLocation", "<path-to-checkpoint>")

.start("<path-to-target")

The above example has CSV for the cloudFiles.Format configuration parameter. You can set it to json for ingesting JSON files.