## CloudFiles in Databricks Best Practise & Learning’s

## Step-by-Step: How cloudFiles Works in Databricks

cloudFiles is the core engine behind Auto Loader. It enables fast, scalable, and incremental ingestion of files from cloud storage — without scanning the directory repeatedly.

Let’s walk through exactly how it works.

### 1. You Define the Source Path and Format

When you write:

python

CopyEdit

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

.load("/mnt/data/autoloader/incoming/")

You're telling Databricks:

“Watch this folder for **new files only**, in CSV format. Don’t rescan the entire folder each time — just tell me what’s new.”

This is very different from traditional .read.format("csv").load() which **scans all files** every time.

### 2. Auto Loader Detects New Files Using Cloud Events or Listing

CloudFiles supports two modes for discovering new files:

|  |  |
| --- | --- |
| **Mode** | **Description** |
| **Notification-based** (recommended) | Uses cloud-native services like S3 EventBridge, Azure Event Grid, or GCS Pub/Sub to detect new files quickly and cheaply |
| **Directory listing** | Scans the folder repeatedly, can be costly with many files |

If you don’t configure notifications, it falls back to listing — which still works well, but costs more at scale.

### 3. Spark Infers or Loads the Schema

When you use:

python

CopyEdit

.option("cloudFiles.schemaLocation", "/mnt/data/schema/people/")

You're telling Spark:

“Save the inferred schema here so you don’t have to re-infer it every time new files arrive.”

Without this, Spark will **try to infer schema repeatedly**, which is **slow and expensive** with millions of files.

### 4. cloudFiles Works with Checkpoints for State Management

CloudFiles integrates tightly with your checkpointLocation. This means:

* It remembers what files were already seen
* Avoids duplicates
* Supports reprocessing when needed

Even if the stream fails and restarts — it picks up from where it left off.

### 5. Supported Formats and File Handling

cloudFiles supports:

* csv, json, parquet, avro, orc, binaryFile
* Can handle nested folders
* Supports partition inference
* Works with structured streaming only (not static reads)

### Full Example

python

CopyEdit

df = (

spark.readStream

.format("cloudFiles")

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

.option("cloudFiles.inferColumnTypes", "true")

.option("cloudFiles.schemaLocation", "/mnt/data/schema/people/")

.load("/mnt/data/autoloader/incoming/")

)

This:

* Reads new CSV files only
* Infers schema once and caches it
* Tracks file discovery incrementally
* Avoids rescanning the folder

### Summary

* cloudFiles is the high-performance file discovery API used by Auto Loader
* It supports scalable, incremental ingestion of files from cloud storage
* Works with checkpointing to ensure fault tolerance
* Supports both notification-based and directory-based discovery

|  |  |
| --- | --- |
|  |  |
| **1. Without It, Everything Is a String**  By default, if you ingest a CSV file like:  bash  CopyEdit  id,name,age  1,Alice,30  2,Bob,27  Without setting cloudFiles.inferColumnTypes, Spark will treat all values as **strings**, like:  python  CopyEdit  StructType([  StructField("id", StringType),  StructField("name", StringType),  StructField("age", StringType)  ])  This might look fine, but it's **not great for downstream logic** — you lose numeric typing, sorting, filtering, and performance benefits. | **2. With cloudFiles.inferColumnTypes = true**  When you set:  python  CopyEdit  .option("cloudFiles.inferColumnTypes", "true")  You're telling Spark:  “Please scan the first few files and **guess the most appropriate data types** for each column.”  Now Spark will infer:  python  CopyEdit  StructType([  StructField("id", IntegerType),  StructField("name", StringType),  StructField("age", IntegerType)  ])  This results in:   * Cleaner schema * Better performance * Easier analytics downstream |
| **3. You Should Still Set cloudFiles.schemaLocation**  python  .option("cloudFiles.schemaLocation", "/mnt/schema/people/")  This tells Spark:  “Once you infer the schema, save it here — don’t redo the work for every file.”  This avoids re-scanning large datasets and keeps schema evolution under control.  **Example: Putting It Together**  python  CopyEdit  df = (  spark.readStream  .format("cloudFiles")  .option("cloudFiles.format", "csv")  .option("cloudFiles.inferColumnTypes", "true")  .option("header", "true")  .option("cloudFiles.schemaLocation", "/mnt/schema/people/")  .load("/mnt/data/autoloader/incoming/")  )   * cloudFiles.inferColumnTypes = true tells Spark to auto-detect the best data types. * Use it when working with CSVs or text files where schema isn’t predefined. * Always combine with cloudFiles.schemaLocation to avoid re-inference and improve performance | |

**CloudFiles Supported by Autoloader for data ingestion:**

**Real-Time Use Case: HR Employee Data Ingestion Scenario:**

**Files** like employee\_1.csv, employee\_2.csv, etc., land in /mnt/data/autoloader/incoming/

They contain **HR records:** id, FirstName, Department

Objective: Load them into a Delta Bronze table with proper tracking and transformation

**Setup Configuration**

source\_path = "/mnt/data/autoloader/incoming/"

schema\_path = "/mnt/data/autoloader/schema/people/"

checkpoint\_path = "/mnt/data/autoloader/checkpoints/people/"

target\_path = "/mnt/data/bronze/people/"

from pyspark.sql.functions import current\_timestamp, input\_file\_name, upper, col

from pyspark.sql.streaming import StreamingQueryListener

class AutoLoaderErrorListener(StreamingQueryListener):

def onQueryProgress(self, event):

print("Batch processed:", event.progress.batchId)

def onQueryTerminated(self, event):

if event.exception:

print("Stream failed:", event.exception)

spark.streams.addListener(AutoLoaderErrorListener())

try:

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

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

.option("cloudFiles.inferColumnTypes", "true") \

.option("cloudFiles.schemaLocation", schema\_path) \

.load(source\_path)

df\_transformed = df.withColumn("Department", upper(col("Department"))) \

.withColumn("ingestion\_timestamp", current\_timestamp()) \

.withColumn("source\_file", input\_file\_name())

df\_transformed.writeStream.format("delta") \

.option("checkpointLocation", checkpoint\_path) \

.outputMode("append") \

.start(target\_path)

except Exception as e:

print("Stream failed to start:", e)

Appendix:

## Common cloudFiles.format Options in Databricks

|  |  |
| --- | --- |
| **Format Name** | **Description** |
| "csv" | Ingests CSV files with optional headers and delimiter options |
| "json" | Reads JSON records; supports multiline and nested JSON |
| "parquet" | High-performance, columnar format — recommended for big data |
| "avro" | Row-based binary format often used for streaming and Kafka |
| "orc" | Optimized Row Columnar format — used often in Hive environments |
| "binaryFile" | Ingests binary data like images or PDFs as base64 — used for ML, document processing |
| "text" | Ingests plain text files (e.g., .txt, .log) as single-column rows |
| "delta" | For streaming from existing Delta tables (less common in Auto Loader scenarios, but supported via cloudFileCatalog)\*\* |

## Most Common .option("cloudFiles.\*") Settings

|  |  |
| --- | --- |
| Option | Description |
| cloudFiles.format | Required. Specifies the file format: csv, json, parquet, avro, orc, text, binaryFile |
| cloudFiles.schemaLocation | Required. Stores inferred or user-defined schema persistently for efficient re-use |
| cloudFiles.inferColumnTypes | Auto-detects data types from files (for CSV and JSON). Defaults to false |
| cloudFiles.includeExistingFiles | Ingests already-existing files when the stream starts. Set to true to include historical files |
| cloudFiles.useNotifications | Enables event-based discovery (via S3, ADLS Gen2, or GCS) instead of directory listing |
| cloudFiles.maxBytesPerTrigger | Limits total size (in bytes) processed per micro-batch |
| cloudFiles.maxFilesPerTrigger | Limits number of new files per micro-batch (default: unlimited) |
| cloudFiles.backfillInterval | Controls rate of discovery for old files when includeExistingFiles=true |
| cloudFiles.partitionColumns | Used to specify partition columns (especially for Hive-style partitions) |
| cloudFiles.resourceGroup | Azure-specific: used for Event Grid setup |
| cloudFiles.subscriptionId | Azure-specific: used for Event Grid setup |
| cloudFiles.connectionString | Azure-specific: connection to Event Grid topic for file notifications |
| cloudFiles.validateOptions | Validates all options at runtime. Default is true |

## Format-Specific Options

Some options also depend on the file type you're ingesting:

### For CSV:

* header → "true" if the file has headers
* delimiter → "," or "|", etc.
* quote, escape, nullValue, etc.

### For JSON:

* multiline → "true" for JSON lines with nested objects
* allowBackslashEscapingAnyCharacter

### For Parquet/Avro/ORC:

* Typically schema is embedded; fewer format-specific options needed

## Example Usage

python

CopyEdit

df = (

spark.readStream

.format("cloudFiles")

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

.option("cloudFiles.inferColumnTypes", "true")

.option("cloudFiles.schemaLocation", "/mnt/schema/logs/")

.option("cloudFiles.includeExistingFiles", "true")

.option("cloudFiles.maxFilesPerTrigger", "100")

.load("/mnt/data/logs/")

)