Accelerating Data Ingestion with Databricks Autoloader

Simon Whiteley

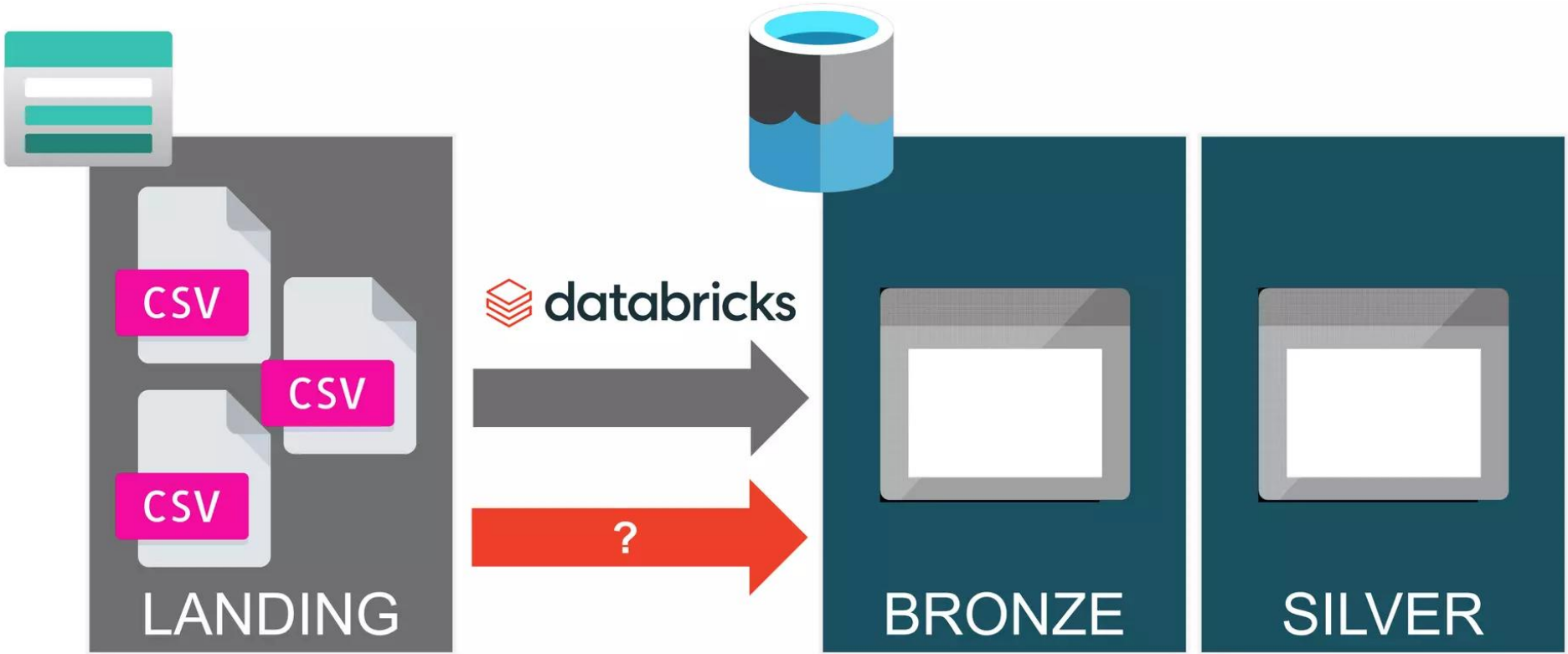
Director of Engineering, Advancing Analytics

Agenda

- Why Incremental is Hard
- Autoloader Components
- Implementation
- Evolution
- Lessons

Why Incremental is Hard

Incremental Ingestion



Incremental Ingestion

- Only Read New Files
- Don’t Miss Files
- Trigger Immediately
- Repeatable Pattern
- Fast over large directories

Existing Patterns – 1) ETL Metadata

{"lastRead":"2021/05/26"}

Contents:

- */2021/05/24/file 1*
- */2021/05/25/file 2*
- */2021/05/26/file 3*
- */2021/05/27/file 4*

.load(f"/{loadDate}/"

Existing Patterns – 2) Spark File Streaming

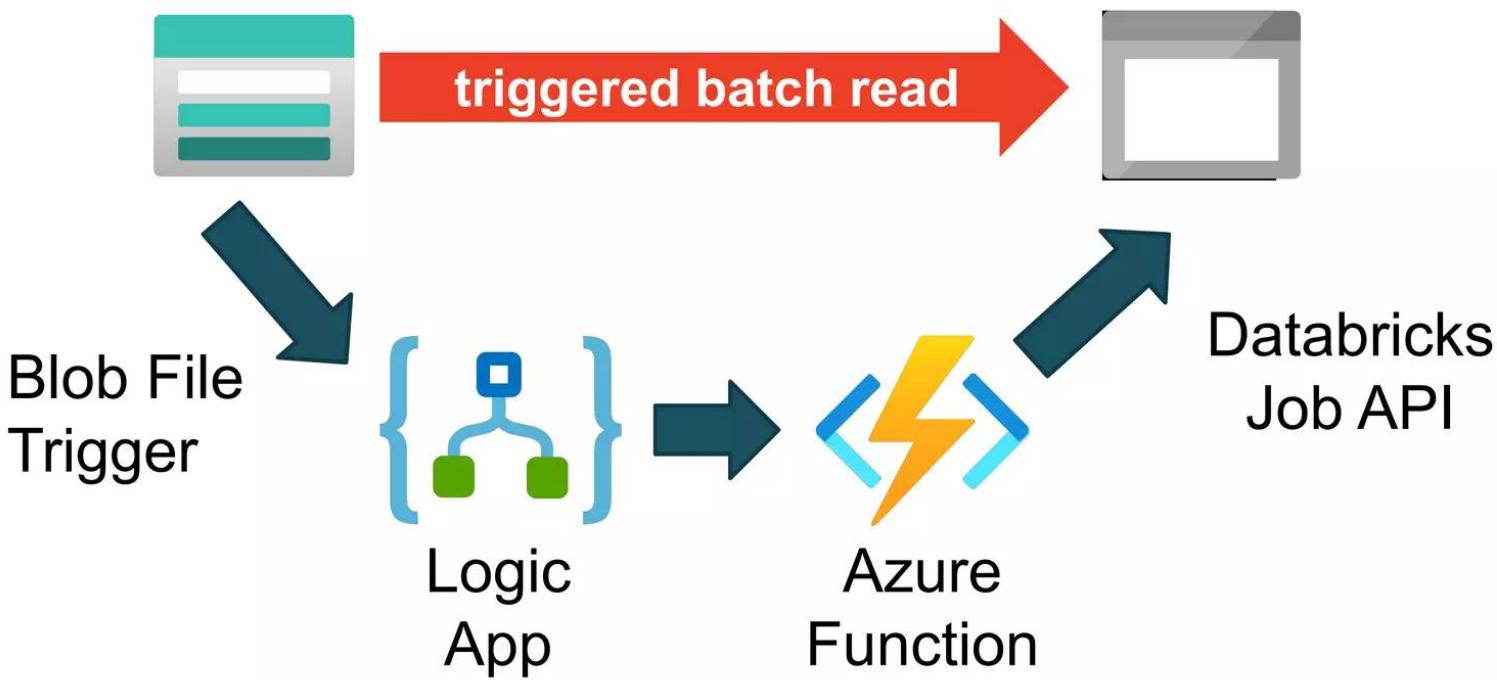
Contents:

- *File 1*
- *File 2*
- *File 3*
- *File 4*

Checkpoint:

- *File 1*
- *File 2*
- *File 3*

Existing Patterns – 3) DIY



Incremental Ingestion Approaches

Approach	Good At	Bad At
Metadata ETL	**Repeatable**	**Not immediate, requires polling**
File Streaming	**Repeatable** **Immediate**	**Slows down over large directories**
DIY Architecture	**Immediate Triggering**	**Not Repeatable**

Databricks Autoloader

Auto Loader is an optimized cloud file source for Apache Spark that loads data continuously and efficiently from cloud storage as new data arrives.

Prakash Chockalingam
Databricks Engineering Blog

What is Autoloader?

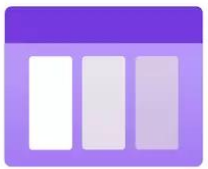
Essentially, Autoloader combines our three approaches of:

- Storing **Metadata** about what has been read
- Using **Structured Streaming** for immediate processing
- Utilising **Cloud-Native Components** to optimise identifying arriving files

There are two parts to the Autoloader job:

- **CloudFiles** DataReader
- **CloudNotification** Services (optional)

Cloudfiles Reader



Blob Storage Queue

{"fileAdded":"/landing/file 4.json"

Blob Storage

- *File 1.json*
- *File 2.json*
- *File 3.json*
- *File 4.json*

Dataframe

Check Files in Queue

Read specific files from source

CloudFiles DataReader

```
df = ( spark
    .readStream
    .format("cloudfiles")
    .option("cloudfiles.format","json")
    .option("cloudfiles.useNotifications","true")
    .schema(mySchema)
    .load("/mnt/landing/")
    )
```

Tells Spark to use Autoloader

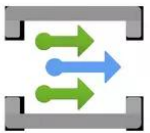
Tells Autoloader to expect JSON files

Should Autoloader use the Notification Queue

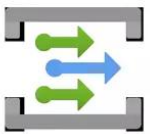
Cloud Notification Services - Azure

Blob Storage

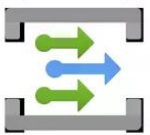
Event Grid Topic



Event Grid Subscription



Event Grid Subscription



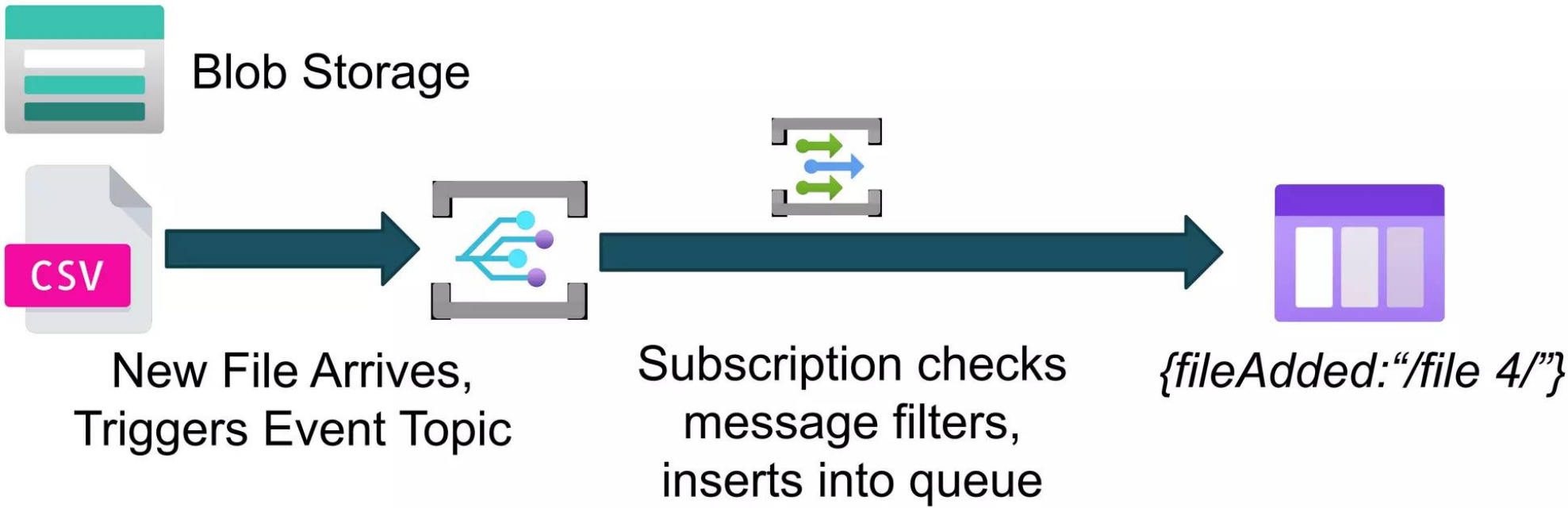
Event Grid Subscription

Blob Storage Queue

Blob Storage Queue

Blob Storage Queue

Cloud Notification Services - Azure



NotificationServices Config

cloudFiles

.useNotifications – *Directory Listing VS Notification Queue*

.queueName – *Use an Existing Queue*

.connectionString – *Queue Storage Connection*

.subscriptionId

.resourceGroup

.tenantId

.clientId

.clientSecret

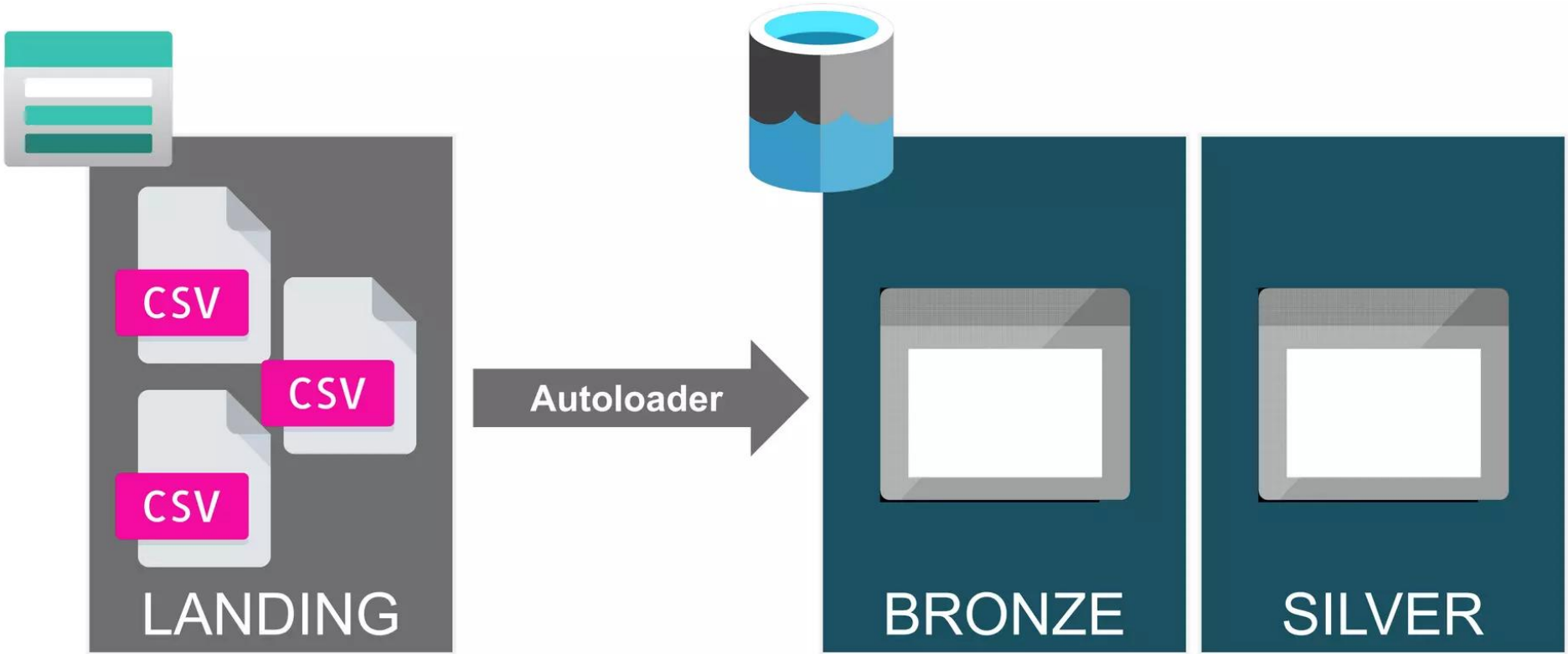
Service Principal for Queue Creation

Implementing Autoloader

- Setup Steps
- Reading New Files
- A Basic ETL Setup

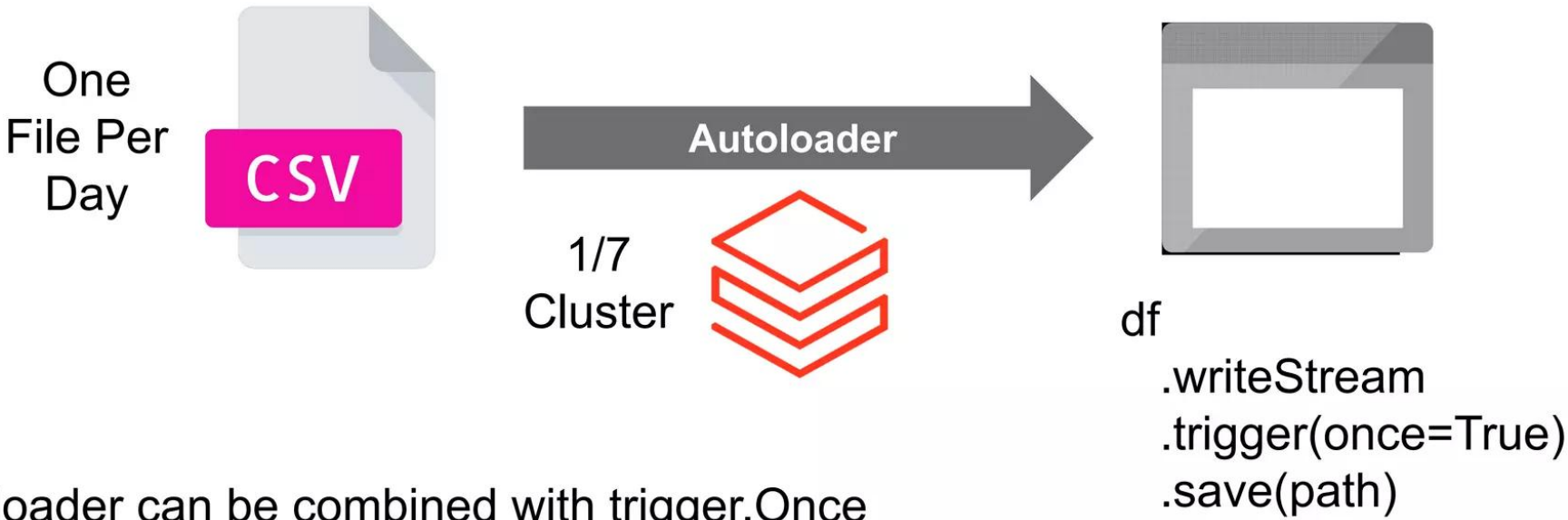
Delta Implementation

Practical Implementations



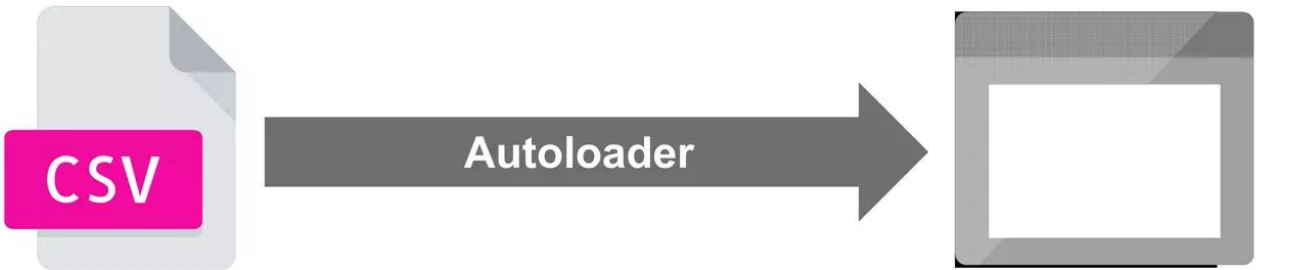
Low Frequency Streams

Low Frequency Streams



Autoloader can be combined with trigger.Once – each run finds only files not processed since last run

Delta Merge



Merge?

Delta Merge

```
df
.writeStream
.foreachBatch(runThis)
.save(path)
```

```
def runThis(df, batchId):
  (df
    .write
    .save(path)
  )
```

Delta Implementation

- Batch ETL Pattern
- Merge Statements
- Logging State

Evolving Schemas

New Features since Databricks Runtime 8.2

What is Schema Evolution?

{“ID”:1,“ProductName”:“Belt”}

{“ID”:2,“ProductName”:“T-Shirt”,”Size”:”XL”}

{“ID”:3,“ProductName”:“Shirt”,“Size”:“14”,
“Care”:{ “DryClean”: “Yes”,
“MachineWash”:“Don’t you dare”
}
}

How do we handle Evolution?

1. Fail the Stream
2. Manually Intervene
3. Automatically Evolve

In order to manage schema evolution, we need to know:

- What the schema is expected to be
- What the schema is now
- How we want to handle any changes in schema

Schema Inference

In Databricks 8.2 Onwards – simply don't provide a Schema to enable Schema Inference. This infers the schema once when the stream is started and stores it as metadata.

cloudfiles

- .**schemaLocation** – where to store the schema
- .**inferColumnTypes** – sample data to infer types
- .**schemaHints** – manually specify data types for certain columns

Schema Metastore

{“ID”:1,
“ProductName”:“Belt”}

On First Read

0

_schemas

```
{
    "type": "struct",
    "fields": [
        {
            "name": "ID",
            "type": “string",
            "nullable": true,
            "metadata": {}
        },
        {
            "name": "ProductName",
            "type": “string",
            "nullable": true,
            "metadata": {}
        }
    ]
}
```

Schema Metastore – DataType Inference

{"ID":1,
"ProductName":"Belt"}

_schemas

On First Read

0

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

```
{
    "type": "struct",
    "fields": [
        {
            "name": "ID",
            "type": "int",
            "nullable": true,
            "metadata": {}
        },
        {
            "name": "ProductName",
            "type": "string",
            "nullable": true,
            "metadata": {}
        }
    ]
}
```

Schema Metastore – Schema Hints

{“ID”:1,
“ProductName”:“Belt”}

_schemas

0

.option(“cloudFiles.schemaHints”,”ID long”)

```
{
    "type": "struct",
    "fields": [
        {
            "name": "ID",
            "type": “long",
            "nullable": true,
            "metadata": {}
        },
        {
            "name": "ProductName",
            "type": “string",
            "nullable": true,
            "metadata": {}
        }
    ]
}
```

Schema Evolution

To allow for schema evolution, we can include a schema evolution mode option:

cloudFiles.schemaEvolutionMode

- **addNewColumns** – Fail the job, update the schema metastore
- **failOnNewColumns** – Fail the job, no updates made
- **rescue** – Do not fail, pull all unexpected data into *_rescued_data*

Evolution Reminder

{“ID”:1,“ProductName”:“Belt”}

{“ID”:2,“ProductName”:“T-Shirt”,”Size”:”XL”}

{“ID”:3,“ProductName”:“Shirt”,“Size”:“14”,
“Care”:{ “DryClean”: “Yes”,
“MachineWash”:“Don’t you dare”
}
}

Schema Evolution - Rescue

1
JSON

ID	Product Name	_rescued_data
1	Belt	

2
JSON

ID	Product Name	_rescued_data
2	T-Shirt	{“Size”:”XL”}

3
JSON

ID	Product Name	_rescued_data
3	Shirt	{“Size”:”14”,”Care”:{*“DryC…*

Schema Evolution – Add New Columns

{"ID":2,
"ProductName":"T-Shirt",
"Size":"XL"}

_schemas

0

1

On Arrival

```
{
    "type": "struct",
    "fields": [
        {
            "name": "ID",
            "type": "string",
        },
        {
            "name": "ProductName",
            "type": "string",
        } ,
        {
            "name": "Size",
            "type": "string",
        }...
```

Schema Evolution

- Inference & The Schema Metastore
- Schema Hints
- Schema Evolution

Lessons from an Autoloader Life

Autoloader Lessons

- EventGrid Quotas & Settings
- Streaming Best Practices
- Batching Best Practices

Quotas and limits

The following limits apply to Azure Event Grid **topics** (system, custom, and partner topics).

Resource	Limit
Custom topics per Azure subscription	100
Event subscriptions per topic	500
Publish rate for a custom or a partner topic (ingress)	5,000 events/sec or 5 MB/sec (whichever is met first)

EventGrid Quota Lessons

- You can have 500 files from a single storage account using the system topic
- Deleting checkpoint will reset the stream ID and create a new Subscription/Queue, leaving an orphan set
- Use the CloudNotification Libraries to manage this more closely with custom topics

Streaming Optimisation

- MaxBytesPerTrigger / MaxFilesPerTrigger

  Manages the size of the streaming microbatch

- FetchParallelism

  Manages the workload on your queue

Batch Lessons – Look for Lost Messages

STORAGE QUEUE MESSAGE TTL

Define the time to live that the message will have once it is delivered to the storage queues. For more information consult the Azure Storage Queue documentation.

☐ Never expires

	Days	Hours	Minutes	Seconds
Storage queue message time to live				

Default 7 days!

DEAD-LETTERING

Save events that cannot be delivered to storage. Learn more

☐ Enable dead-lettering

Databricks Autoloader

- Reduces complexity of ingesting files
- Has some quirks in implementing ETL processes
- Growing number of schema evolution features

ADVANCING ANALYTICS

Microsoft Partner

Microsoft

Gold Cloud Platform
Gold Data Platform
Gold Data Analytics
Silver Application Integration

databricks partner

Simon Whiteley
Director of Engineering

hello@advancinganalytics.co.uk

@MrSiWhiteley

www.youtube.com/c/AdvancingAnalytics

MVP Microsoft® Most Valuable Professional

Feedback

Your feedback is important to us.

Don’t forget to rate and review the sessions.

DATA+AI SUMMIT 2021

FORMERLY SPARK+AI SUMMIT