



# Working within the Data Lake

## With AWS Glue

Justin Yenser

Solutions Architect  
Amazon Web Services

# Agenda

Optimizing for Cost and Performance

AWS Glue Components:

Data Catalog

Job Authoring

Job Execution

Job Workflow

# Optimizing for Cost and Performance

# Optimizing for Cost and Performance

```
/user/hive/warehouse/logs
├── dt=2001-01-01/
│   ├── country=GB/
│   │   ├── file1
│   │   └── file2
│   └── country=US/
│       └── file3
└── dt=2001-01-02/
    ├── country=GB/
    │   ├── file4
    │   └── country=US/
    │       ├── file5
    │       └── file6
```

## Partitioning

Pay for data your query **needs**,  
not to scan **all** of your data



## Compression

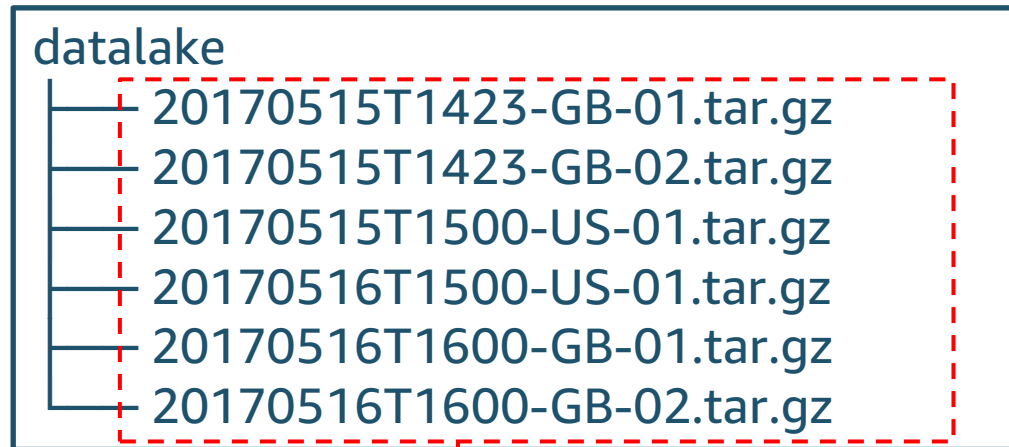
Pay for what you **store**,  
not for what you **process**



## Managed Services

Pay for what you **use**,  
not for what you **run**

# Partitioning



where dt=20170515 and  
country=US



select \* from datalake where  
**dt=20170515** and **country=US**

# Partitioning

	select count(*) from datalake where dt='20170515'		select count(*) from datalake where dt >= '20170515' and dt < '20170516'	
	Non-Partitioned	Partitioned	Non-Partitioned	Partitioned
Run Time	9.71 sec	2.16 sec	10.41 sec	2.73 sec
Data Scanned	74.1 GB	29.06 MB	74.1 GB	871.39 MB
Cost	\$0.36	\$0.0001	\$0.36	\$0.004
Results	77% faster, 99% cheaper		73% faster, 98% cheaper	

<https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-tips-for-amazon-athena/>



# Compression

- Compressing your data can speed up your queries significantly
- Split-able formats enable parallel processing across nodes

Algorithm	Splittable	Compression Ratio	Algorithm Speed	Good For
Gzip (DEFLATE)	No	High	Medium	Raw Storage
bzip2	Yes	Very High	Slow	Very Large Files
LZO	Yes	Low	Fast	Slow Analytics
Snappy	Yes and No *	Low	Very Fast	Slow & Fast Analytics

\* Depends on if the source format is splittable and can output each record into a Snappy Block



# Compression - Example

Snappy Compression with Parquet File Format

Format	Size on S3	Run Time	Data Scanned	Cost
Text	1.15 TB	3m 56s	1.15 TB	\$5.75
Parquet	130 GB	6.78s	2.51 GB	\$0.013
Result	87% less	34x faster	99% less	99.7% savings



# Split content

Fewer, larger files are better than many, smaller files (when split-able)

- Faster Listing Operations
- Fewer Requests to Amazon S3
- Less Metadata to Manage
- Faster Query Performance

Format	Size on S3	Run Time
select count(*) from datalake	5000 files	8.4 sec
select count(*) from datalake	1 file	2.31 sec
Result		72% Faster



# Why managed service for ETL ?

# Transforming Data

Over 90% of ETL jobs in the cloud are hand-coded

Which is good ...

- Flexible
- Powerful
- Unit Tests
- CI/CD
- Developer Tools ...

... but also bad!

- Brittle
- Error-Prone
- Laborious
- Sources change
- Schemas change
- Volume changes
- EVERYTHING KEEPS CHANGING !!!

# Glue Flex

Execution option for AWS Glue that allows customers to reduce the costs by up to 35%



## Standard execution-class

10x faster job start times  
Predictable job latencies

Enables micro-batching  
Latency-sensitive workloads

New!



## Flex execution-class

Up to  
35% cost savings (\$0.29/DPU)

Cost effective for non-time  
sensitive workloads

# AWS Glue Overview

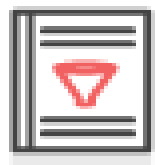


# AWS Glue

**Discover** Automatically **discover and categorize** your data making it immediately searchable and queryable across data sources

**Develop** Visually build or utilize auto-generated code to **integrate data** from multiple sources to **clean, enrich, and reliably move data** from sources to targets.

**Deploy** Run your jobs on a **serverless, fully managed, scale-out environment**. No compute resources to provision or manage.



**Data Catalog**



**Job Authoring**



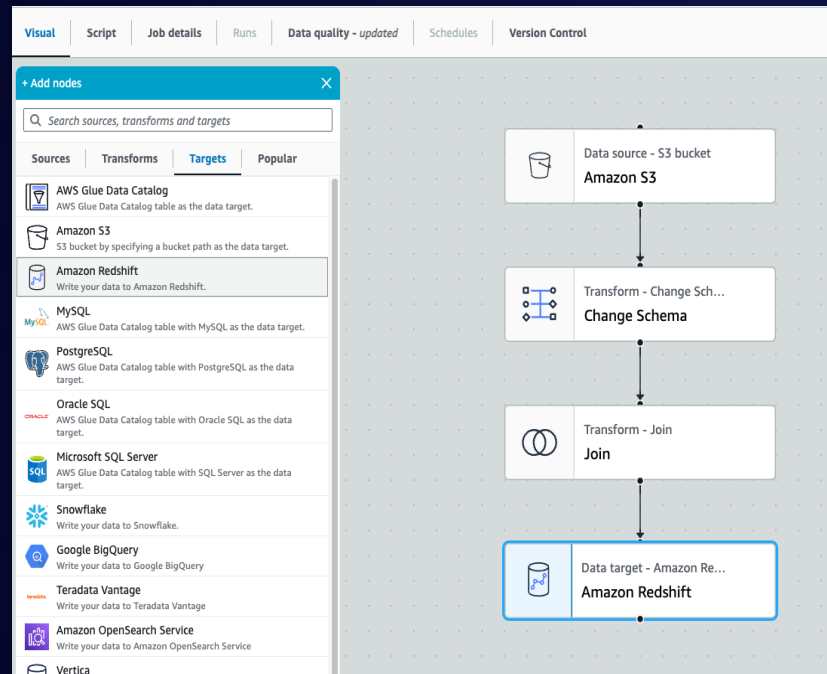
**Job Execution**



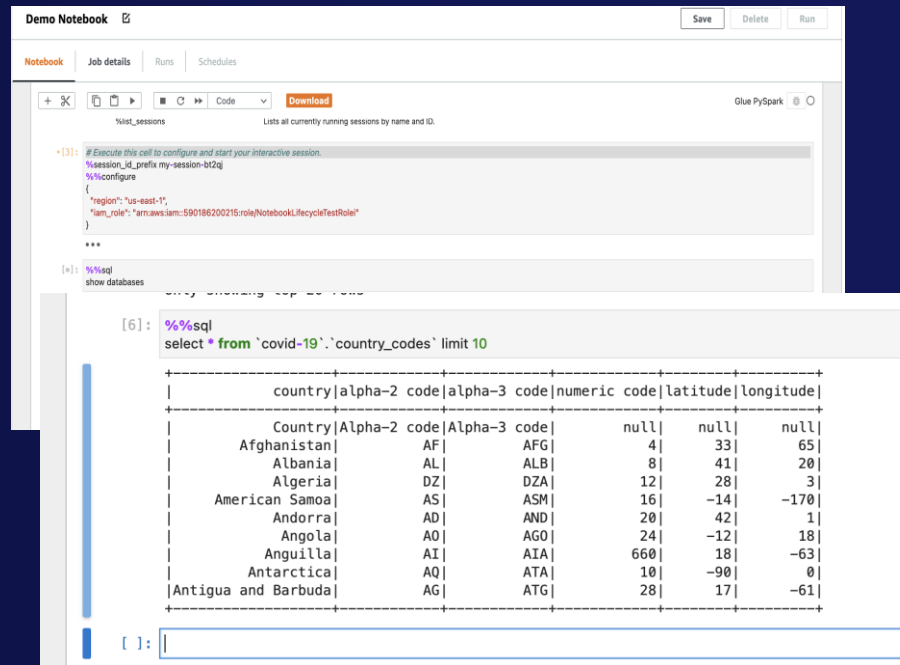
**Job Workflow**

# Built for multiple personas

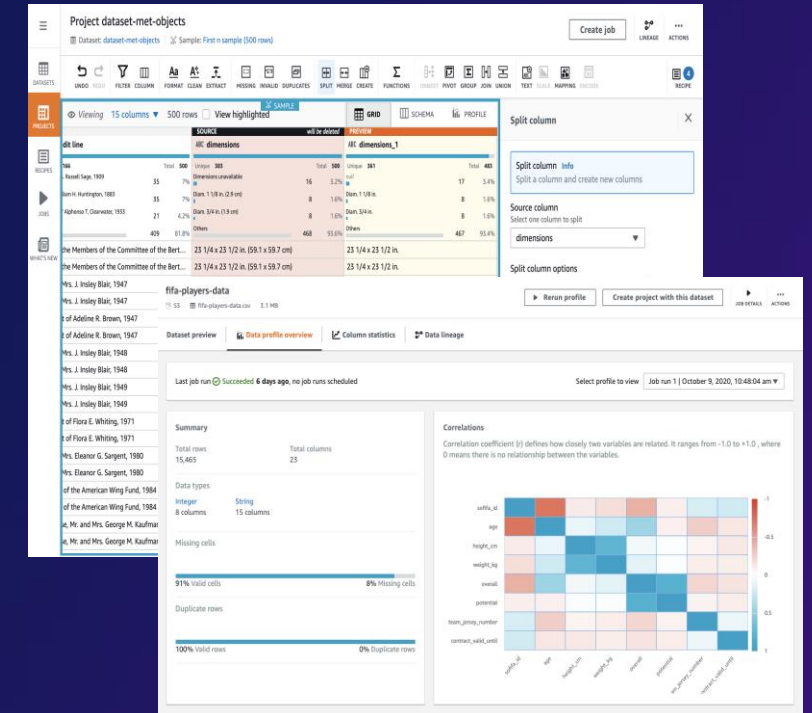
Customers migrate to promote self service



**Glue Studio** for Data Engineers who prefer visual low-code development experience



**Glue Notebooks & interactive sessions** for Data Engineers who prefer code based experience



**Glue Databrew** for Analysts who prefer a no-code experience

# AWS Glue Data Catalog

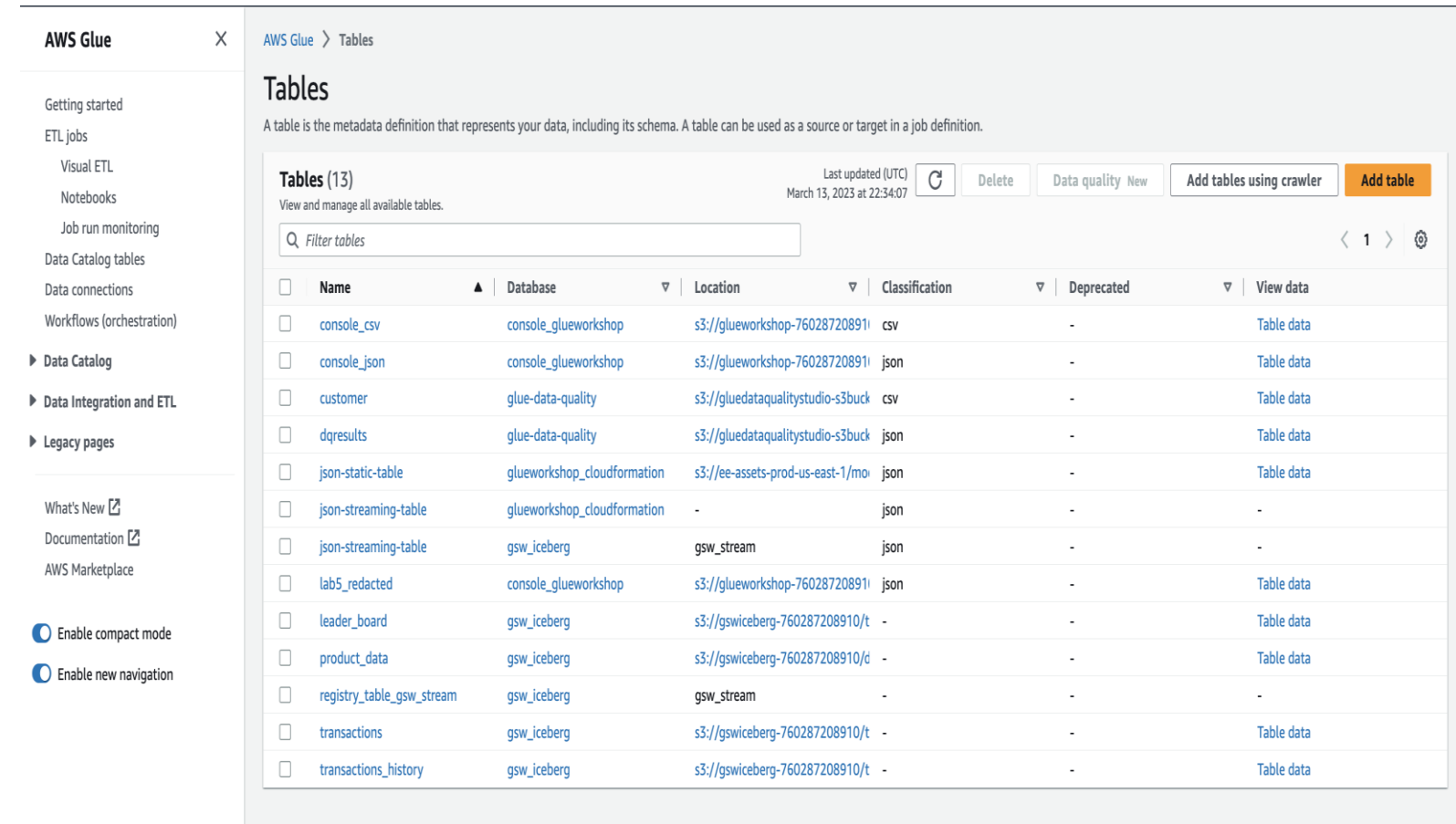




# AWS Glue : Data Catalog

Features include:

- No movement of data- **Low Costs/Admin**
- **Increased Productivity**- All metadata centrally available for search and query, no movement of data
- **Connections** to S3, RDS, Redshift, JDBC, DynamoDB and many more
- **Versioning** of table metadata as schemas

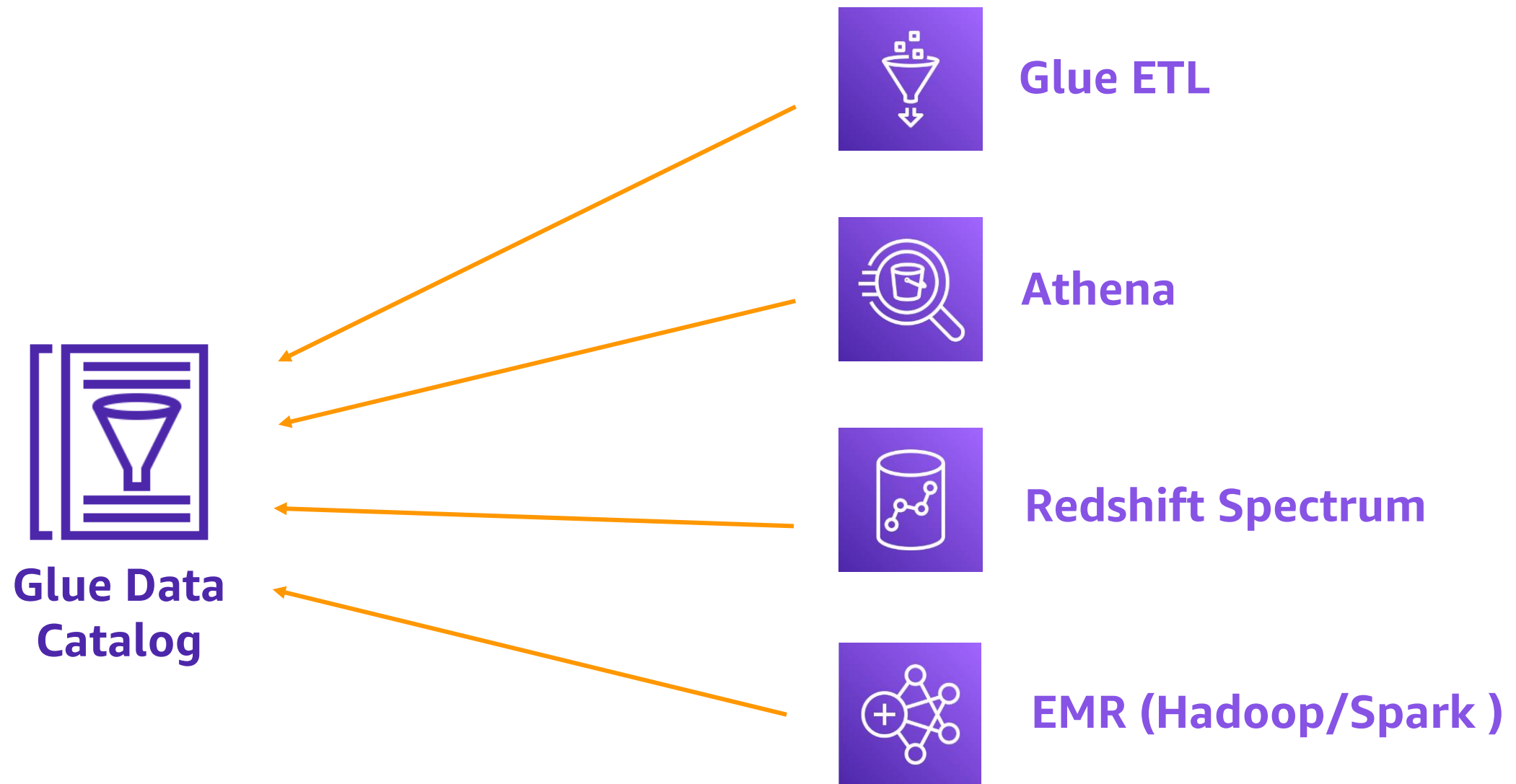


The screenshot displays the AWS Glue Data Catalog console. On the left is a navigation sidebar with options like 'Getting started', 'ETL jobs', 'Visual ETL', 'Notebooks', 'Job run monitoring', 'Data Catalog tables', 'Data connections', 'Workflows (orchestration)', 'Data Catalog', 'Data Integration and ETL', and 'Legacy pages'. The main panel is titled 'Tables' and shows a list of 13 tables. A search bar and filters are at the top. The table list includes columns for Name, Database, Location, Classification, and Deprecated status. Each row has a 'View data' link.

Name	Database	Location	Classification	Deprecated	View data
console_csv	console_glueworkshop	s3://glueworkshop-760287208910	csv	-	Table data
console_json	console_glueworkshop	s3://glueworkshop-760287208910	json	-	Table data
customer	glue-data-quality	s3://gluedataqualitystudio-s3buck	csv	-	Table data
dqresults	glue-data-quality	s3://gluedataqualitystudio-s3buck	json	-	Table data
json-static-table	glueworkshop_cloudformation	s3://ee-assets-prod-us-east-1/mo	json	-	Table data
json-streaming-table	glueworkshop_cloudformation	-	json	-	-
json-streaming-table	gsw_iceberg	gsw_stream	json	-	-
lab5_redacted	console_glueworkshop	s3://glueworkshop-760287208910	json	-	Table data
leader_board	gsw_iceberg	s3://gswiceberg-760287208910/t	-	-	Table data
product_data	gsw_iceberg	s3://gswiceberg-760287208910/d	-	-	Table data
registry_table_gsw_stream	gsw_iceberg	gsw_stream	-	-	-
transactions	gsw_iceberg	s3://gswiceberg-760287208910/t	-	-	Table data
transactions_history	gsw_iceberg	s3://gswiceberg-760287208910/t	-	-	Table data



# AWS Glue : Data Catalog – Queryable by many services



# AWS Glue : Data Catalog - Crawlers

## Features Include:

- Built-in classifiers
  - Detect file type
  - Extract schema
  - Identify partitions
- On-Demand or Scheduled Execution
- Build-your-own classifiers
  - Grok for ease of use

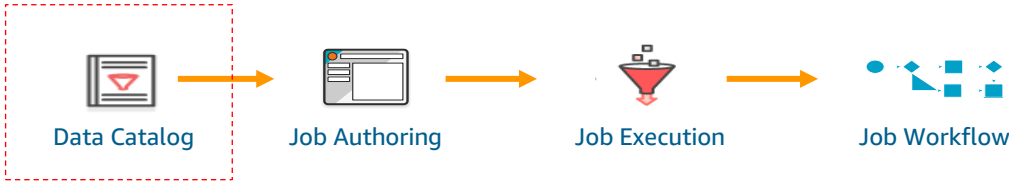
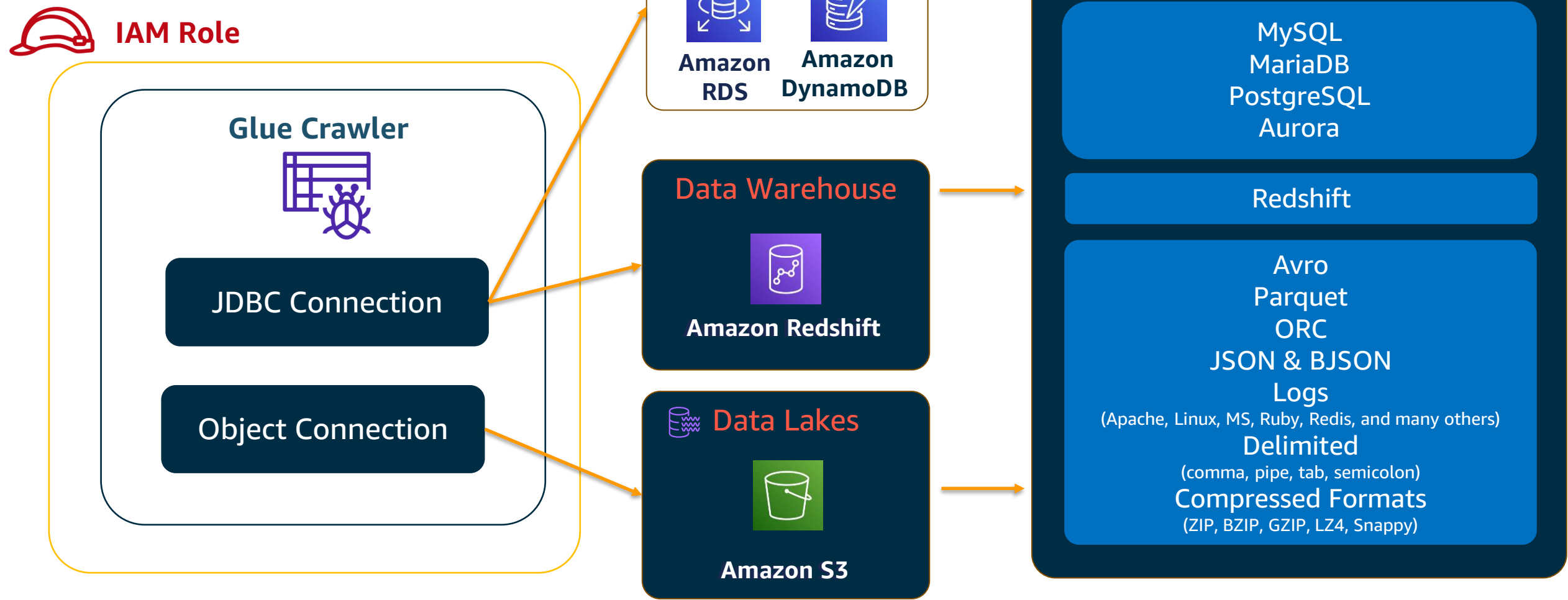
The screenshot displays the AWS Glue console interface. On the left is a navigation sidebar with categories like Data catalog, ETL, Security, and Tutorials. The 'Crawlers' link is highlighted. The main panel shows a table of crawlers with columns for Name, Schedule, Status, Logs, Last runtime, Median runtime, Tables updated, and Tables added. Two crawlers are listed: 'Crawler' and 'demo-crawler', both with a status of 'Ready'. Above the table are buttons for 'Add crawler', 'Run crawler', and 'Action', along with a search filter and pagination controls.

	Name	Schedule	Status	Logs	Last runtime	Median runtime	Tables updated	Tables added
<input type="checkbox"/>	Crawler		Ready		0 secs	0 secs	0	0
<input type="checkbox"/>	demo-crawler		Ready		0 secs	0 secs	0	0



# AWS Glue: Crawlers – Classifiers

Build your own custom classifiers with Grok!



# AWS Glue : Data Catalog – Detecting Schema and Partitions

S3 bucket hierarchy

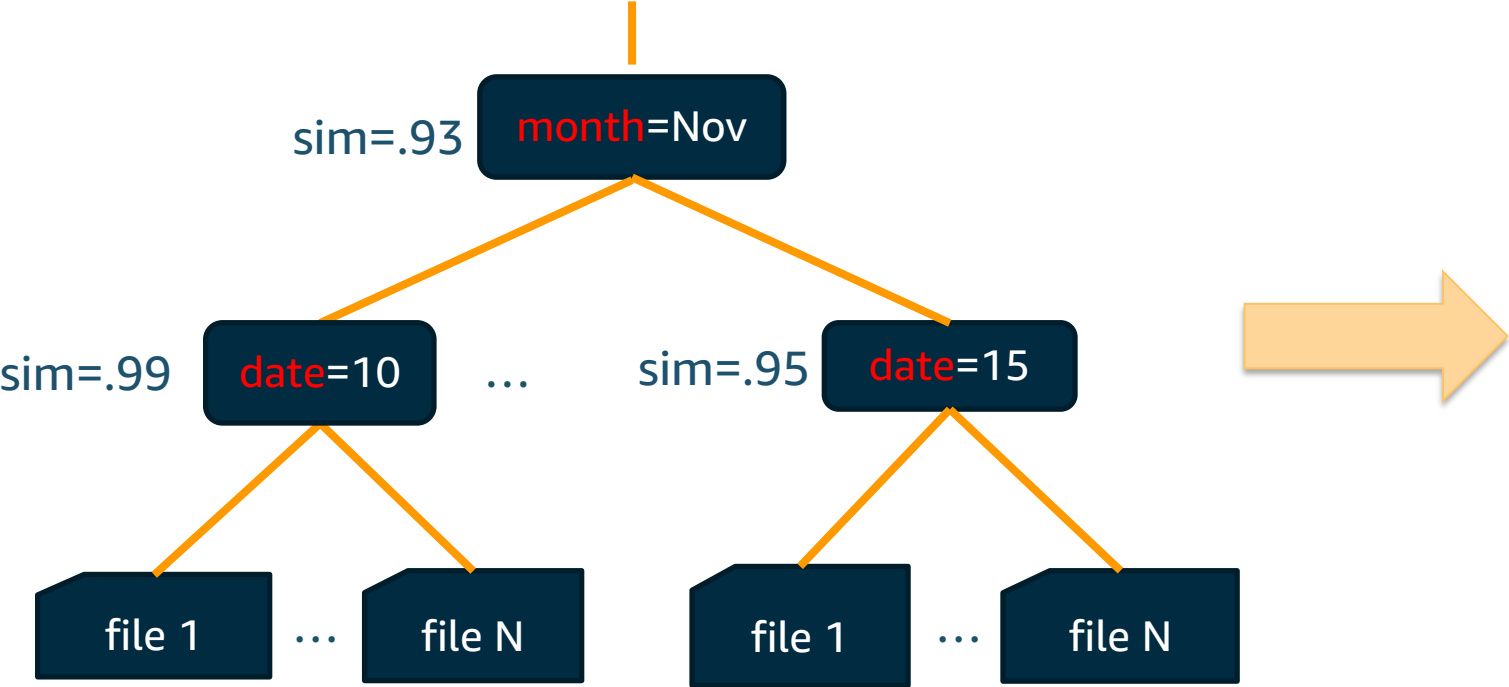
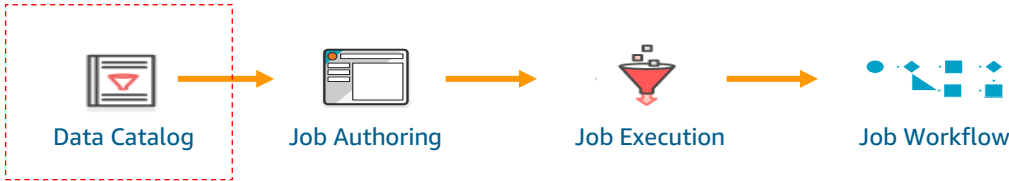


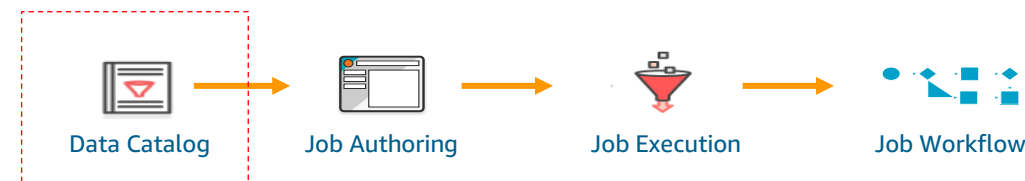
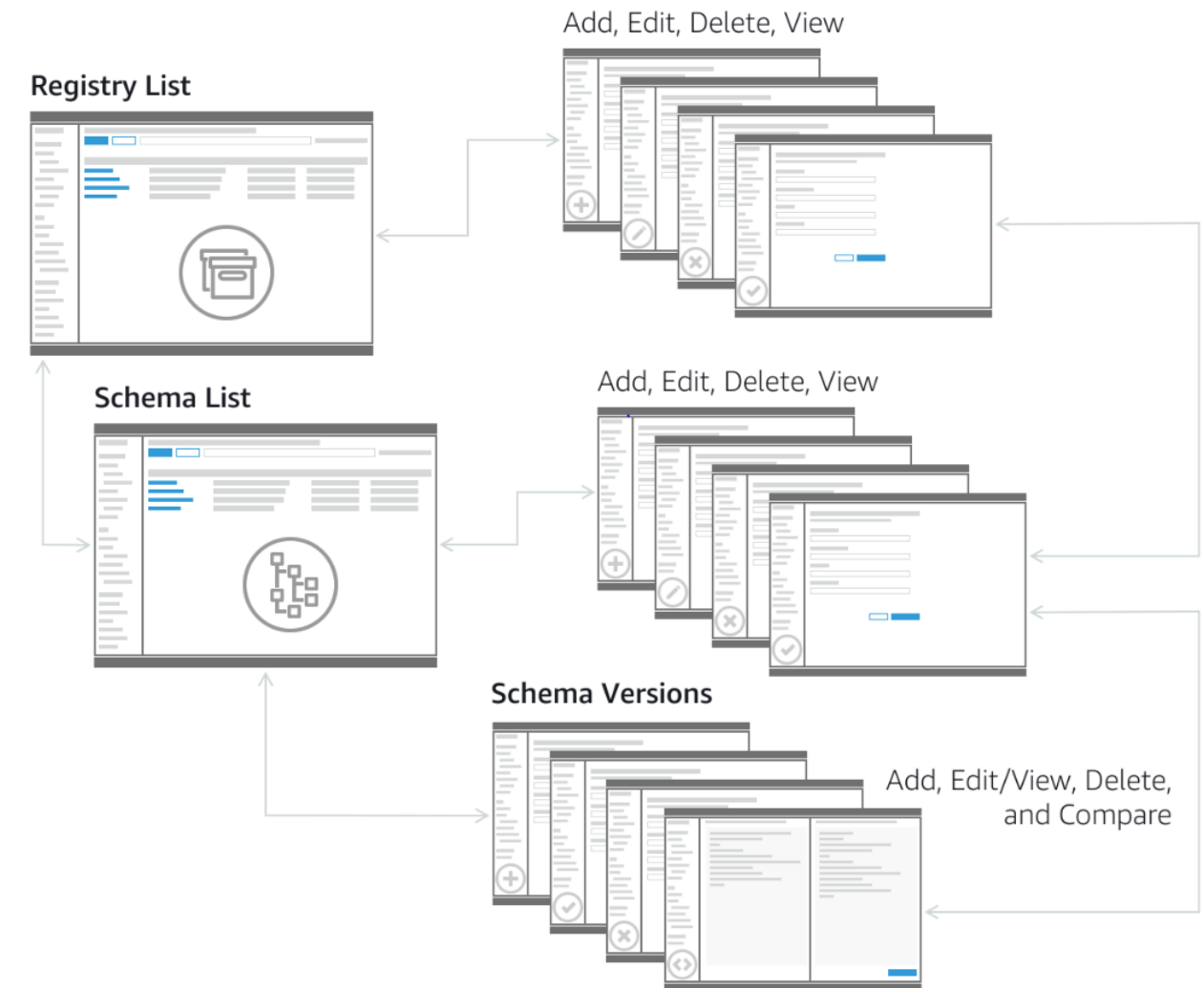
Table definition

Column	Type
month	str
date	str
col 1	int
col 2	float
⋮	⋮



# AWS Glue : Schema Registry for Streaming Data

- Improve data quality for your data streaming applications.
- **Enforce schemas and schema evolution** to prevent downstream application failures
- Easily integrates with Amazon Managed Streaming for Apache Kafka (MSK), Amazon Kinesis Data Streams, and Amazon Managed Service for Apache Flink for convenient setup
- Use provided open source libraries to compress data and save on storage and data transfer costs



# AWS Glue: Job Authoring

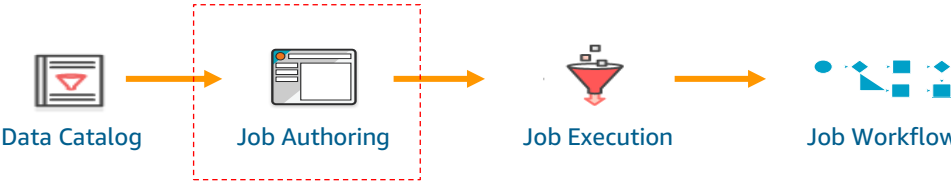
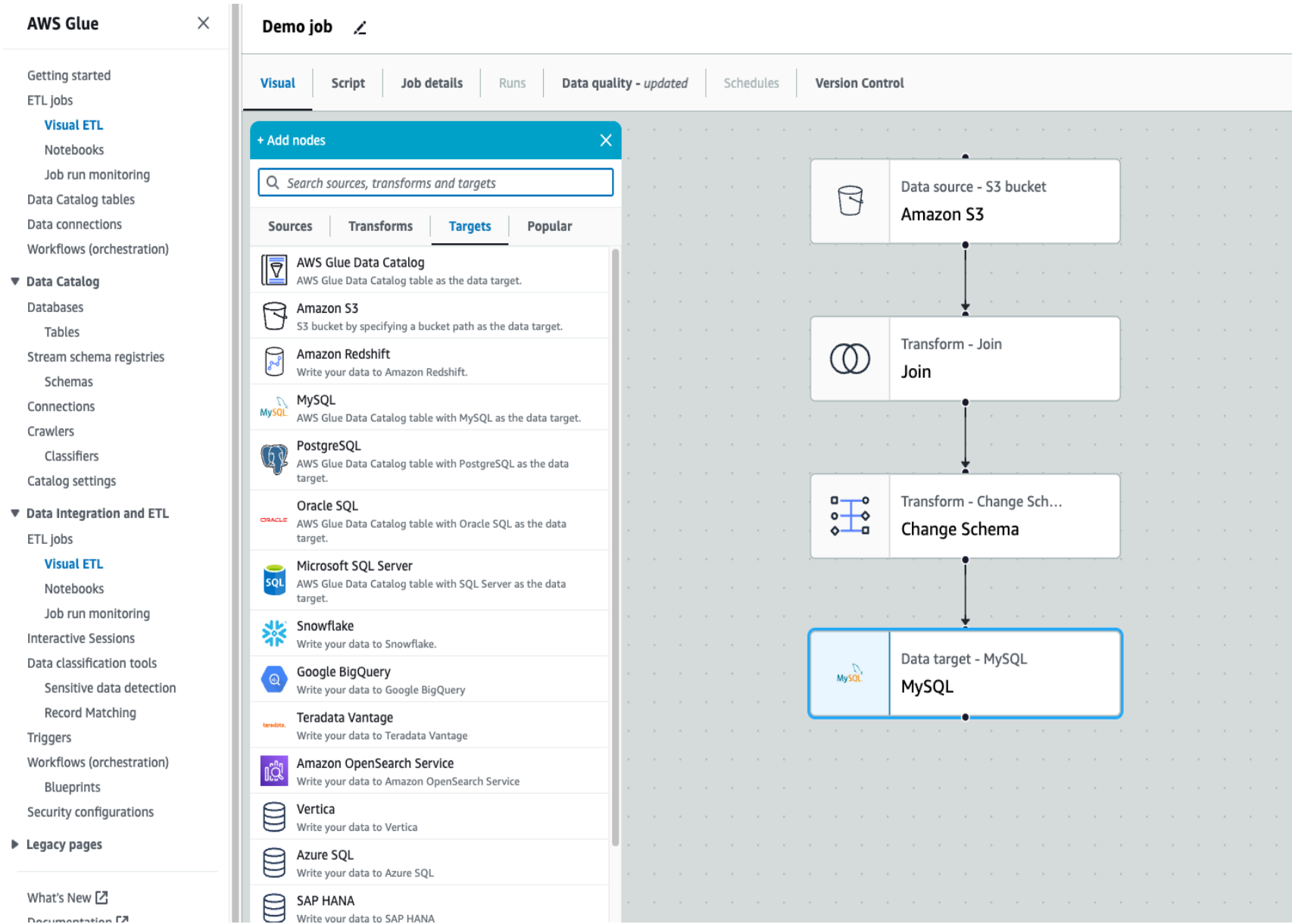
# AWS Glue : Glue Studio

Visual authoring of ETL jobs  
without writing code

Monitor thousands of jobs through  
a single pane of glass

Distributed processing without  
the learning curve

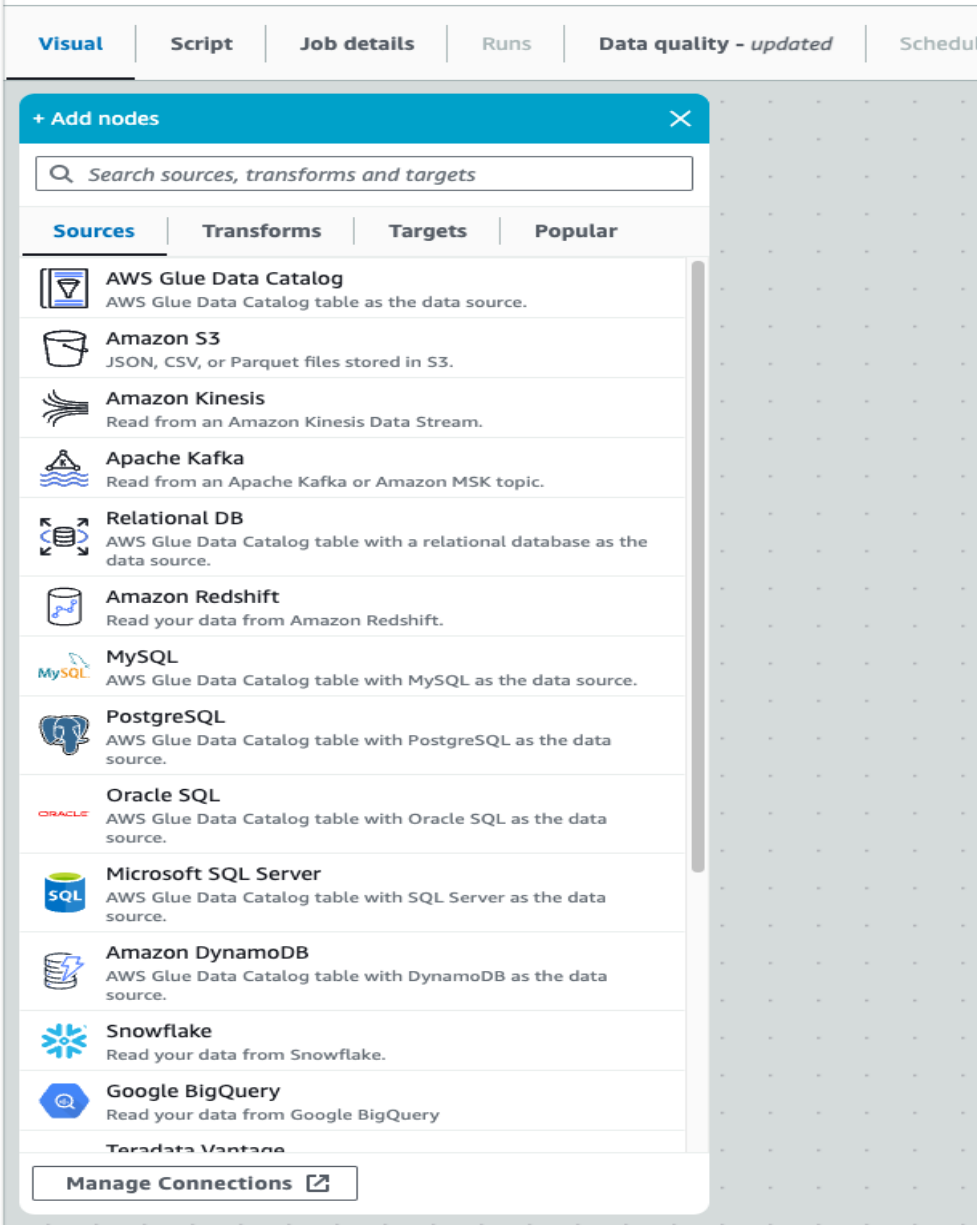
Advanced transforms  
though code snippets














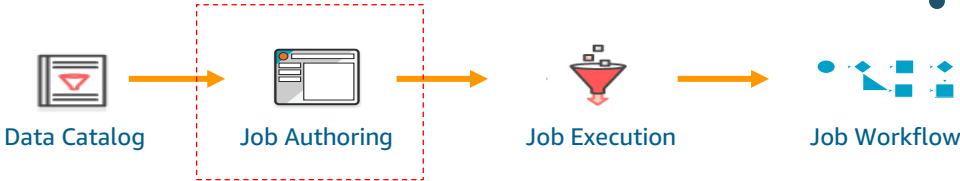
# Glue : Authoring – Sources

## Choose from Manually-Defined or Crawler-Generated Sources

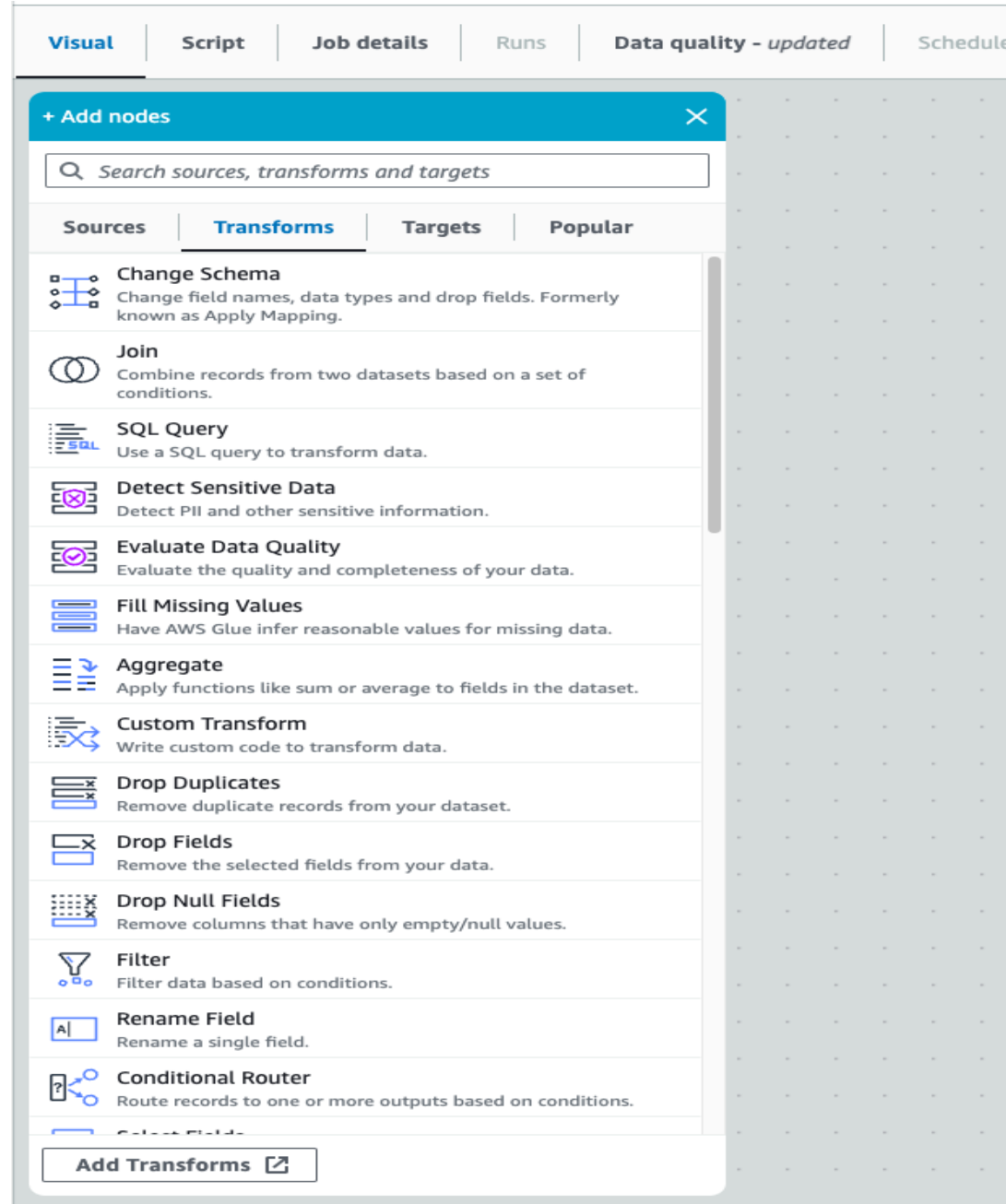


-  **S3 Bucket**
-  **RDBMS**
-  **Redshift**
-  **RDS**
-  **DocumentDB**
-  **MongoDB**
-  **DynamoDB**
-  **Kinesis**
-  **MSK**

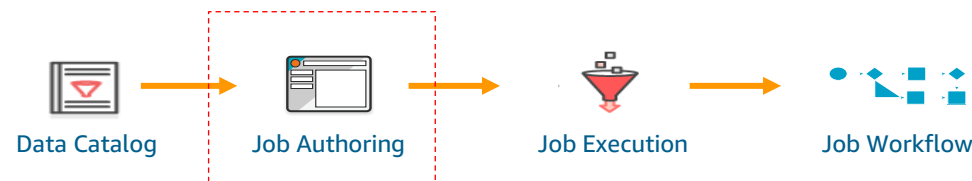
• And more



# AWS Glue : Authoring – Transformations

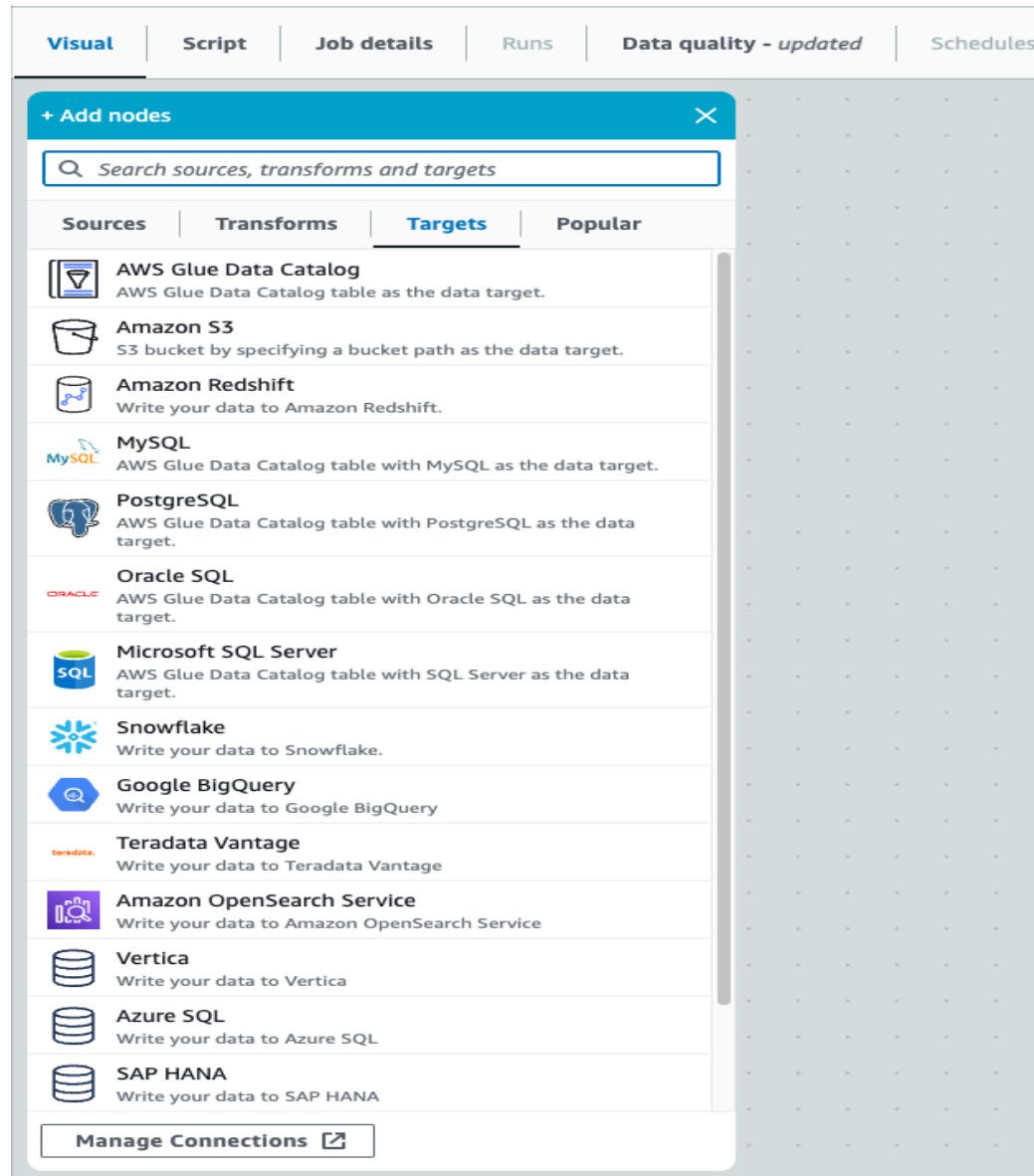


- **Pre-Built Transformation** click and add to your job with simple configuration
- **Spigot** writes sample data from DynamicFrame to S3 in JSON format
- **Expanding** with more transformations to come



# AWS Glue : Authoring – Targets

Crawler-Defined or Manually-Created



 **S3 Bucket**

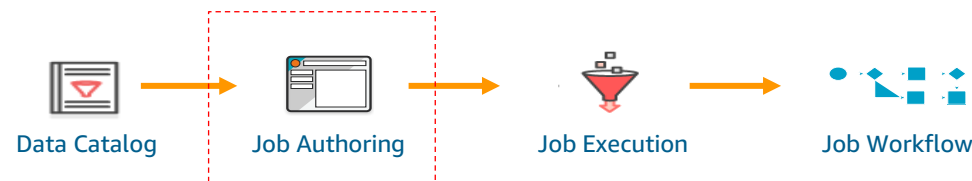
 **RDBMS**

 **Redshift**

 **Document DB**

 **RDS**

• **And more**



# AWS Glue : Authoring – Apply Mapping Transform

Visual

Script

Job details

Runs

Data quality New

Schedules

Version Control

Source

Action

Target

Undo

Redo

Remove

Data source - S3 bucket  
Amazon S3

Transform - ApplyMapping  
Change Schema

Data target - S3 bucket  
Amazon S3

Node properties

Transform

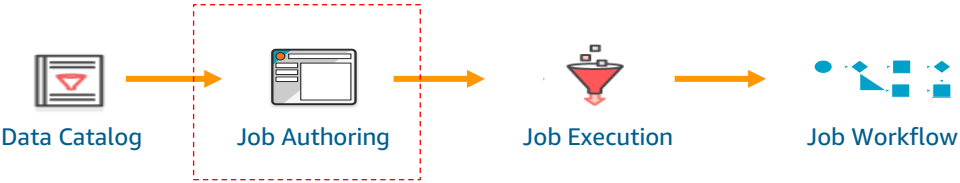
Output schema

Data preview

Change Schema (Apply mapping)

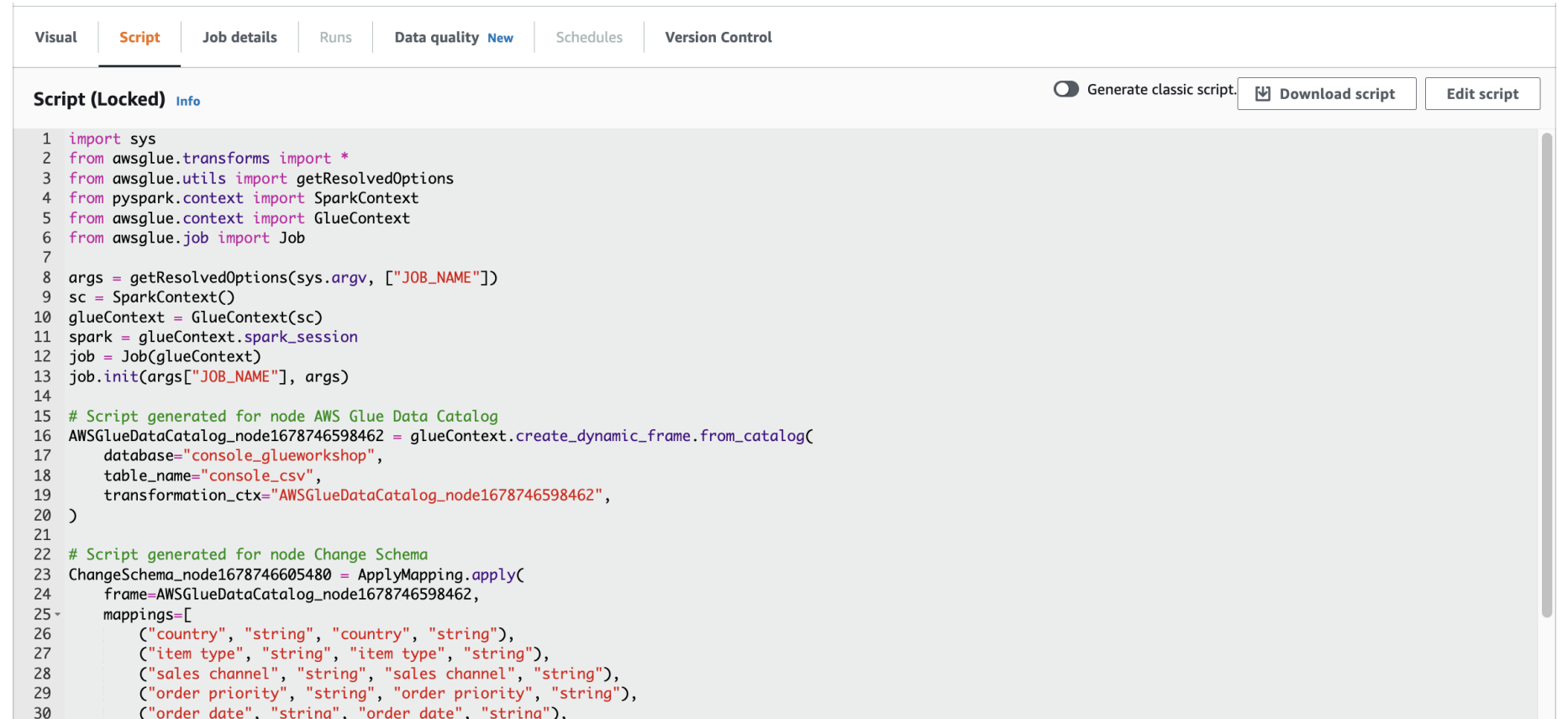
Source key	Target key	Data type	Drop
uuid	uuid	long	<input type="checkbox"/>
country	country	string	<input type="checkbox"/>
item type	item type	string	<input type="checkbox"/>
sales channel	sales channel	string	<input type="checkbox"/>
order priority	order priority	string	<input type="checkbox"/>
order date	order date	string	<input type="checkbox"/>
region	region	string	<input type="checkbox"/>
ship date	ship date	string	<input type="checkbox"/>
units sold	units sold	long	<input type="checkbox"/>

Existing  
columns  
in target

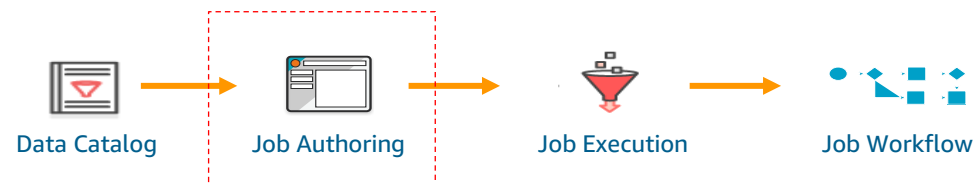


# AWS Glue : Authoring – Code Generation

1. **Human-readable, editable, and portable** PySpark or Scala code
2. **Customizable**: Use native PySpark / Scala, import custom libraries, and/or leverage Glue's libraries
3. **Collaborative**: share code snippets via GitHub, reuse code across jobs



```
1 import sys
2 from awsglue.transforms import *
3 from awsglue.utils import getResolvedOptions
4 from pyspark.context import SparkContext
5 from awsglue.context import GlueContext
6 from awsglue.job import Job
7
8 args = getResolvedOptions(sys.argv, ["JOB_NAME"])
9 sc = SparkContext()
10 glueContext = GlueContext(sc)
11 spark = glueContext.spark_session
12 job = Job(glueContext)
13 job.init(args["JOB_NAME"], args)
14
15 # Script generated for node AWS Glue Data Catalog
16 AWSGlueDataCatalog_node1678746598462 = glueContext.create_dynamic_frame.from_catalog(
17     database="console_glueworkshop",
18     table_name="console_csv",
19     transformation_ctx="AWSGlueDataCatalog_node1678746598462",
20 )
21
22 # Script generated for node Change Schema
23 ChangeSchema_node1678746605480 = ApplyMapping.apply(
24     frame=AWSGlueDataCatalog_node1678746598462,
25     mappings=[
26         ("country", "string", "country", "string"),
27         ("item type", "string", "item type", "string"),
28         ("sales channel", "string", "sales channel", "string"),
29         ("order priority", "string", "order priority", "string"),
30         ("order date", "string", "order date", "string"),
```



# AWS Glue : Authoring – Import Custom Modules

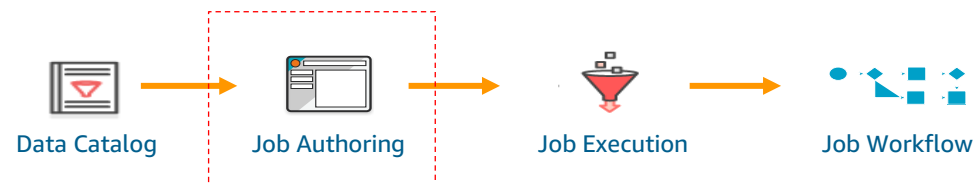
- **Add external Python libraries and modules**
- **Java JARs required by the script**
- **Additional files such as configuration, etc.**

Libraries [Info](#)

Python library path

Dependent JARs path

Referenced files path



# AWS Glue : Authoring – Apache Spark and Glue ETL

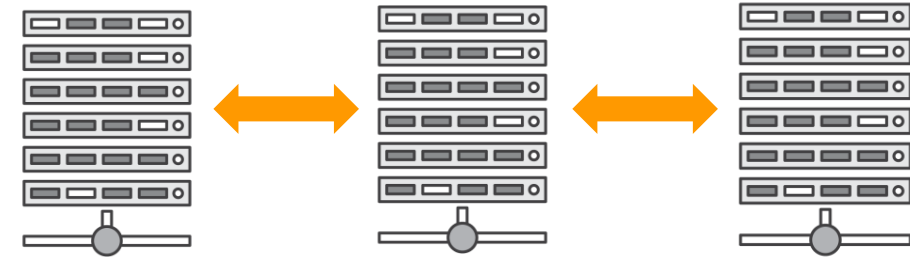
## What is Apache Spark?

Parallel, scale-out data processing engine

Fault-tolerance built-in

Flexible interface: Python scripting, SQL

Rich eco-system: ML, Graph, analytics, ...



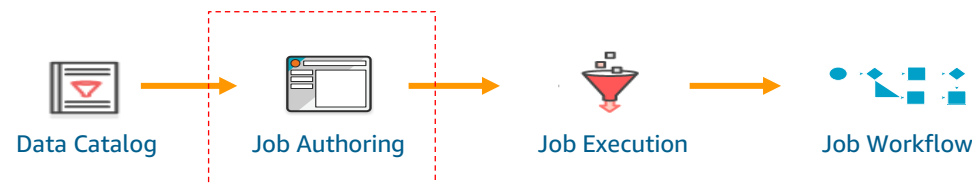
## AWS Glue ETL libraries

Integration: Data Catalog, job orchestration, code-generation, job bookmarks, S3, RDS

ETL transforms, more connectors & formats

New data structure: Dynamic Frames

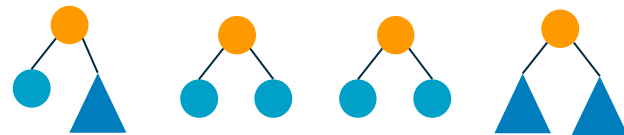
SparkSQL	AWS Glue ETL
Dataframes	Dynamic Frames
Spark core: RDDs	



# AWS Glue : Authoring – DataFrames and DynamicFrames

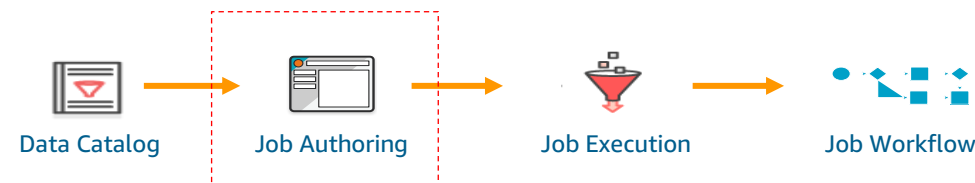
## Spark DataFrames

- Core data structure for SparkSQL
- Similar to structured tables  
Need schema up-front  
Each row has same structure
- Suited for SQL-like analytics



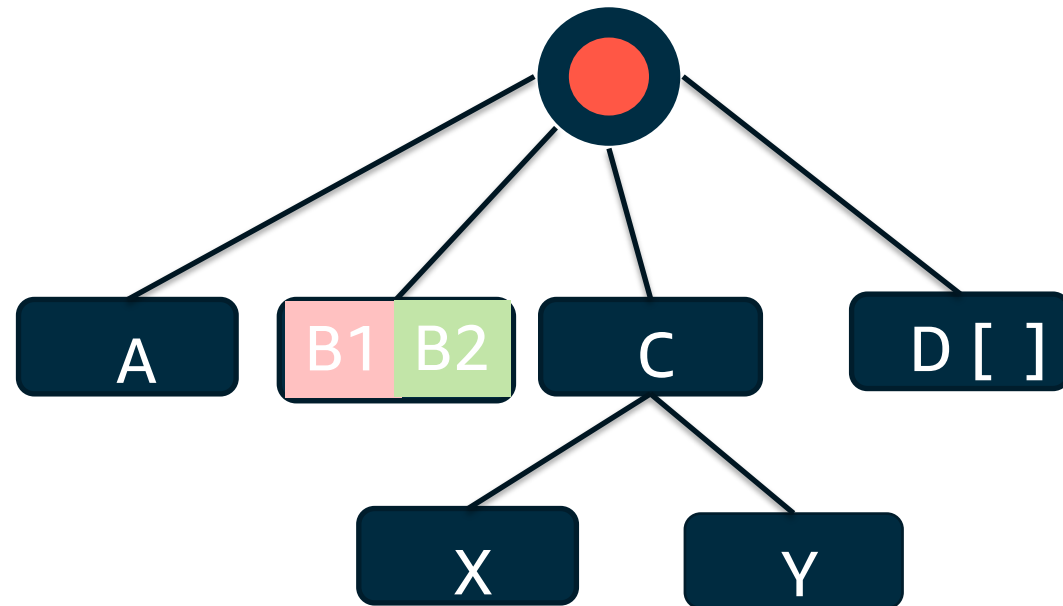
## Glue DynamicFrames

- Similar to DataFrames for ETL
- Designed for processing **semi-structured** data, e.g. JSON, Avro, Apache logs ...





# AWS Glue : Authoring – DynamicFrames



Like Spark's DataFrames, but better for:

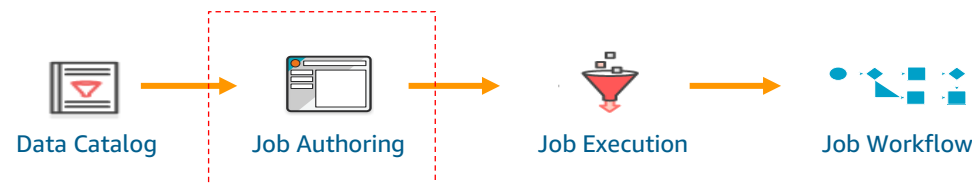
- Cleaning and (re)-structuring **semi-structured** data sets, e.g. JSON, Avro, Apache logs ...

No upfront schema needed:

- Infers schema on-the-fly, enabling transformations in a **single pass**

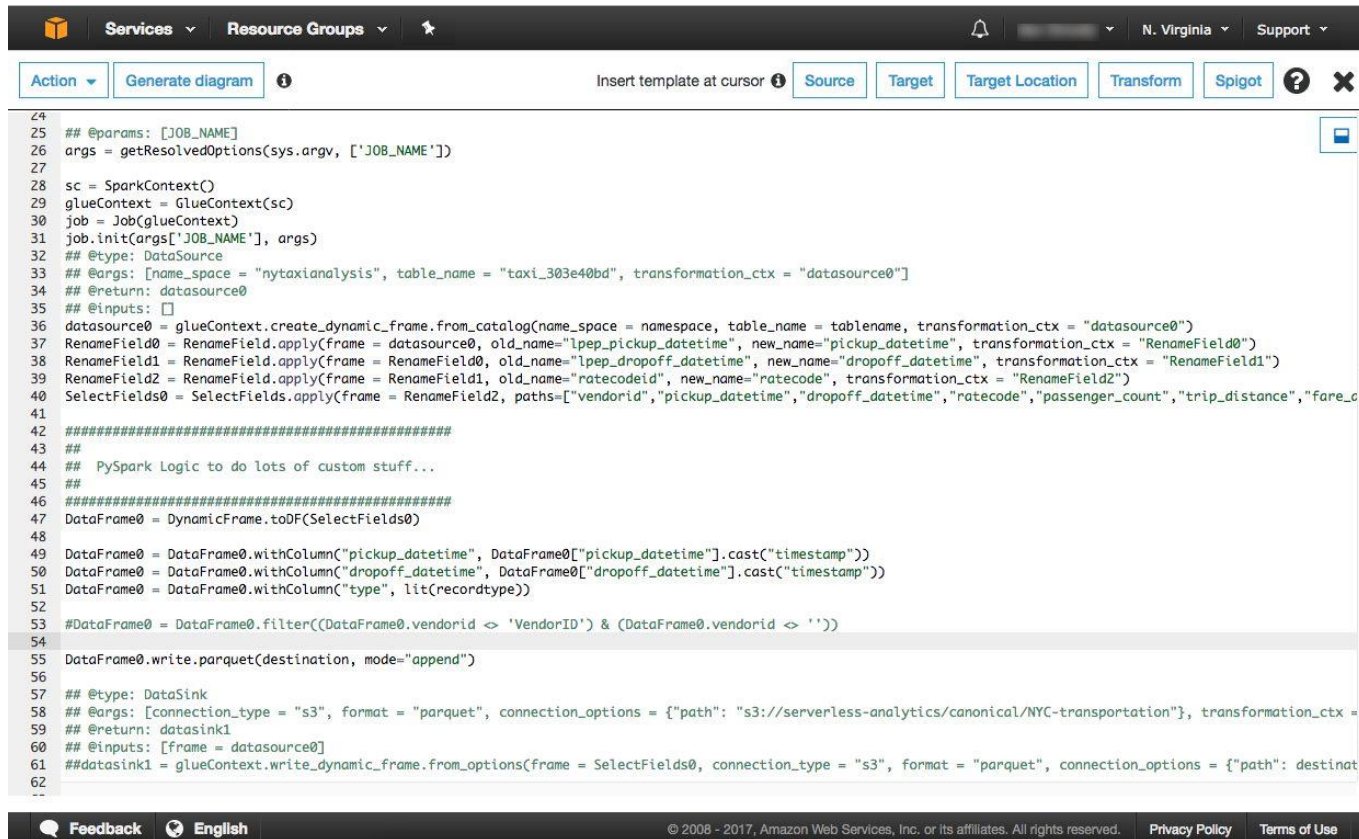
Easy to handle the unexpected:

- Tracks new fields, and inconsistent changing data types with **choices**, e.g. integer or string
- Automatically mark and separate error records



# AWS Glue : Authoring – “DynamicFrame” High-Level API

- Convert to and from Spark DataFrames
- Run with custom code and libraries



```
24
25 ## @params: [JOB_NAME]
26 args = getResolvedOptions(sys.argv, ['JOB_NAME'])
27
28 sc = SparkContext()
29 glueContext = GlueContext(sc)
30 job = Job(glueContext)
31 job.init(args['JOB_NAME'], args)
32 ## @type: DataSource
33 ## @args: [name_space = "nytaxianalysis", table_name = "taxi_303e40bd", transformation_ctx = "datasource0"]
34 ## @return: datasource0
35 ## @inputs: []
36 datasource0 = glueContext.create_dynamic_frame.from_catalog(name_space = namespace, table_name = tablename, transformation_ctx = "datasource0")
37 RenameField0 = RenameField.apply(frame = datasource0, old_name="lpep_pickup_datetime", new_name="pickup_datetime", transformation_ctx = "RenameField0")
38 RenameField1 = RenameField.apply(frame = RenameField0, old_name="lpep_dropoff_datetime", new_name="dropoff_datetime", transformation_ctx = "RenameField1")
39 RenameField2 = RenameField.apply(frame = RenameField1, old_name="ratecodeid", new_name="ratecode", transformation_ctx = "RenameField2")
40 SelectFields0 = SelectFields.apply(frame = RenameField2, paths=["vendorid", "pickup_datetime", "dropoff_datetime", "ratecode", "passenger_count", "trip_distance", "fare_c
41
42 #####
43 ##
44 ## PySpark Logic to do lots of custom stuff...
45 ##
46 #####
47 DataFrame0 = DynamicFrame.toDF(SelectFields0)
48
49 DataFrame0 = DataFrame0.withColumn("pickup_datetime", DataFrame0["pickup_datetime"].cast("timestamp"))
50 DataFrame0 = DataFrame0.withColumn("dropoff_datetime", DataFrame0["dropoff_datetime"].cast("timestamp"))
51 DataFrame0 = DataFrame0.withColumn("type", lit(recordtype))
52
53 #DataFrame0 = DataFrame0.filter((DataFrame0.vendorid <> 'VendorID') & (DataFrame0.vendorid <> ''))
54
55 DataFrame0.write.parquet(destination, mode="append")
56
57 ## @type: DataSink
58 ## @args: [connection_type = "s3", format = "parquet", connection_options = {"path": "s3://serverless-analytics/canonical/NYC-transportation"}, transformation_ctx =
59 ## @return: datasink1
60 ## @inputs: [frame = datasource0]
61 ##datasink1 = glueContext.write_dynamic_frame.from_options(frame = SelectFields0, connection_type = "s3", format = "parquet", connection_options = {"path": destinat
62
--
```

## fromDF

**fromDF(dataframe, glue\_ctx, name)**

Converts a DataFrame to a DynamicFrame by converting DynamicRecords to Rows. Returns the new DynamicFrame.

- dataframe – The spark SQL dataframe to convert. Required
- glue\_ctx – The [GlueContext \(p. 164\)](#) object that specifies the context for this transform. Required.
- name – The name of the resulting DynamicFrame. Required

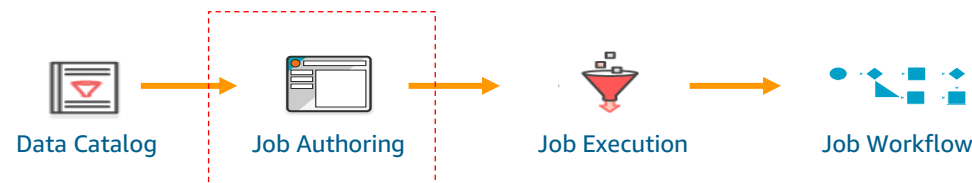
## toDF

**toDF(options)**

Converts a DynamicFrame to an Apache DataFrame. Returns the new DataFrame.

- options – A list of options. Please specify the target type if you choose the Project and Cast action type. Examples are:

```
>>>toDF([ResolveOption("a.b.c", "KeepAsStruct")])
>>>toDF([ResolveOption("a.b.c", "Project", DoubleType())])
```



# AWS Glue: Job Execution



# AWS Glue: Progress and History

Glue-Lab-SportTeamParquet

Save

Delete

Run

Visual

Script

Job details

Runs

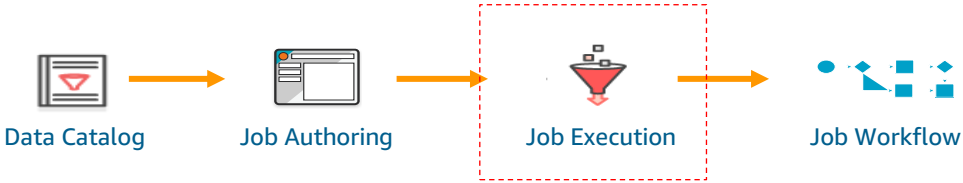
Schedules

Recent job runs (2) [Info](#)

July 21, 2021 12:23:45 AM

Job name	Id	Run status	Glue version
Glue-Lab-SportTeamParquet	jr_26dbd948fb668520cf658290f866b9ad99e1f5b66af59e2fb7934749428cae52	Succeeded	2.0
Retry attempt number	Start time	End time	Start-up time
Initial run	July 21, 2021 12:23:45 AM	July 21, 2021 12:24:43 AM	7 seconds
Execution time	Last modified on	Trigger name	Security configuration
51 seconds	July 21, 2021 12:24:43 AM	-	-
Timeout	Max capacity	Number of workers	Worker type
2880 minutes	10 DPUs	10	G.1X
Execution class	Log group name	Cloudwatch logs	Performance and debugging recommendations
-	/aws-glue/jobs	<a href="#">All logs</a>	

- Track ETL job progress and inspect logs directly from the console.
- Logs are written to CloudWatch for simple access.
- Errors are automatically extracted and presented in the Error Logs for easy troubleshooting of jobs.



# AWS Glue: Job Bookmarks

Glue keeps track of data that has already been processed by a previous run of an ETL job. This persisted state information is called a **bookmark**.

**For example,** you get new files everyday in your S3 bucket. By default, AWS Glue keeps track of which files have been successfully processed by the job to prevent data duplication.

Option	Behavior
Enable	Pick up from where you left off
Disable	Ignore and process the entire dataset every time
Pause	Temporarily disable advancing the bookmark

VisualScriptJob detailsRunsData quality NewSchedulesVersion Control

Type

The type of ETL job. This is set automatically based on the types of data sources you have selected.

Spark

Glue version Info

Glue 3.0 - Supports spark 3.1, Scala 2, Python 3

Language

Python 3

Worker type

Set the type of predefined worker that is allowed when a job runs.

G 1X

(4vCPU and 16GB RAM)

☐ Automatically scale the number of workers

AWS Glue will optimize costs and resource usage by dynamically scaling the number of workers up and down throughout the job run. Requires Glue 3.0 or later.

Requested number of workers

The number of workers you want AWS Glue to allocate to this job.

10

☒ Generate job insights

AWS Glue will analyze your job runs and provide insights on how to optimize your jobs and the reasons for job failures.

Job bookmark Info

Specifies how AWS Glue processes job bookmark when the job runs. It can remember previously processed data (Enable), update state information (Pause), or ignore state information (Disable).

Disable

```
graph LR; DC[Data Catalog] --> JA[Job Authoring]; JA --> JE[Job Execution]; JE --> JW[Job Workflow]; style JE stroke-dasharray: 5 5; style JW stroke-dasharray: 5 5;
```

© 2024, Amazon Web Services, Inc. or its Affiliates.

# AWS Glue: Monitoring and Notifications

External

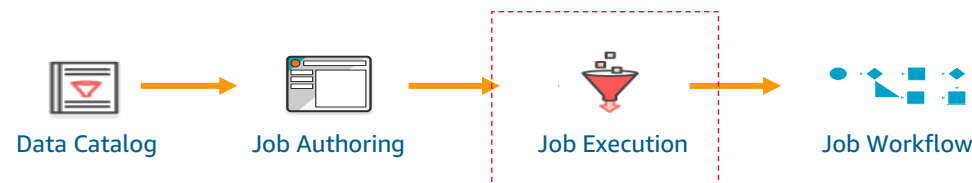
Publish crawler and job notifications into **Amazon CloudWatch**.  
Use Amazon CloudWatch events to control downstream workflows

Conditions

'**ANY**' and '**ALL**' operators in Trigger conditions  
Additional job states '**failed**', '**stopped**', or '**timeout**'

Control

Configure **job timeout**  
**Job delay notifications**





# AWS Glue: Job Metrics

▼ **Advanced properties**

Script filename

Glue-Lab-SportTeamParquet.py

Script path

S3 location of the script. Path must be in the form s3://bucket/prefix/path/. It must end with a slash (/) or

🔍

s3://aws-glue-assets-424559074886-1

✕

View

🔗

Browse S3

☒ **Job metrics** [Info](#)

Enable the creation of CloudWatch metrics when this job runs.

☒ **Continuous logging** [Info](#)

Enable logs in CloudWatch.

☒ **Spark UI** [Info](#)

Enable using Spark UI for monitoring this job.

Spark UI logs path

🔍

s3://aws-glue-assets-424559074886-1

✕

View

🔗

Browse S3

AWS Glue

Data catalog

Databases

Tables

Connections

Crawlers

Classifiers

Settings

ETL

Jobs

Triggers

Dev endpoints

Notebooks

Security

Security configurations

Tutorials

Add crawler

Explore table

Add job

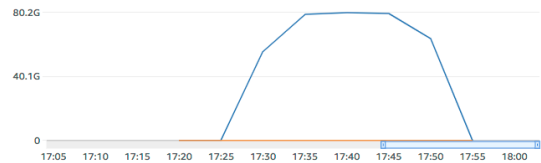
Resources

What's new 10+

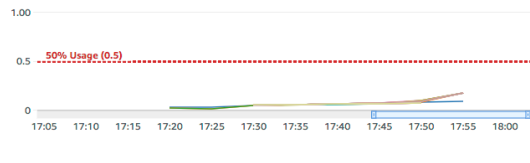
Jobs > githubevent\_year

Detailed job metrics

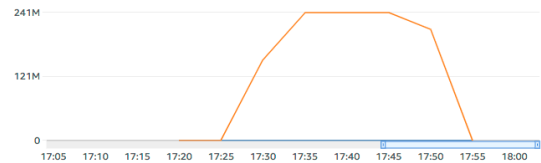
ETL Data Movement



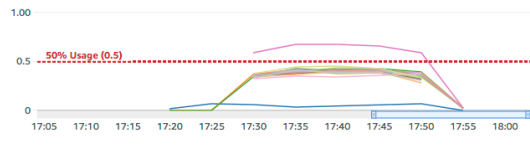
Memory Profile: Driver and Executors



Data Shuffle Across Executors



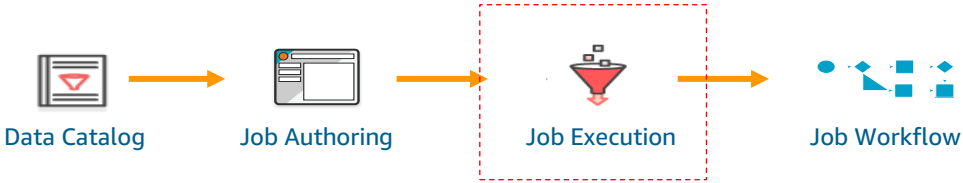
CPU Load: Driver and Executors



Job Execution: Active Executors, Completed Stages & Maximum Needed Executors

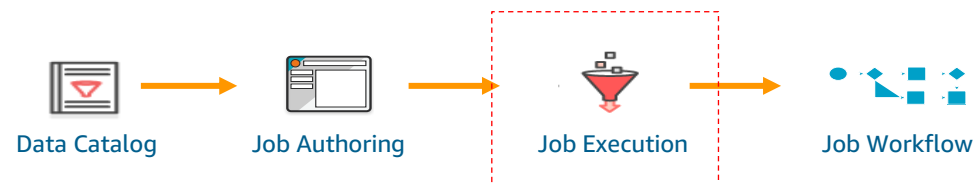
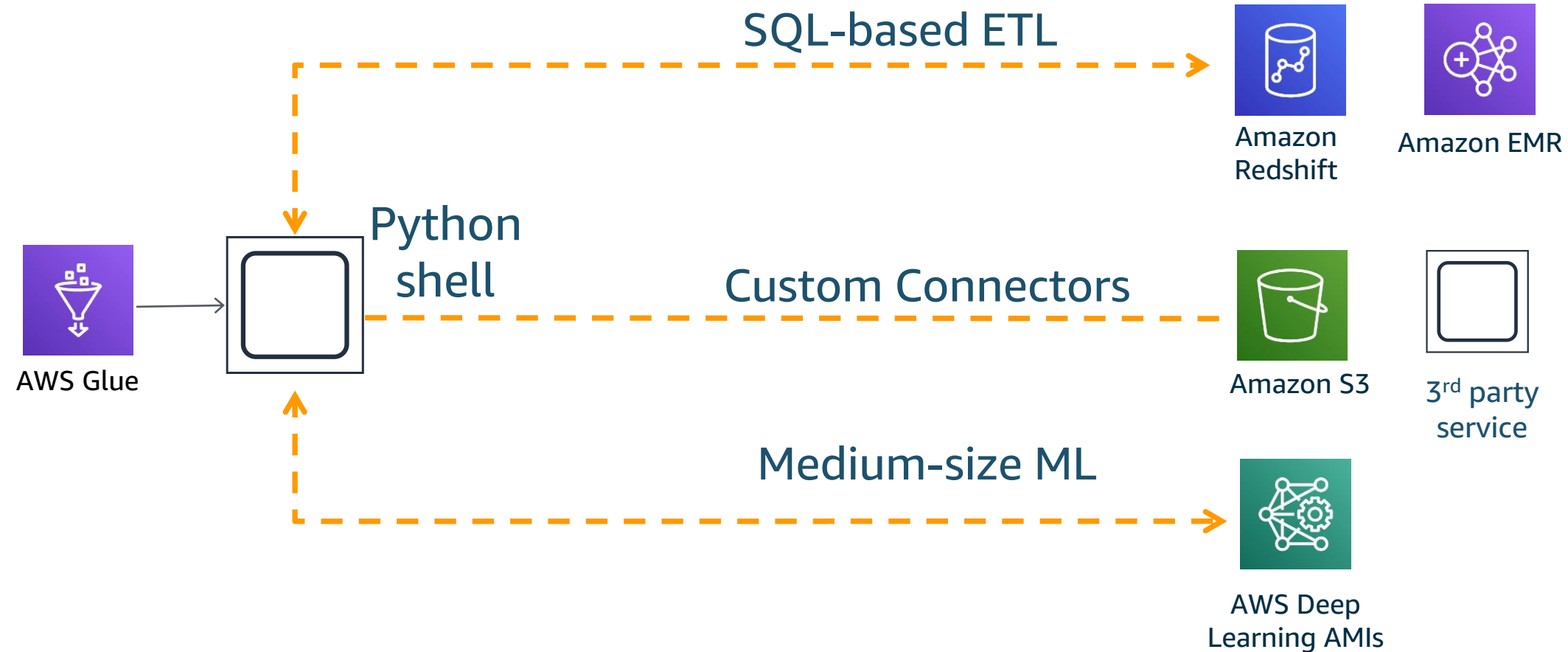


Metrics can be enabled in the AWS Command Line Interface (AWS CLI) and AWS SDK by passing `--enable-metrics` as a job parameter key.



# AWS Glue: Python Shell Job

*A cost-effective ETL primitive for small to medium tasks*

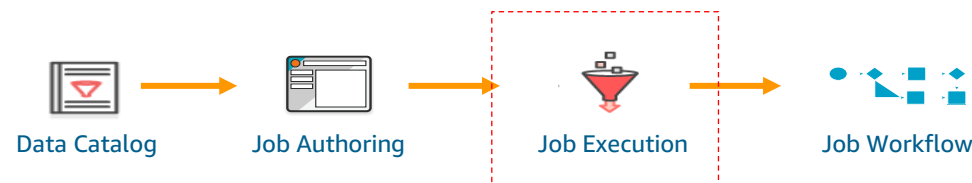




# AWS Glue: Streaming ETL

ETL operations on streaming data using continuously-running jobs.

- Built on the Apache Spark Structured Streaming engine.
- Streaming ETL can clean and transform streaming data and load it into Amazon S3 or JDBC data stores.
- Use Streaming ETL in AWS Glue to process event data like IoT streams, clickstreams, and network logs.
- Automatically detect the schema of incoming records and gracefully handle schema changes on a per-record basis
- Use both AWS Glue built-in transforms and transforms that are native to Apache Spark Structured Streaming
  - Selection, projection, aggregation, window operations, joins, deduplication, etc.



# AWS Glue: Optimizations

## Workload Partitioning

Make complex ETL pipelines significantly more resilient to errors using **Workload Partitioning**:

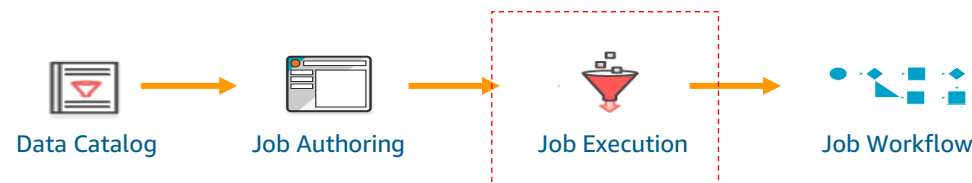
### Partitioning:

- limits the maximum number of files or dataset size, processed incrementally.
- Increments are orchestrated sequentially or in parallel.

## Pushdown Predicates

Reduce the number of files read by a job by filtering by pushdown predicates:

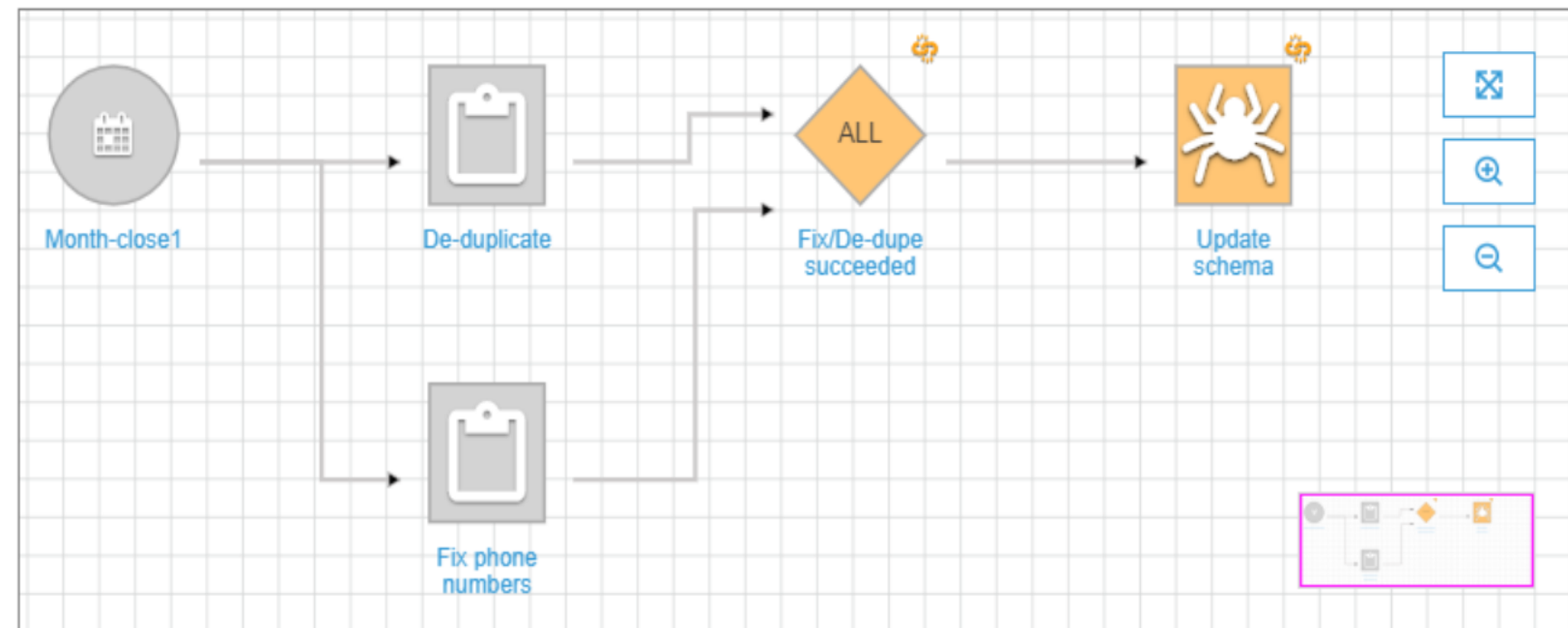
- Only read filtered partitions into your dynamic frames.
- Predicate expression can be any Boolean expression supported by Spark SQL



# Glue Job Orchestration

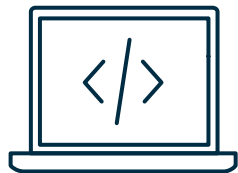
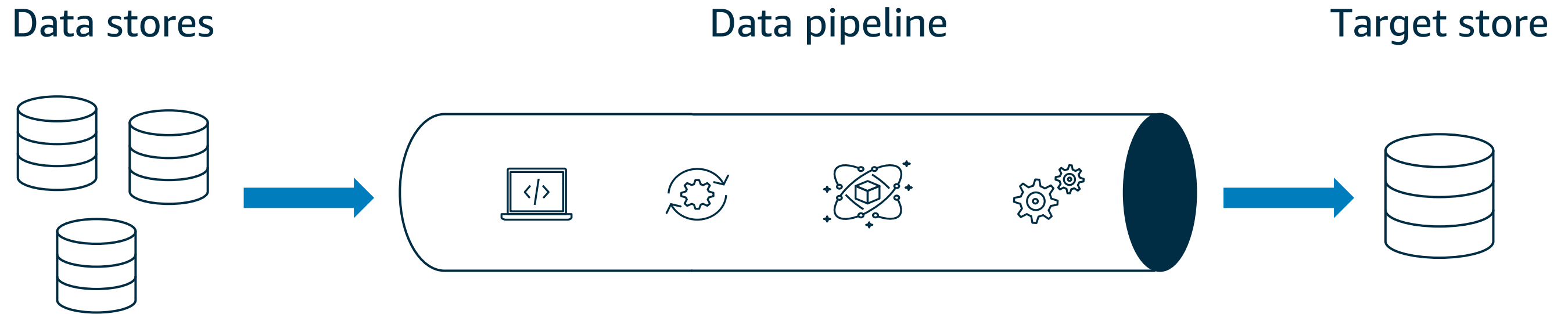
# AWS Glue: Job Orchestration Workflows

- Create and visualize complex ETL activities involving multiple crawlers, jobs, and triggers
- Records execution progress and status
- Provide both static and dynamic view
- Multiple triggering mechanisms
  - **Schedule-based:** e.g., time of day
  - **Event-based:** e.g., job completion
  - **On-demand:** e.g., AWS Lambda
- Logs and alerts are available in Amazon CloudWatch

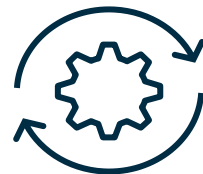


# Data Cleansing, Transformation and Lineage with **AWS Glue DataBrew**

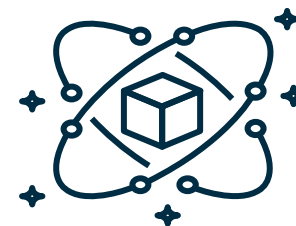
# Combining and replicating data between multiple stores can be challenging



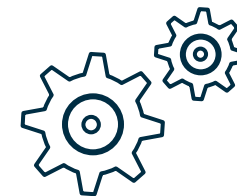
**COMPLEX  
APPLICATION CODE**



**CUMBERSOME  
RETRY LOGIC**



**PIPELINE  
MANAGEMENT**



**NEED SPECIALIZED  
ETL SKILLS**



**BUSINESS  
ANALYST**

**DATA SCIENTIST**

## AWS Glue DataBrew

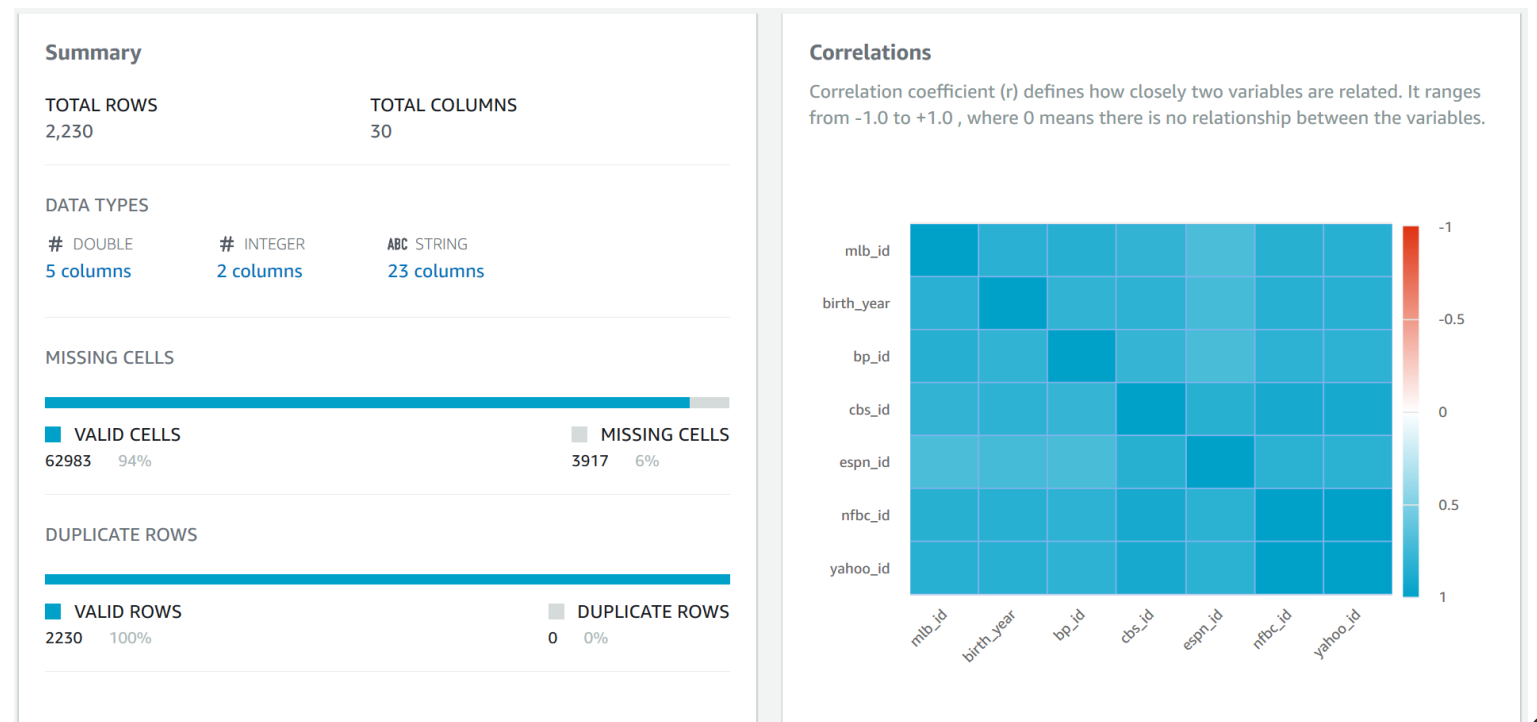
Clean and normalize data with  
a rich visual interface

Choose from 250+ built-in  
transformations to automate tasks

Profile data to understand data  
patterns and anomalies

Work on large datasets at scale

Format, Clean, Extract, Find Missing  
Values, Invalid Data, Duplicates, Split,  
Merge, Create Columns, Apply Functions,  
Unnest, Pivot, Group, Join, Union and  
more...



**Thank you!**

Justin Yenser  
qyensjus@amazon.com