Working within the Data Lake

With AWS Glue

Justin Yenser

Solutions Architect
Amazon Web Services

Agenda

Optimizing for Cost and Performance

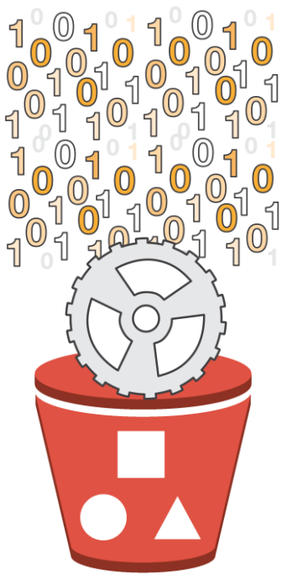
AWS Glue Components:

- 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

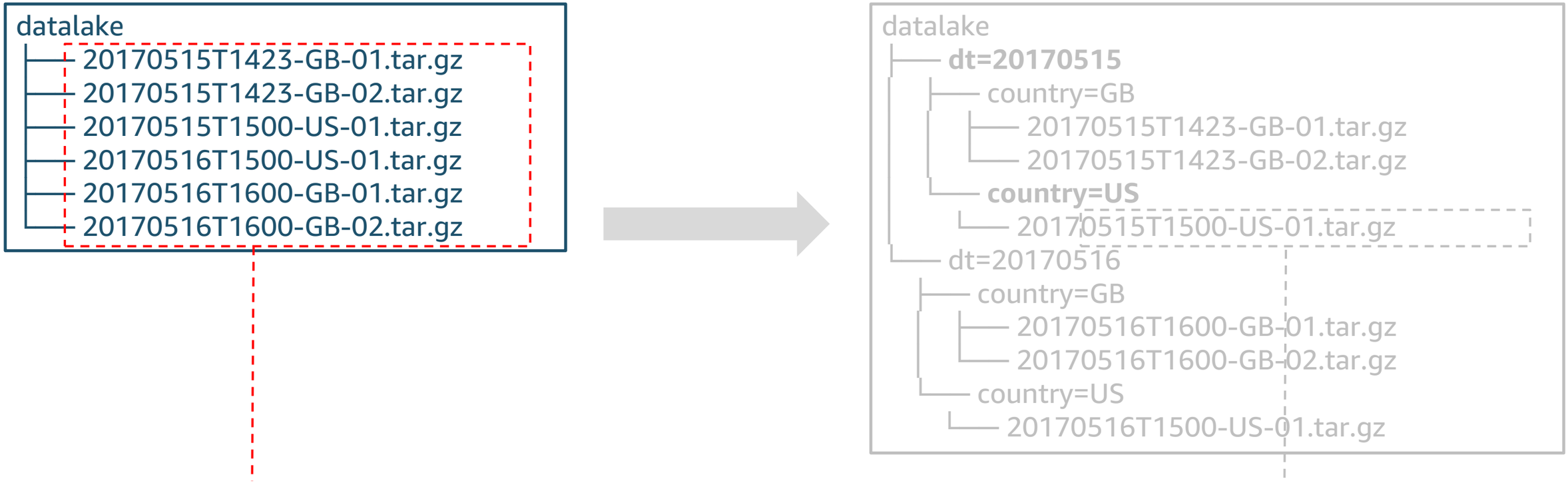
Compression

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

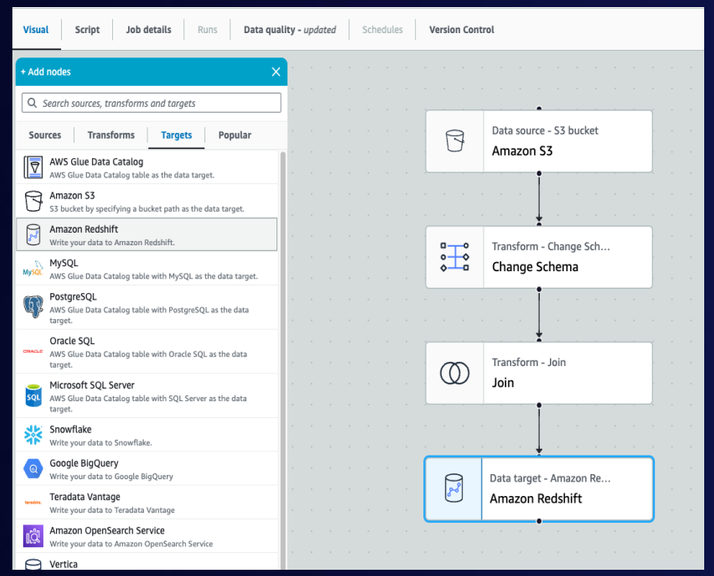
Job Authoring

Job Execution

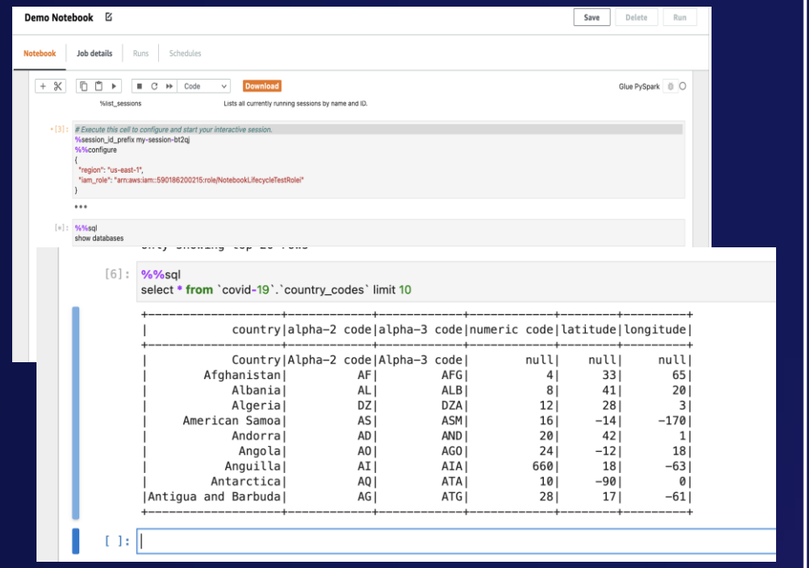
Job Workflow

Built for multiple personas

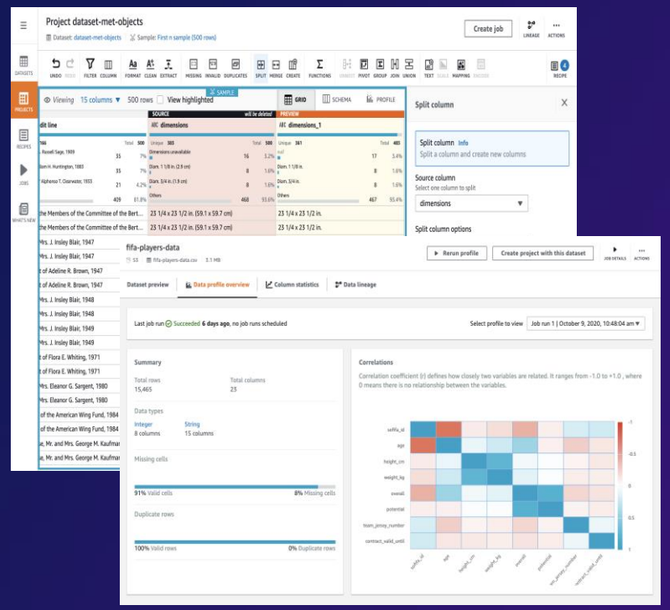
Customers migrate to promote self service



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



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



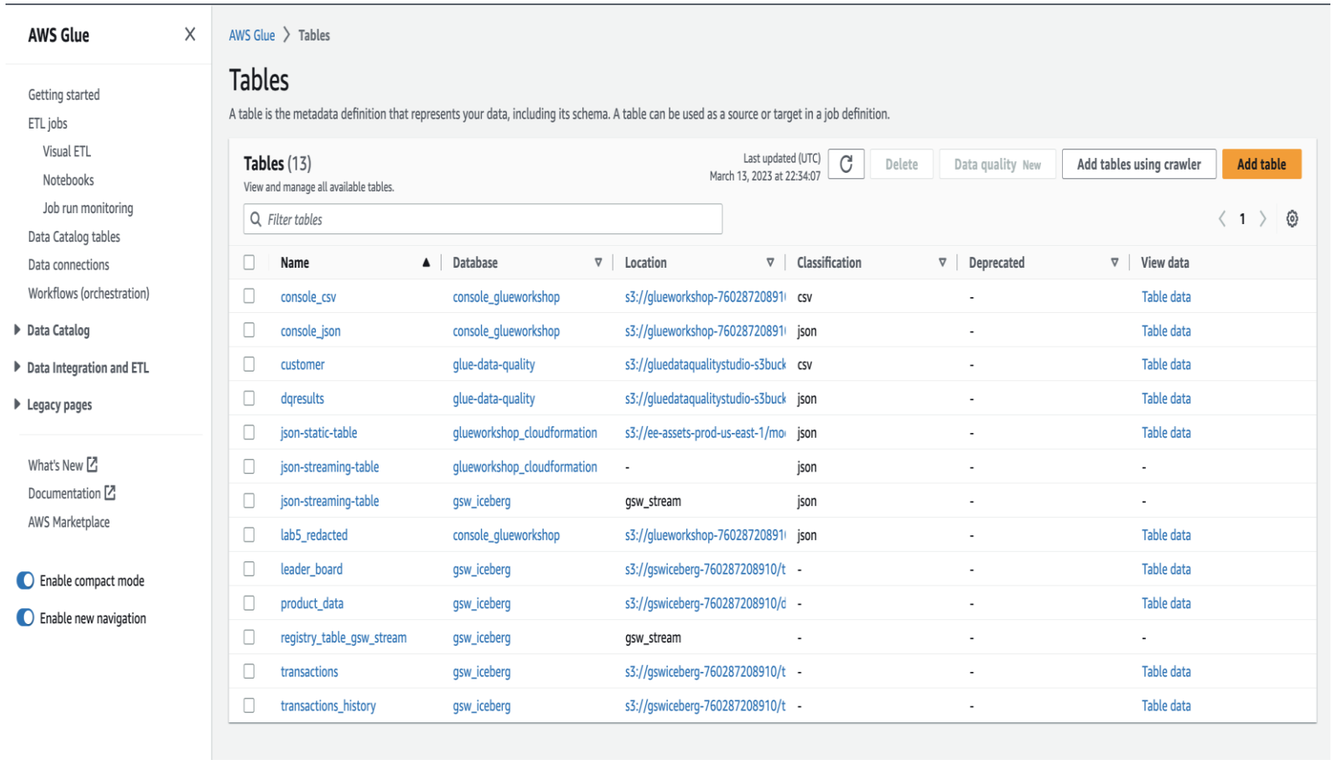
Glue Databrew for Analysts who prefer a no-code experience

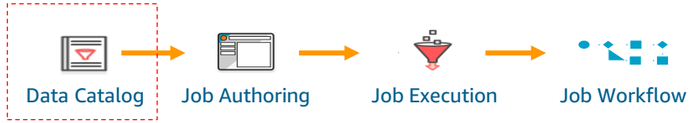
AWS Glue Data Catalog

AWS Glue : Data Catalog

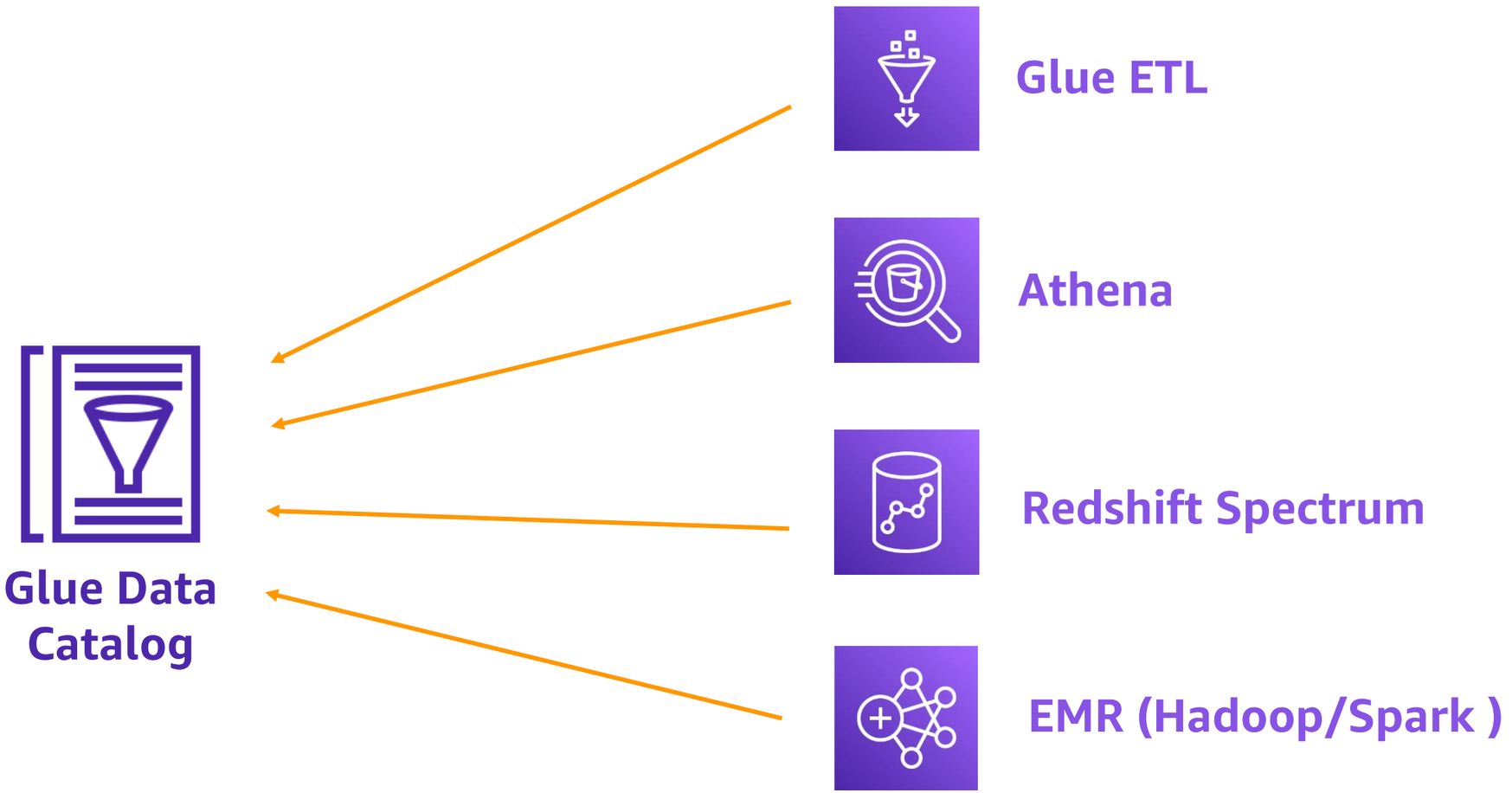
Features include:

- 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





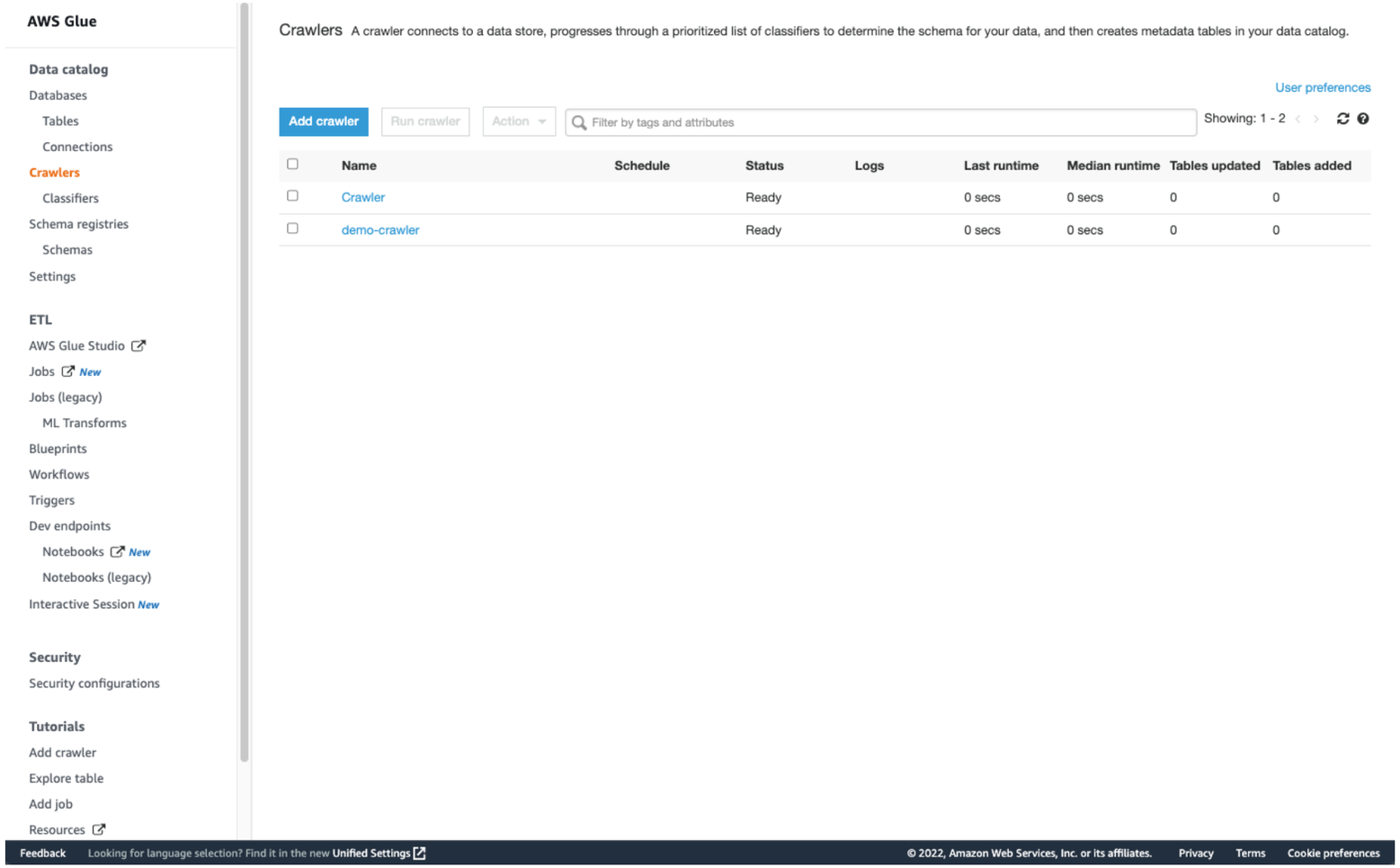
AWS Glue : Data Catalog – Queryable by many services



AWS Glue : Data Catalog - Crawlers

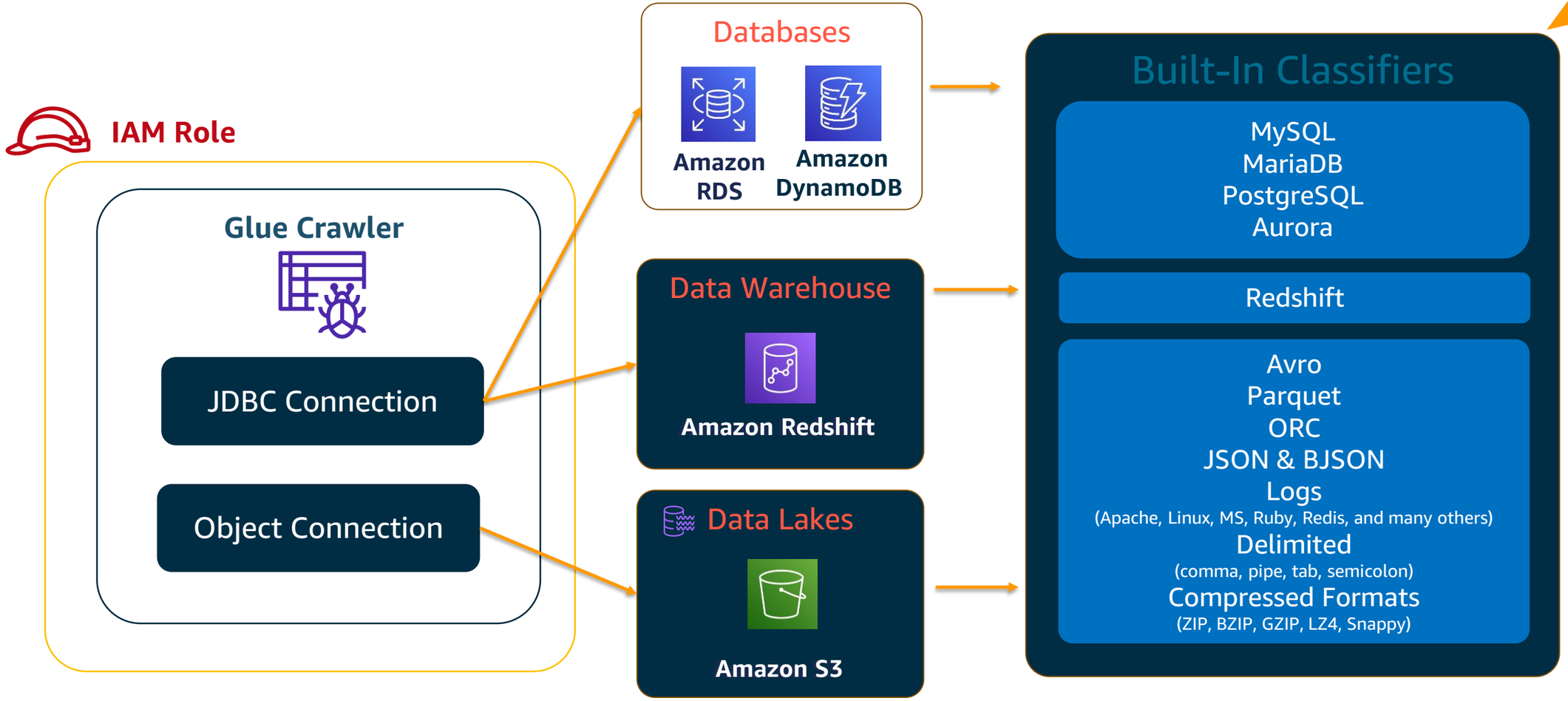
Features Include:

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



AWS Glue: Crawlers – Classifiers

Build your own custom classifiers with Grok!



AWS Glue : Data Catalog – Detecting Schema and Partitions

S3 bucket hierarchy

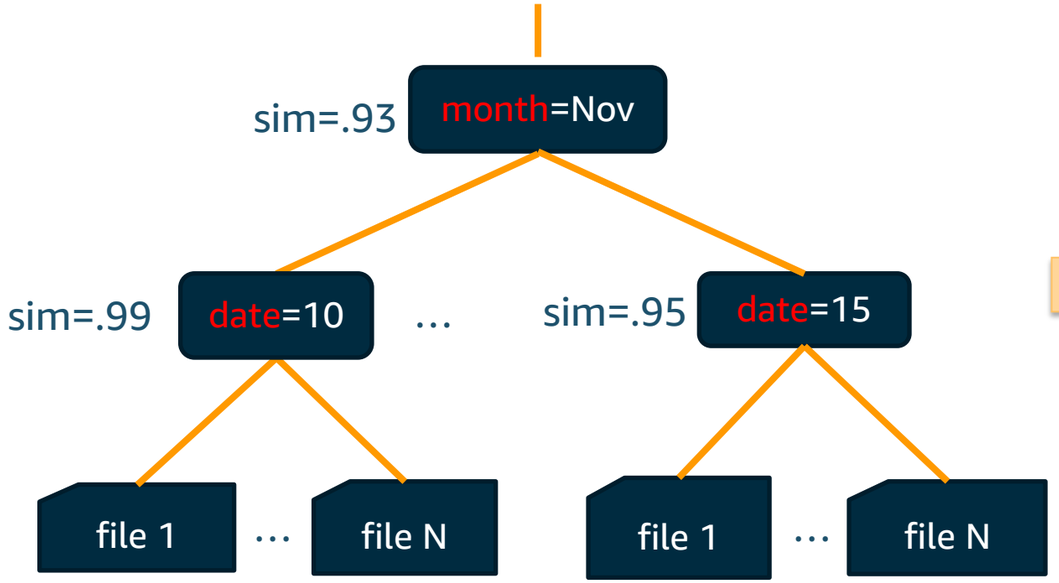
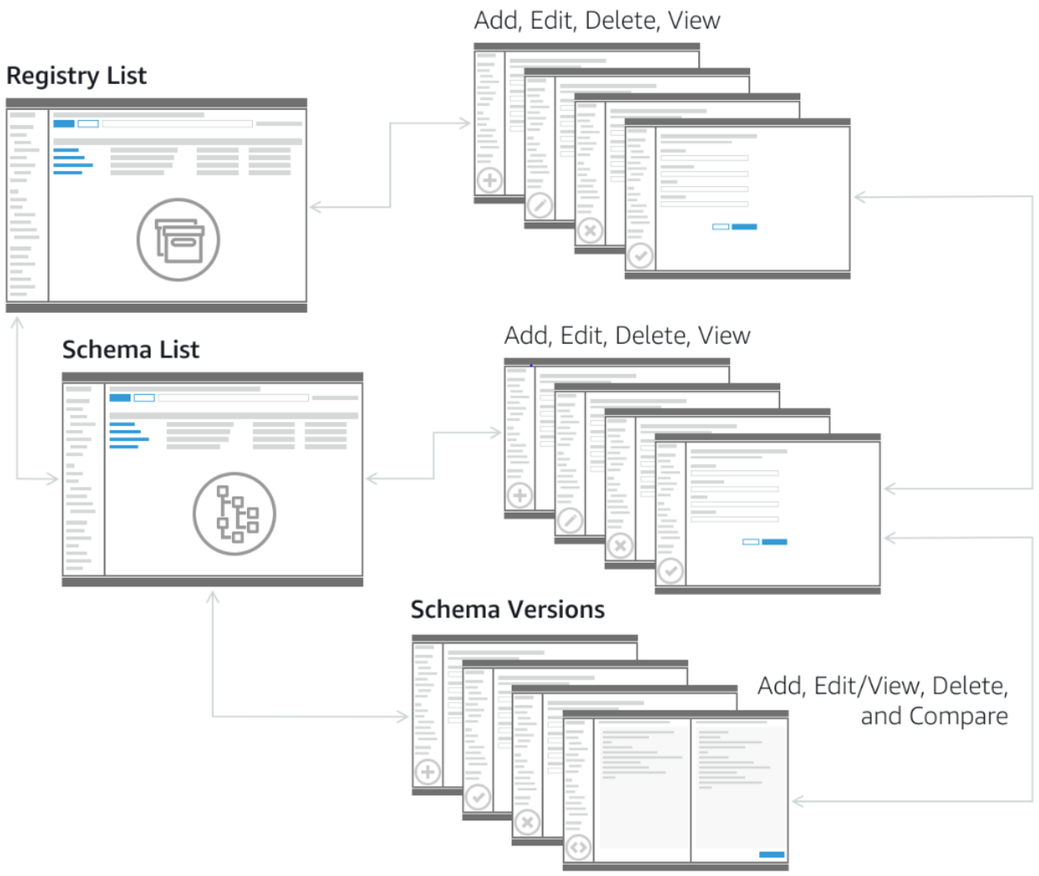


Table definition

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

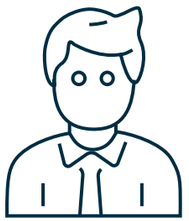
AWS Glue : Schema Registry for Streaming Data

- 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



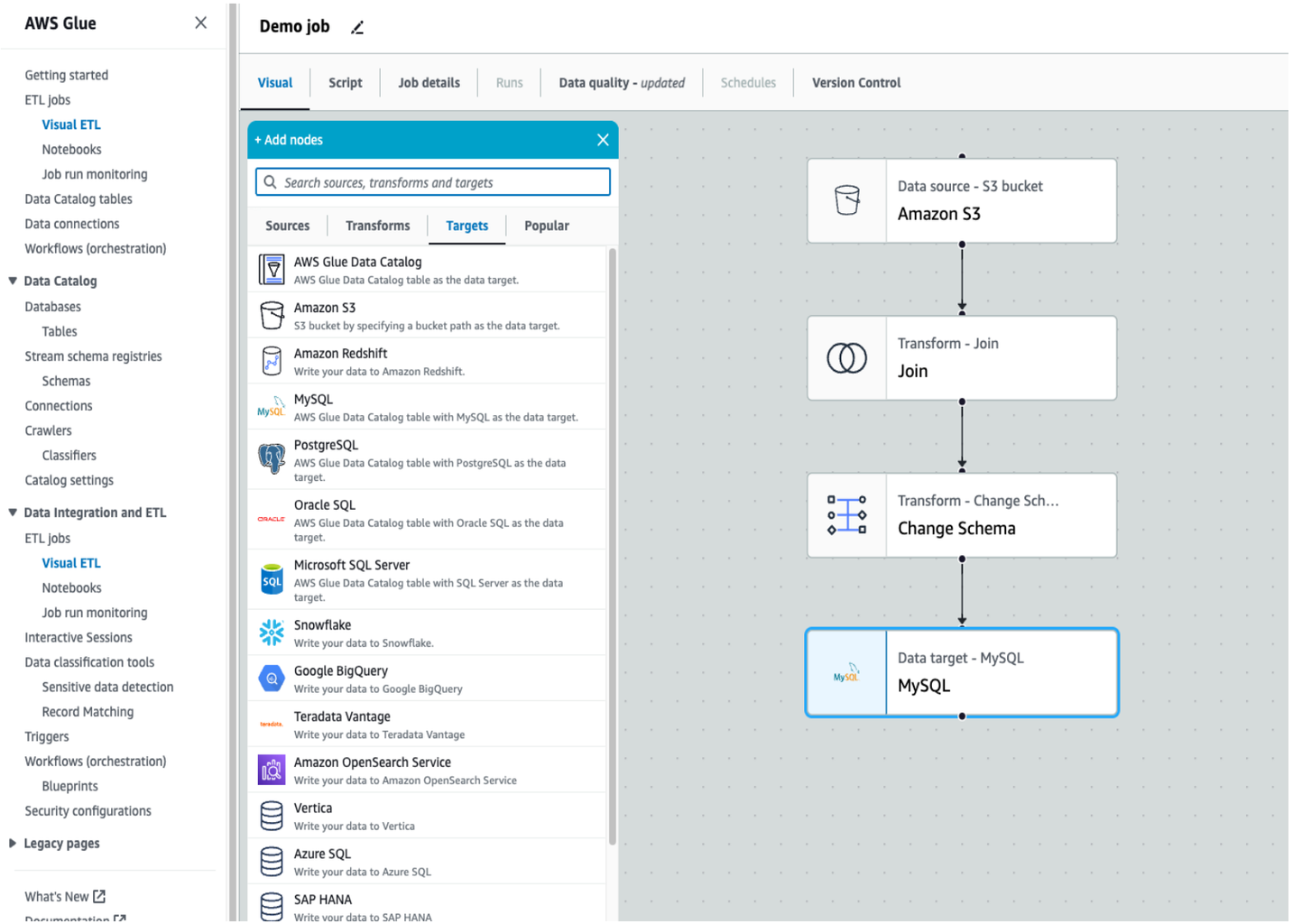
ETL DEVELOPER

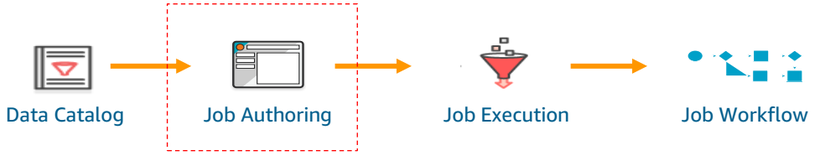
Visual authoring of ETL jobs without writing code

Monitor thousands of jobs through a single pane of glass

Distributed processing without the learning curve

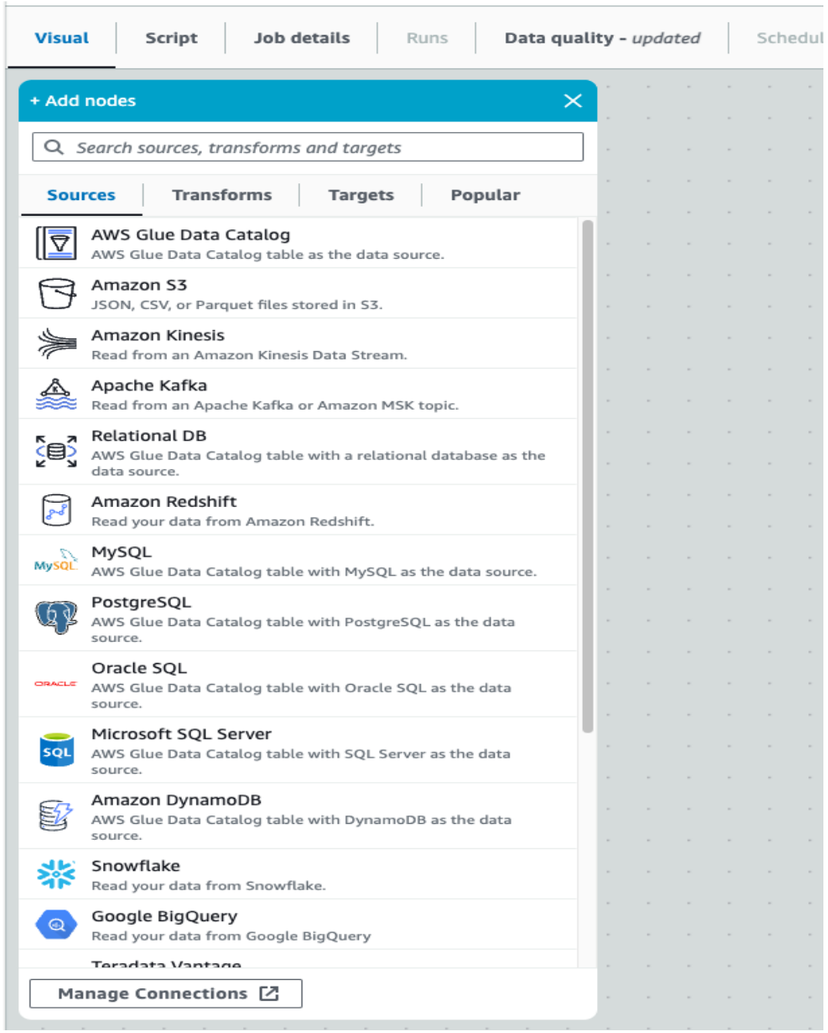
Advanced transforms though code snippets



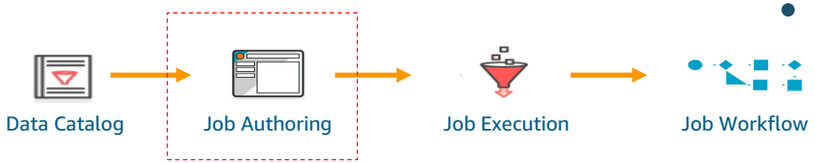


Glue : Authoring – Sources

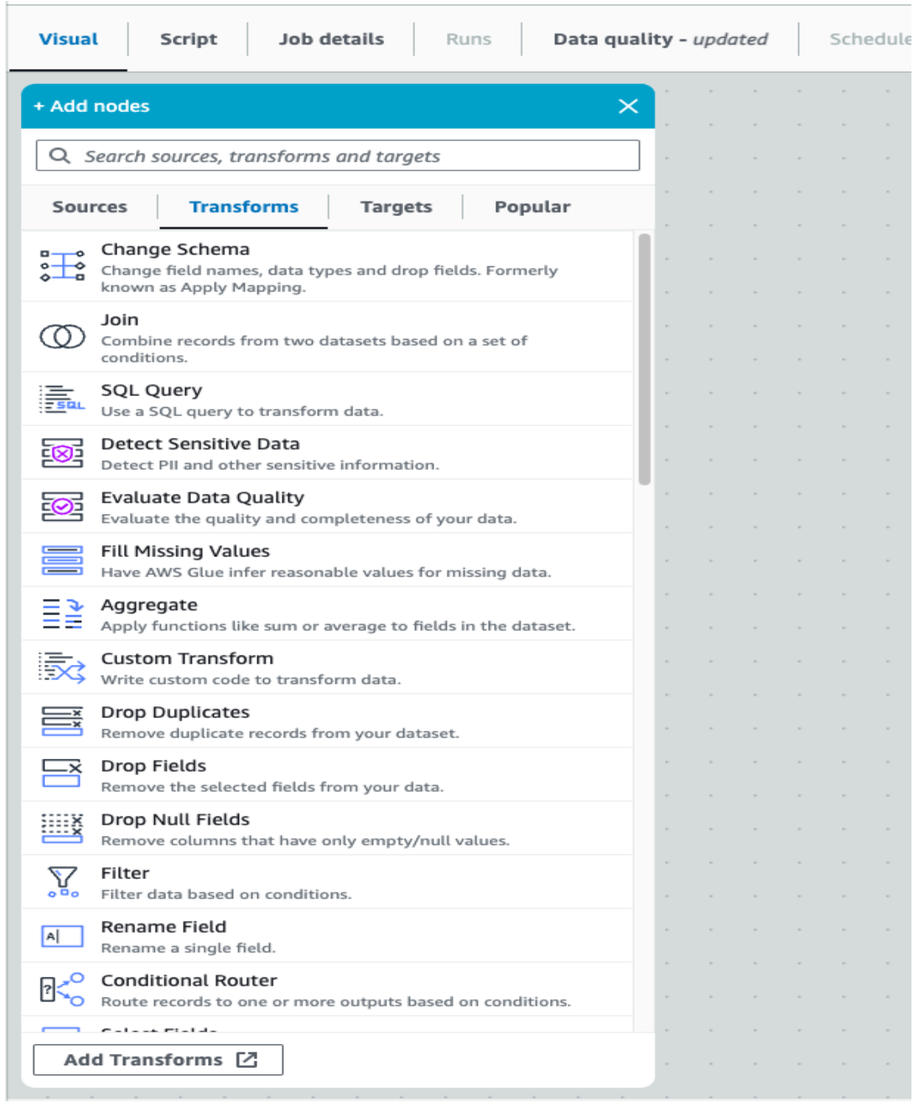
Choose from Manually-Defined or Crawler-Generated Sources



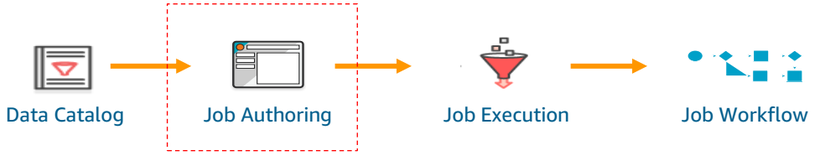
- S3 Bucket
- RDBMS
- Redshift
- RDS
- DocumentDB
- MongoDB
- DynamoDB
- Kinesis
- MSK
- And more



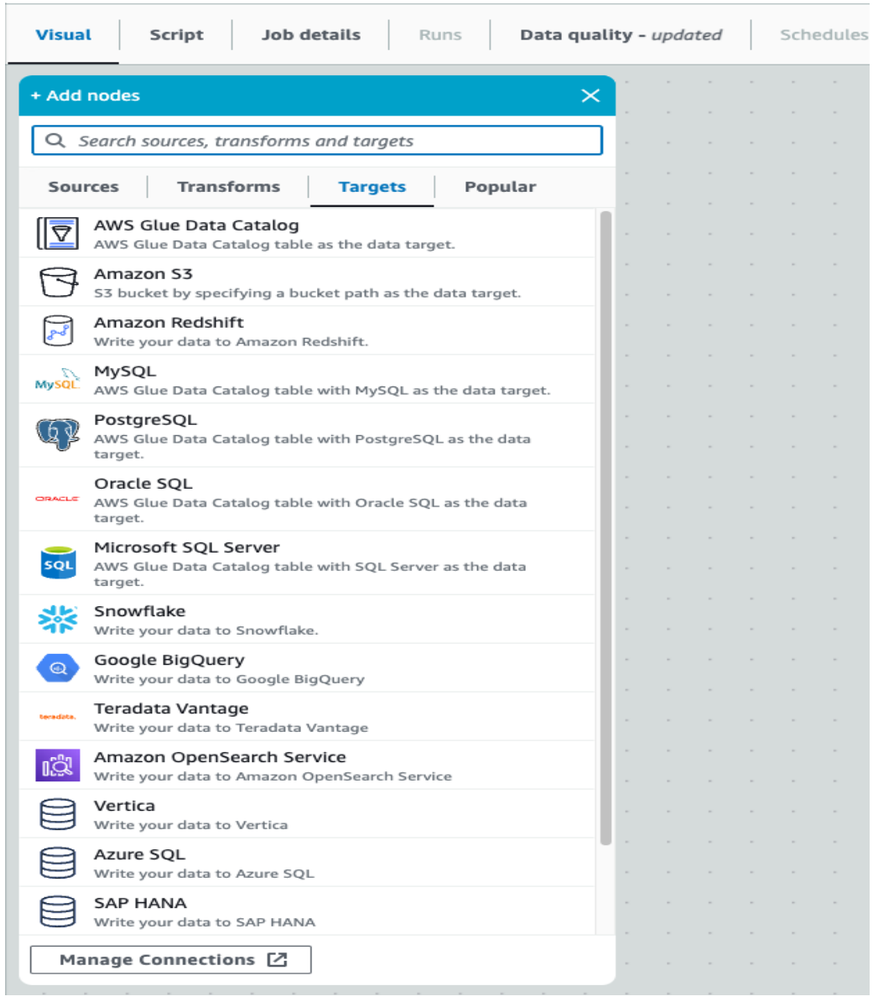
AWS Glue : Authoring – Transformations



- **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

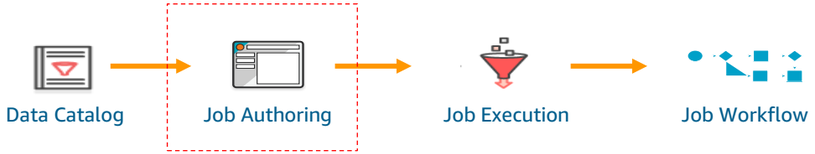


AWS Glue : Authoring – Targets

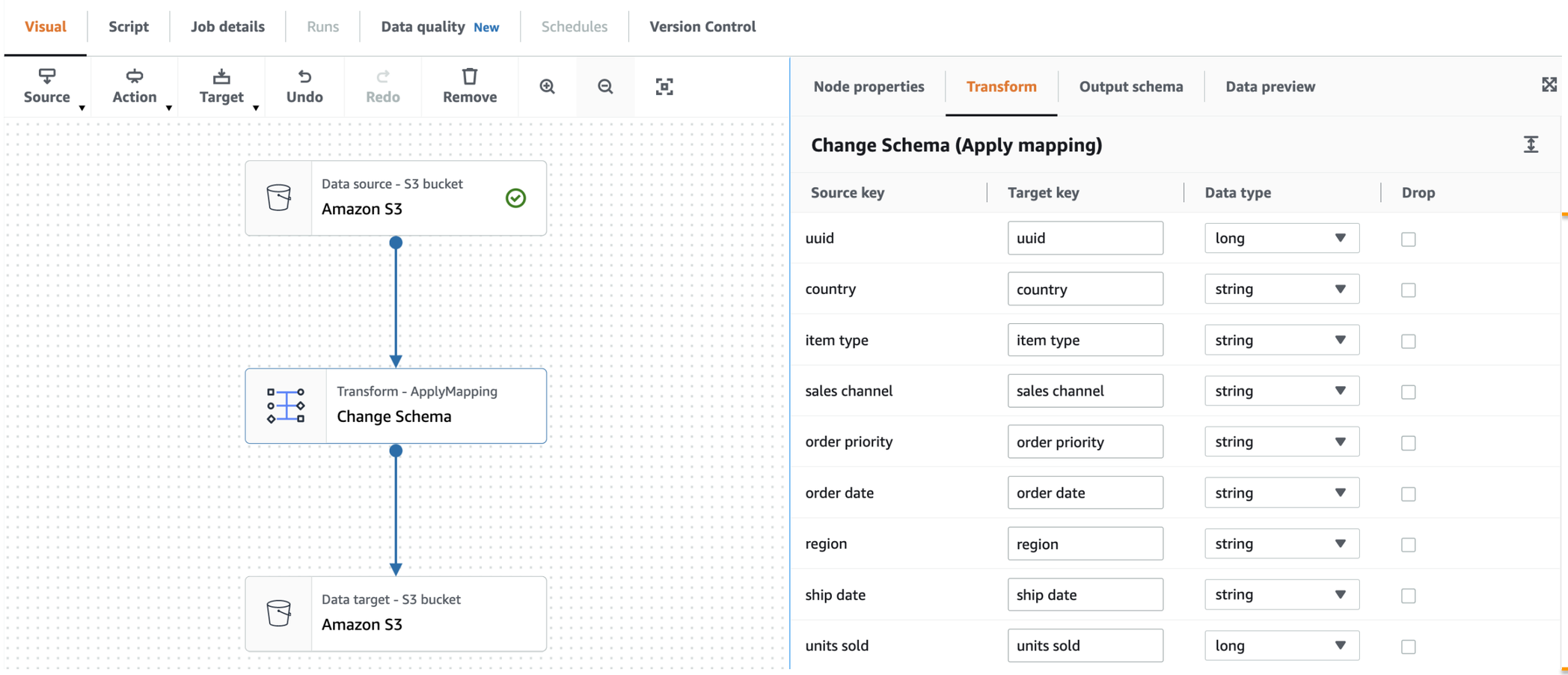


Crawler-Defined or Manually-Created

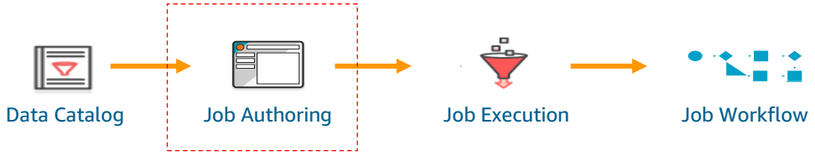
- S3 Bucket
- RDBMS
- Redshift
- Document DB
- RDS
- And more



AWS Glue : Authoring – Apply Mapping Transform



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

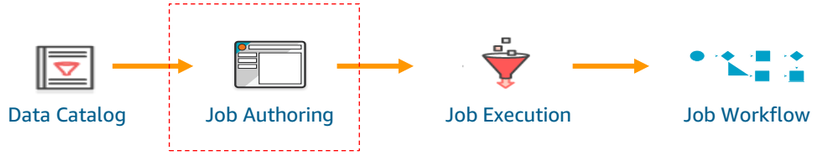
3. Collaborative: share code snippets via GitHub, reuse code across jobs

Visual | Script | Job details | Runs | Data quality New | Schedules | Version Control

Script (Locked) Info

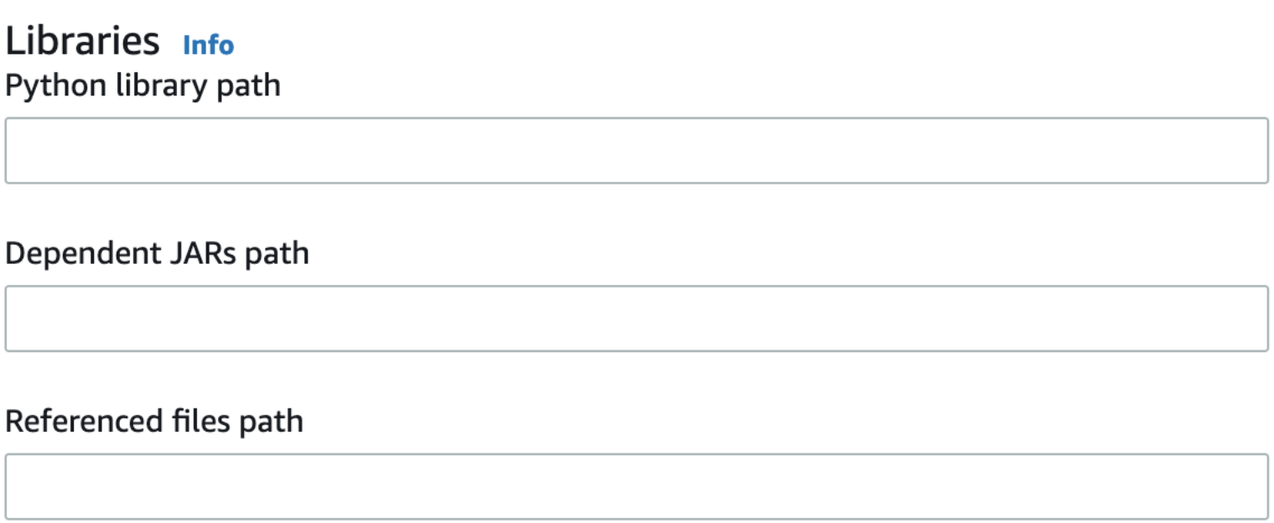
Generate classic script. Download script Edit script

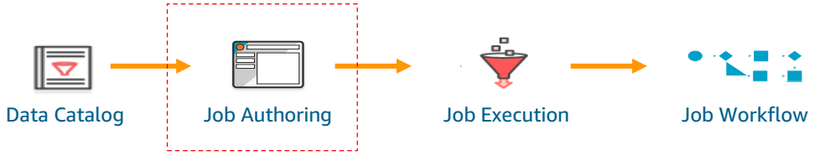
```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Script generated for node AWS Glue Data Catalog
AWSGlueDataCatalog_node1678746598462 = glueContext.create_dynamic_frame.from_catalog(
    database="console_glueworkshop",
    table_name="console_csv",
    transformation_ctx="AWSGlueDataCatalog_node1678746598462",
)

# Script generated for node Change Schema
ChangeSchema_node1678746605480 = ApplyMapping.apply(
    frame=AWSGlueDataCatalog_node1678746598462,
    mappings=[
        ("country", "string", "country", "string"),
        ("item type", "string", "item type", "string"),
        ("sales channel", "string", "sales channel", "string"),
        ("order priority", "string", "order priority", "string"),
        ("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.**

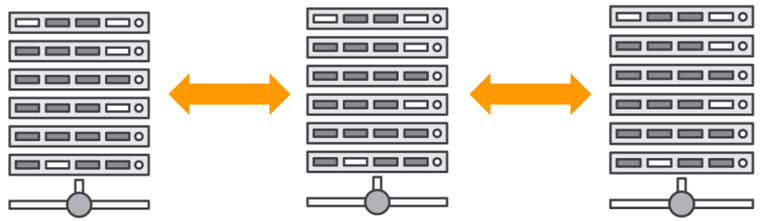




AWS Glue : Authoring – Apache Spark and Glue ETL

What is Apache Spark?

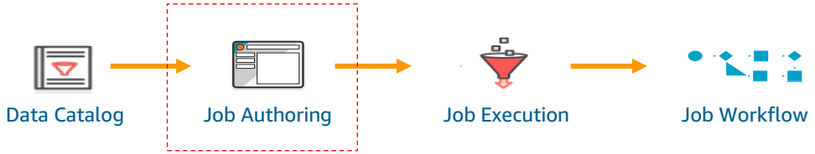
- Parallel, scale-out data processing engine
- Fault-tolerance built-in
- Flexible interface: Python scripting, SQL
- Rich eco-system: ML, Graph, analytics, …



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

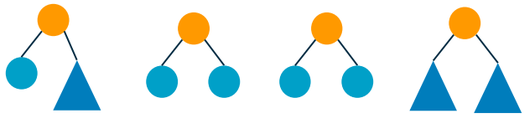
SparkSQL	AWS Glue ETL
Dataframes	Dynamic Frames
Spark core: RDDs	

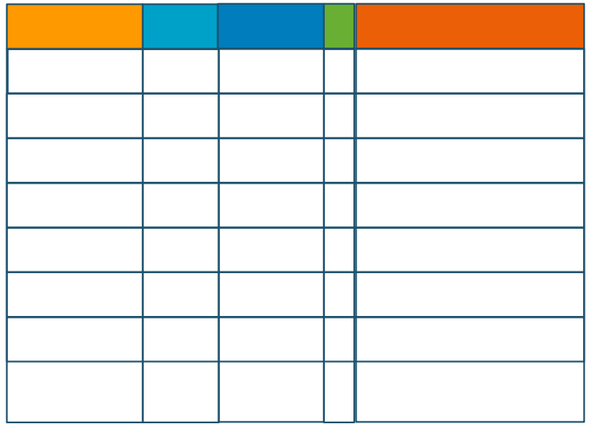


AWS Glue : Authoring – DataFrames and DynamicFrames

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

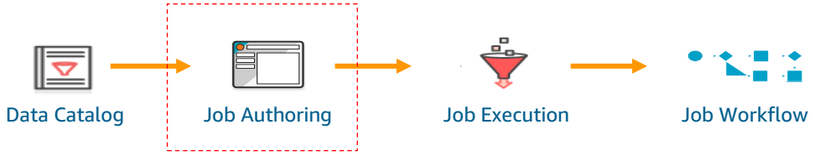




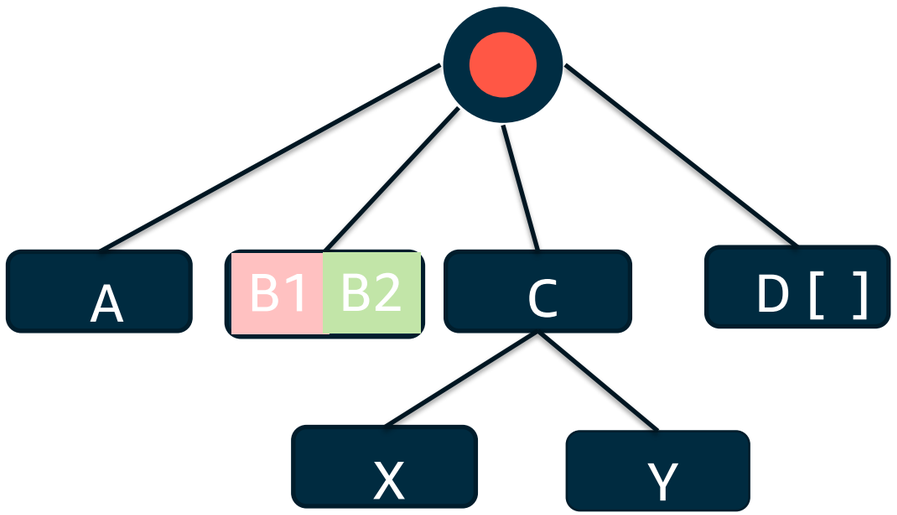
Glue DynamicFrames

- 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:

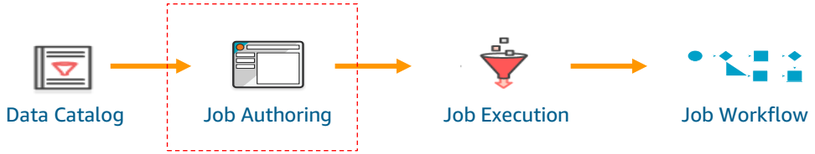
- Cleaning and (re)-structuring **semi-structured** data sets, e.g. JSON, Avro, Apache logs …

No upfront schema needed:

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

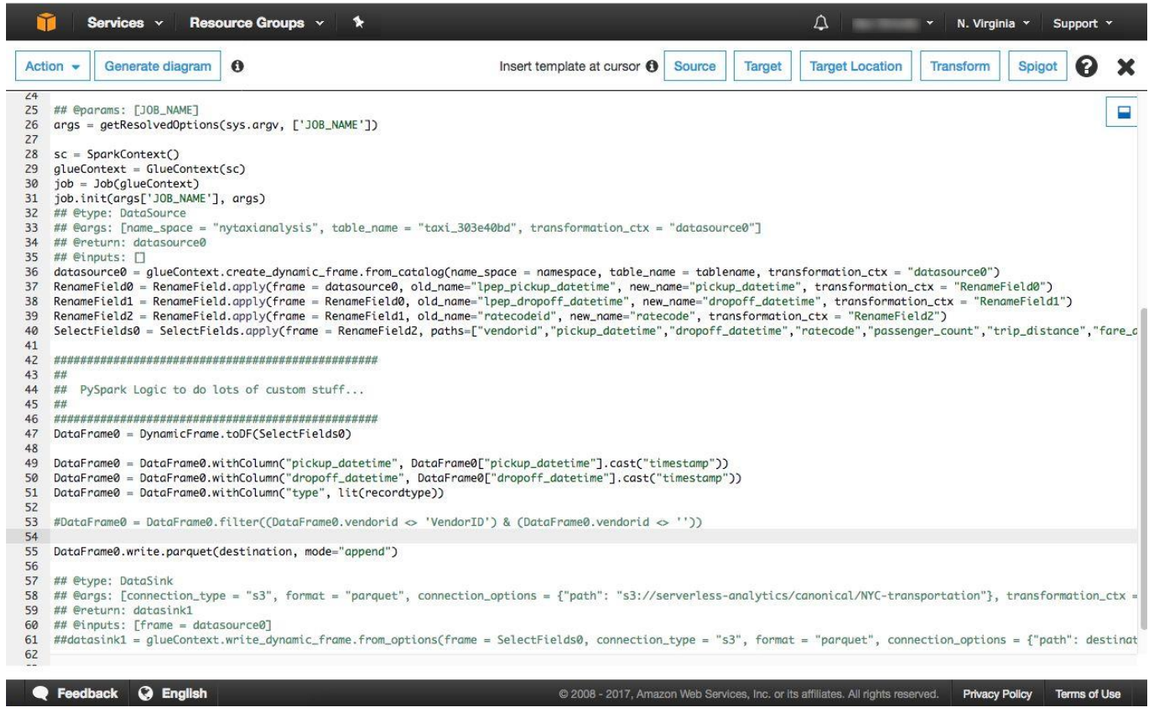
Easy to handle the unexpected:

- 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**




```
## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
## @type: DataSource
## @args: [name_space = "nytaxianalysis", table_name = "taxi_303e40bd", transformation_ctx = "datasource0"]
## @return: datasource0
## @inputs: []
datasource0 = glueContext.create_dynamic_frame.from_catalog(name_space = namespace, table_name = tablename, transformation_ctx = "datasource0")
RenameField0 = RenameField.apply(frame = datasource0, old_name="lpep_pickup_datetime", new_name="pickup_datetime", transformation_ctx = "RenameField0")
RenameField1 = RenameField.apply(frame = RenameField0, old_name="lpep_dropoff_datetime", new_name="dropoff_datetime", transformation_ctx = "RenameField1")
RenameField2 = RenameField.apply(frame = RenameField1, old_name="ratecodeid", new_name="ratecode", transformation_ctx = "RenameField2")
SelectFields0 = SelectFields.apply(frame = RenameField2, paths=["vendorid","pickup_datetime","dropoff_datetime","ratecode","passenger_count","trip_distance","fare_c

##################################################
##
##  PySpark Logic to do lots of custom stuff...
##
##################################################
DataFrame0 = DynamicFrame.toDF(SelectFields0)

DataFrame0 = DataFrame0.withColumn("pickup_datetime", DataFrame0["pickup_datetime"].cast("timestamp"))
DataFrame0 = DataFrame0.withColumn("dropoff_datetime", DataFrame0["dropoff_datetime"].cast("timestamp"))
DataFrame0 = DataFrame0.withColumn("type", lit(recordtype))

#DataFrame0 = DataFrame0.filter((DataFrame0.vendorid <> 'VendorID') & (DataFrame0.vendorid <> ''))

DataFrame0.write.parquet(destination, mode="append")

## @type: DataSink
## @args: [connection_type = "s3", format = "parquet", connection_options = {"path": "s3://serverless-analytics/canonical/NYC-transportation"}, transformation_ctx =
## @return: datasink1
## @inputs: [frame = datasource0]
##datasink1 = glueContext.write_dynamic_frame.from_options(frame = SelectFields0, connection_type = "s3", format = "parquet", connection_options = {"path": destinat
```

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

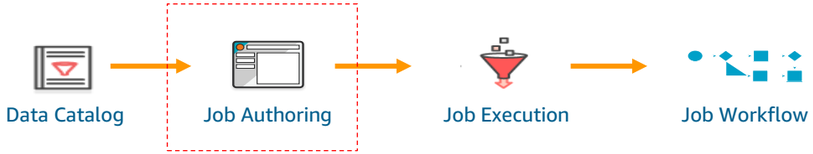
toDF

`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

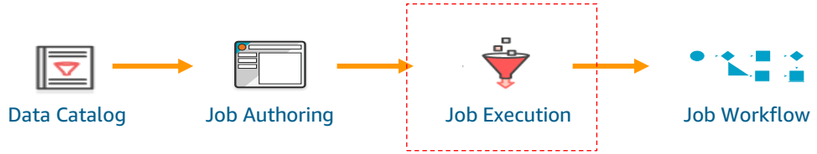
Visual | Script | Job details | Runs | Schedules

Recent job runs (2) Info

July 21, 2021 12:23:45 AM

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

- 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.

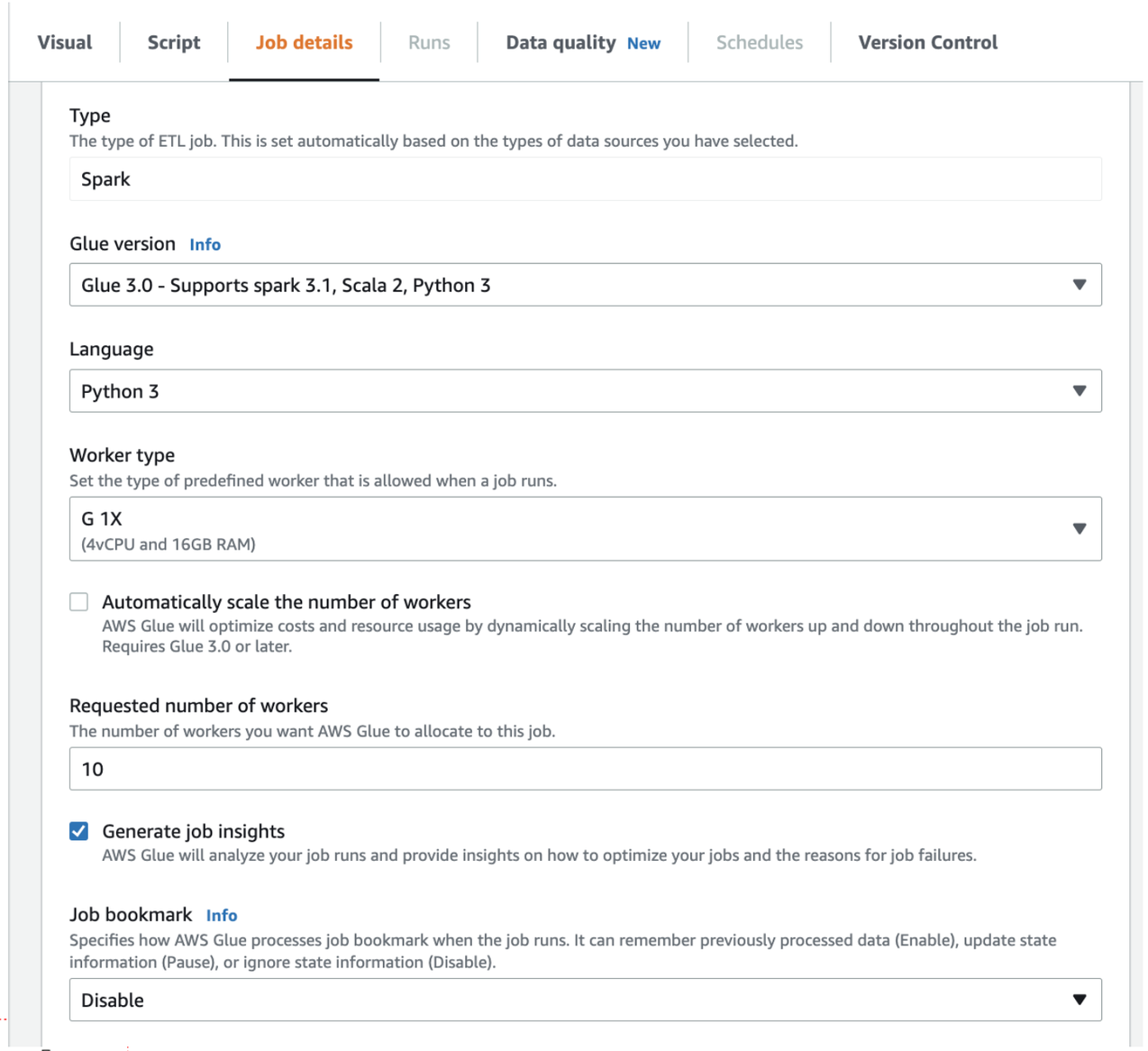


AWS Glue: Job Bookmarks

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



AWS Glue: Monitoring and Notifications

External

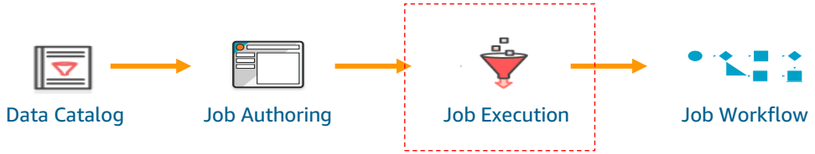
Publish crawler and job notifications into **Amazon CloudWatch**.
Use Amazon CloudWatch events to control downstream workflows

Conditions

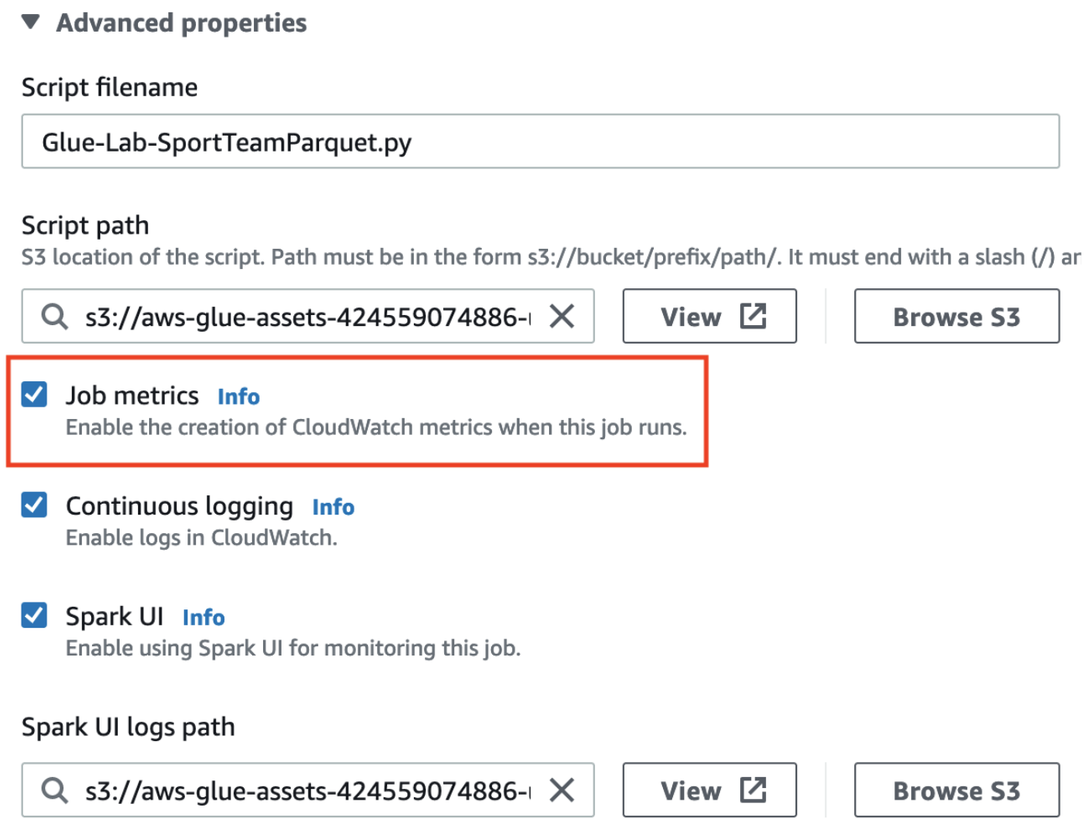
'**ANY**' and '**ALL**' operators in Trigger conditions
Additional job states **'failed'**, '**stopped**', or '**timeout**'

Control

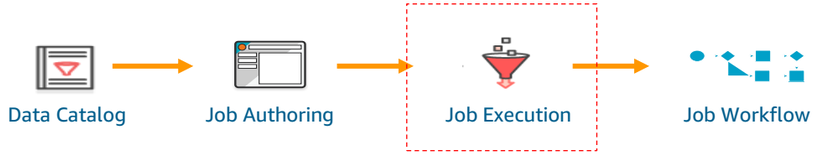
Configure **job timeout**
Job delay notifications



AWS Glue: Job Metrics

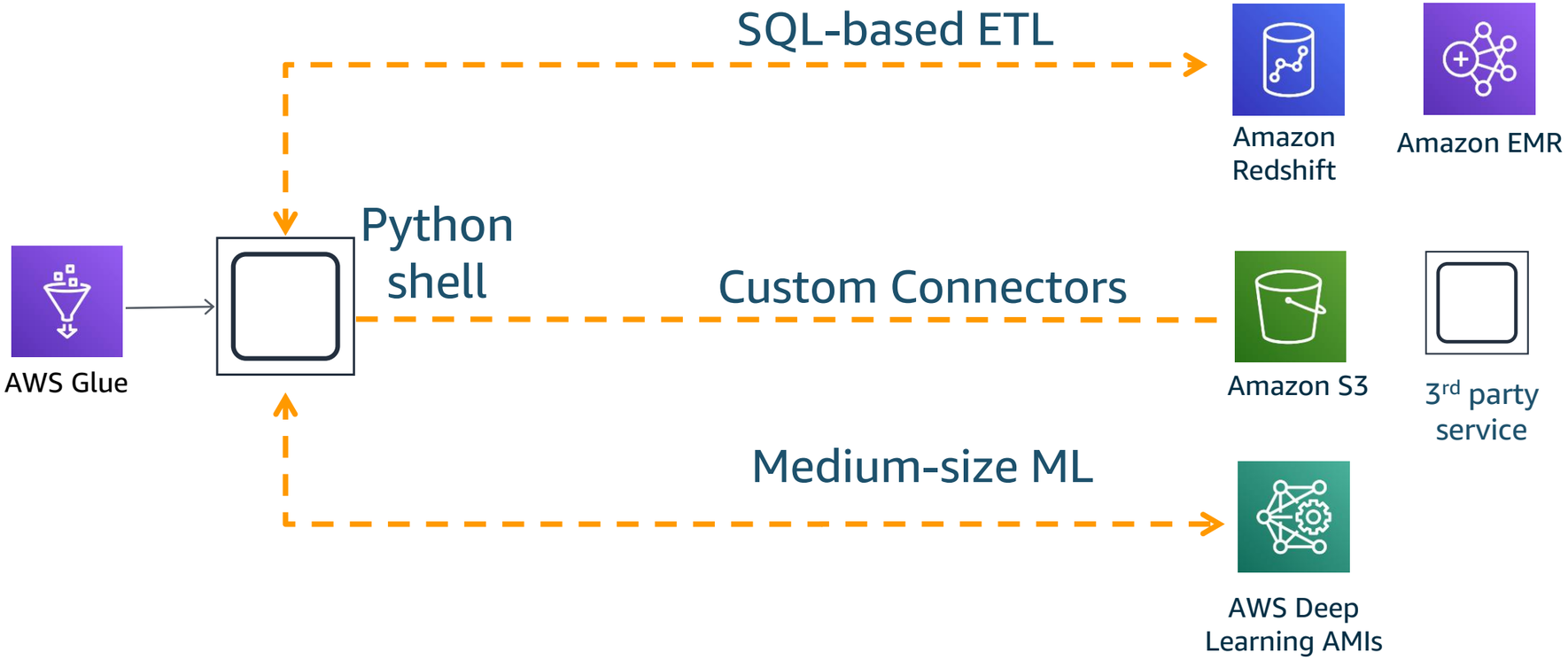


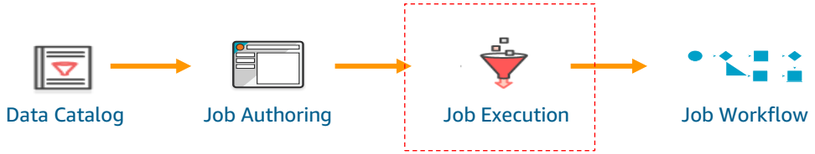
Metrics can be enabled in the AWS Command Line Interface (AWS CLI) and AWS SDK by passing --enable-metrics as a job parameter key.



AWS Glue: Python Shell Job

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

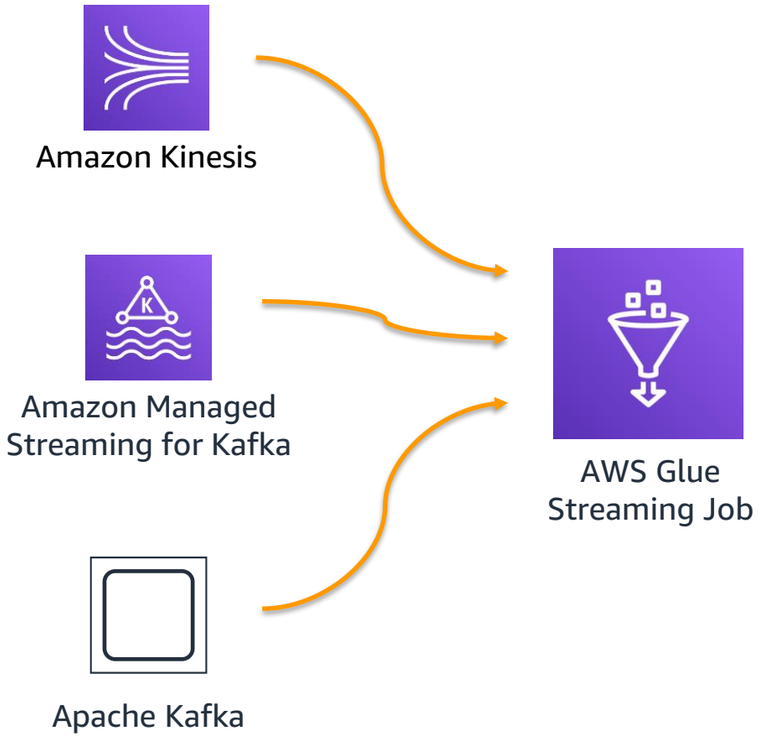


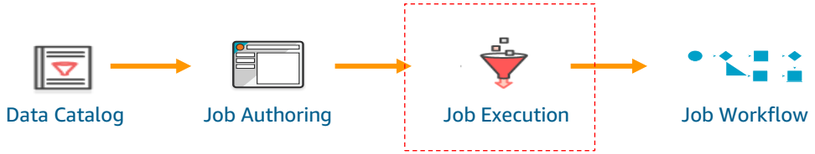


AWS Glue: Streaming ETL

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

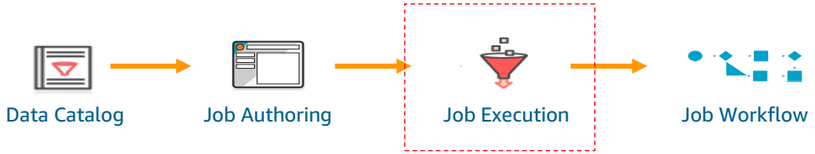
Make complex ETL pipelines significantly more resilient to errors using Workload Partitioning:

- 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:

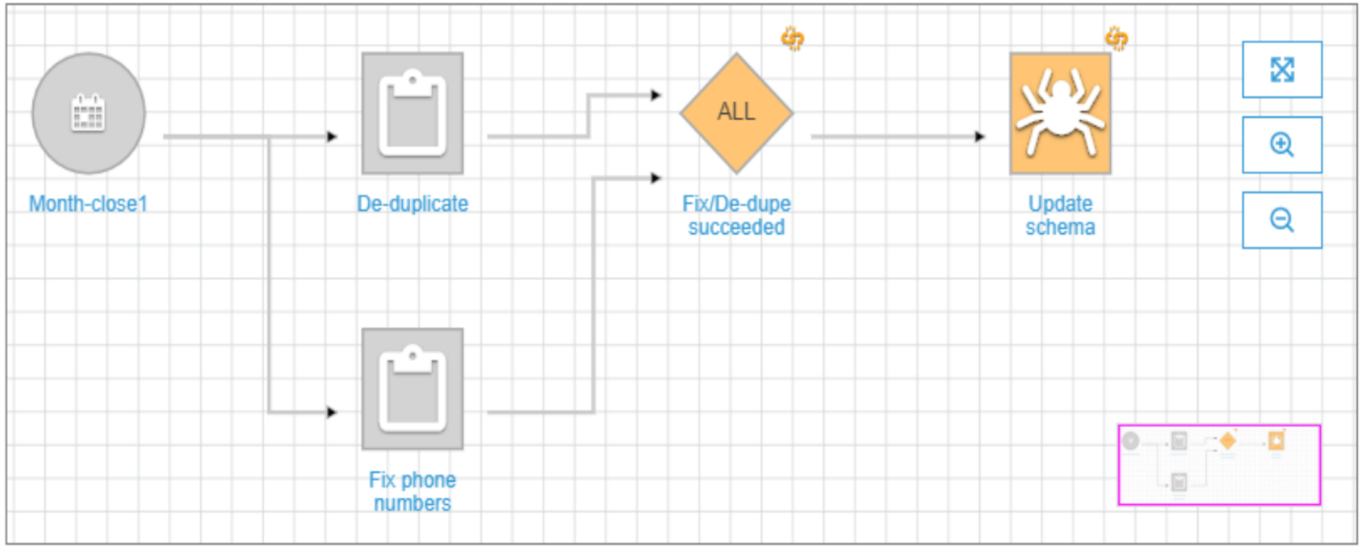
- Only read filtered partitions into your dynamic frames.
- Predicate expression can be any Boolean expression supported by Spark SQL



Glue Job Orchestration

AWS Glue: Job Orchestration Workflows

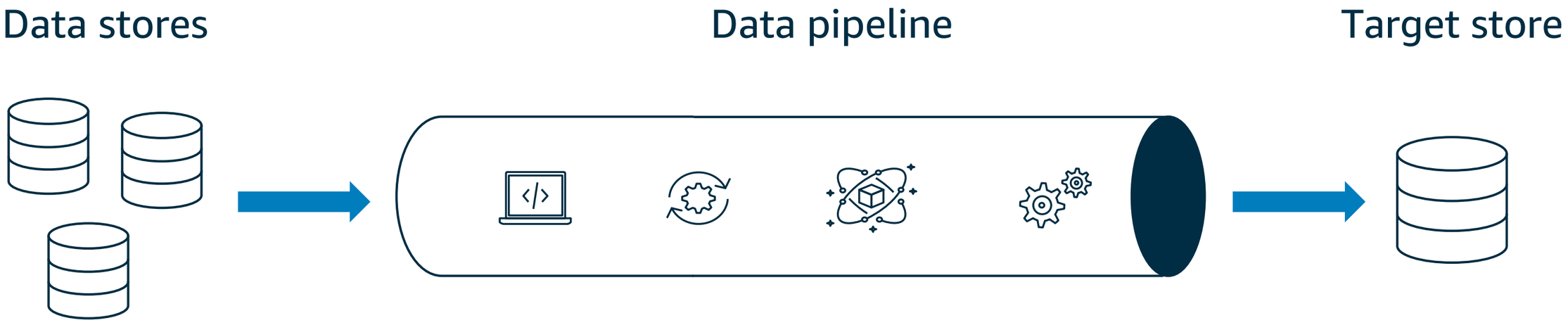
- Create and visualize complex ETL activities involving multiple crawlers, jobs, and triggers
- Records execution progress and status
- Provide both static and dynamic view
- Multiple triggering mechanisms
  - **Schedule-based:** e.g., time of day
  - **Event-based:** e.g., job completion
  - **On-demand:**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



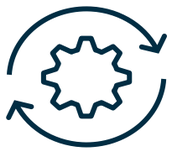
- Logs and alerts are available in Amazon CloudWatch

Data Cleansing, Transformation and Lineage with AWS Glue DataBrew

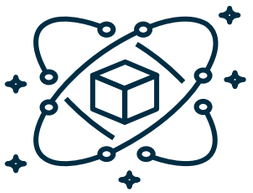
Combining and replicating data between multiple stores can be challenging



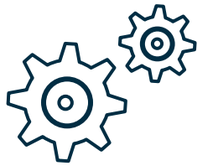
COMPLEX APPLICATION CODE



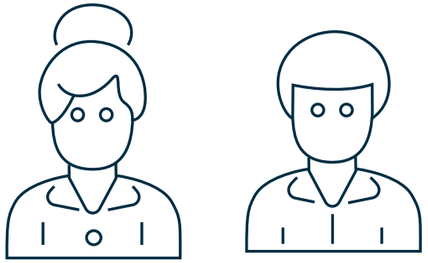
CUMBERSOME RETRY LOGIC



PIPELINE MANAGEMENT



NEED SPECIALIZED ETL SKILLS



BUSINESS ANALYST

DATA SCIENTIST

AWS Glue DataBrew

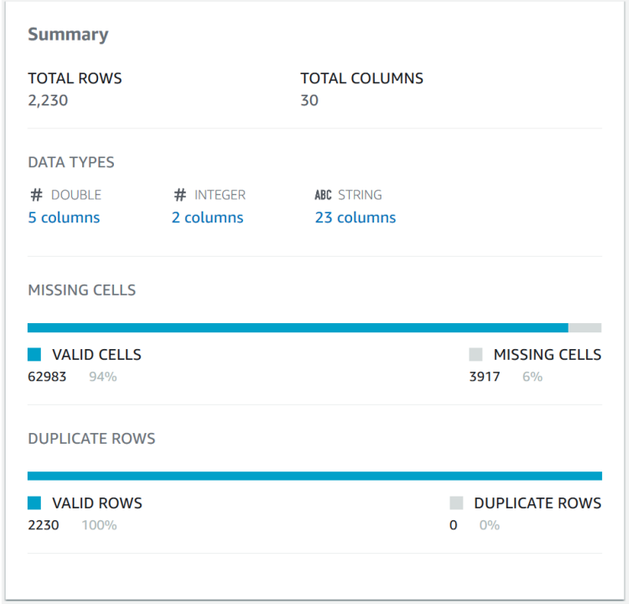
Clean and normalize data with a rich visual interface

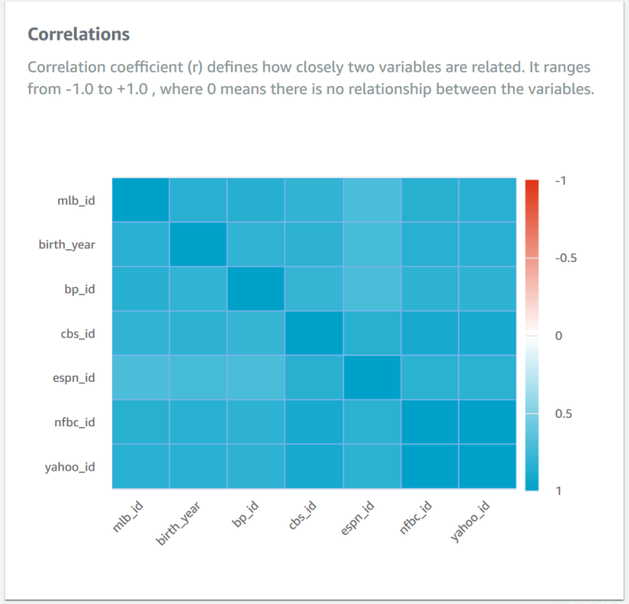
Choose from 250+ built-in transformations to automate tasks

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…





The picture can't be displayed.

Thank you!

Justin Yenser

qyensjus@amazon.com