



Working Within the Data Lake

With AWS Glue

Team or presenters name

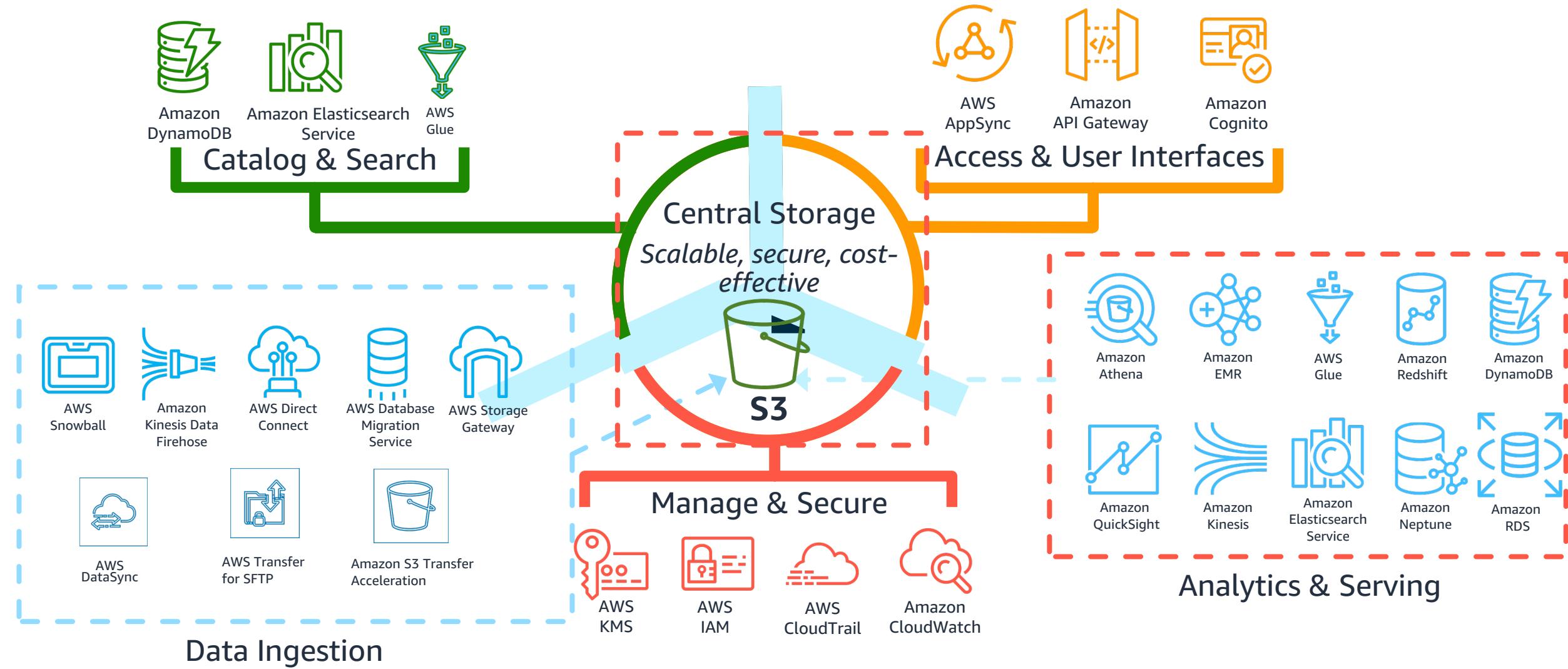
Date



Table of contents

1. Optimizing for Cost and Performance
2. Cataloging Data Schemas with AWS Glue
3. Transforming Data with AWS Glue
4. AWS Glue ML Transform and Workflows

Session's Focus – Working In The Data Lake



Optimizing for Cost and Performance

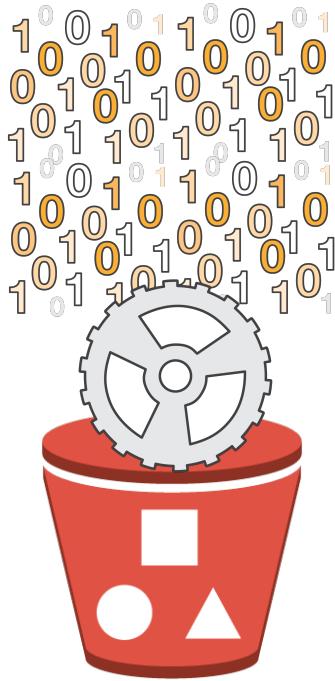


Optimizing for Cost and Performance

```
/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

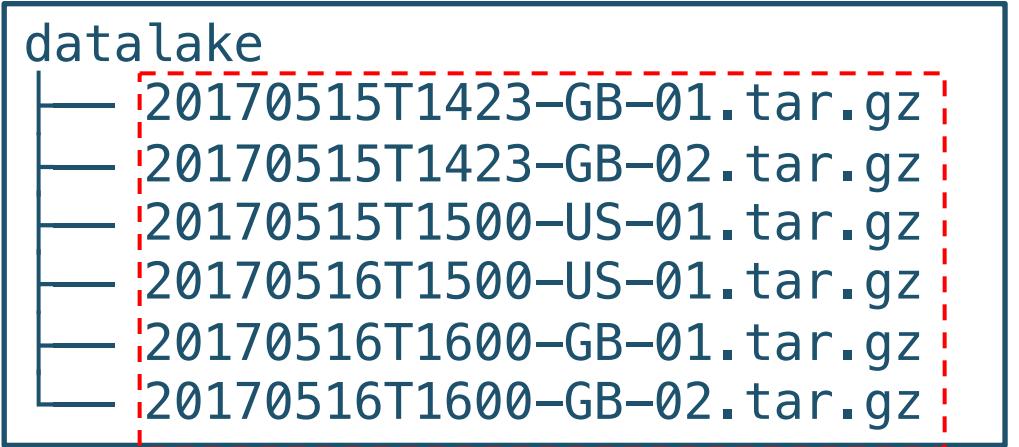
Pay for what you **store**,
not for what you **process**



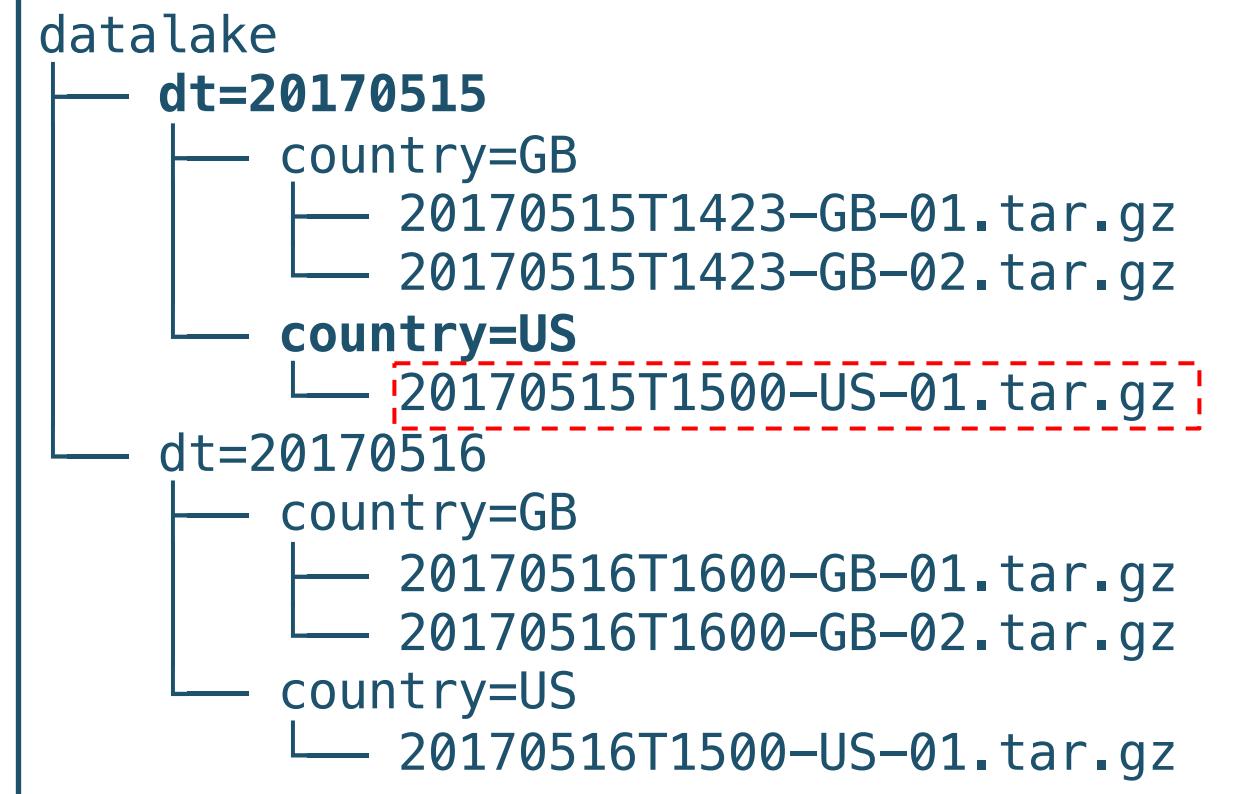
Managed Services

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

Partitioning



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



Partitioning - Advantages

	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	

Compression

- Compressing your data can speed up your queries significantly
- Splittable 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

Compression – File Counts

Fewer, larger files are better than many, smaller files (when splittable)

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

Query	# files	Run Time
select count(*) from datalake	5000 files	8.4 sec
select count(*) from datalake	1 file	2.31 sec
Result		72% Faster

Cataloging and Transforming Data

Working with AWS Glue

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 !!!



AWS Glue

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

Develop Generate code to clean, enrich, and reliably move data between various data sources

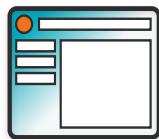
Deploy Run your jobs on a serverless, fully managed, scale-out environment. No compute resources to provision or manage.

AWS Glue: Components



Data Catalog

- Hive Metastore compatible with enhanced functionality
- Crawlers automatically extracts metadata and creates tables
- Integrated with Amazon Athena, Amazon EMR and many more



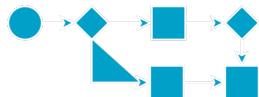
Job Authoring

- Auto-generates ETL code
- Build on open frameworks – Python/Scala and Apache Spark
- Developer-centric – editing, debugging, sharing



Job Execution

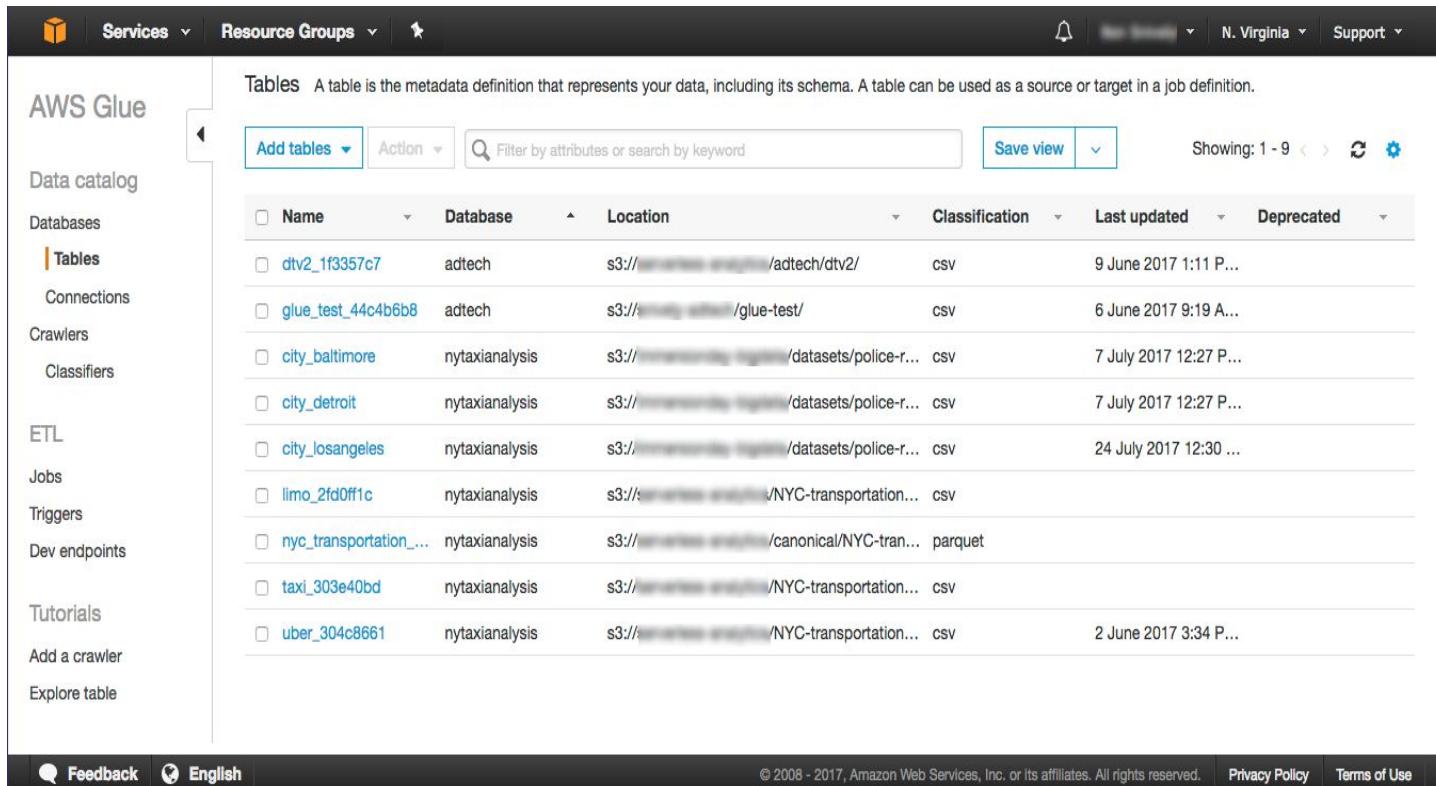
- Run jobs on a serverless Spark platform
- Provides flexible scheduling , Job monitoring and alerting



Job Workflow

- Orchestrate triggers, crawlers & jobs
- Author & monitor entire flows and Integrated alerting

Glue: Data Catalog



The screenshot shows the AWS Glue Data Catalog interface. On the left, there's a sidebar with navigation links: AWS Glue, Data catalog, Databases, Tables (which is selected), Connections, Crawlers, Classifiers, ETL, Jobs, Triggers, Dev endpoints, Tutorials, Add a crawler, and Explore table. The main area is titled "Tables" with a sub-instruction: "A table is the metadata definition that represents your data, including its schema. A table can be used as a source or target in a job definition." It includes buttons for "Add tables", "Action", "Save view", and a search bar. Below is a table listing nine tables:

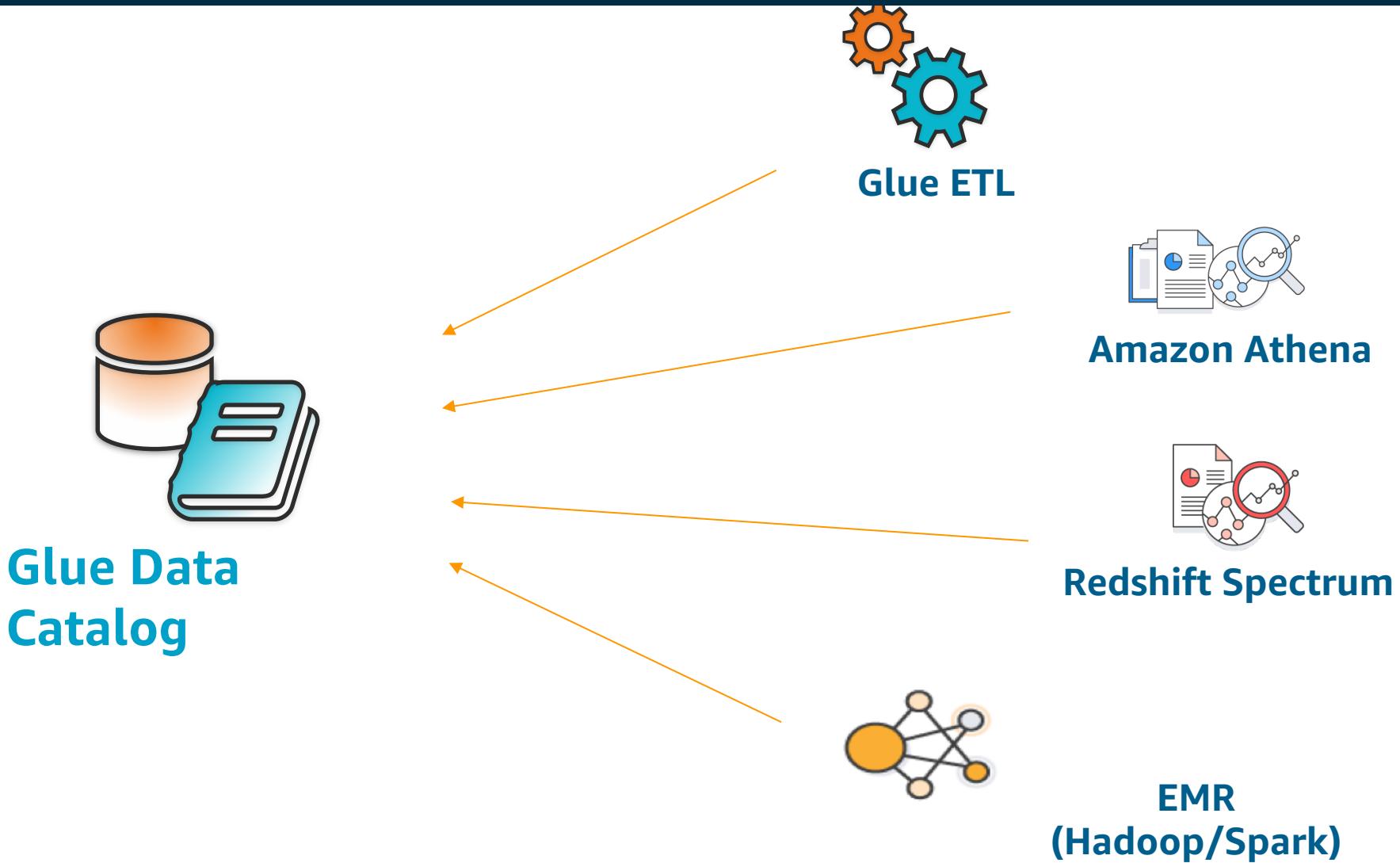
Name	Database	Location	Classification	Last updated
dtv2_1f3357c7	adtech	s3://.../adtech/dtv2/	csv	9 June 2017 1:11 P...
glue_test_44c4b6b8	adtech	s3://.../glue-test/	csv	6 June 2017 9:19 A...
city_baltimore	nytaxianalysis	s3://.../datasets/police-r...	csv	7 July 2017 12:27 P...
city_detroit	nytaxianalysis	s3://.../datasets/police-r...	csv	7 July 2017 12:27 P...
city_losangeles	nytaxianalysis	s3://.../datasets/police-r...	csv	24 July 2017 12:30 ...
limo_2fd0ff1c	nytaxianalysis	s3://.../NYC-transportation...	csv	
nyc_transportation_...	nytaxianalysis	s3://.../canonical/NYC-tran...	parquet	
taxi_303e40bd	nytaxianalysis	s3://.../NYC-transportation...	csv	
uber_304c8661	nytaxianalysis	s3://.../NYC-transportation...	csv	2 June 2017 3:34 P...

At the bottom, there are links for Feedback, English, and legal notices: © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved., Privacy Policy, and Terms of Use.

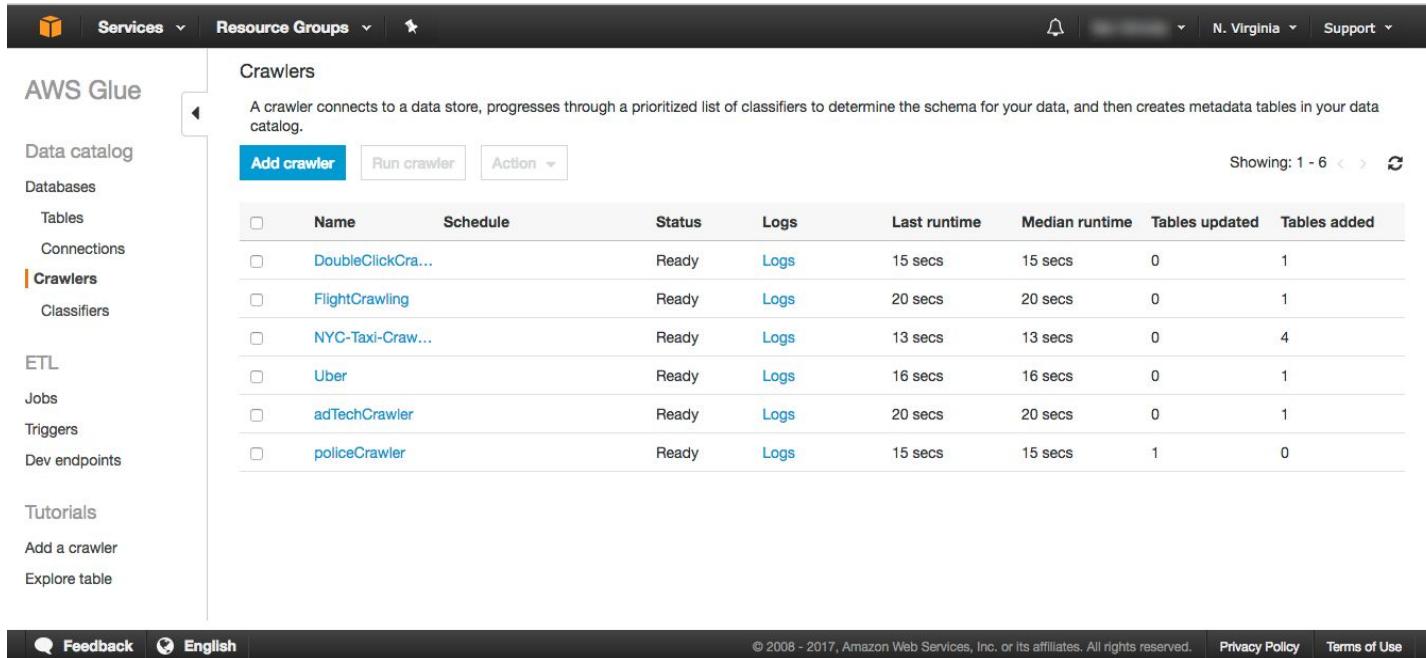
Features Include

- **Search** metadata for data discovery
- **Connections** to S3, RDS, Redshift, JDBC, and DynamoDB
- **Classification** for identifying and parsing files
- **Versioning** of table metadata as schemas

Glue: Data Catalog – Queryable by Many Services



Glue: Data Catalog - Crawlers



The screenshot shows the AWS Glue Data Catalog - Crawlers interface. On the left, there's a sidebar with navigation links for Services, Resource Groups, AWS Glue, Data catalog, Databases, Tables, Connections, Crawlers (which is selected and highlighted in orange), Classifiers, ETL, Jobs, Triggers, Dev endpoints, Tutorials, Add a crawler, and Explore table. The main content area has a header "Crawlers" with a sub-header: "A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog." Below this are buttons for "Add crawler", "Run crawler", and "Action". A table lists six crawlers with columns: Name, Schedule, Status, Logs, Last runtime, Median runtime, Tables updated, and Tables added. The crawlers listed are: DoubleClickCra..., FlightCrawling, NYC-Taxi-Craw..., Uber, adTechCrawler, and policeCrawler. All crawlers are in a "Ready" status. The table also includes a "Showing: 1 - 6" link and a refresh icon.

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

Crawlers: Classifiers

Create additional Custom Classifiers
with Grok!



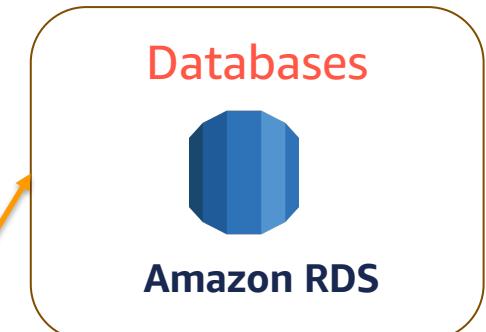
IAM Role

Glue Crawler



JDBC Connection

Object Connection



Built-In Classifiers

MySQL
MariaDB
PostgreSQL
Aurora

Redshift

Avro
Parquet
ORC
JSON & BSON

Logs
(Apache, Linux, MS, Ruby, Redis, and many others)

Delimited

(comma, pipe, tab, semicolon)

Compressed Formats
(ZIP, BZIP, GZIP, LZ4, Snappy)

Data Catalog: Table details

Table properties

Data statistics

Table schema

AWS Glue Data catalog

Tables > simpletweets_json

Last updated 10 Aug 2017 Table Version (Current version)

Edit table Delete table View properties Compare versions Edit schema

Name: simpletweets_json
Description: analytics
Database: analytics
Classification: json
Location: s3://gluesampleddata/simpletweets.json
Connection: S3Crawler
Deprecated: No
Last updated: Thu Aug 10 16:25:24 GMT-700 2017

Properties: sizeKey 456580, objectCount 1, UPDATED_BY_CRAWLER, recordCount 1001, averageRecordSize 456, CrawlerSchemaDeserializerVersion 1.0, compressionType none, typeOfData file

Schema:

	Column name	Data type
1	entities	struct
2	id	bigint
3	retweeted	boolean
4	text	string
5	user	struct

user schema details

```
STRUCT
contributors_enabled:BOOLEAN
description:STRING
favourites_count:INT
followers_count:INT
friends_count:INT
id:INT
lang:STRING
location:STRING
name:STRING
profile_background_tile:BOOLEAN
```

Close

© 2020, Amazon Web Services, Inc. or its Affiliates.

Data Catalog: Version control

Compare schema versions

Last updated 10 Aug 2017 Table Version 0

Name	simpletweets_json
Description	
Database	simpletweets_json
Classification	json
Location	s3://gluesampleddata/simpletweets.json
Connection	
Deprecated	No
Last updated	Thu Aug 10 16:25:24 GMT-700 2017
sizeKey	456580
objectCount	1
UPDATED_BY_CRAWLER	S3Crawler
Properties	CrawlerSchemaSerializerVersion 1.0 recordCount 1001
averageRecordSize	456
CrawlerSchemaDeserializerVersion	1.0
compressionType	none
typeOfData	file

Change Column name Data type Key

Changed	entities	struct	
	id	bigint	
	retweeted	boolean	
	text	string	
	user	struct	

Last updated 10 Aug 2017 Table Version 1 (Current version)

Name	simpletweets_json
Description	
Database	simpletweets_json
Classification	json
Location	s3://gluesampleddata/simpletweets.json
Connection	
Deprecated	No
Last updated	Thu Aug 10 16:25:24 GMT-700 2017
Properties	CrawlerSchemaSerializerVersion 1.0 recordCount 1001
averageRecordSize	456
CrawlerSchemaDeserializerVersion	1.0
compressionType	none
typeOfData	file

Change Column name Data type Key

Changed	entities	struct	
	id	STRING	
	retweeted	boolean	
	text	string	
	user	struct	

List of table versions

Showing: 1 - 2 < >

Version	Created:	Created by:
1	10 August 2017 ...	arn:aws:sts::413... role:Glue_Default... Crawler
0	10 August 2017 ...	arn:aws:sts::413... role:Glue_Default... Crawler

Glue: Crawlers – Detecting Schema and Partitions

S3 bucket hierarchy

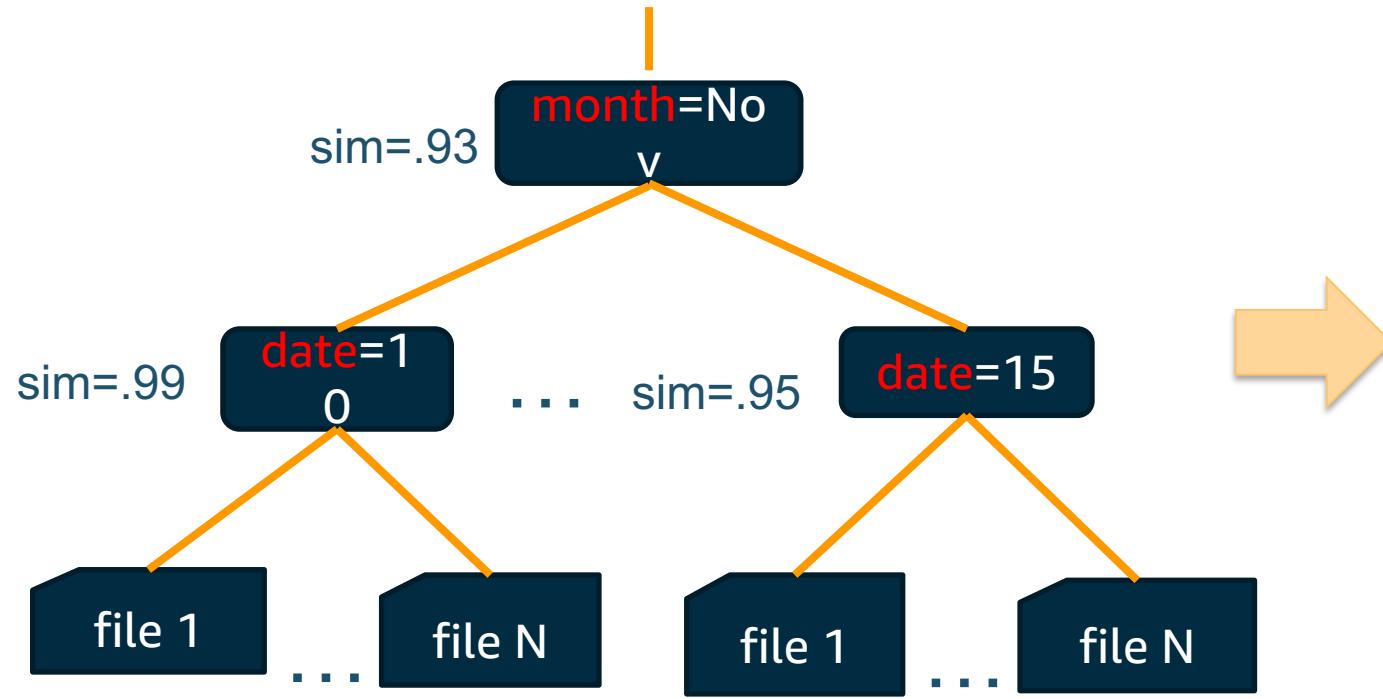


Table definition

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

Data Catalog: Automatic partition detection

12	errorcode	string	
13	region	string	Partition (0)
14	year	string	Partition (1)
15	month	string	Partition (2)
16	day	string	Partition (3)

Table partitions

Automatically register available partitions

Showing: 1 - 3 < >

region	year	month	day		
us-west-2	2016	02	18	View files	View properties
us-west-2	2016	02	17	View files	View properties
us-west-2	2016	02	16	View files	View properties

AWS Glue: Components



Data Catalog

- Hive Metastore compatible with enhanced functionality
- Crawlers automatically extracts metadata and creates tables
- Integrated with Amazon Athena, Amazon EMR and many more



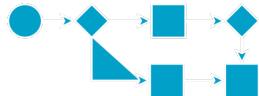
Job Authoring

- Auto-generates ETL code
- Build on open frameworks – Python/Scala and Apache Spark
- Developer-centric – editing, debugging, sharing



Job Execution

- Run jobs on a serverless Spark platform
- Provides flexible scheduling , Job monitoring and alerting



Job Workflow

- Orchestrate triggers, crawlers & jobs
- Author & monitor entire flows and Integrated alerting

AWS Glue: Job Authoring - Pick a Source

Job properties

GlueDemo

Data source

Data target

Schema

Review

Choose your data sources

Filter by attributes or search by keyword

Name	Database	Location	Classification
auroragluetestdb_sample_test_table	aurora_db	gluetestdb.sample_test_table	mysql

Choose from Manually-Defined or Crawler-Generated Sources:

- S3 Bucket
- RDBMS
- Redshift
- DynamoDB

AWS Glue: Job Authoring – Pick a Target

Write output results into an existing table.

The screenshot shows the AWS Glue Job Authoring interface. On the left, a sidebar lists steps: Job properties (checked), Data source (checked), Data target (checked), Schema, and Review. The main area is titled "Choose your data targets". It shows two radio button options: "Create tables in your data target" (unchecked) and "Use tables in the data catalog and update your data target" (checked). Below this is a search bar with placeholder text "Filter by attributes or search by keyword". A table lists a single table entry:

Name	Database	Location	Classification
auroragluetestdb_sample_test_table	aurora_db	gluetestdb.sample_test_table	mysql

At the bottom left, there is a panel with two radio button options: "Create tables in your data target" (checked) and "Use tables in the data catalog and update your data target" (unchecked). Below this are sections for "Data store" (set to "Amazon S3"), "Format" (set to "Parquet"), and "Target path" (set to "s3://my-bucket/").

Crawler-Defined or Manually-Created:

- **S3 Bucket**
- **RDBMS**
- **Redshift**

AWS Glue: Job Authoring – Apply Transformation

The screenshot shows the AWS Glue "Add job: schema" interface. On the left, there's a sidebar with checked items: "Job properties" (DeleteMe), "Data source" (city_baltimore), "Data target" (canonical), and "Schema". Below these are "Add column" and "Review" buttons. The main area has two tables: "Column name" and "Data type" for both source and target. Arrows indicate mappings between columns. The source table includes columns like crimedate, crimetime, crimecode, location, description, inside/outside, weapon, post, district, neighborhood, location 1, premise, and total incidents. The target table includes columns like crimedate, crime_time, crimecode, location, description, weapon, district, neighborhood, and total incidents. A "Map to target" dropdown is present for each source column.

Column name	Data type	Map to target
crimedate	string	crimedate
crimetime	string	crime_time
crimecode	string	crimecode
location	string	location
description	string	description
inside/outside	string	-
weapon	string	weapon
post	bigint	-
district	string	district
neighborhood	string	neighborhood
location 1	string	-
premise	string	-
total incidents	bigint	total incidents

Column name	Data type
crimedate	string
crime_time	string
crimecode	string
location	string
description	string
weapon	string
district	string
neighborhood	string
total incidents	long

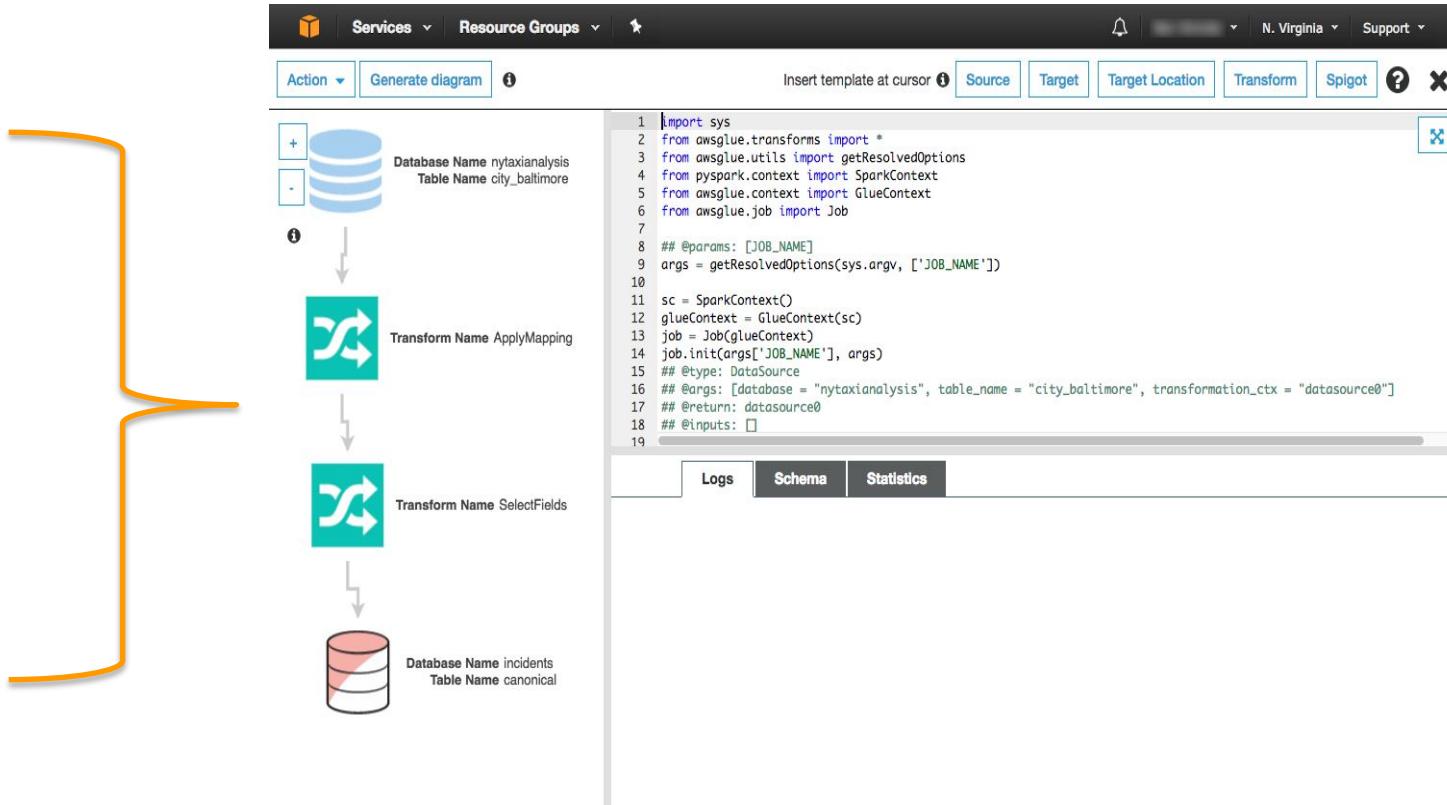
Existing columns
in target

Can extend/add
new columns to
target

AWS Glue: Job Authoring – Code Generation

The screenshot shows the 'Add job: schema' interface in AWS Glue. It displays two sets of columns with their data types and mapping options. A large orange curly brace on the left side of the diagram indicates that the entire schema mapping section is generated from the transformation graph shown above.

Column name	Data type	Map to target	Column name	Data type
crimedate	string	crimedate	crimedate	string
crimetype	string	crime_time	crime_time	string
crimecode	string	crimecode	crimecode	string
location	string	location	location	string
description	string	description	description	string
inside/outside	string	-	weapon	string
weapon	string	weapon	district	string
post	bigint	-	neighborhood	string
district	string	district	total incidents	long
neighborhood	string	neighborhood		
location 1	string	-		
premise	string	-		
total incidents	bigint	total incidents		



1. Customize the mappings
2. Glue generates transformation graph and either *Python* or *Scala* code
3. Specify trigger condition

Job authoring: ETL code

- Human-readable, editable, and portable PySpark or Scala code

```
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")
```

- Flexible: Glue's ETL library simplifies manipulating complex, semi-structured data
- Customizable: Use native PySpark / Scala, import custom libraries, and/or leverage Glue's libraries

```
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))
```

- Collaborative: share code snippets via GitHub, reuse code across jobs

AWS Glue: Job Authoring – Import Custom Modules

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

Parameters (optional)

▶ Advanced properties

▼ Script libraries and job parameters (optional)

Server-side encryption

Python library path

s3://bucket-name/folder-name/file-name

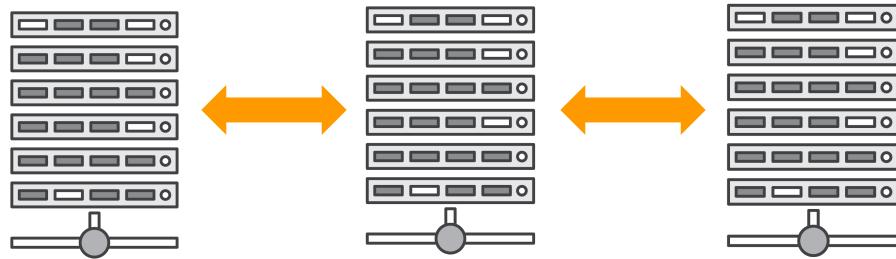
Dependent jars path

s3://bucket-name/folder-name/file-name

Referenced files path

s3://bucket-name/folder-name/file-name

Apache Spark and AWS 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, ...

SparkSQL	AWS Glue ETL
Dataframes	Dynamic Frames
Spark core: RDDs	

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

Dataframes and Dynamic Frames

Orange	Cyan	Blue	Green	Orange

Dataframes

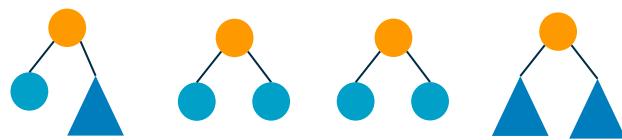
Core data structure for SparkSQL

Like structured tables

Need schema up-front

Each row has same structure

Suited for SQL-like analytics



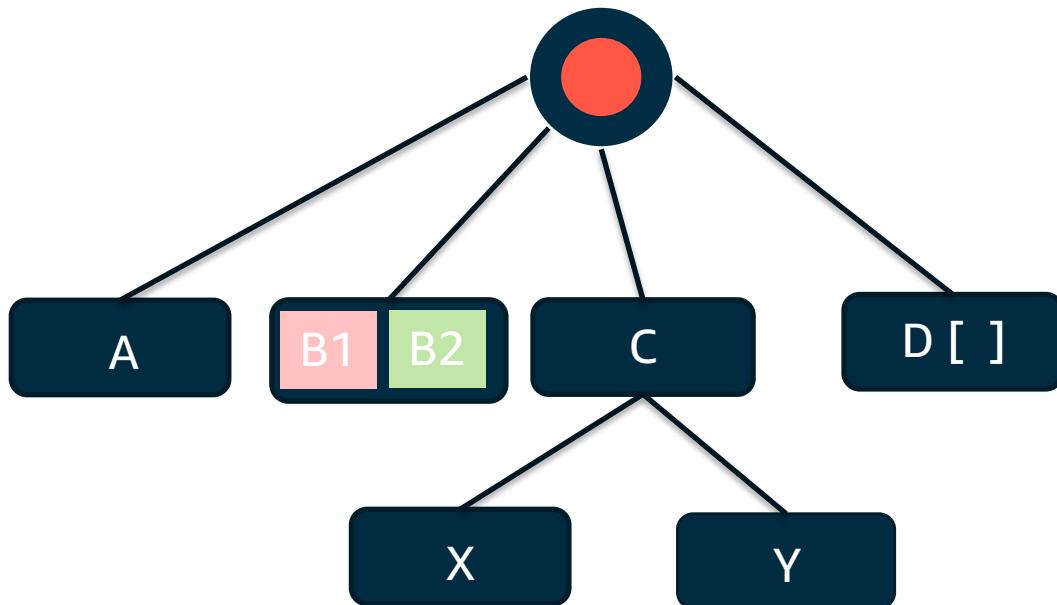
Dynamic Frames

Like dataframes for ETL

Designed for processing **semi-structured** data,

e.g. JSON, Avro, Apache logs ...

Job Authoring: Glue Dynamic Frames



Like Spark's Data Frames, but better for:

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

No upfront schema needed:

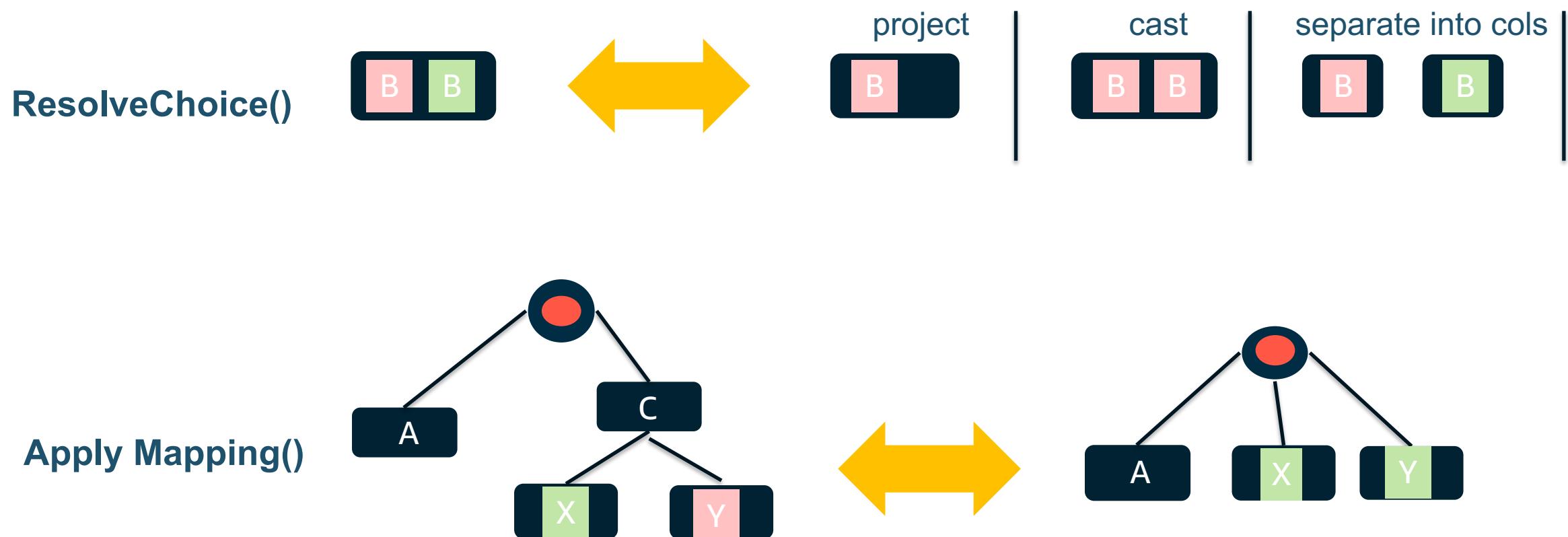
- Infers schema on-the-fly, enabling transformations in a **single pass**

Easy to handle the unexpected:

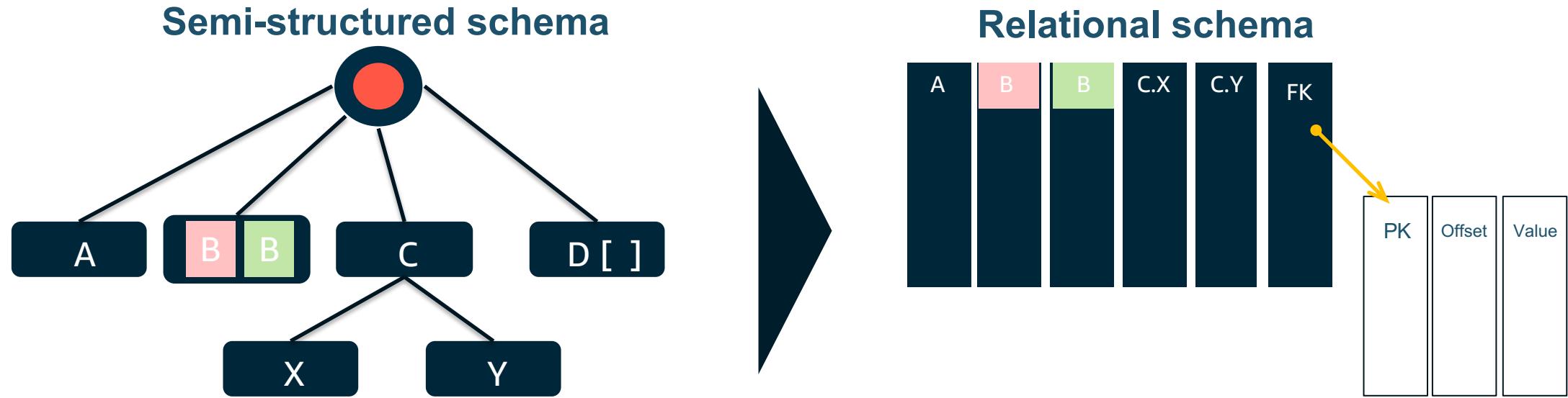
- Tracks new fields, and inconsistent changing data types with **choices**, e.g. integer or string
- Automatically mark and separate error records

Job Authoring: Glue transforms

Adaptive and flexible



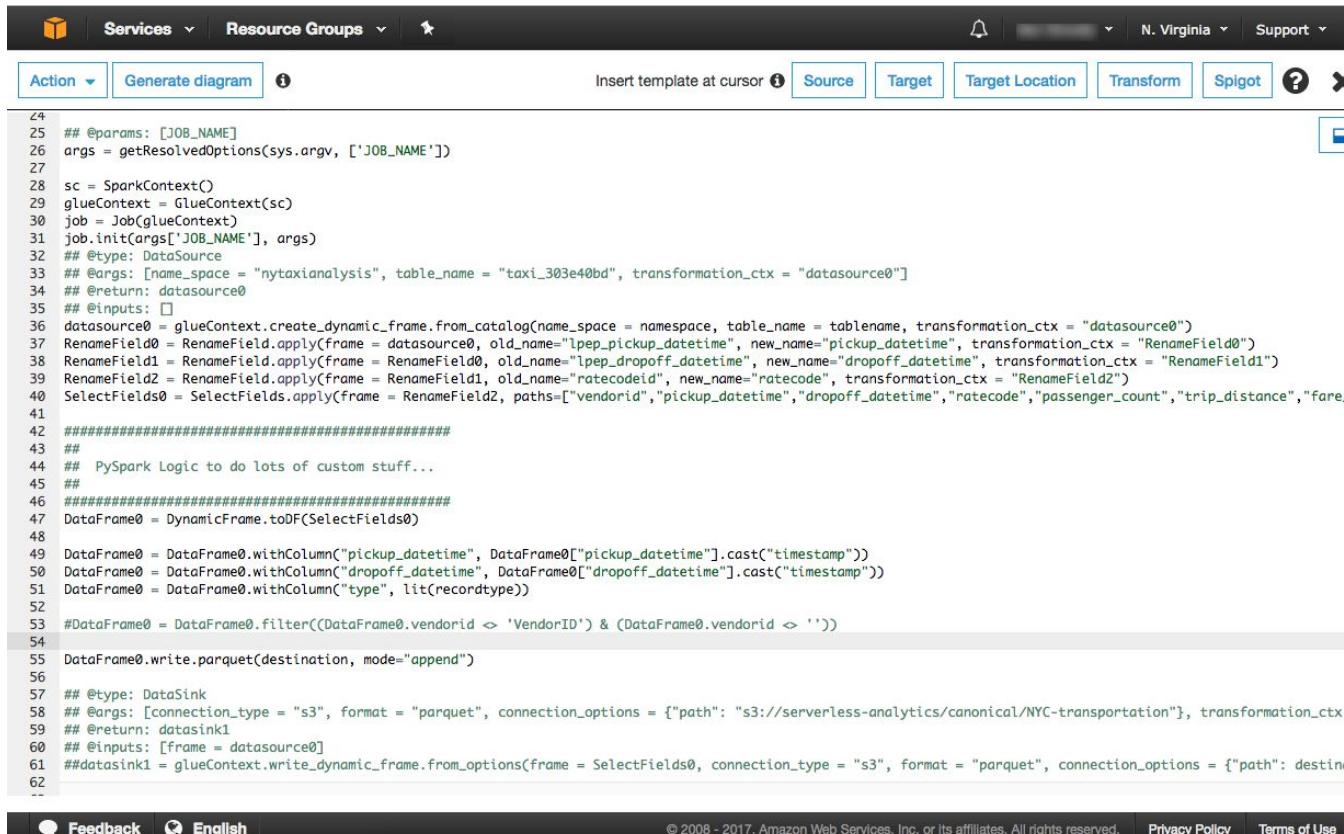
Job authoring: Relationalize() transform



- Transforms and **adds new** columns, types, and tables on-the-fly
- Tracks **keys** and **foreign keys** across runs
- SQL on the relational schema is orders of **magnitude faster** than JSON processing

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

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



The screenshot shows the AWS Glue Job Editor interface. At the top, there are tabs for Action, Generate diagram, and a help icon. Below the tabs, there are buttons for Source, Target, Target Location, Transform, Spigot, and another help icon. The main area contains a Python script for job authoring:

```
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_amount"])
41
42 ##### PySpark Logic to do lots of custom stuff...
43
44 DataFrame0 = DynamicFrame.toDF(SelectFields0)
45
46 DataFrame0 = DataFrame0.withColumn("pickup_datetime", DataFrame0["pickup_datetime"].cast("timestamp"))
47 DataFrame0 = DataFrame0.withColumn("dropoff_datetime", DataFrame0["dropoff_datetime"].cast("timestamp"))
48 DataFrame0 = DataFrame0.withColumn("type", lit(recordtype))
49
50 #DataFrame0 = DataFrame0.filter((DataFrame0.vendorid > 'VendorID') & (DataFrame0.vendorid < ''))
51
52 DataFrame0.write.parquet(destination, mode="append")
53
54 ## @type: DataSink
55 ## @args: [connection_type = "s3", format = "parquet", connection_options = {"path": "s3://serverless-analytics/canonical/NYC-transportation"}, transformation_ctx =
56 ## @return: datasink1
57 ## @inputs: [frame = datasource0]
58 ## datasink1 = glueContext.write_dynamic_frame.from_options(frame = SelectFields0, connection_type = "s3", format = "parquet", connection_options = {"path": destinat
59
60
61
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 Authoring – “DynamicFrame” High-Level API

Add transform

Name	Description
<input type="radio"/> DropFields	Drop fields from a DynamicFrame
<input type="radio"/> DropNullFields	DynamicFrame without null fields
<input type="radio"/> Join	Join two DynamicFrames
<input type="radio"/> MapToCollection	Apply a transform to each DynamicFrame in this DynamicFrameCollection
<input type="radio"/> Relationalize	Flatten nested schema and pivot out array columns from the flattened frame
<input type="radio"/> RenameField	Rename a field within a DynamicFrame
<input type="radio"/> SelectFields	Select fields from a DynamicFrame
<input type="radio"/> SelectFromCollection	Select one DynamicFrame from a DynamicFrameCollection
<input type="radio"/> SplitFields	Split fields within a DynamicFrame
<input type="radio"/> SplitRows	Split rows within a DynamicFrame based on comparators
<input type="radio"/> Unbox	Unbox a string field

- **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: Components



Data Catalog

- Hive Metastore compatible with enhanced functionality
- Crawlers automatically extracts metadata and creates tables
- Integrated with Amazon Athena, Amazon EMR and many more



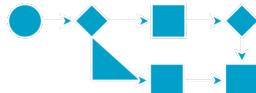
Job Authoring

- Auto-generates ETL code
- Build on open frameworks – Python/Scala and Apache Spark
- Developer-centric – editing, debugging, sharing



Job Execution

- Run jobs on a serverless Spark platform
- Provides flexible scheduling , Job monitoring and alerting



Job Workflow

- Orchestrate triggers, crawlers & jobs
- Author & monitor entire flows and Integrated alerting

AWS Glue Orchestration

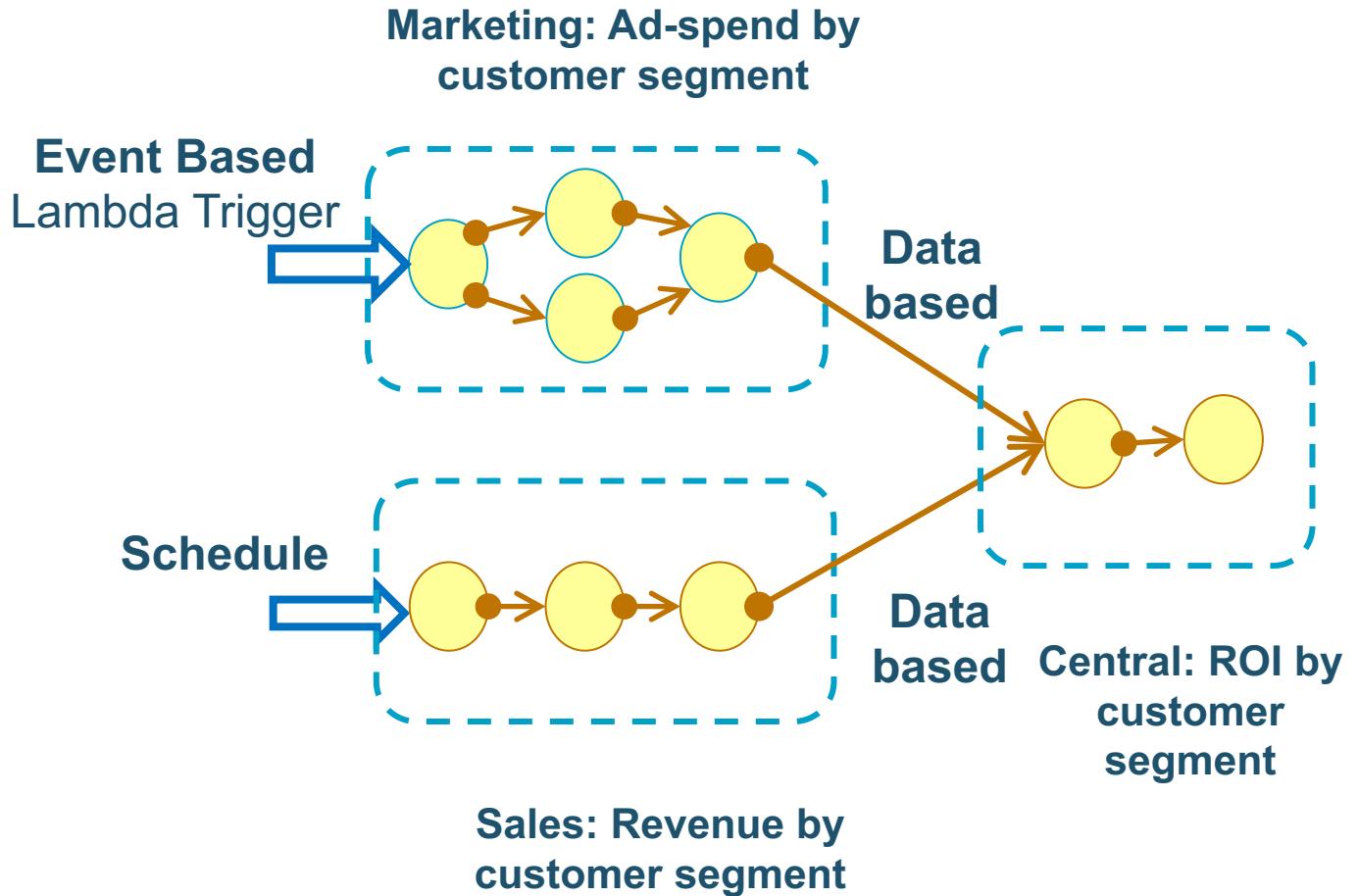
Compose jobs globally with event-based dependencies

- Easy to reuse and leverage work across organization boundaries

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

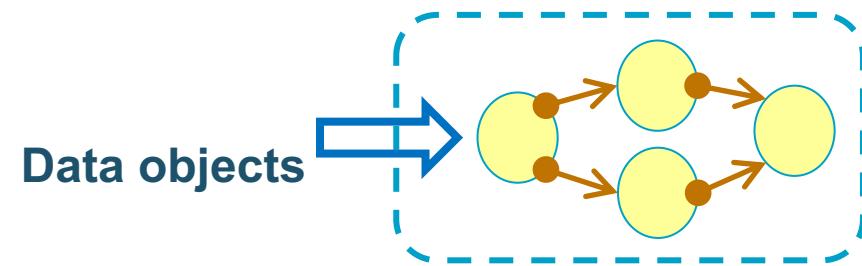


Job execution: Job bookmarks

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**.

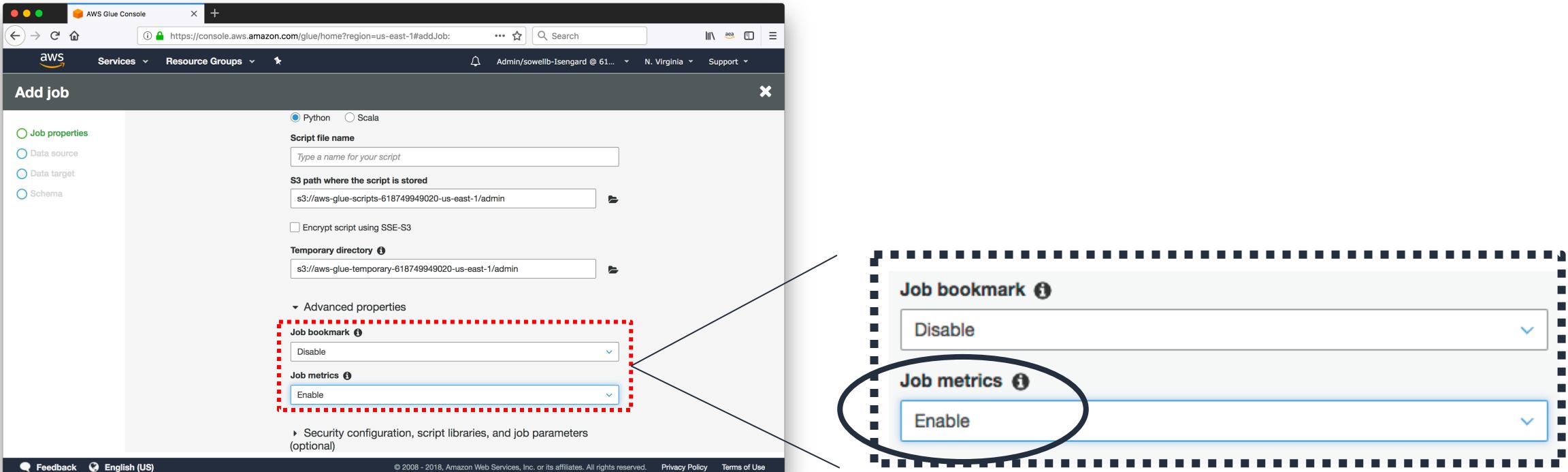
Option	Behavior
Enable	Pick up from where you left off
Disable	Ignore and process the entire dataset every time
Pause	Temporarily disable advancing the bookmark

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.



Marketing: Ad-spend by customer segment

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: Job Execution – Progress and History

Run ID	Retry attempt	Run status	Logs	Errors	Duration
jr_5cc040d697b...	-	Starting	Logs	Error logs	0 secs
jr_5c8fb8846bd...	-	Succeeded	Logs	Error logs	4 mins
!	jr_77c546040fe...	Failed	Logs	Error logs	2 mins

- 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.

Job Monitoring and Notification

External

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

Conditions

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

Control

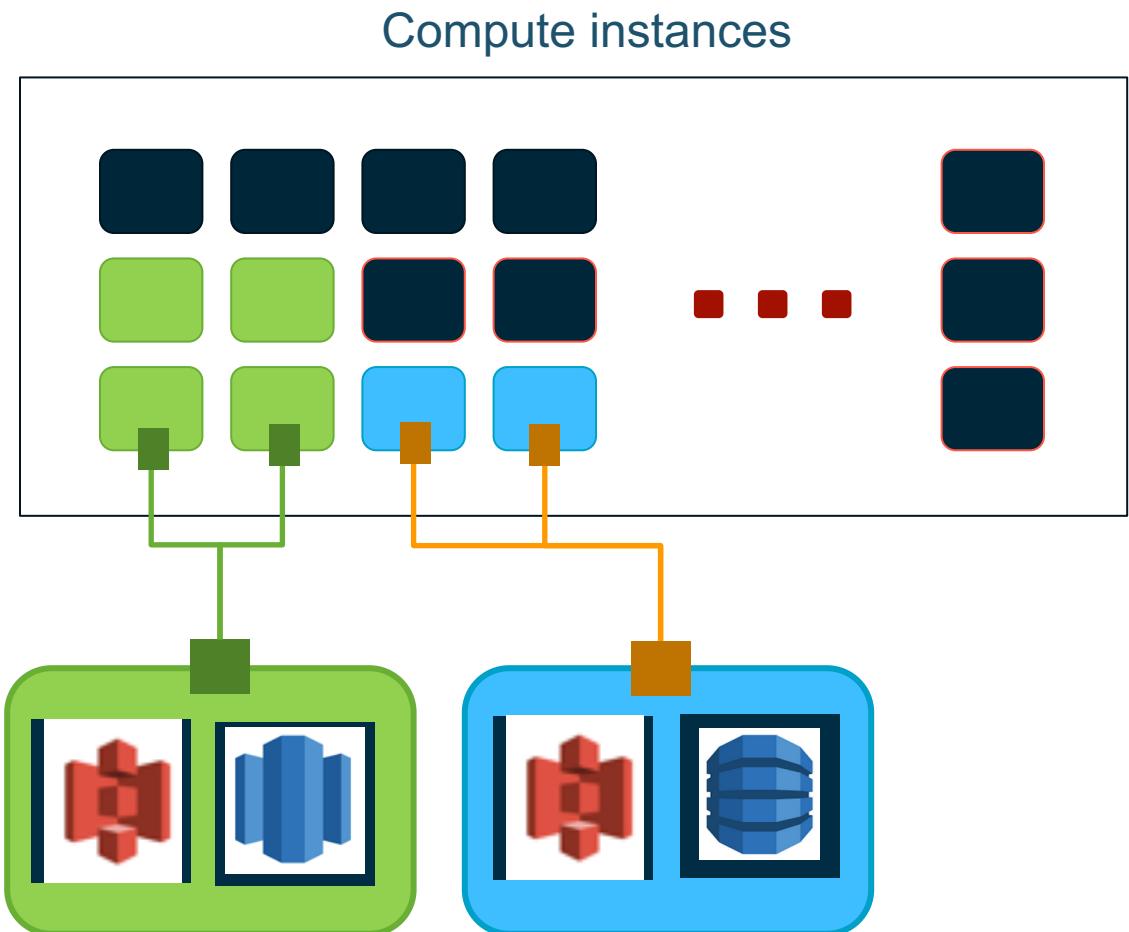
Configure job timeout
Job delay notifications

[Build and automate a serverless data lake using an AWS Glue trigger for the Data Catalog and ETL jobs](#)

AWS Glue: Job Execution - Serverless

There is no need to provision, configure, or manage servers

- Auto-configure VPC and role-based access
- Customers can specify the capacity that gets allocated to each job
- Automatically scale resources (on post-GA roadmap)
- You pay only for the resources you consume while consuming them



Glue worker types

Worker maps to 1 DPU

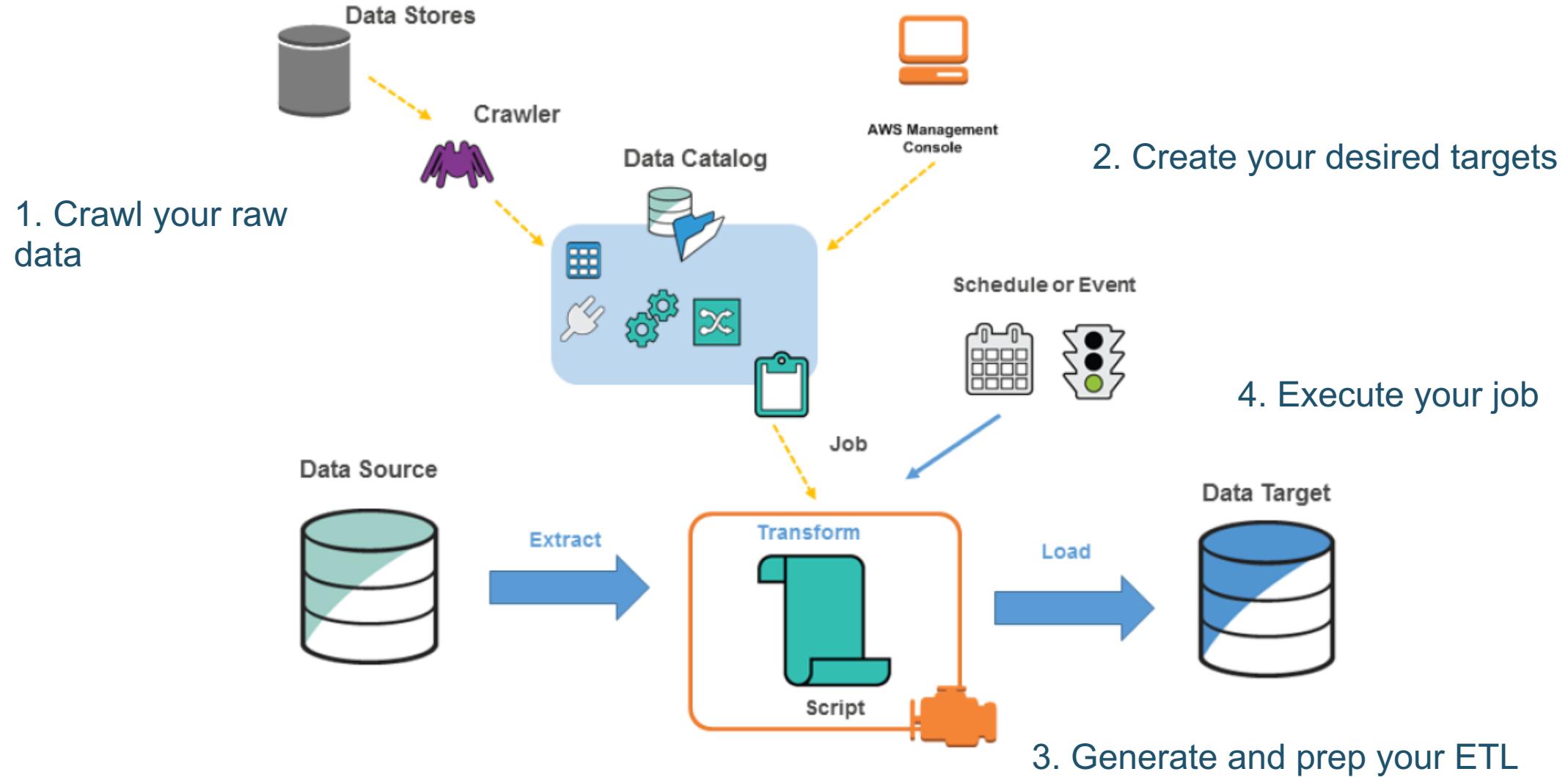
Standard – 2 executors/worker: 16 GB

More memory per executor

G.1X – 1 executor/worker: 16 GB

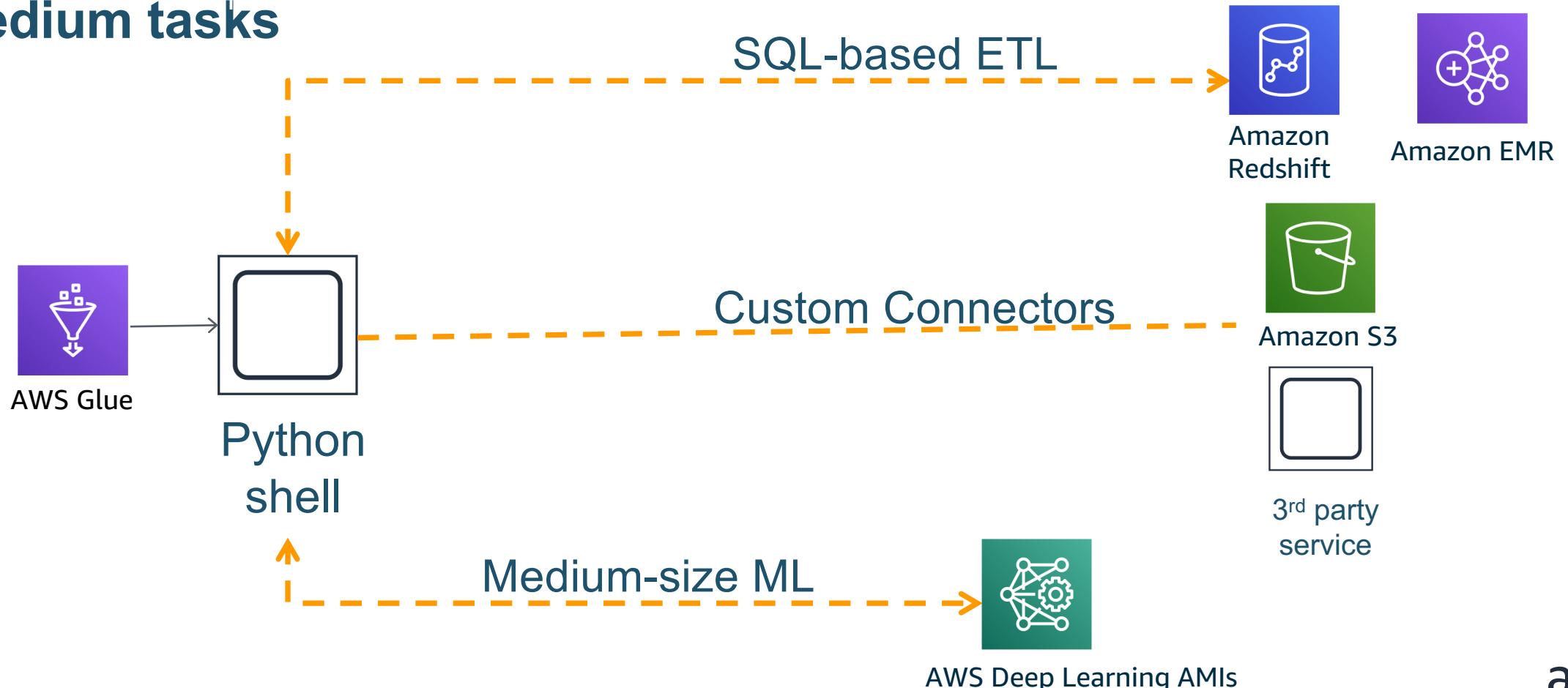
G.2X – 1 executor/worker: 32 GB

AWS Glue: Overall Flow



Python shell Job

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



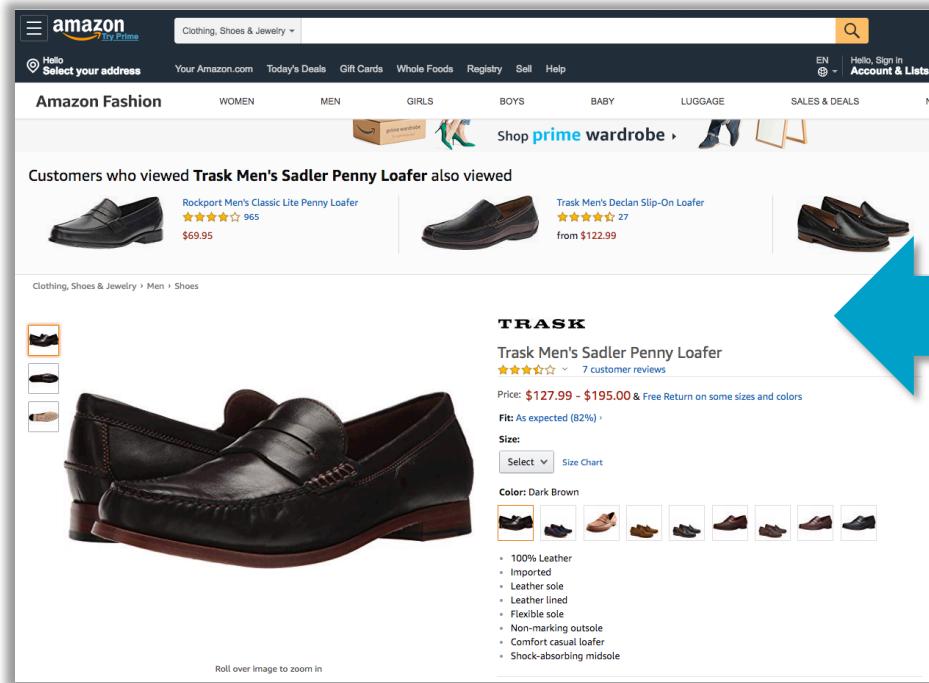
Glue Provides FindMatches ML Transform

- By adding the FindMatches transformation to your Glue ETL jobs, you can find related products, places, suppliers, customers, and more.
- You can also use the FindMatches transformation for deduplication
- Improve fraud detection. Identifying duplicate customer accounts, determining when a newly created account is (or might be) a match for a previously known fraudulent user.

Leverage machine learning to solve hard problems

De-duplication

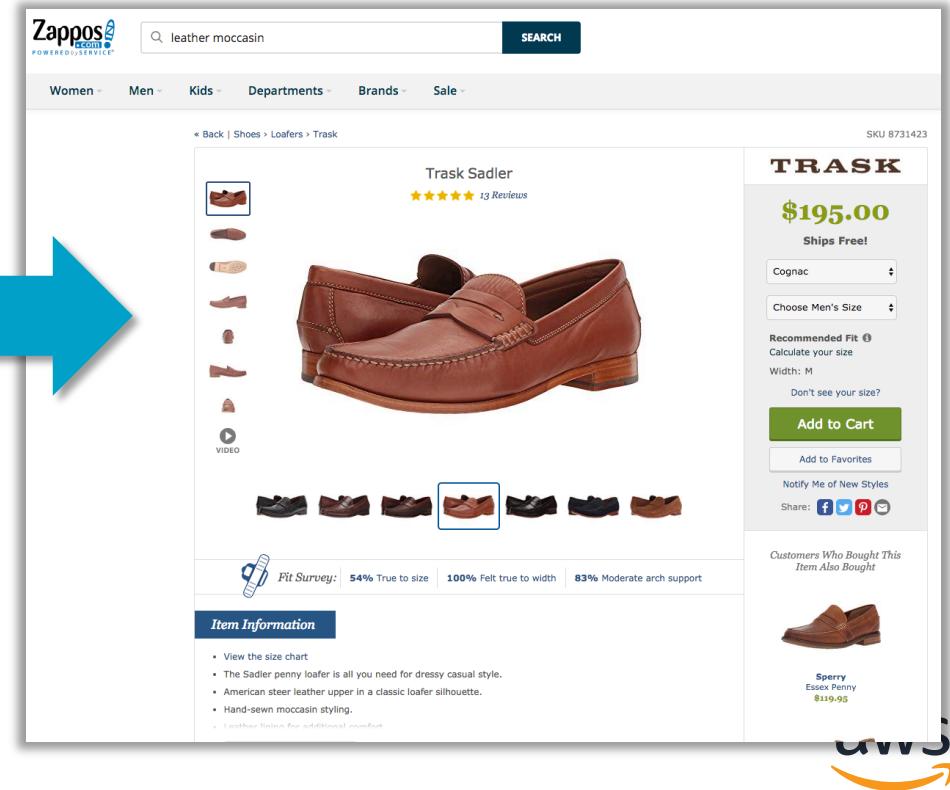
Transforming a dataset that has multiple rows referring to the *same actual thing* into a dataset where no two rows refer to the *same actual thing*



ML FindMatches

Record matching

Finding the relationships between multiple datasets, even when those datasets do not share an identifier (or when their identifier is unreliable)



AWS Glue: Components



Data Catalog

- Hive Metastore compatible with enhanced functionality
- Crawlers automatically extracts metadata and creates tables
- Integrated with Amazon Athena, Amazon EMR and many more



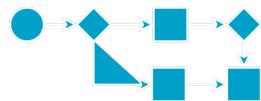
Job Authoring

- Auto-generates ETL code
- Build on open frameworks – Python/Scala and Apache Spark
- Developer-centric – editing, debugging, sharing



Job Execution

- Run jobs on a serverless Spark platform
- Provides flexible scheduling , Job monitoring and alerting

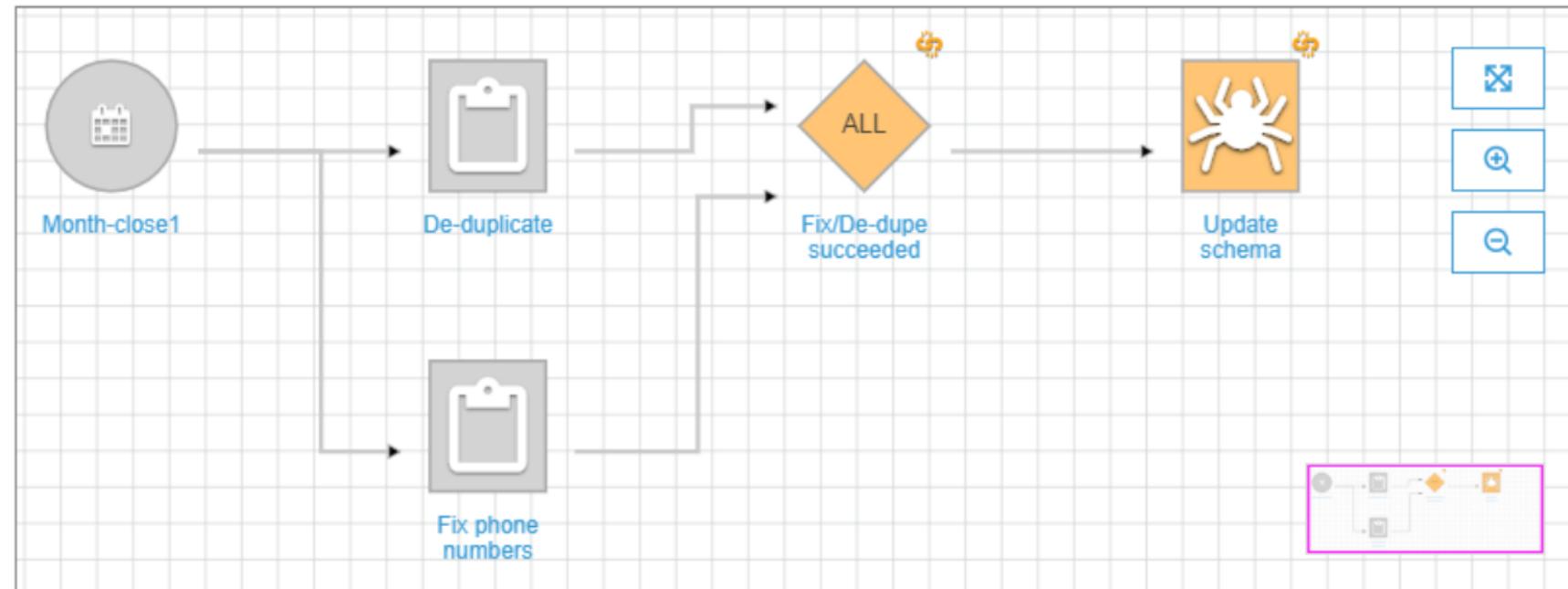


Job Workflow

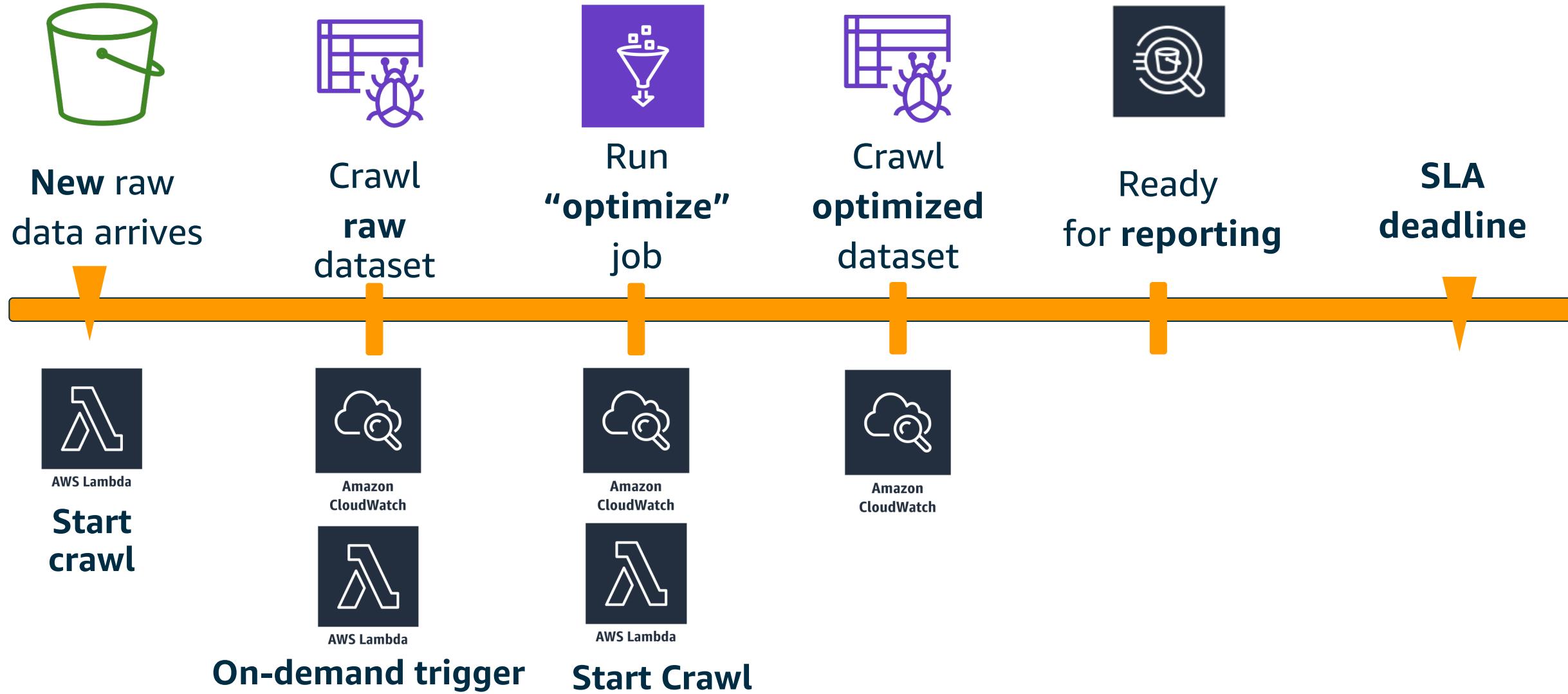
- Orchestrate triggers, crawlers & jobs
- Author & monitor entire flows and Integrated alerting

Glue Workflows

- Create and visualize complex ETL activities involving multiple crawlers, jobs, and triggers
- Records execution progress and status
- Provide both static and dynamic view



Example event-driven workflow



Q & A