Working Within the Data Lake

With AWS Glue

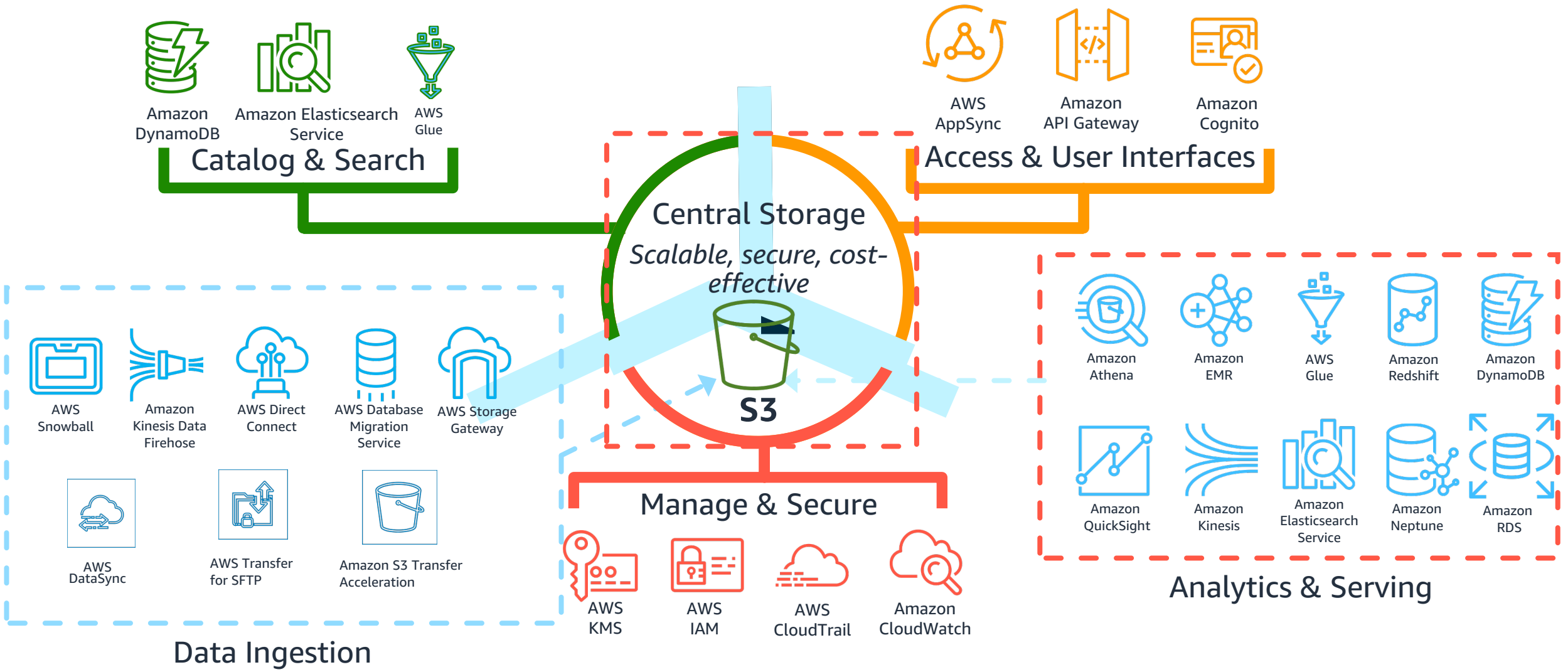
Team or presenters name
Date

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

aws

Session's Focus – Working In The Data Lake

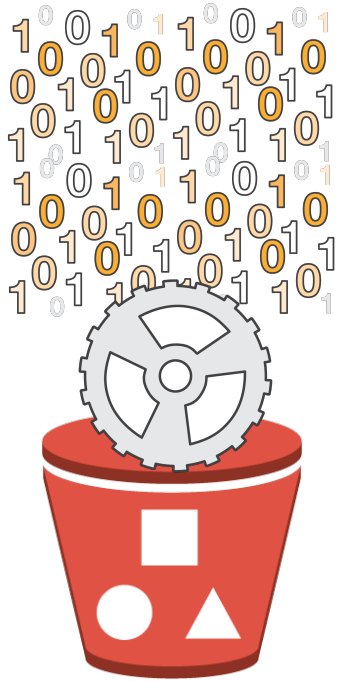


Optimizing for Cost and Performance

aws

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**

aws

Partitioning

```
datalake
├── 20170515T1423-GB-01.tar.gz
├── 20170515T1423-GB-02.tar.gz
├── 20170515T1500-US-01.tar.gz
├── 20170516T1500-US-01.tar.gz
├── 20170516T1600-GB-01.tar.gz
└── 20170516T1600-GB-02.tar.gz
```

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

```
datalake
├── dt=20170515
│   ├── country=GB
│   │   ├── 20170515T1423-GB-01.tar.gz
│   │   └── 20170515T1423-GB-02.tar.gz
│   └── country=US
│       └── 20170515T1500-US-01.tar.gz
└── dt=20170516
    ├── country=GB
    │   ├── 20170516T1600-GB-01.tar.gz
    │   └── 20170516T1600-GB-02.tar.gz
    └── country=US
        └── 20170516T1500-US-01.tar.gz
```


aws

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

aws

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**

aws

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**

aws

Cataloging and Transforming Data

Working with AWS Glue

aws

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

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

AWS Glue: Components

Data Catalog

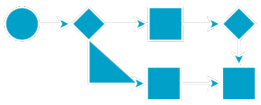
- Hive Metastore compatible with enhanced functionality
- Crawlers automatically extracts metadata and creates tables
- Integrated with Amazon Athena, Amazon EMR and many more

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

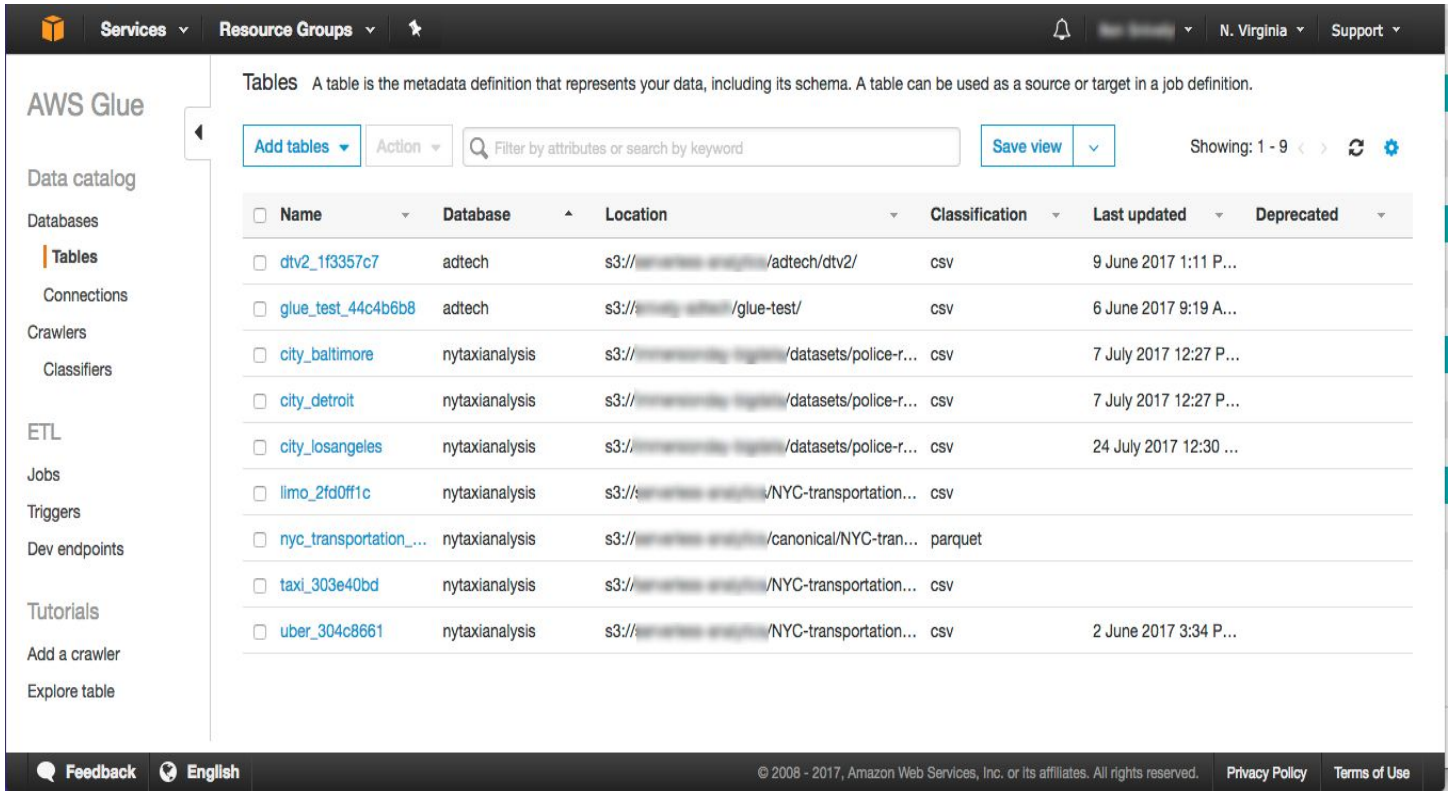


Job Workflow

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

aws

Glue: Data Catalog



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

aws

Glue: Data Catalog – Queryable by Many Services

Glue ETL

Amazon Athena

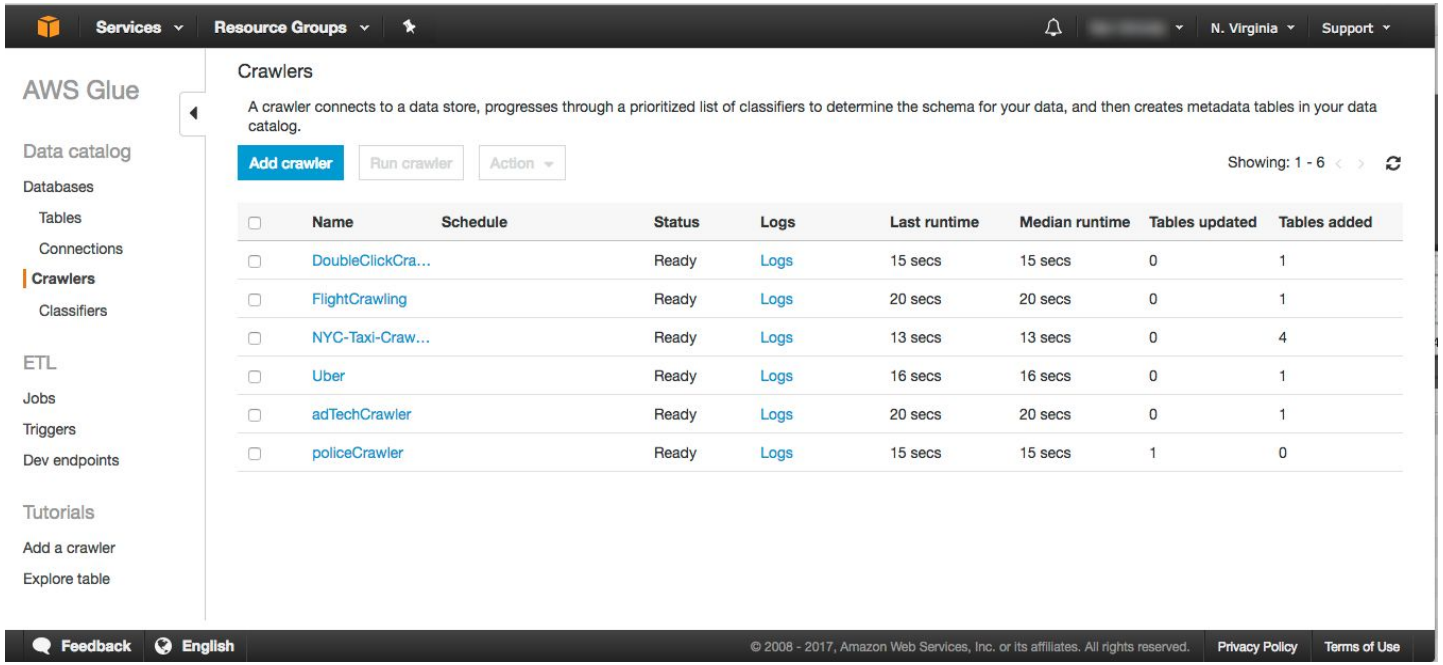
Glue Data Catalog

Redshift Spectrum

EMR (Hadoop/Spark)

aws

Glue: Data Catalog - Crawlers



Features Include:

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

aws

Crawlers: Classifiers

Create additional Custom Classifiers with Grok!

IAM Role

Glue Crawler

JDBC Connection

Object Connection

Databases

Amazon RDS

Data Warehouse

Amazon Redshift

Data Lakes

Amazon S3

Built-In Classifiers

MySQL
MariaDB
PostreSQL
Aurora

Redshift

Avro
Parquet
ORC
JSON & BJSON
Logs
(Apache, Linux, MS, Ruby, Redis, and many others)
Delimited
(comma, pipe, tab, semicolon)
Compressed Formats
(ZIP, BZIP, GZIP, LZ4, Snappy)

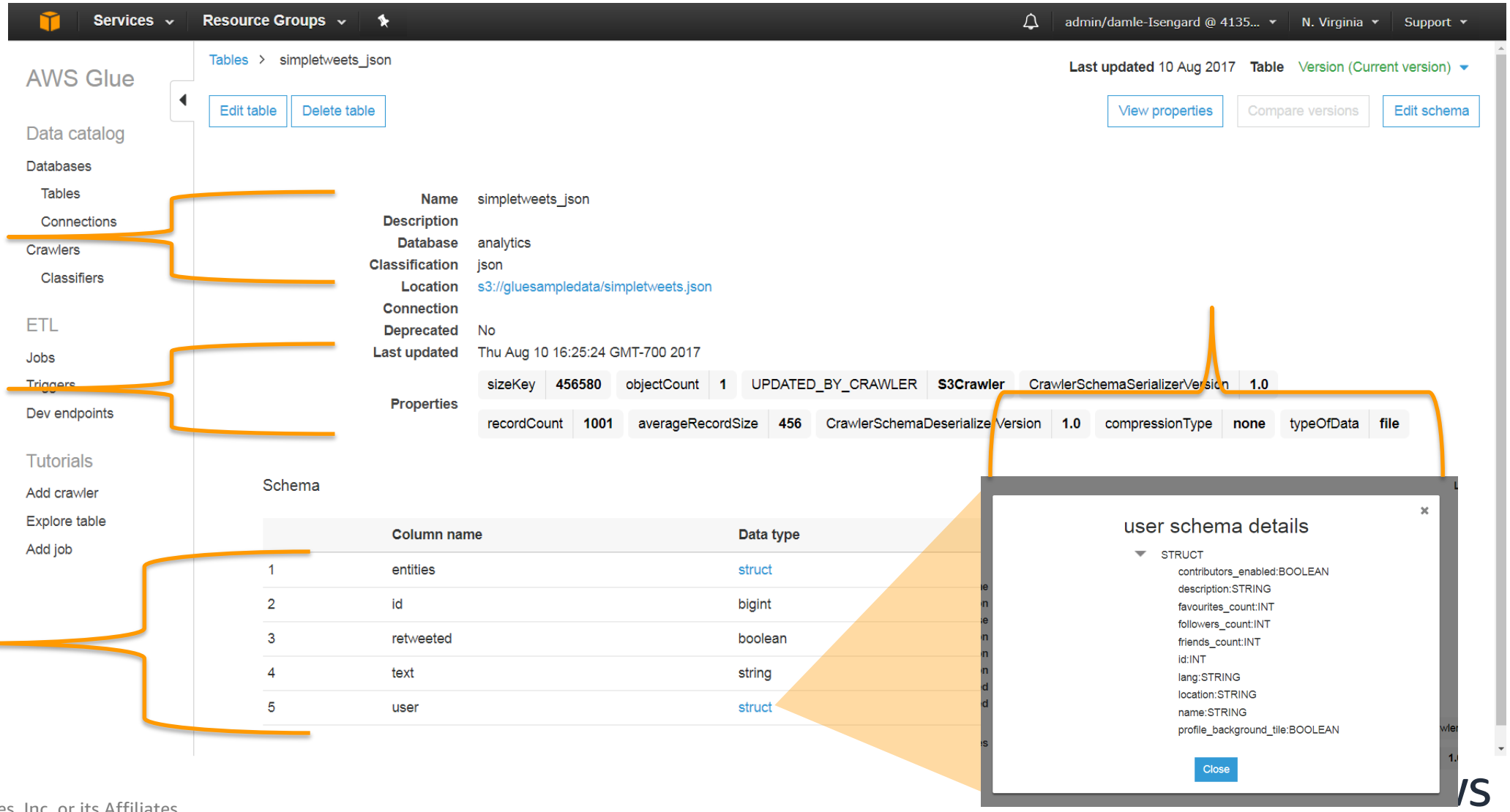
aws

Data Catalog: Table details

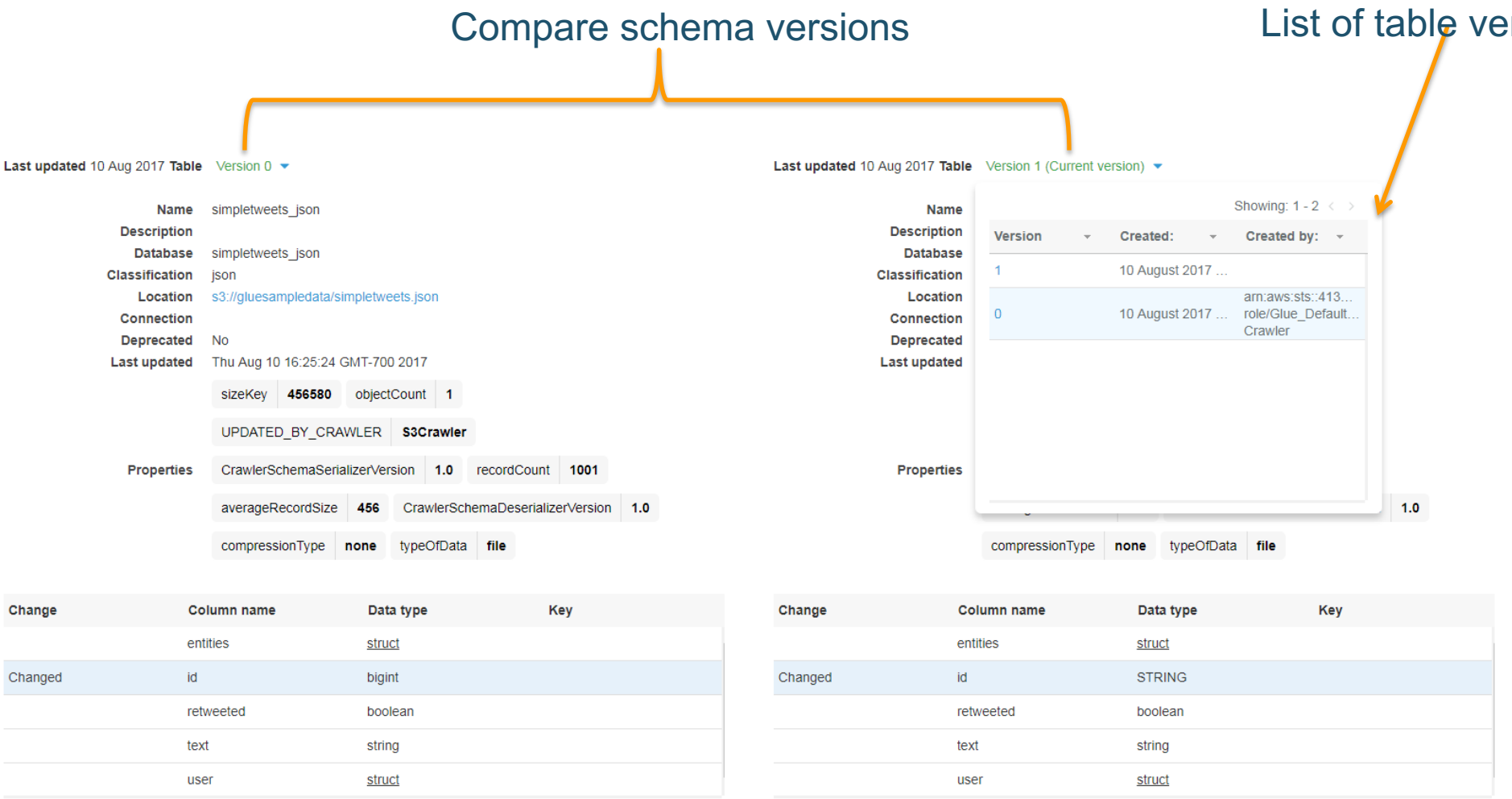
Table properties

Data statistics

Table schema



Data Catalog: Version control



Glue: Crawlers – Detecting Schema and Partitions

S3 bucket hierarchy

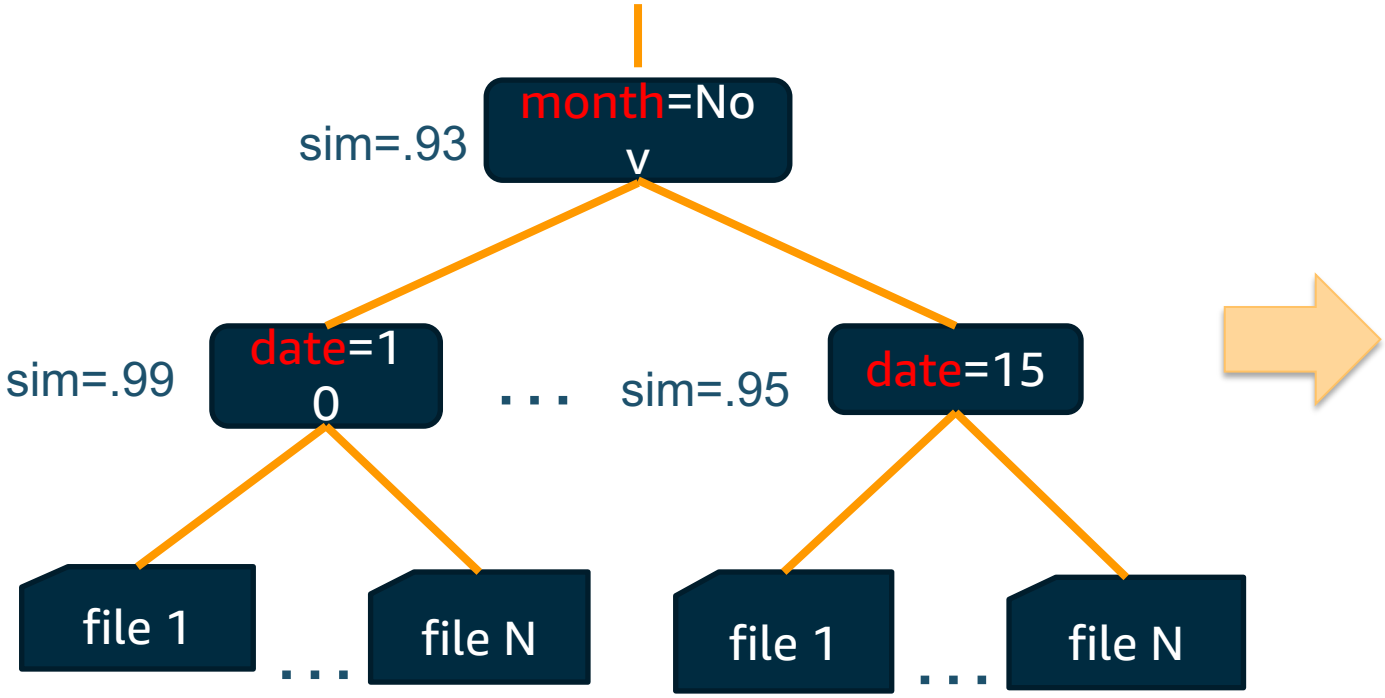


Table definition

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

aws

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

AWS Glue: Components

Data Catalog

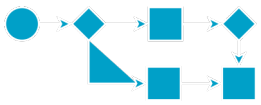
- Hive Metastore compatible with enhanced functionality
- Crawlers automatically extracts metadata and creates tables
- Integrated with Amazon Athena, Amazon EMR and many more

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

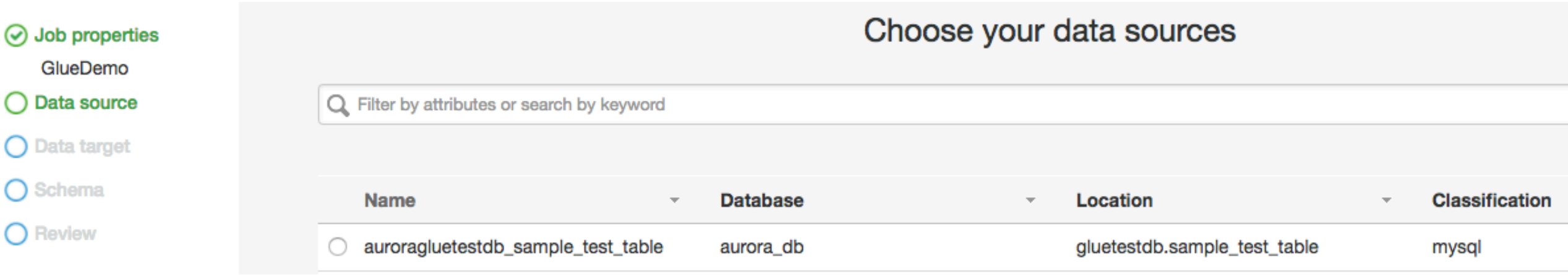


Job Workflow

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

aws

AWS Glue: Job Authoring – Pick a Source



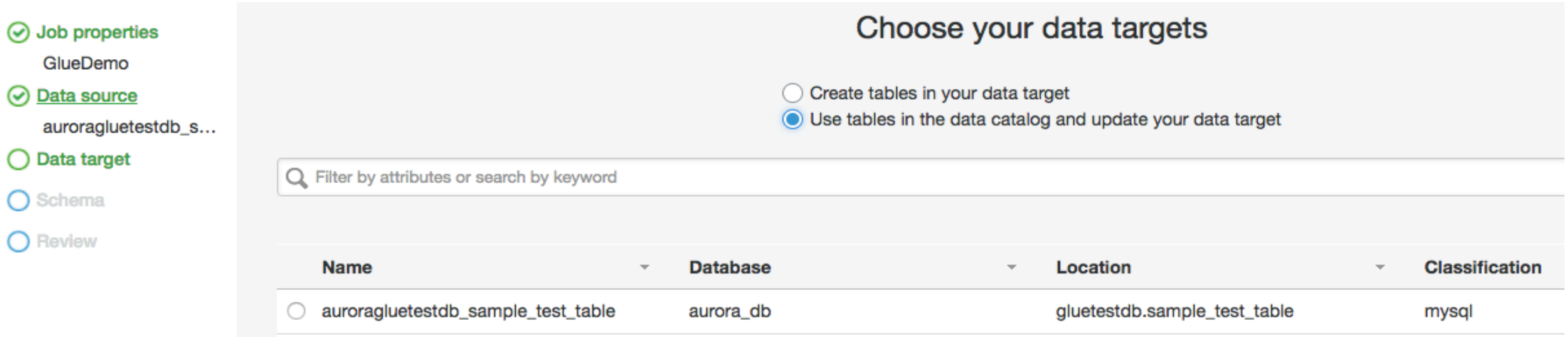
Choose from Manually-Defined or Crawler-Generated Sources:

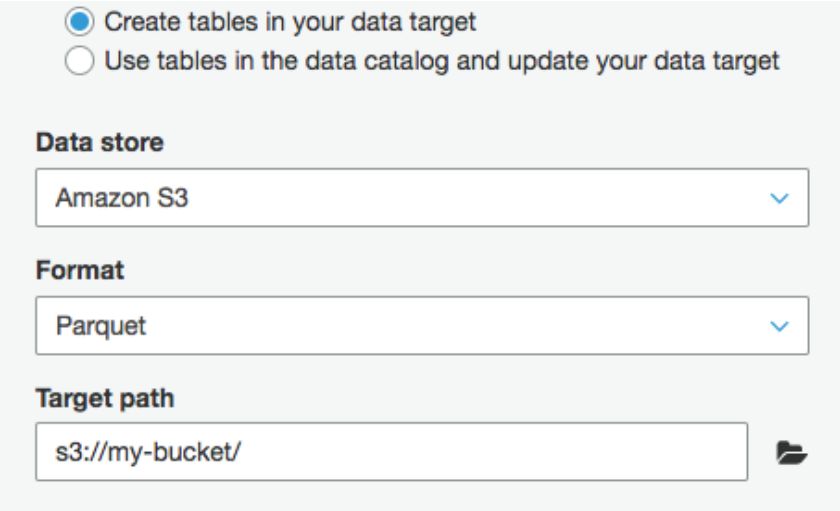
- S3 Bucket
- RDBMS
- Redshift
- DynamoDB

aws

AWS Glue: Job Authoring – Pick a Target

Write output results into an existing table.



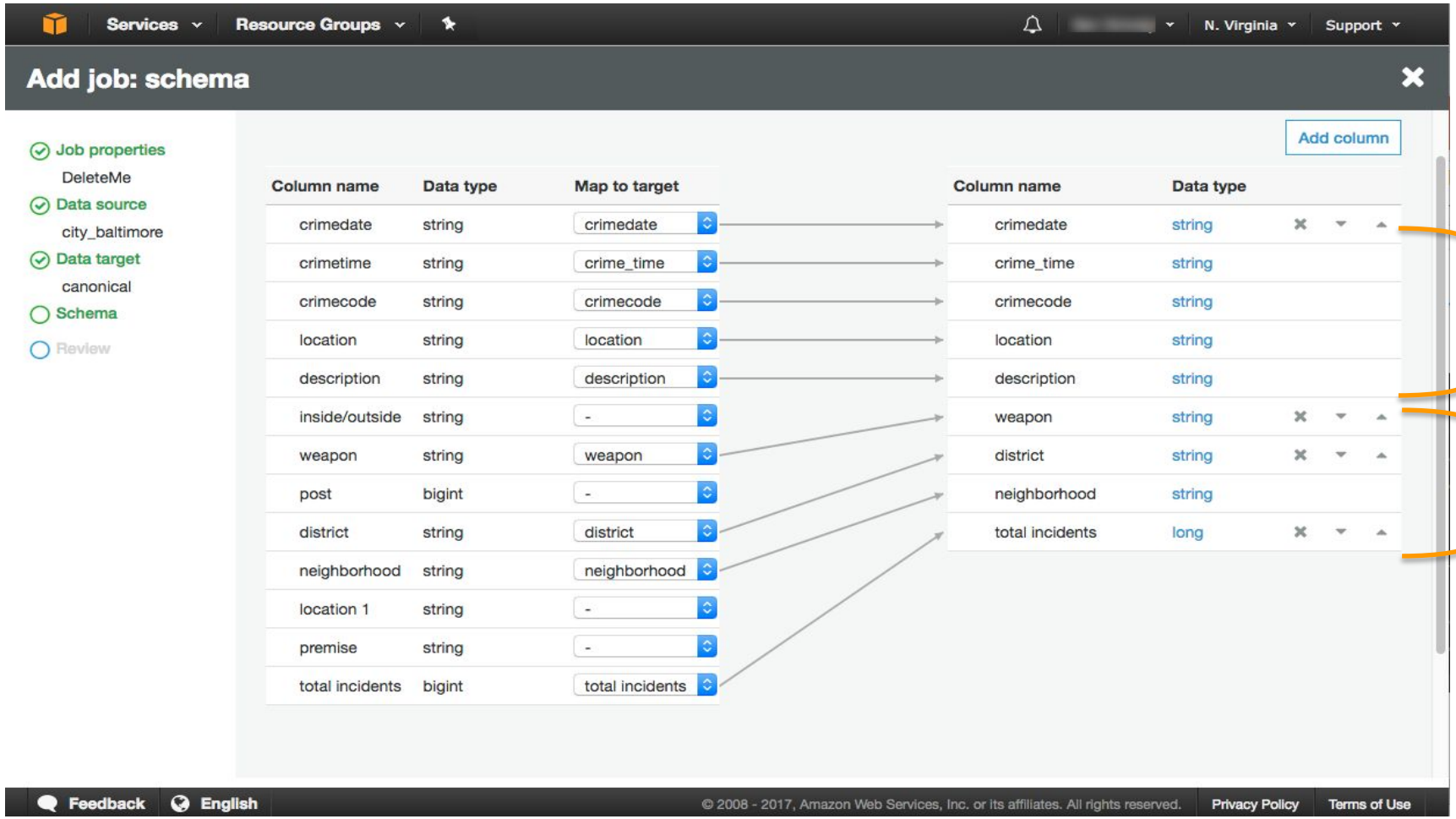


Crawler-Defined or Manually-Created:

- S3 Bucket
- RDBMS
- Redshift

aws

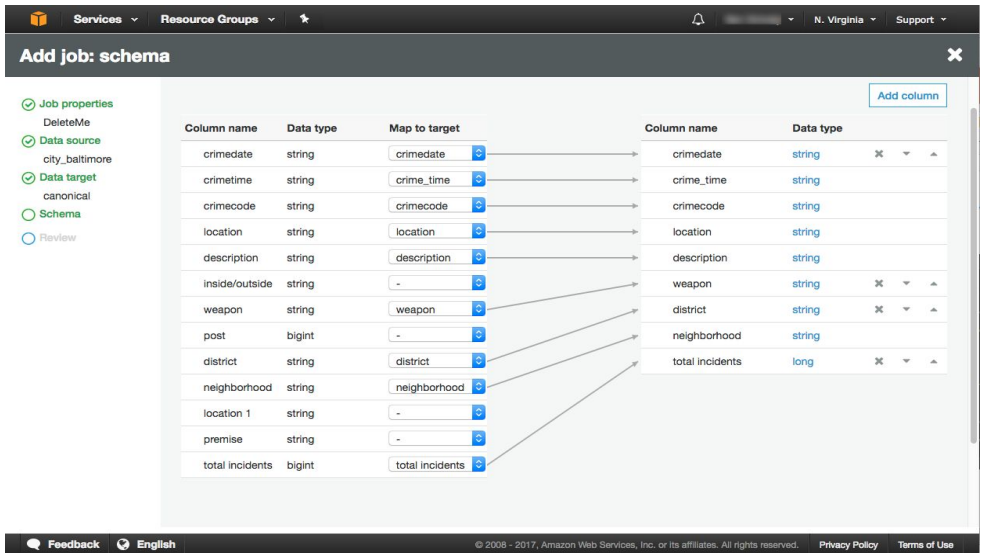
AWS Glue: Job Authoring – Apply Transformation

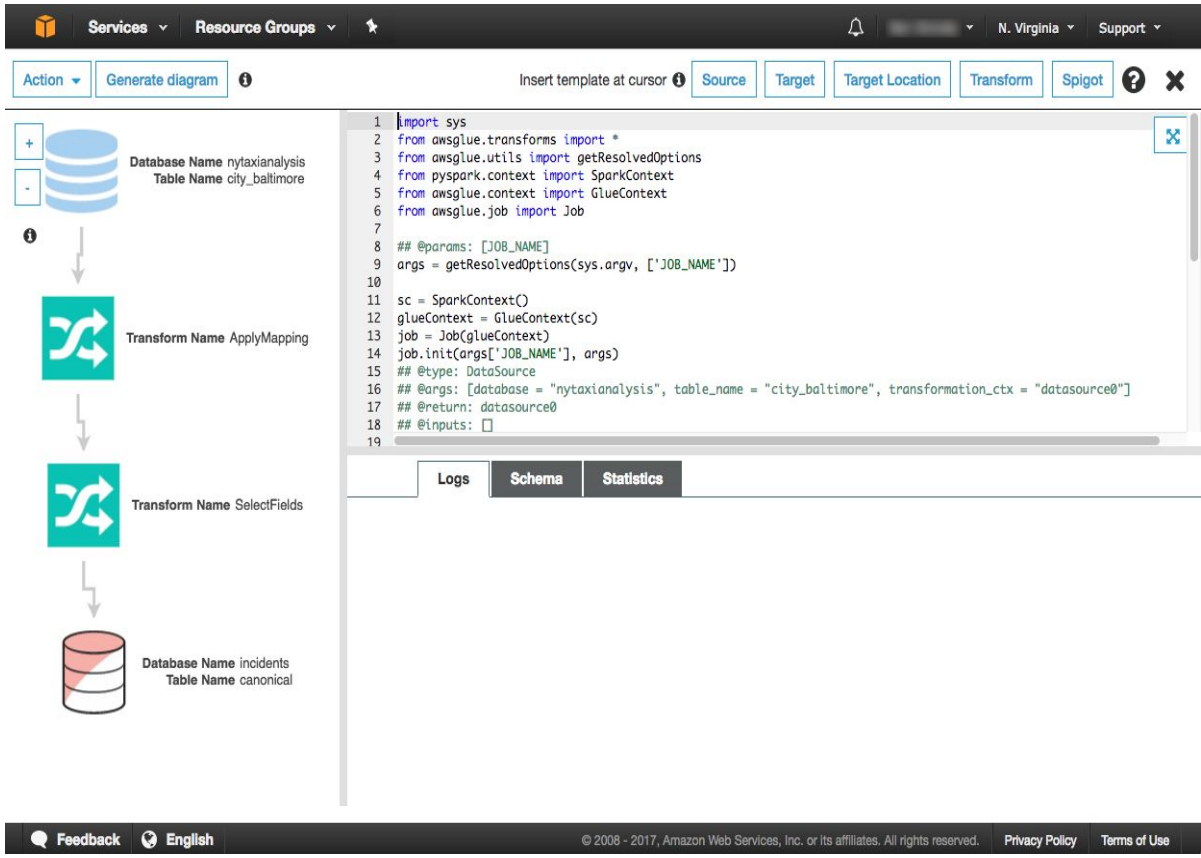


Existing columns in target

Can extend/add new columns to target

AWS Glue: Job Authoring – Code Generation





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

aws

Job authoring: ETL code

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

```
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")
```

- **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

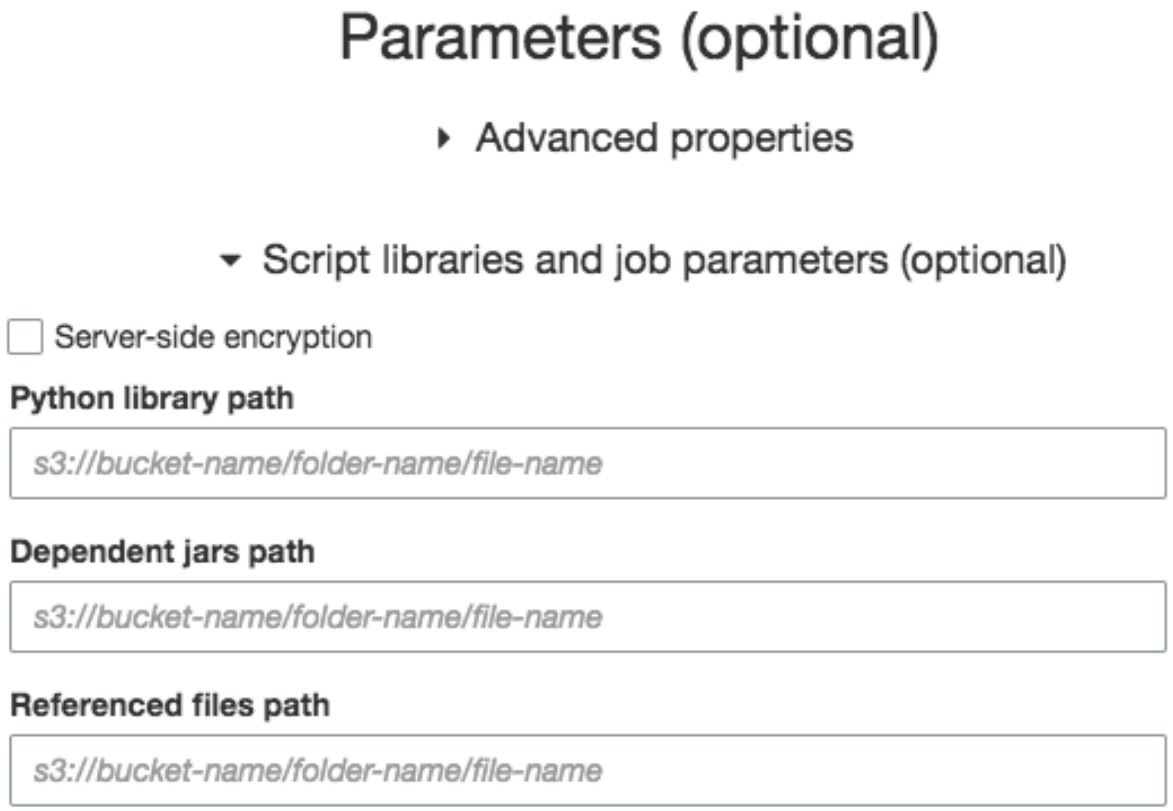
```
##################################################
##
##  PySpark Logic to do lots of custom stuff...
##
##################################################
DataFrame0 = DynamicFrame.toDF(SelectFields0)

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))
```

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

aws

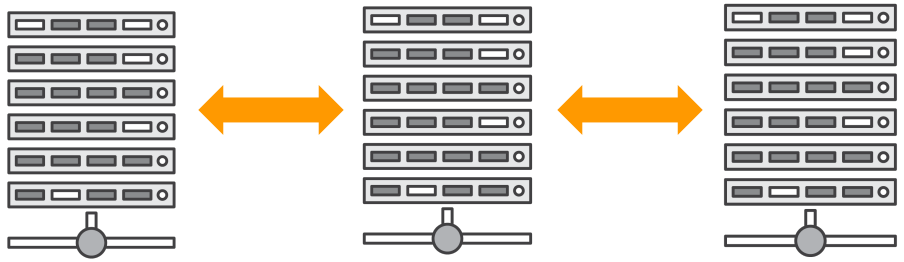
AWS Glue: Job Authoring – Import Custom Modules

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



aws

Apache Spark and AWS Glue ETL



SparkSQL	AWS Glue ETL
Dataframes	Dynamic Frames
Spark core: RDDs	

What is Apache Spark?

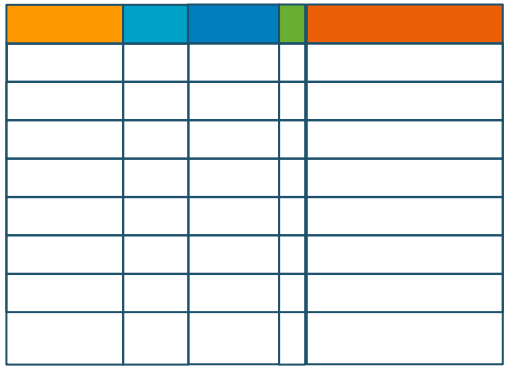
- Parallel, scale-out data processing engine
- Fault-tolerance built-in
- Flexible interface: Python scripting, SQL
- Rich eco-system: ML, Graph, analytics, …

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

aws

Dataframes and Dynamic Frames



Dataframes

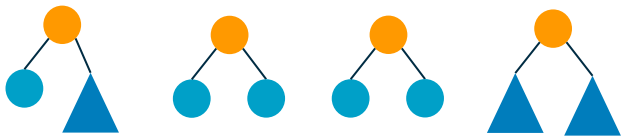
Core data structure for SparkSQL

Like structured tables

Need schema up-front

Each row has same structure

Suited for SQL-like analytics



Dynamic Frames

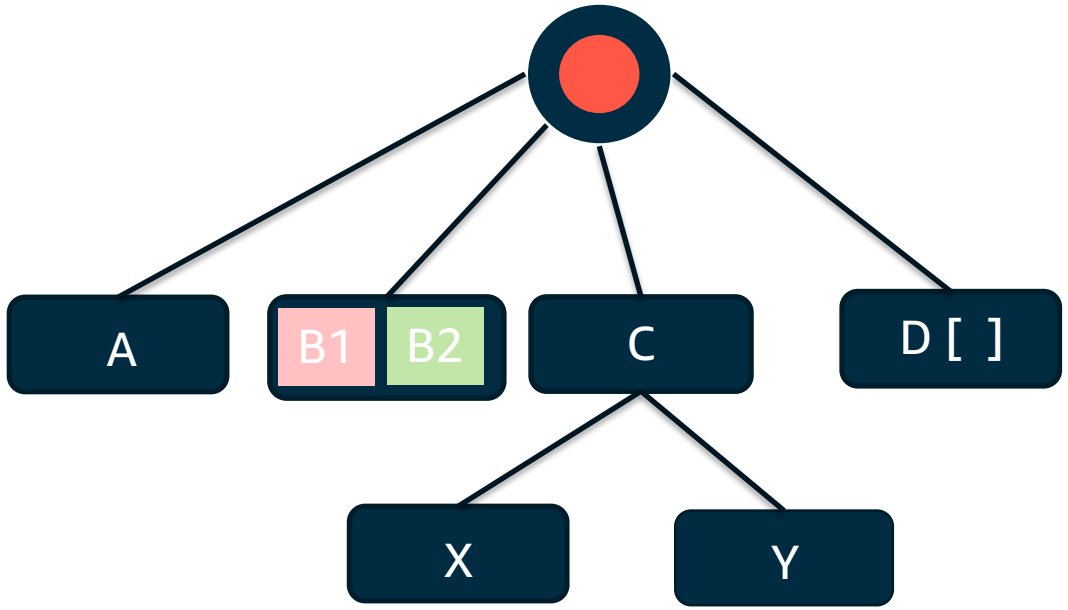
Like dataframes for ETL

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

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

aws

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:

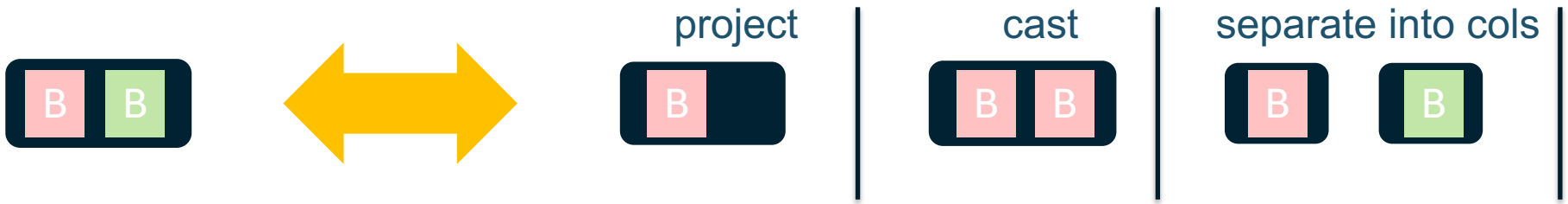
- Tracks new fields, and inconsistent changing data types with **choices**, e.g. integer or string
- Automatically mark and separate error records

aws

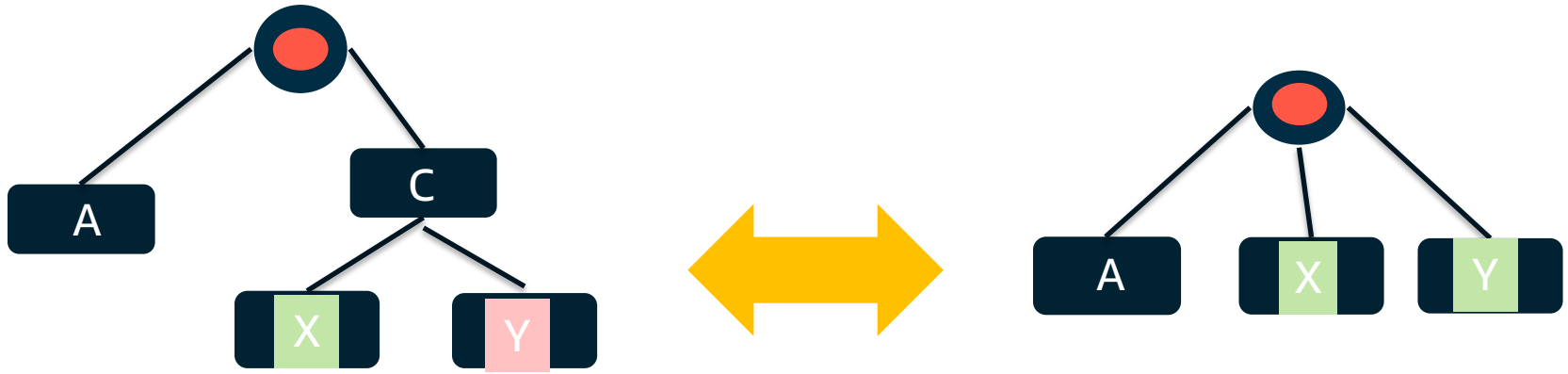
Job Authoring: Glue transforms

Adaptive and flexible

ResolveChoice()

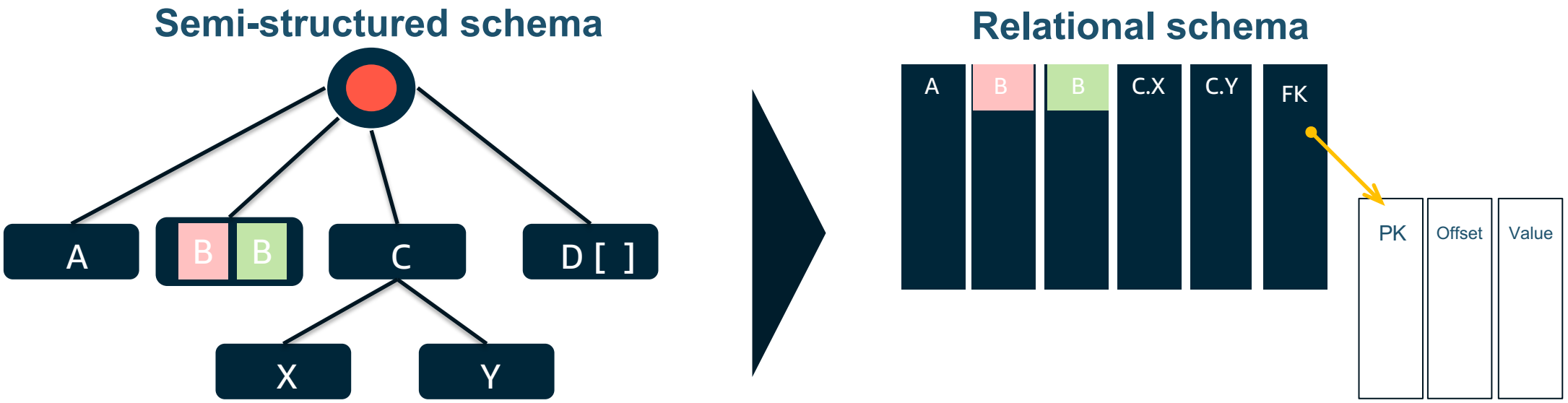


Apply Mapping()



aws

Job authoring: Relationalize() transform

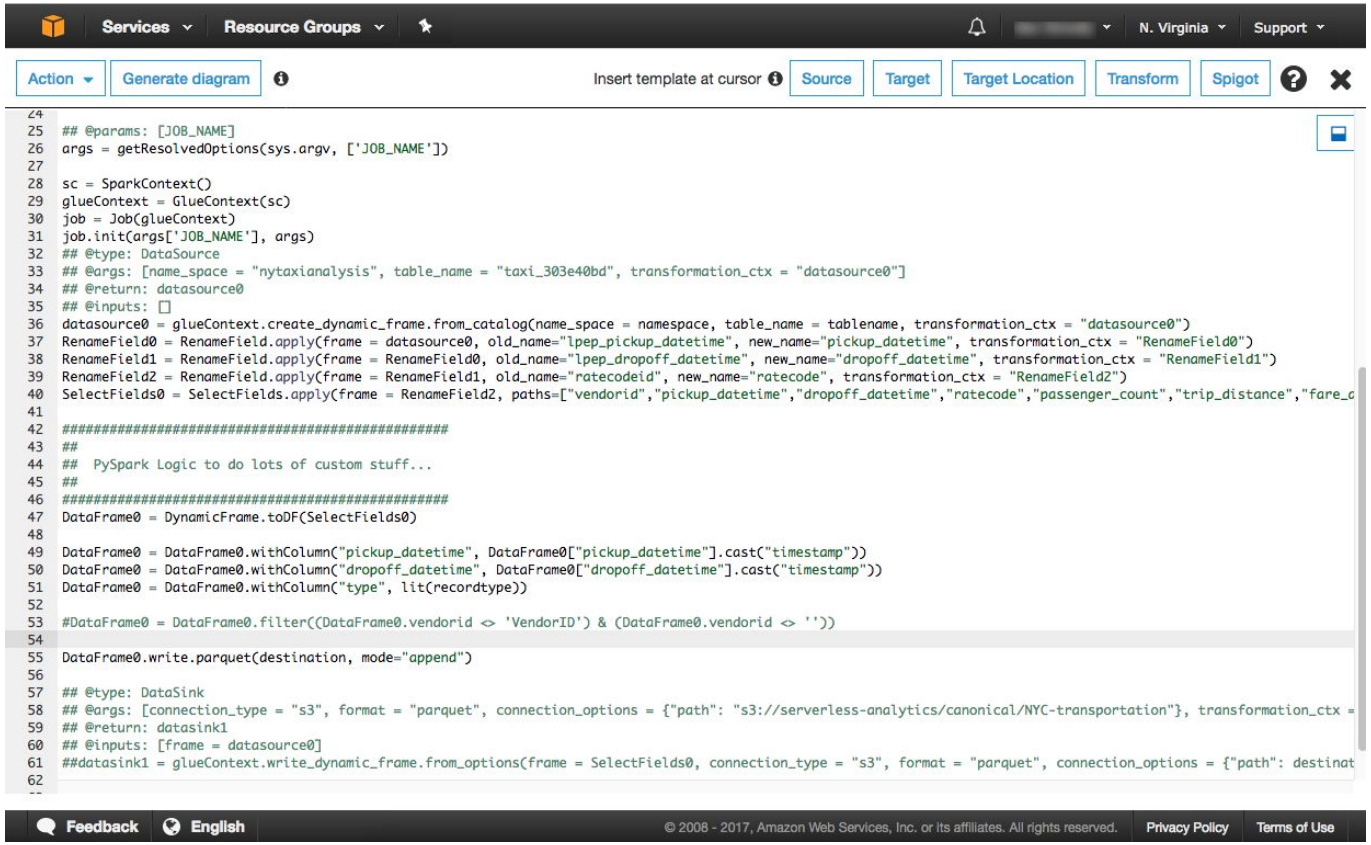


- Transforms and **adds new** columns, types, and tables on-the-fly
- Tracks **keys** and **foreign keys** across runs
- SQL on the relational schema is orders of **magnitude faster** than JSON processing

aws

AWS Glue: Job Authoring – "DynamicFrame" High-Level API

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



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

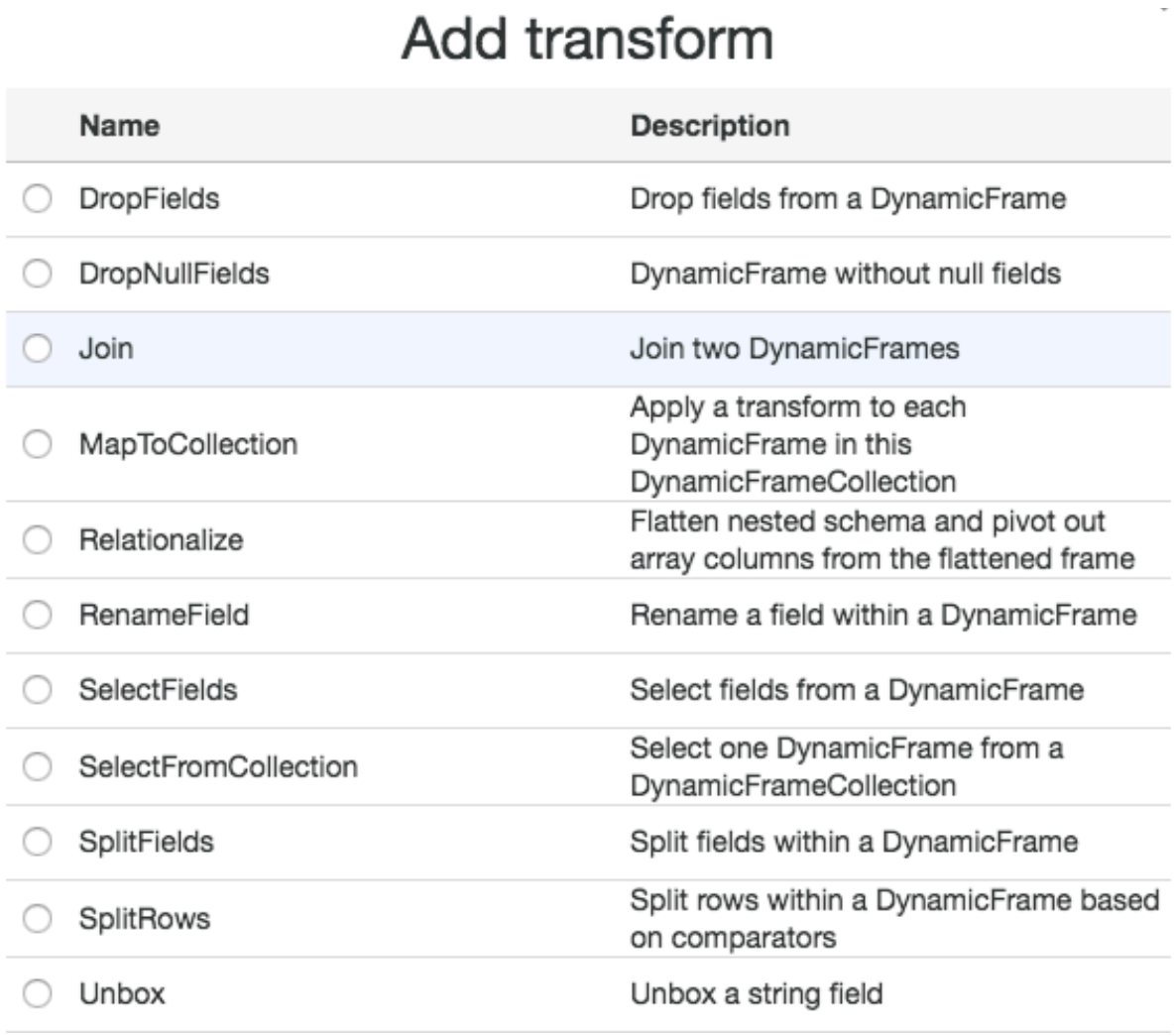
`toDF(options)`

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



- **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

AWS Glue: Components

Data Catalog

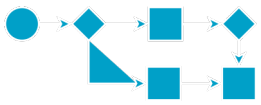
- Hive Metastore compatible with enhanced functionality
- Crawlers automatically extracts metadata and creates tables
- Integrated with Amazon Athena, Amazon EMR and many more

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

AWS Glue Orchestration

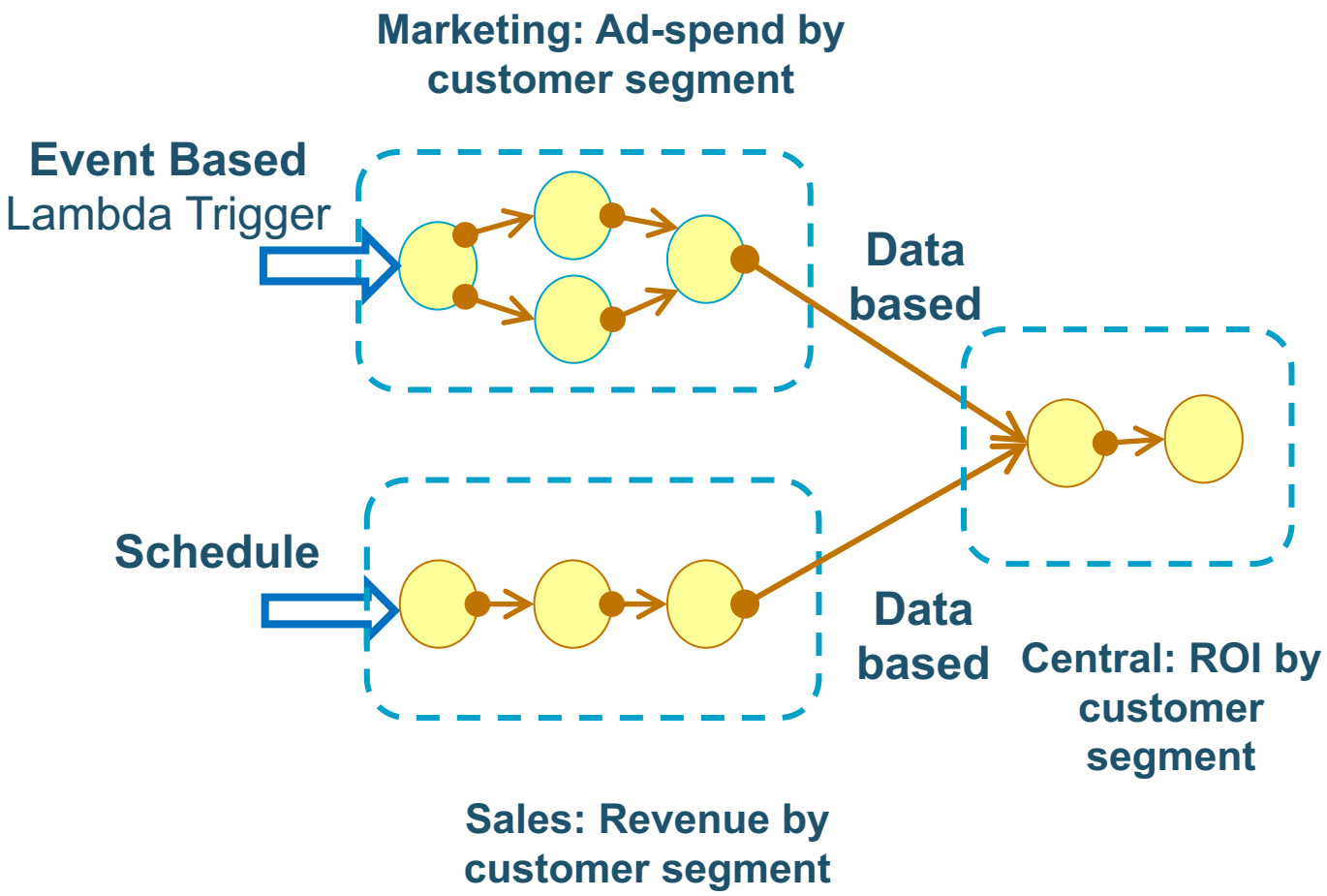
Compose jobs globally with event-based dependencies

- 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

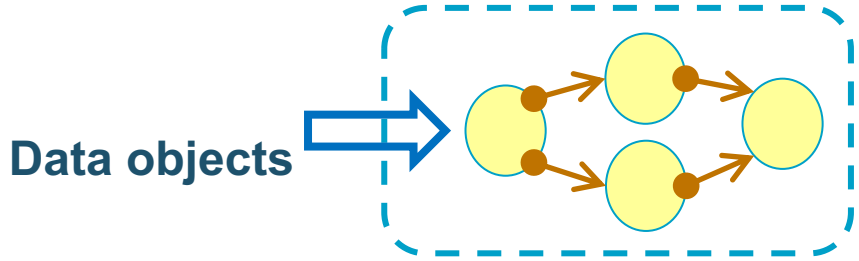


Job execution: Job bookmarks

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

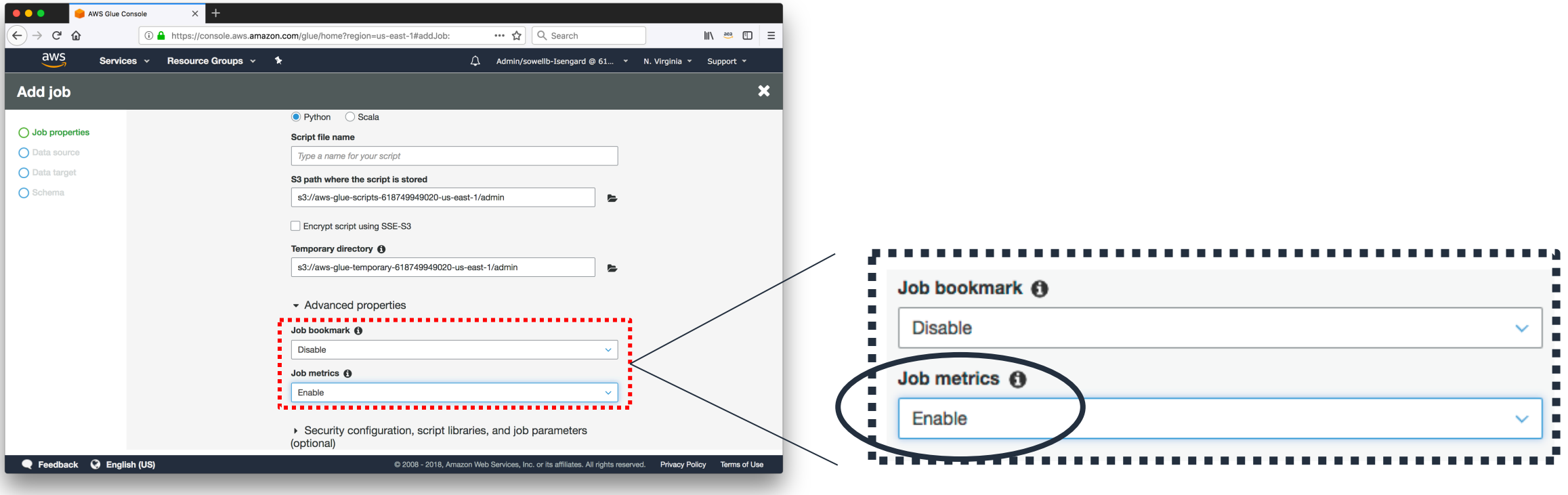
For example, you get new files everyday in your S3 bucket. By default, AWS Glue keeps track of which files have been successfully processed by the job to prevent data duplication.



Marketing: Ad-spend by customer segment

aws

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.

aws

Job Monitoring and Notification

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

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

Configure job timeout
Job delay notifications

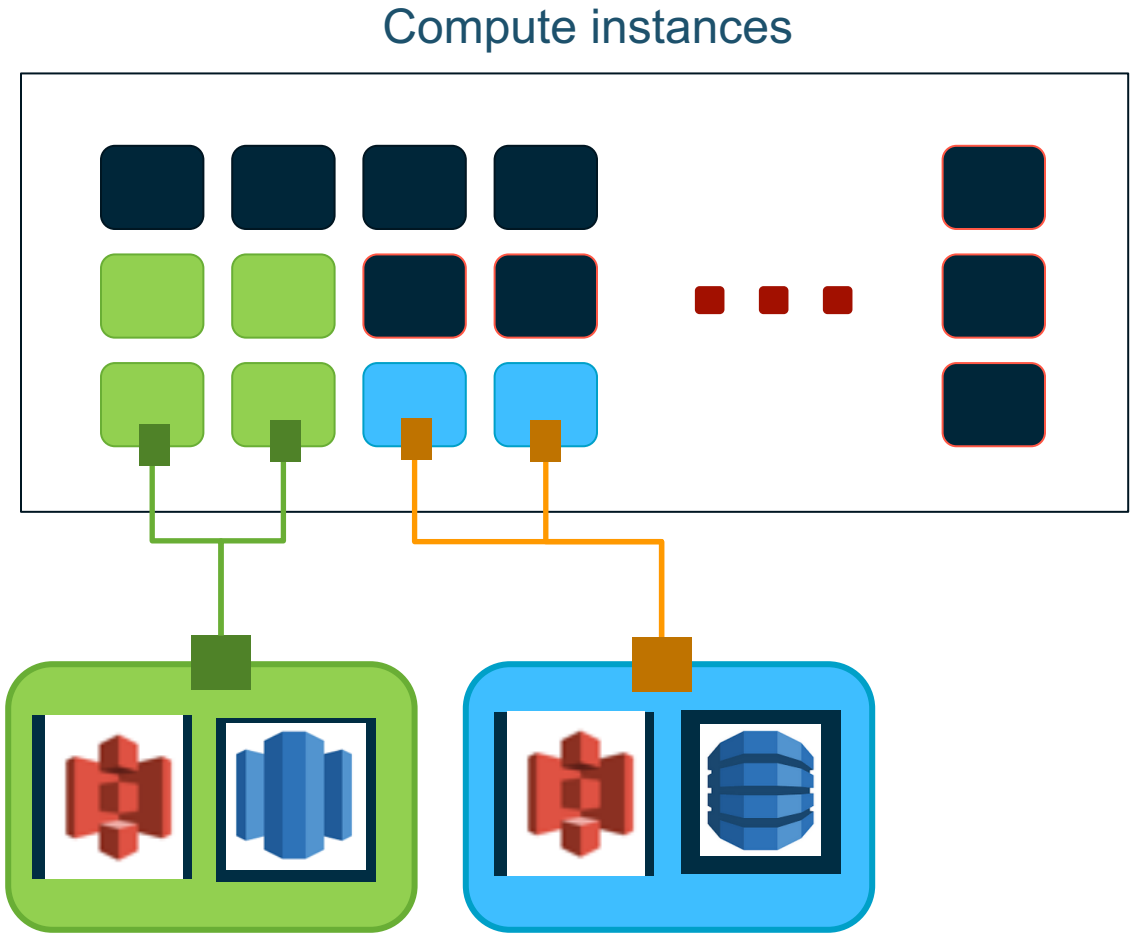
[Build and automate a serverless data lake using an AWS Glue trigger for the Data Catalog and ETL jobs]

aws

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



aws

Glue worker types

Worker maps to 1 DPU

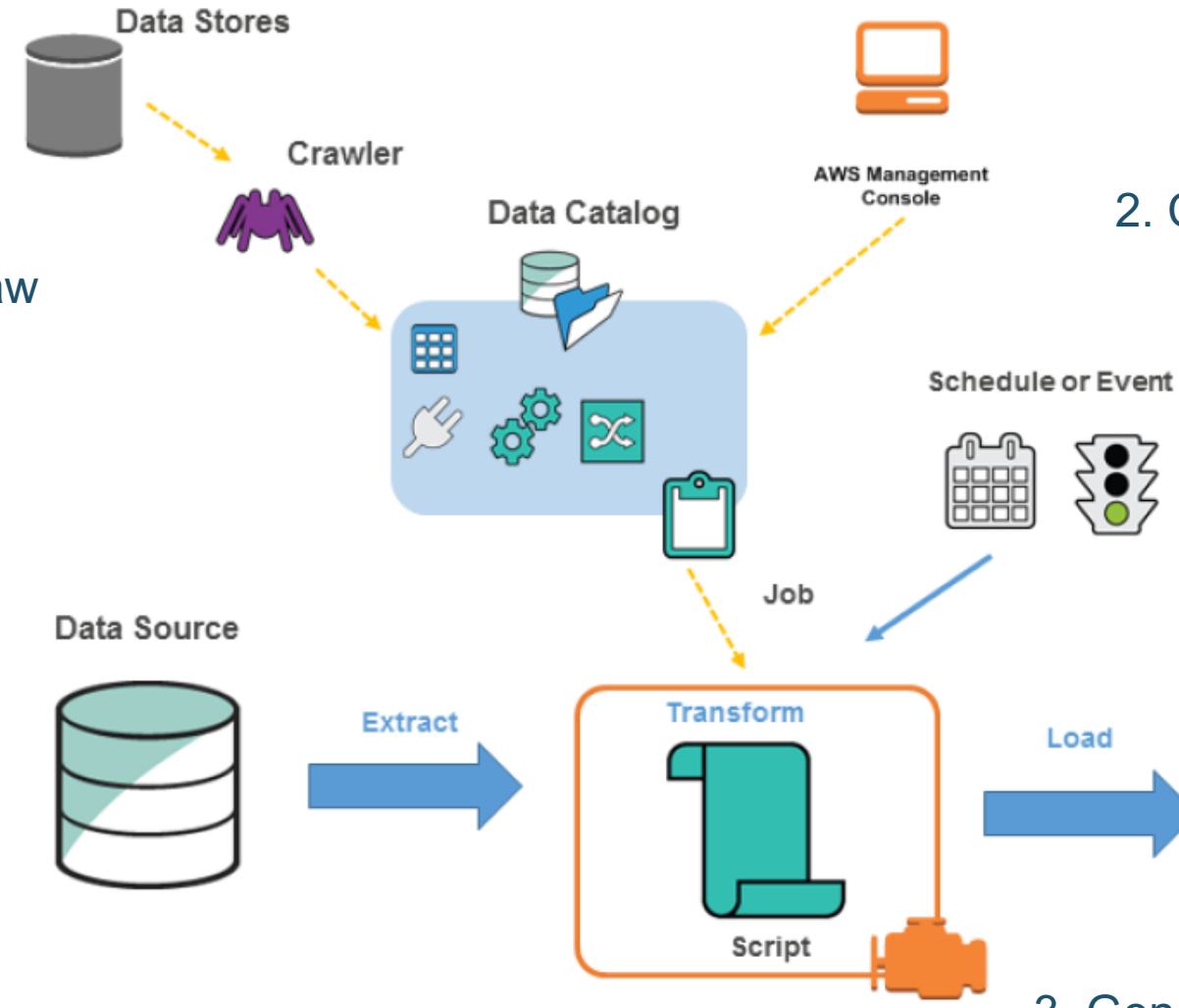
Standard – 2 executors/worker: 16 GB

More memory per executor

G.1X – 1 executor/worker: 16 GB

G.2X – 1 executor/worker: 32 GB

AWS Glue: Overall Flow



1. Crawl your raw data

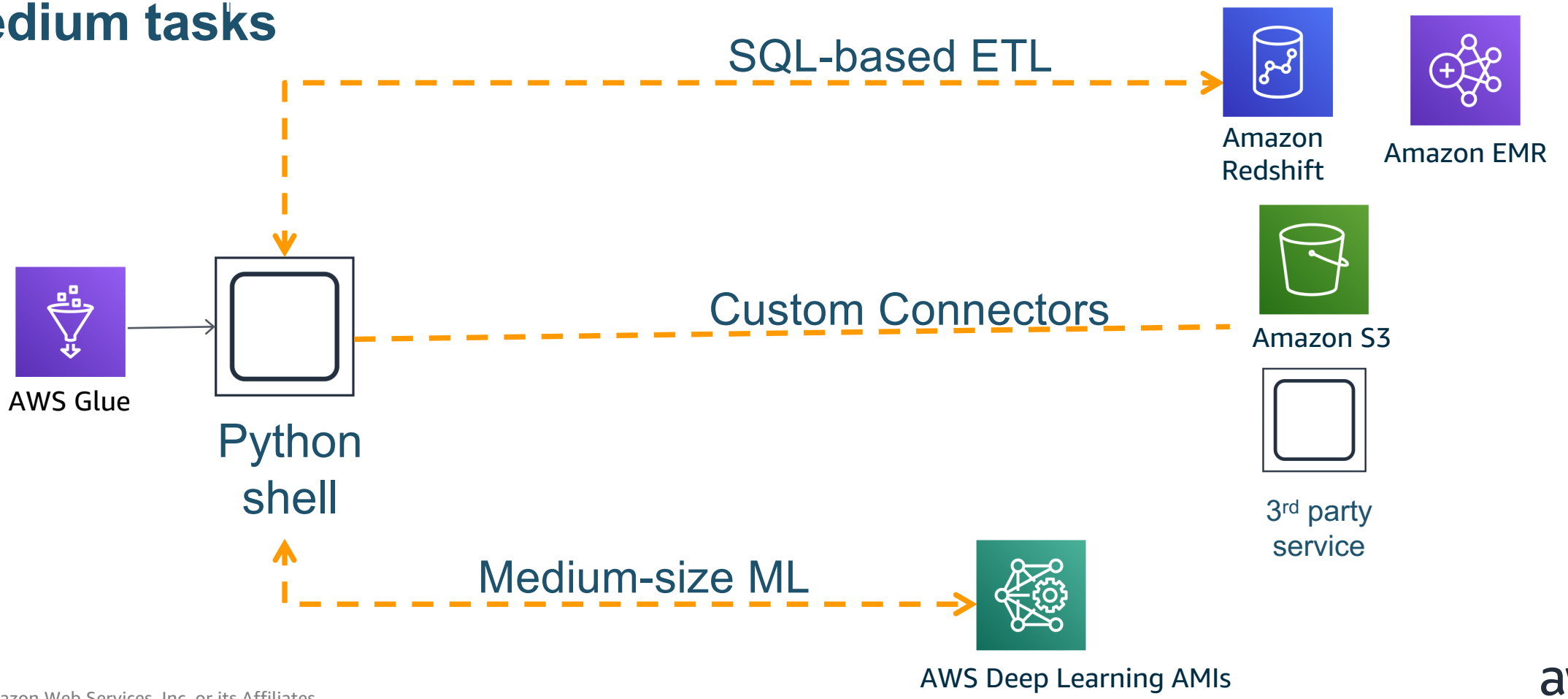
2. Create your desired targets

4. Execute your job

3. Generate and prep your ETL

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.

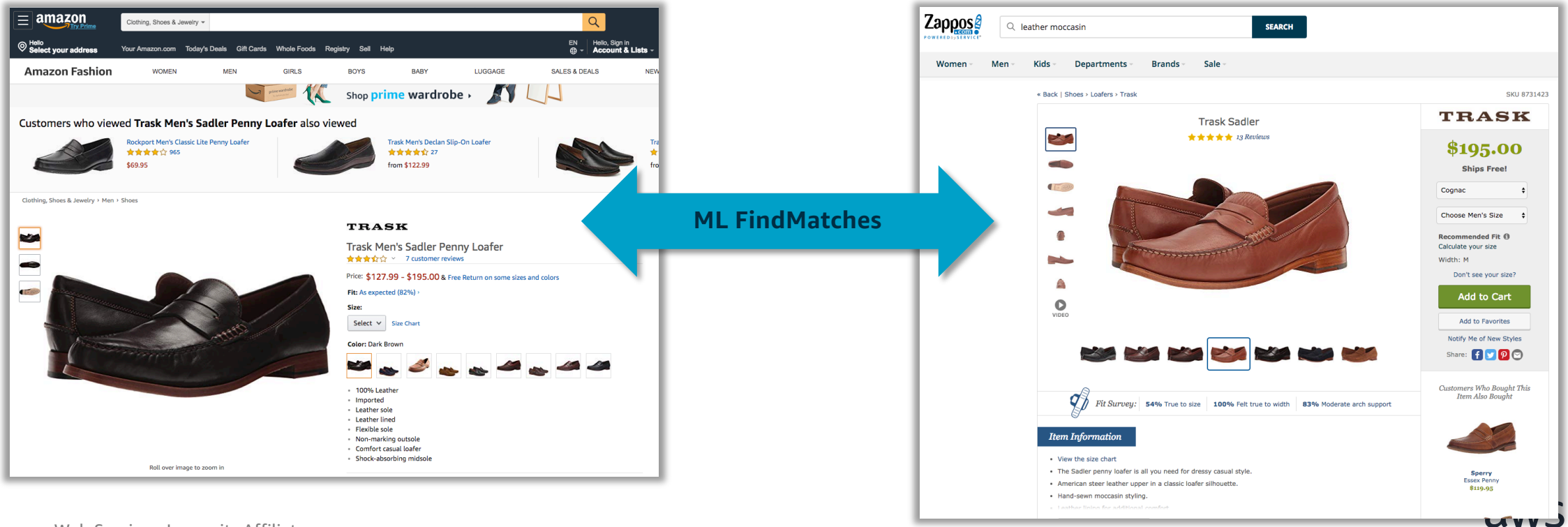
Leverage machine learning to solve hard problems

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*

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

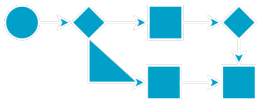
- Hive Metastore compatible with enhanced functionality
- Crawlers automatically extracts metadata and creates tables
- Integrated with Amazon Athena, Amazon EMR and many more

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



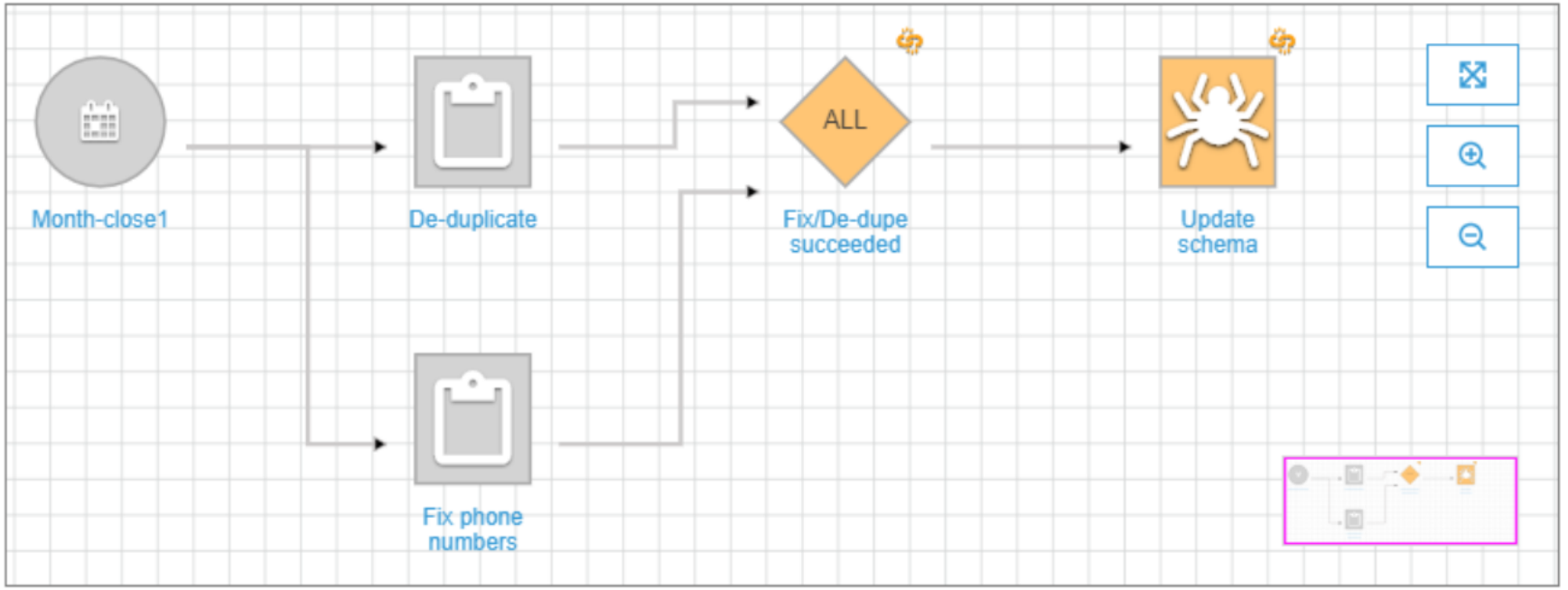
Job Workflow

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

aws

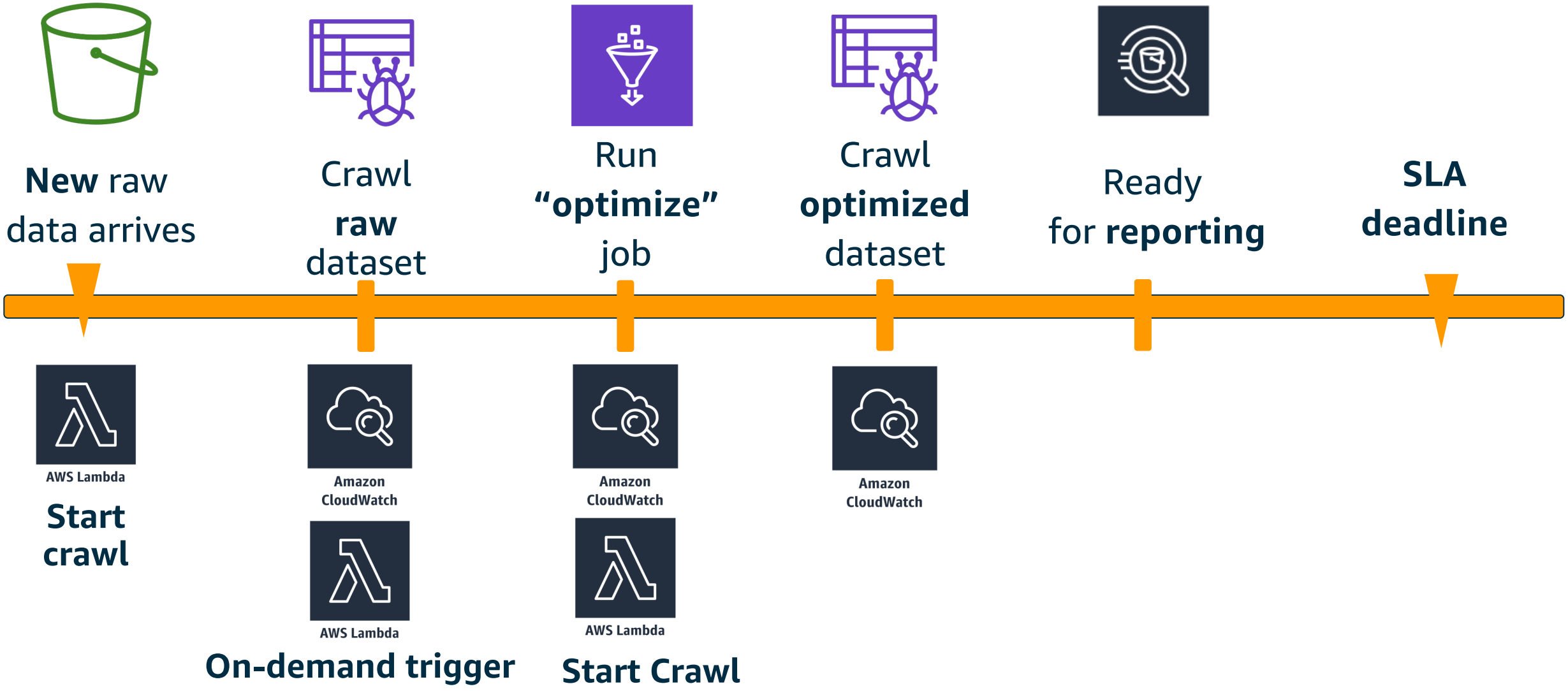
Glue Workflows

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



aws

Example event-driven workflow



Q & A