AWS SUMMIT

Introduction to AWS Glue

Simple, Flexible, Cost-Effective ETL

Debanjan Saha, General Manager
Amazon Aurora, Amazon RDS MySQL, AWS Glue

August 14, 2017

Today's agenda

- Why did we build AWS Glue?
- Main components of AWS Glue
- What did we announce today?

Why would AWS get into the ETL space?

We have lots of ETL partners

Amazon Redshift Partner Page for Data Integration

ATTUNITY CloudBeam

ironSource .atom

infoteria

bryte Systems

alooma

MATILLION

informatica Cloud

talend integration at any scale

Segment

Fivetran

etleap

xplenty

FlyData

TREASURE DATA

Qubole

RJMETRICS

snapLogic

amazon web services

The problem is

70% of ETL jobs are hand-coded

With **no** use of ETL tools.

Actually…

It’s **over 90%** in the cloud

Why do we see so much hand-coding?

Code is flexible | Code is powerful

You can unit test | You can deploy with other code | You know your dev tools

Hand-coding involves a lot of undifferentiated heavy lifting…

Brittle | Error-prone | Laborious

- As data formats change
- As target schemas change
- As you add sources
- As data volume grows

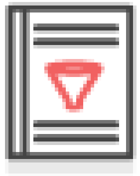
AWS Glue automates
the undifferentiated heavy lifting of ETL

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; you can also use their favorite tools to build ETL jobs

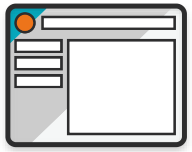
Deploy Run your jobs on a serverless, fully managed, scale-out environment. No compute resources to provision or manage.

AWS Glue: Components



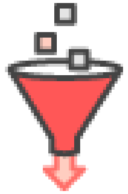
Data Catalog

- Hive Metastore compatible with enhanced functionality
- Crawlers automatically extracts metadata and creates tables
- Integrated with Amazon Athena, Amazon Redshift Spectrum



Job Authoring

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

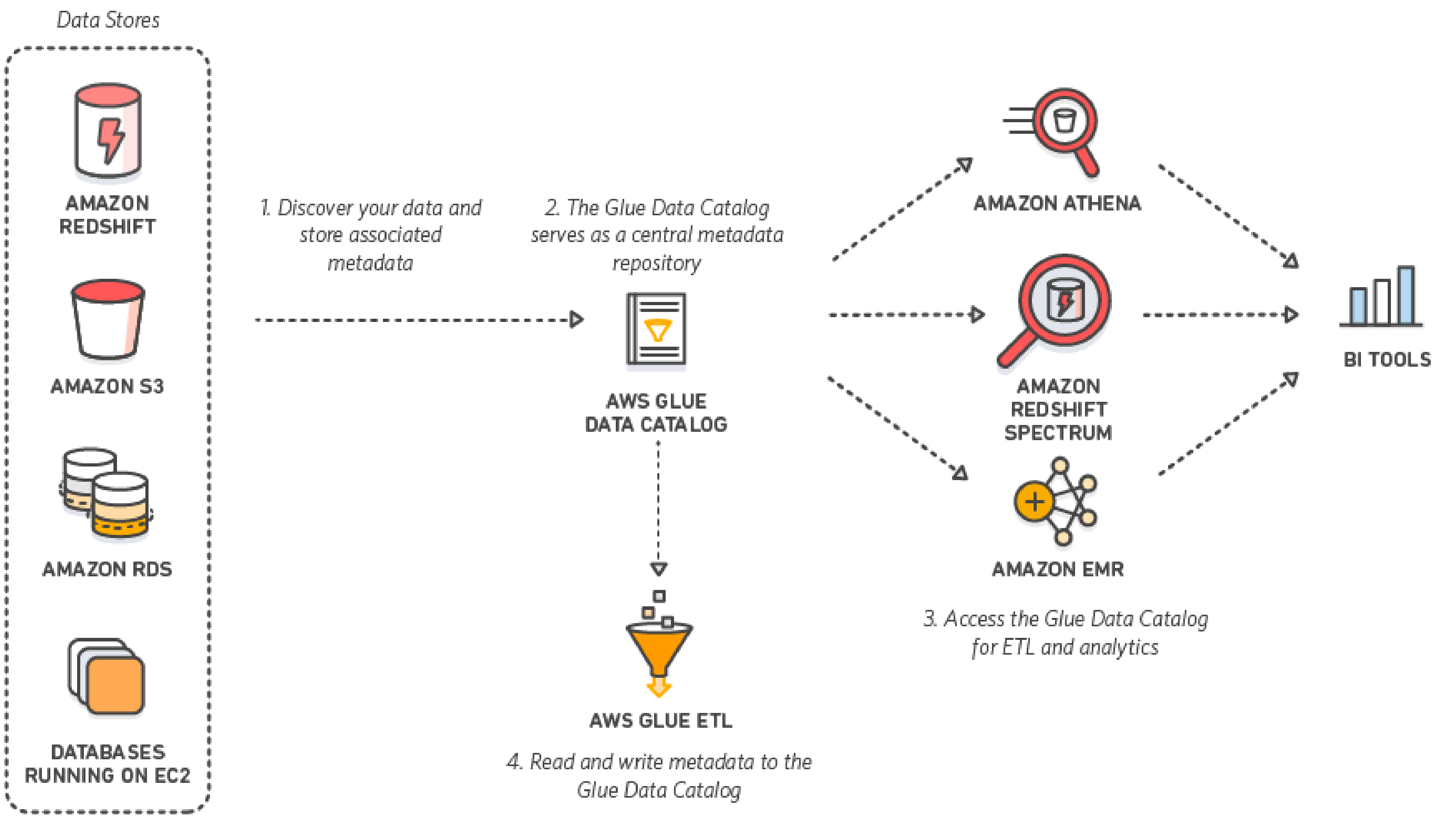


Job Execution

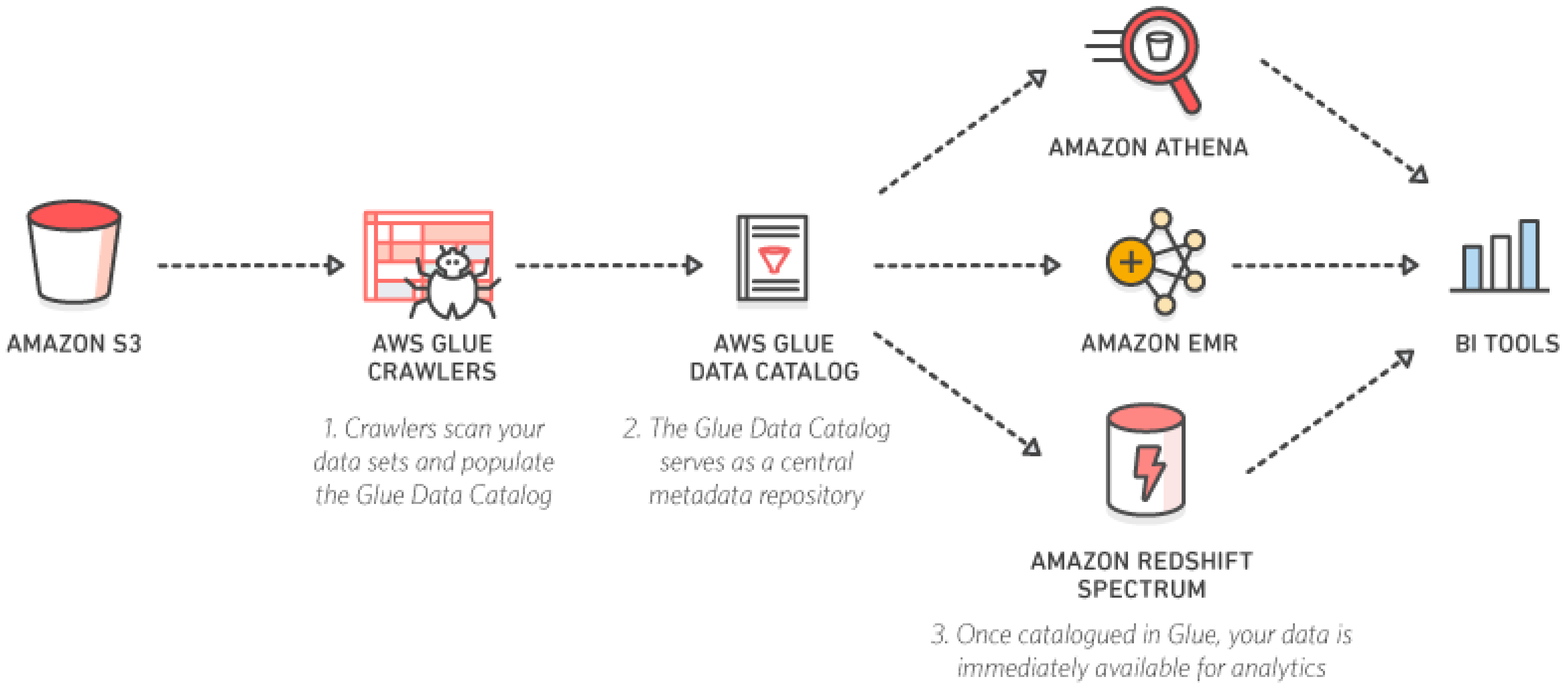
- Run jobs on a serverless Spark platform
- Provides flexible scheduling
- Handles dependency resolution, monitoring and alerting

Common use cases for AWS Glue

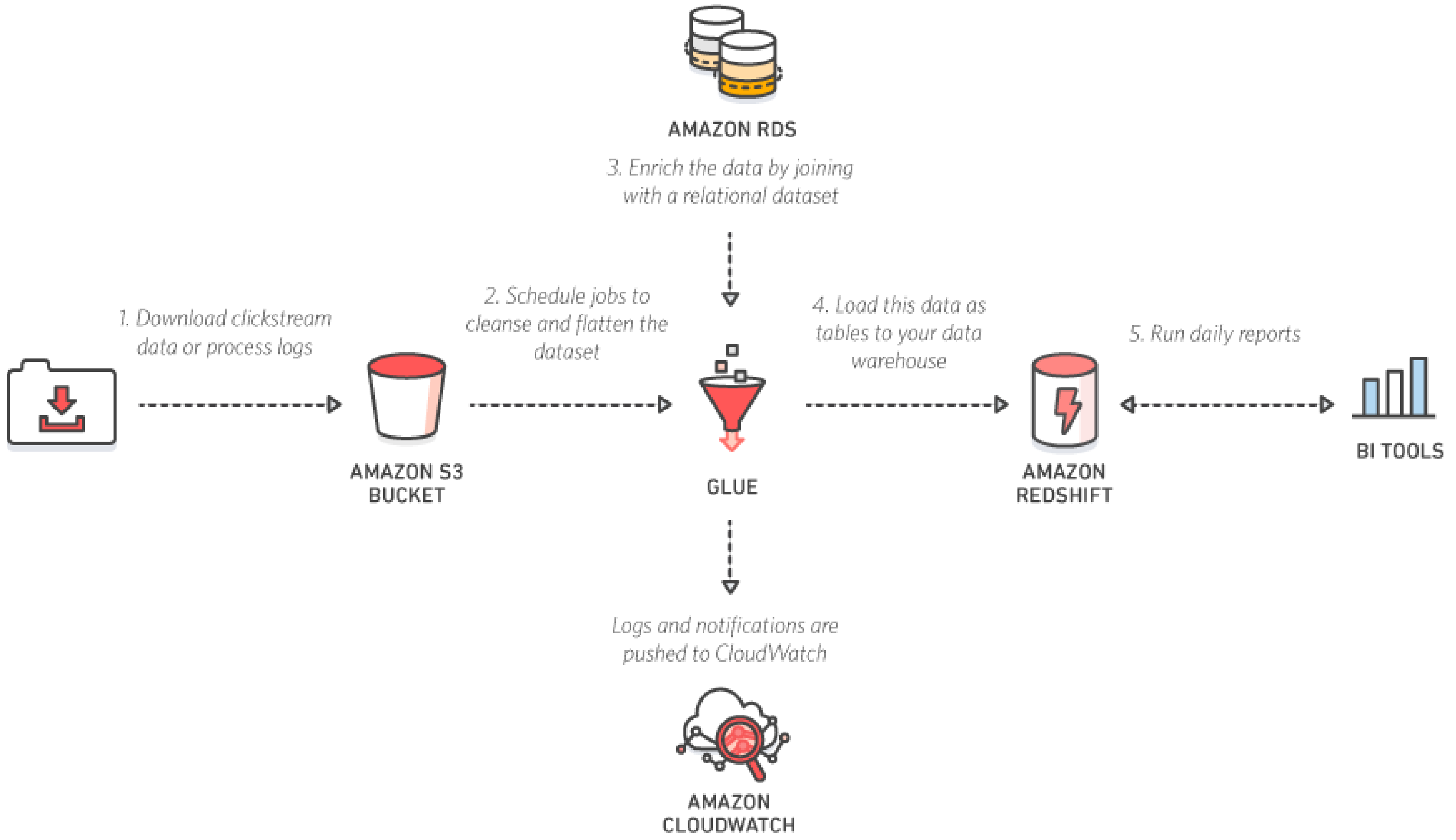
Understand your data assets



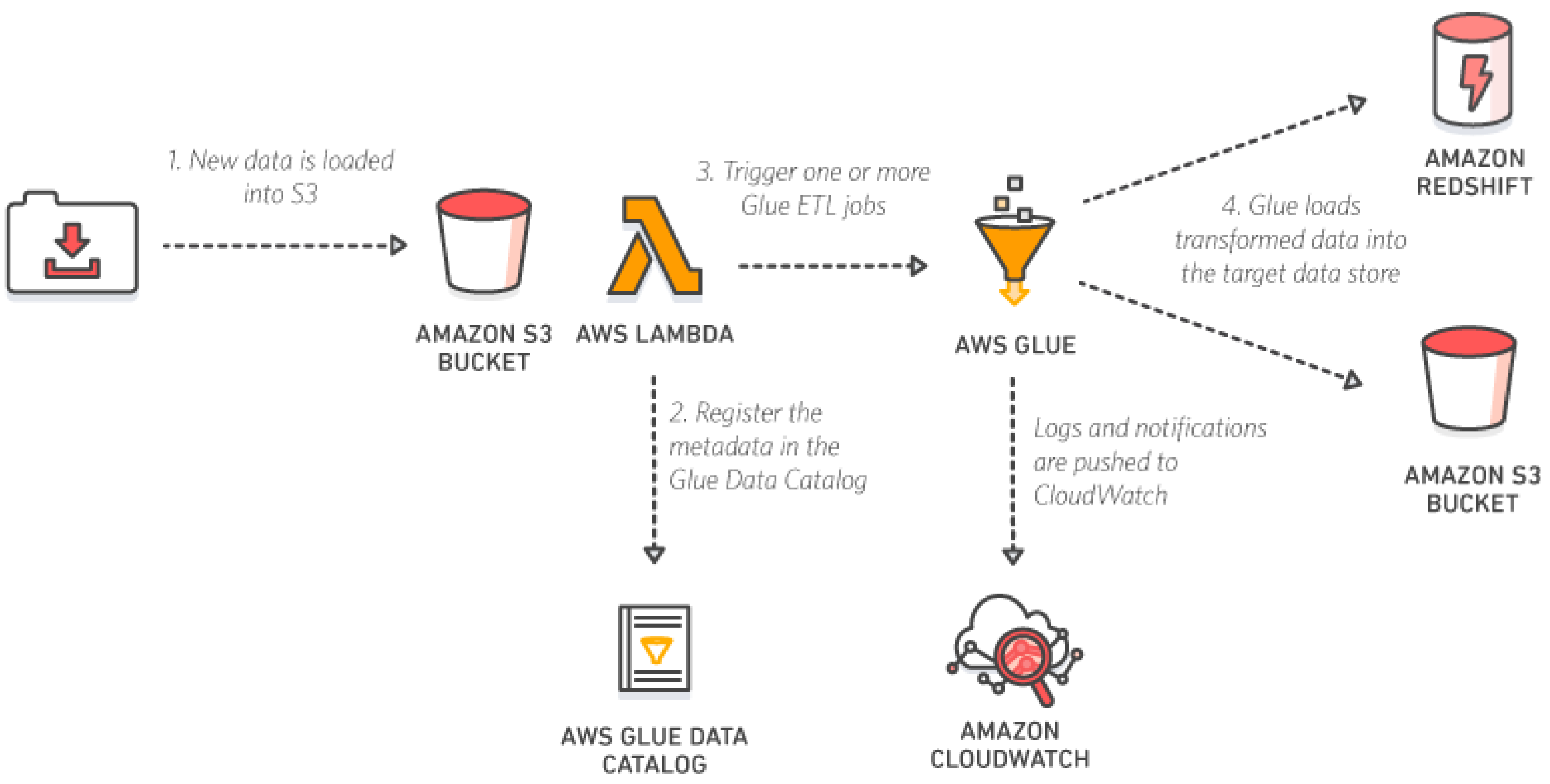
Instantly query your data lake on Amazon S3



ETL data into your data warehouse



Build event-driven ETL pipelines



Main components of AWS Glue

AWS Glue Data Catalog

Discover and organize your data

Glue data catalog

Manage table metadata through a Hive metastore API or Hive SQL.
Supported by tools like Hive, Presto, Spark etc.

We added a few extensions:

- **Search** over metadata for data discovery
- **Connection info** – JDBC URLs, credentials
- **Classification** for identifying and parsing files
- **Versioning** of table metadata as schemas evolve and other metadata are updated

Populate using Hive DDL, bulk import, or automatically through **Crawlers**.

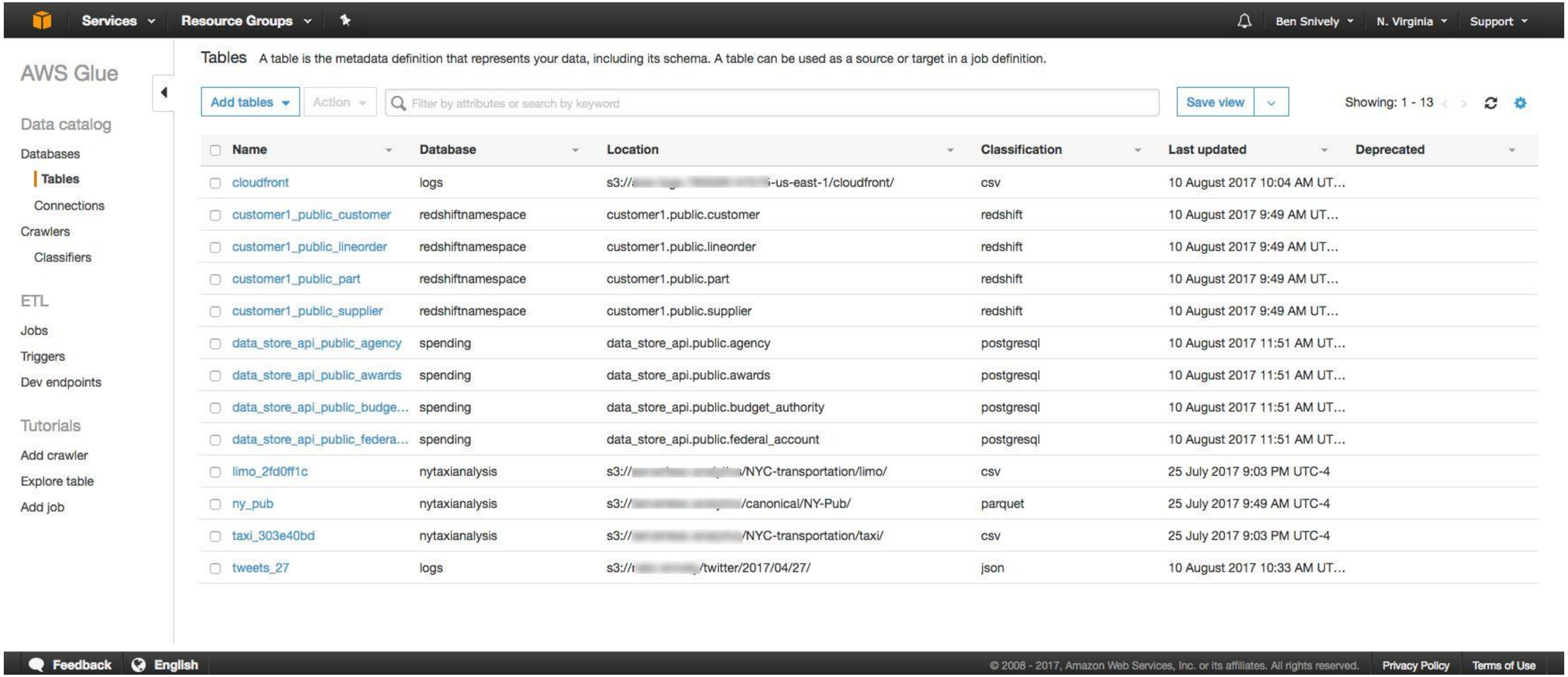
Data Catalog: Crawlers

Crawlers automatically build your Data Catalog and keep it in sync

- Automatically discover new data, extracts schema definitions
  - Detect schema changes and version tables
  - Detect Hive style partitions on Amazon S3
- Built-in classifiers for popular types; custom classifiers using Grok expressions
- Run ad hoc or on a schedule; serverless – only pay when crawler runs

AWS Glue Data Catalog

Bring in metadata from a variety of data sources (Amazon S3, Amazon Redshift, etc.) into a single categorized list that is searchable



Data Catalog: Table details

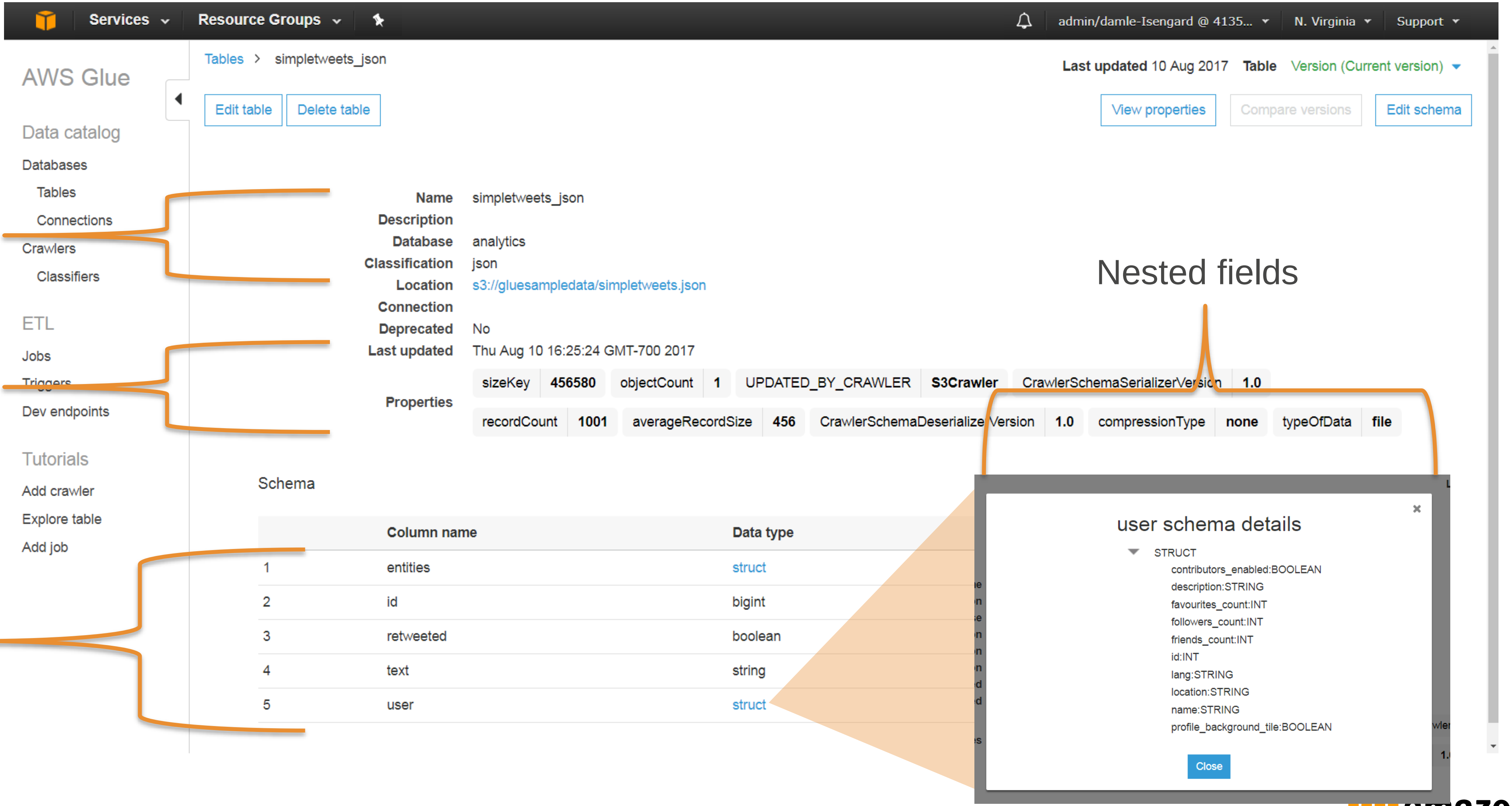


Table properties

Data statistics

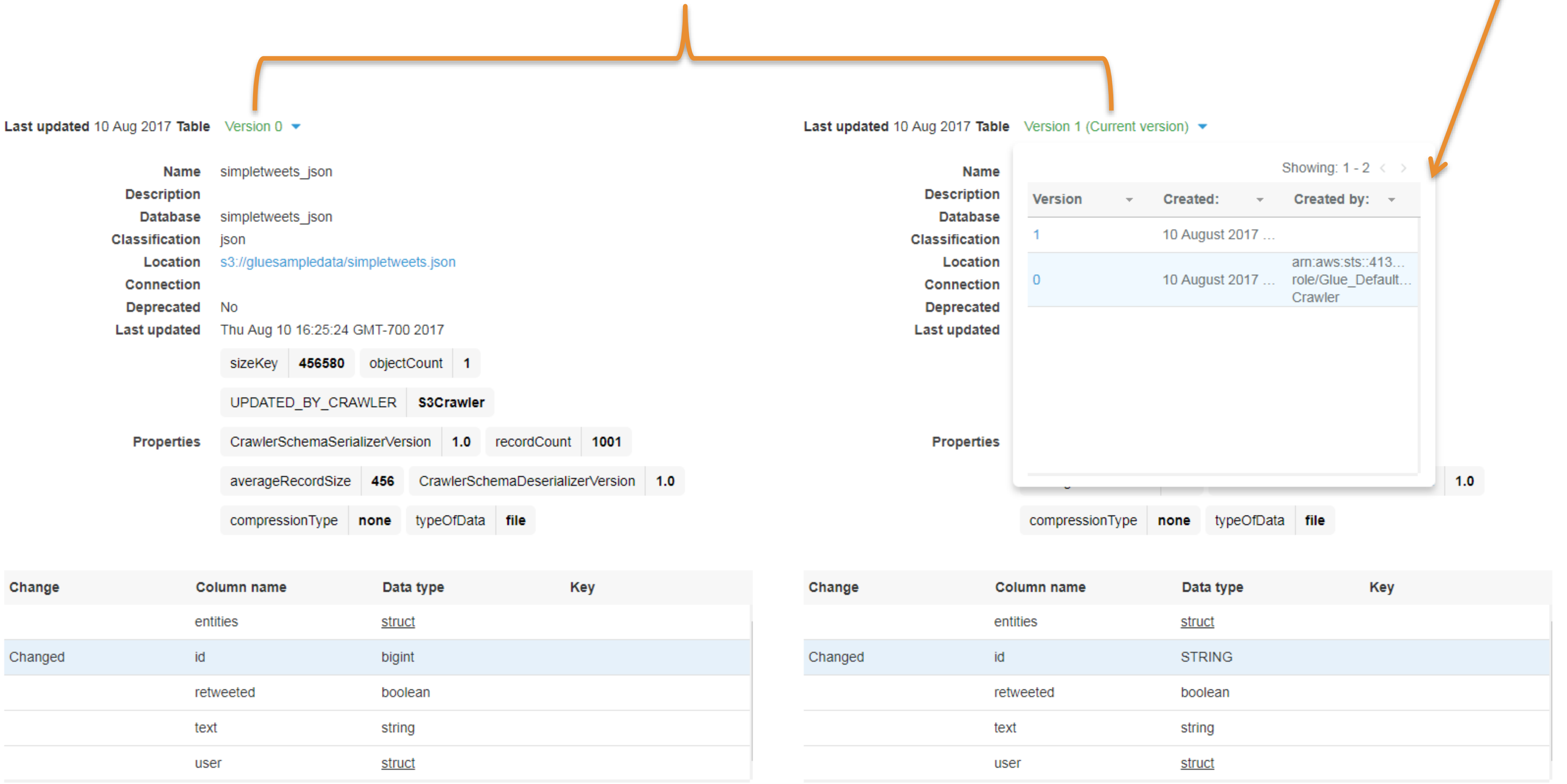
Nested fields

Table schema

Data Catalog: Version control

Compare schema versions

List of table versions



Data Catalog: Detecting partitions

S3 bucket hierarchy

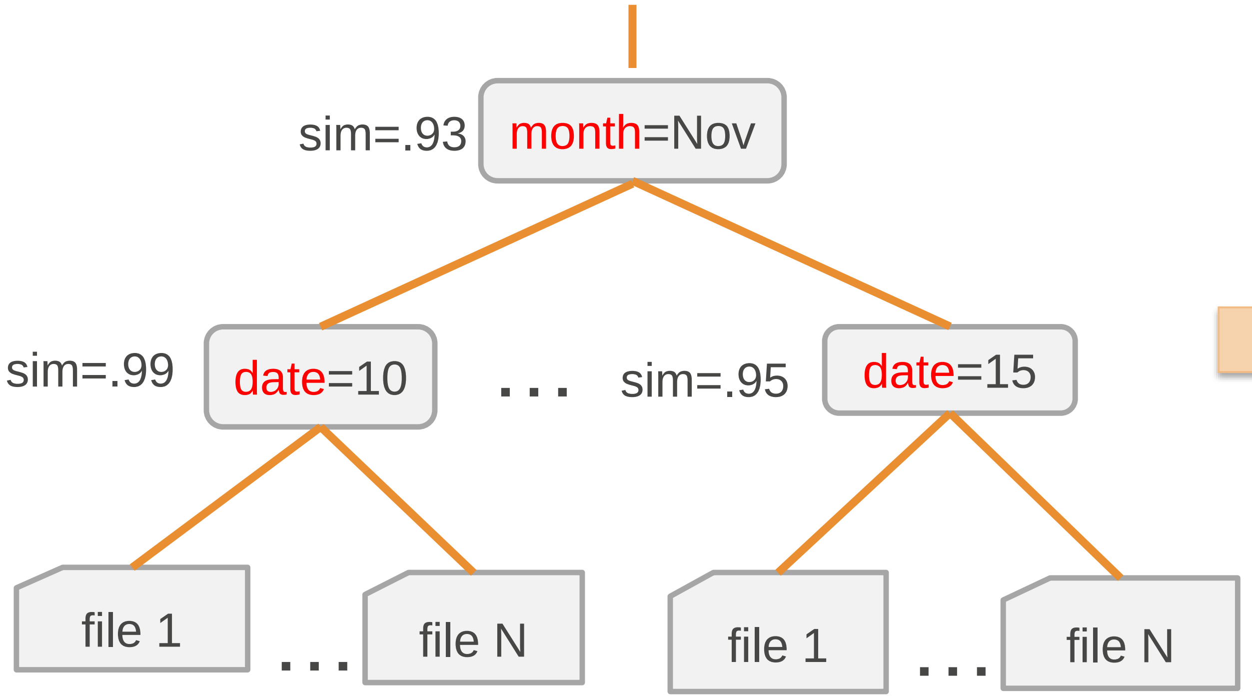


Table definition

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

Estimate schema similarity among files at each level to handle semi-structured logs, schema evolution…

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

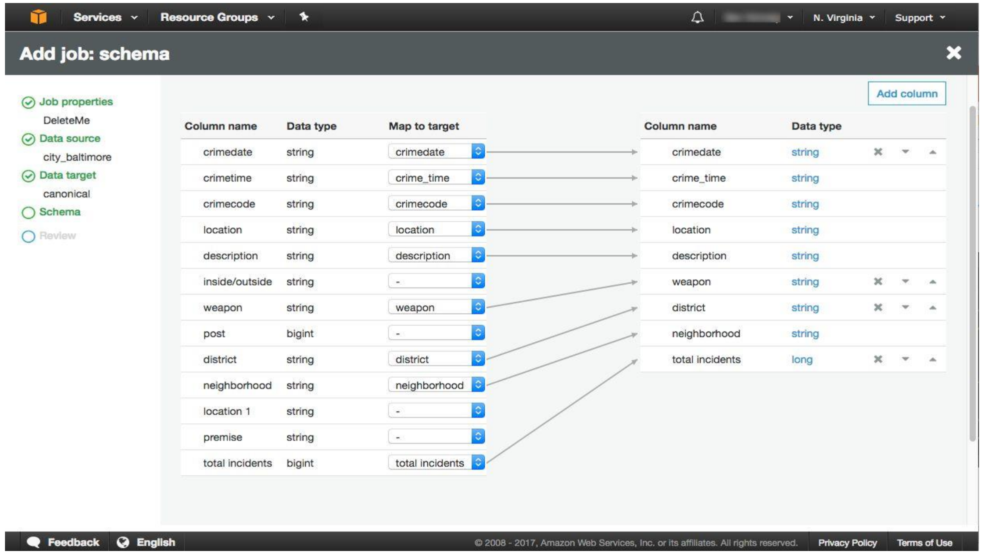
region	year	month	day		
us-west-2	2016	02	18	View files	View properties
us-west-2	2016	02	17	View files	View properties
us-west-2	2016	02	16	View files	View properties

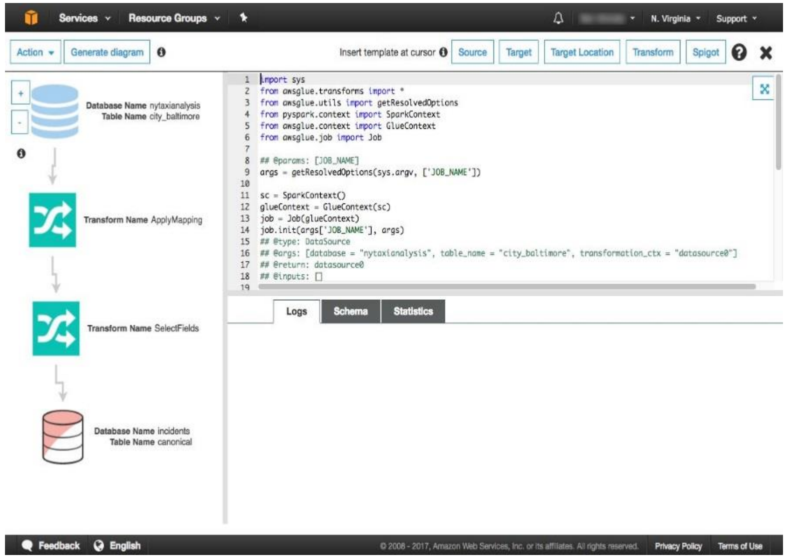
Job authoring in AWS Glue

You have choices on how to get started

- Python code generated by AWS Glue
- Connect a notebook or IDE to AWS Glue
- Existing code brought into AWS Glue

Job authoring: Automatic code generation





1. Customize the mappings
2. Glue generates transformation graph and **Python** code
3. Connect your **notebook** to development endpoints to customize your code

Job authoring: ETL code

- **Human-readable**, editable, and portable PySpark 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

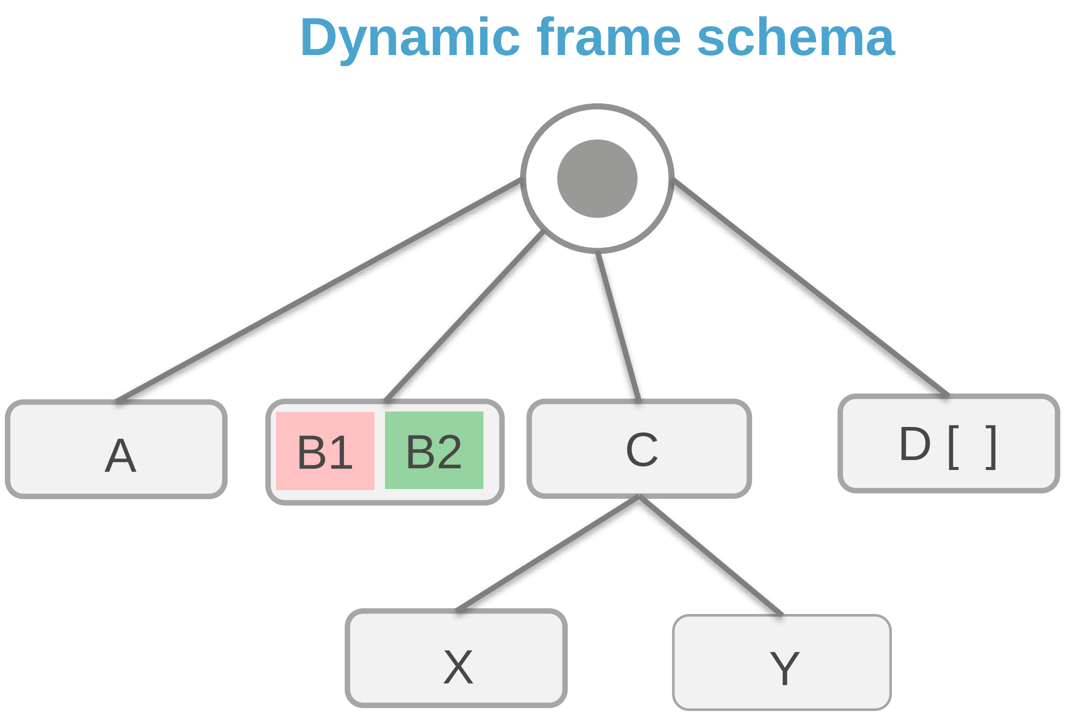
- **Customizable:** Use native PySpark, import custom libraries, and/or leverage Glue's libraries

```
####################################################
##
##  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

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**

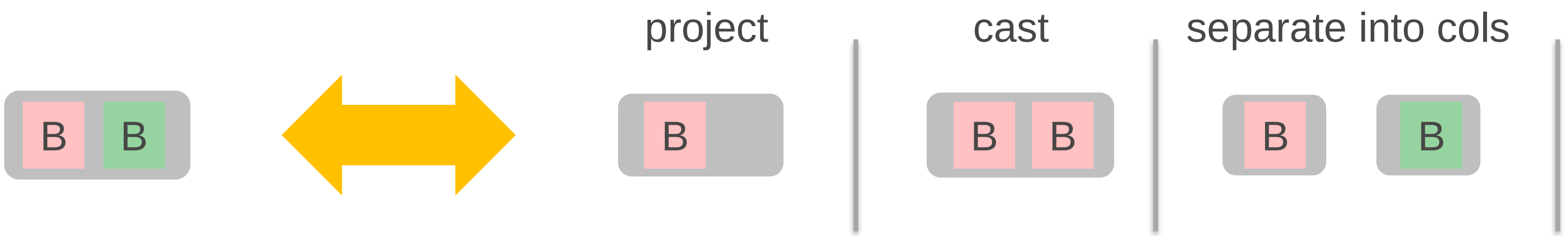
Easy to handle the unexpected:

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

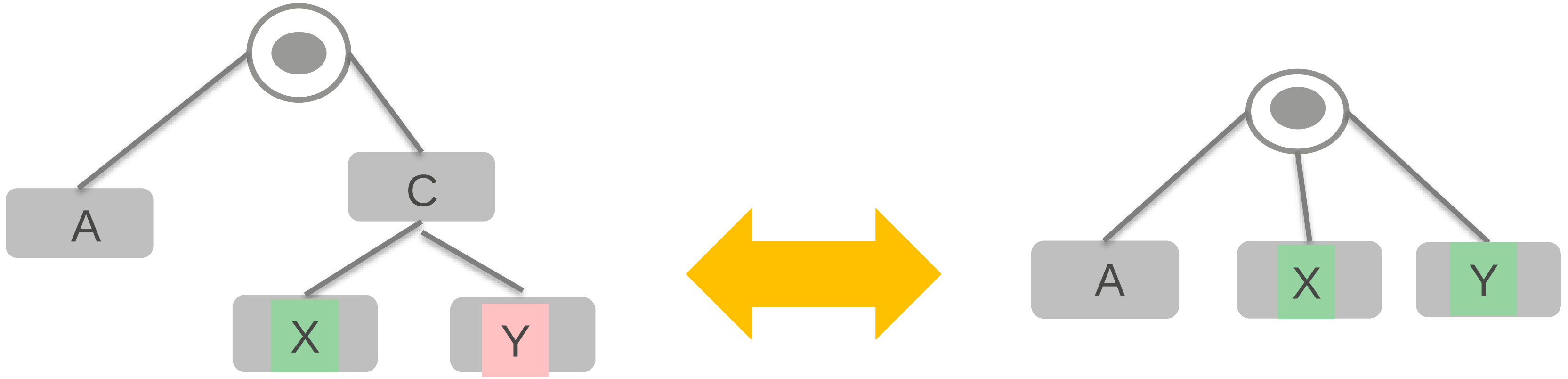
Job Authoring: Glue transforms

Adaptive and flexible

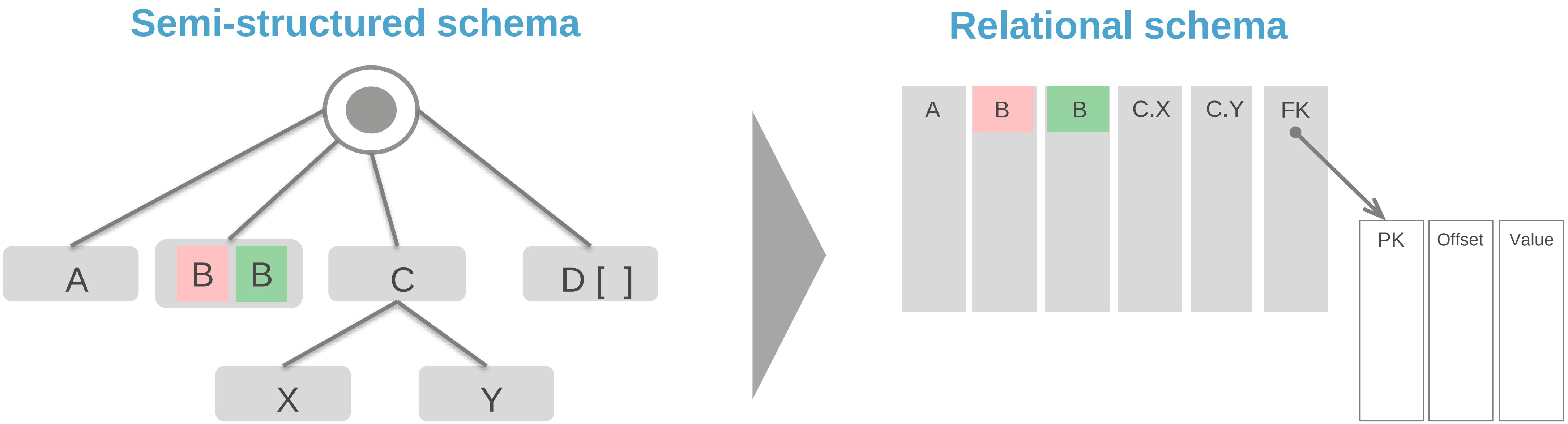
ResolveChoice()



Apply Mapping()



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

Job authoring: Glue transformations

Add transform

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

- **Prebuilt transformation:** Click and add to your job with simple configuration
- **Spigot** writes sample data from DynamicFrame to S3 in JSON format
- **Expanding…** more transformations to come

Job authoring: Write your own scripts

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())])
```

Convert to a Spark Data Frame
for complex SQL-based ETL

Convert back to Glue Dynamic Frame
for semi-structured processing and AWS Glue connectors

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

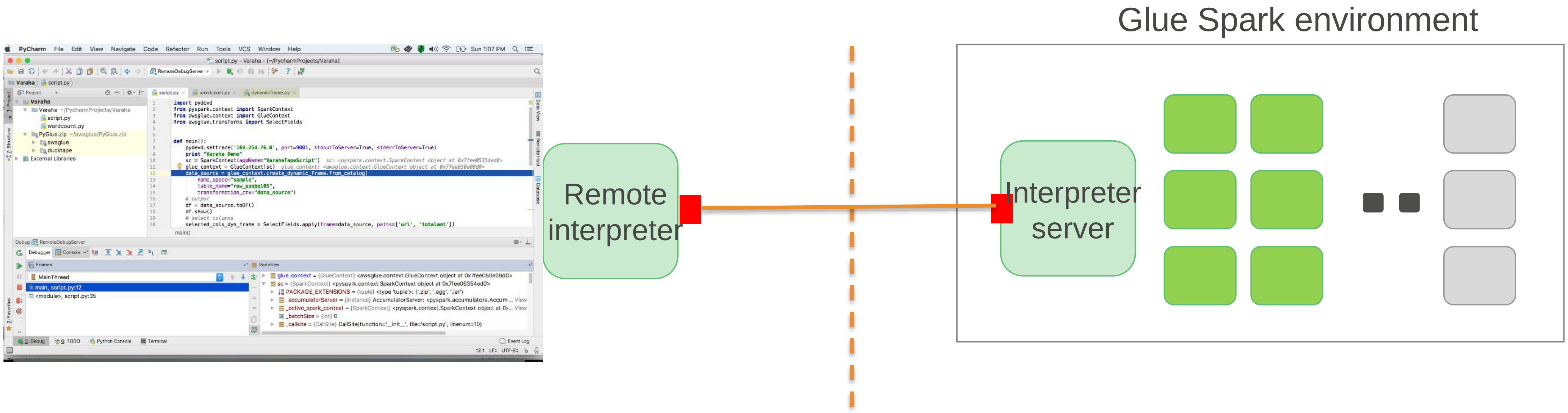
s3://bucket-name/folder-name/file-name

Referenced files path

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

Import custom libraries required by your code

Job authoring: Developer endpoints



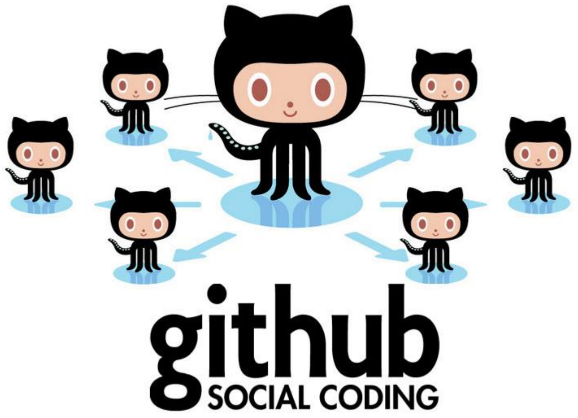
- Environment to **iteratively develop** and test ETL code.
- Connect your IDE or notebook (e.g. **Zeppelin**) to a Glue development endpoint.
- When you are satisfied with the results you can create an ETL job that runs your code.

Job Authoring: Leveraging the community

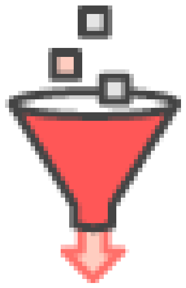
No need to start from scratch.

Use **Glue samples** stored in Github to share, reuse, contribute: https://github.com/awslabs/aws-glue-samples

- Migration scripts to import existing Hive Metastore data into AWS Glue Data Catalog
- Examples of how to use Dynamic Frames and Relationalize() transform
- Examples of how to use arbitrary PySpark code with Glue's Python ETL library



Download **Glue's Python ETL library** to start developing code in your IDE: https://github.com/awslabs/aws-glue-libs



Orchestration and resource management

Fully managed, serverless job execution

Job execution: Scheduling and monitoring

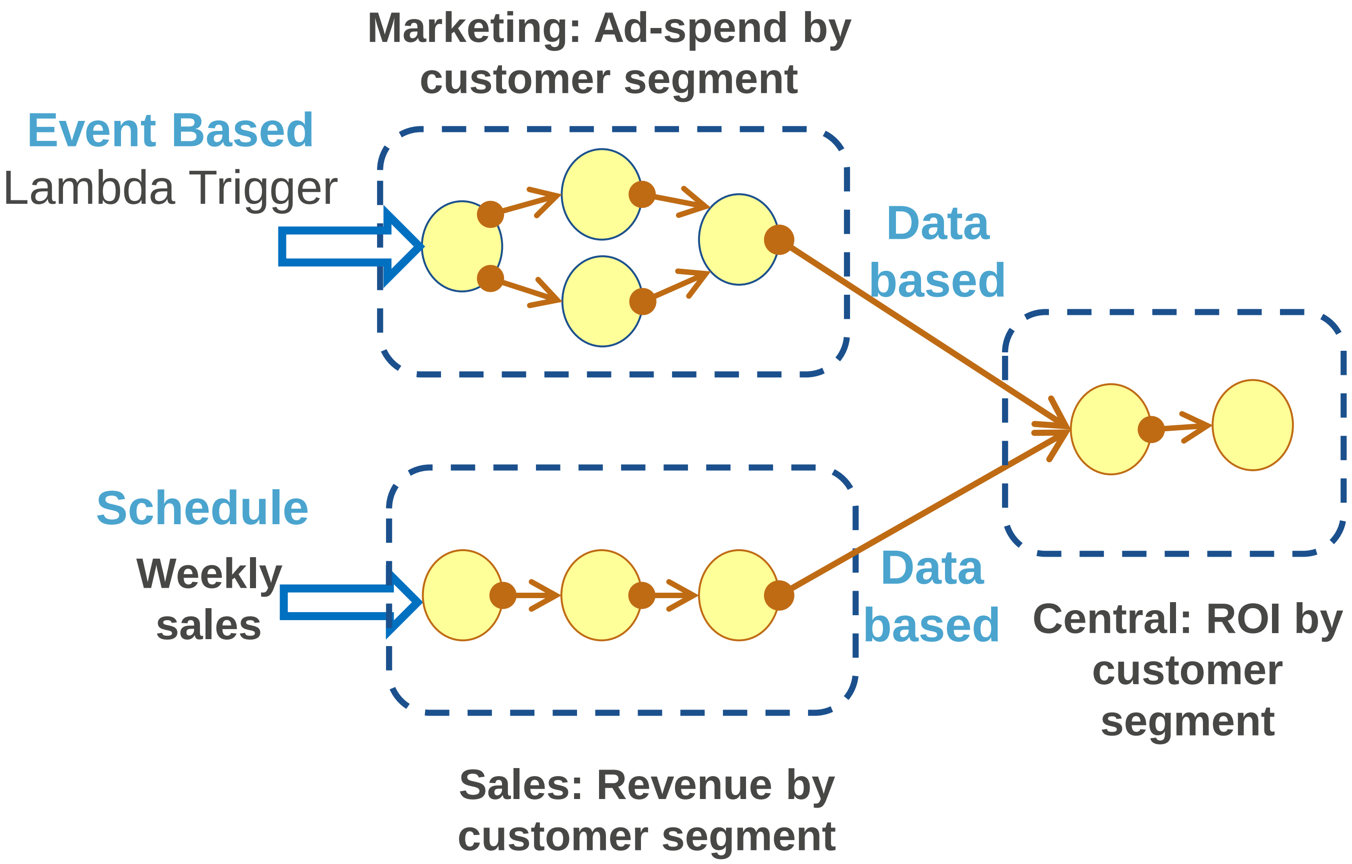
Compose jobs globally with event-based dependencies

- 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
- **More coming soon:** Data Catalog based events, S3 notifications and Amazon CloudWatch events

Logs and alerts are available in Amazon CloudWatch

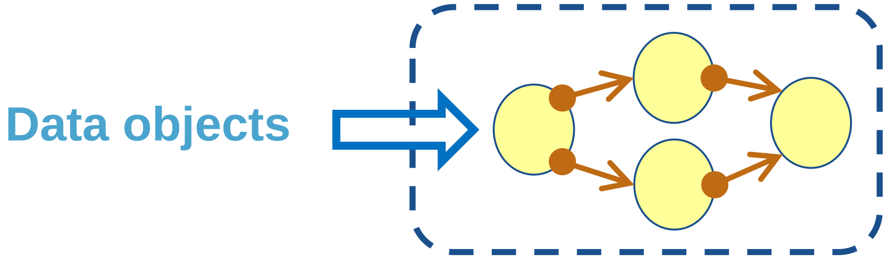


Job execution: Job bookmarks

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.

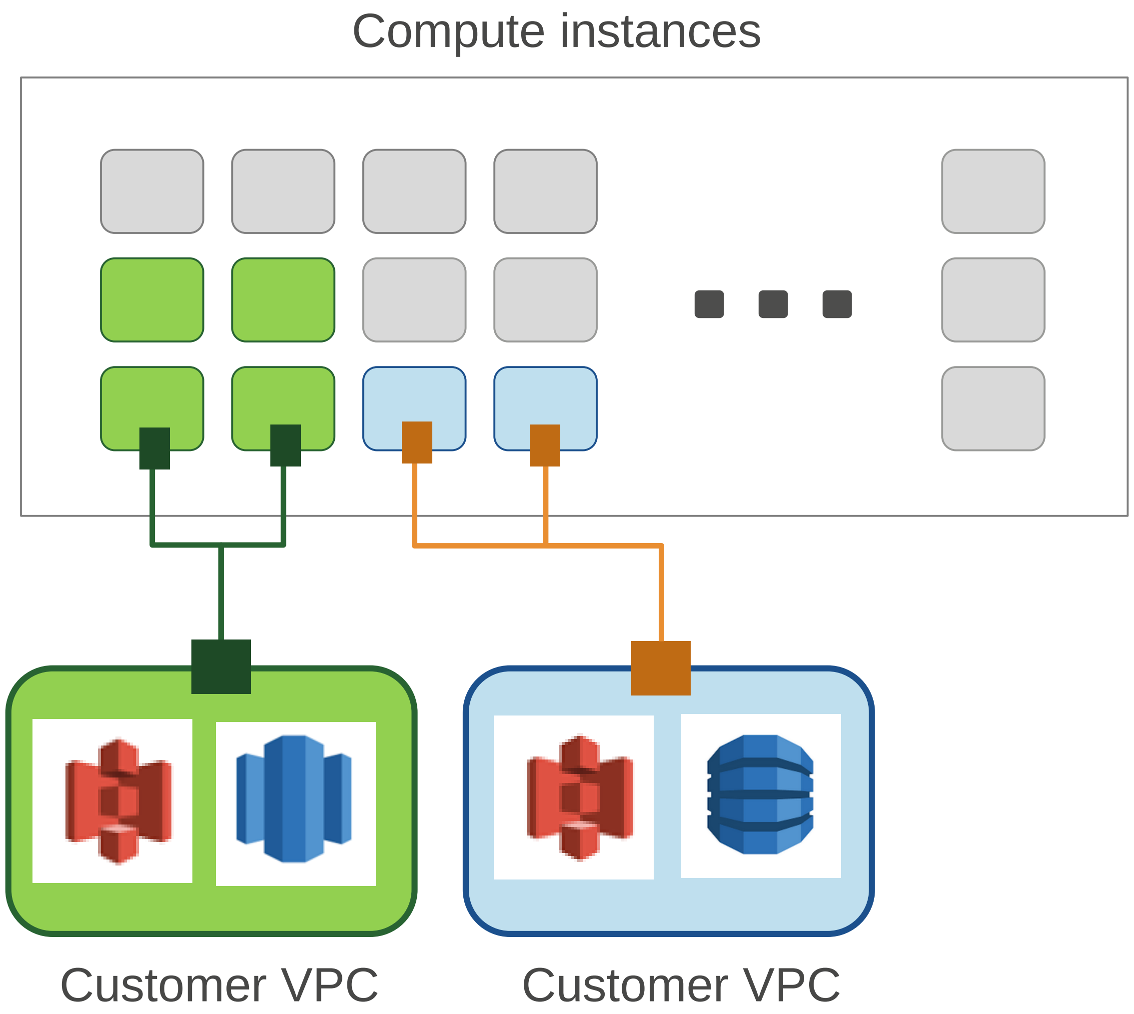


Marketing: Ad-spend by customer segment

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 pricing examples

AWS Glue pricing

Compute based usage:

ETL jobs, development endpoints, and crawlers

$0.44 per DPU-Hour, 1 minute increments.10-minute minimum

A single DPU Unit = 4 vCPU and 16 GB of memory

Data Catalog usage:

Data Catalog Storage:

Free for the **first million objects** stored

$1 per 100,000 objects, per month, stored above 1M

Data Catalog Requests:

Free for the **first million requests** per month

$1 per million requests above 1M

Glue ETL pricing example

Consider an ETL job that ran for **10 minutes** on a **6 DPU** environment.

The price of 1 DPU-Hour in US East (N. Virginia) is **$0.44**.
The cost for this job run = 6 DPUs * (10/60) hour * $0.44 per DPU-Hour or **$0.44**.

Now consider you provision a development endpoint to debug the code for this job and keep the development endpoint active for **24 min**.

Each development endpoint is provisioned with **5 DPUs**
The cost to use the development endpoint = 5 DPUs * (24/ 60) hour * 0.44 per DPU-Hour or **$0.88**.

Glue Data Catalog pricing example

Let’s consider that you store **1 million tables** in your Data Catalog in a given month and make **1 million requests** to access these tables.

You pay **$0** for using data catalog. You are covered under the **Data Catalog free tier**.

Now consider your requests double to **2 million requests.**

You will only be paying for one million requests above the free tier, which is **$1**

If you use crawlers to find new tables and they run for **30 min** and use **2 DPUs**.

You will pay for 2 DPUs * (30/60) hour * $0.44 per DPU-Hour or **$0.44**.
Your total monthly bill = $0 + $1 + $0.44 or **$1.44**

AWS Glue regional availability

AWS Glue regional availability plan

Planned schedule	Regions
At launch	US East (N. Virginia)
Q3 2017	US East (Ohio), US West (Oregon)
Q4 2017	EU (Ireland), Asia Pacific (Tokyo), Asia Pacific (Sydney)
2018	Rest of the public regions

Q&A

AWS

SUMMIT

Thank you!