



Building Serverless ETL Pipelines With AWS Glue

Ben Thurgood

Principal Solutions Architect, Amazon Web Services

Can I get you
to go ahead
and...



...prepare
our data for
analysis





Generate

Connected
devices



Web logs /
cookies



Social
media



Transactions



ERP



Collect

Store

**Extract
Transform
Load**

Analyse

**Visualise/
Report**

Collect



Polling Application

Generate



Amazon
Kinesis Firehose

Store

**Extract
Transform
Load**

Analyse

**Visualise/
Report**



Amazon
Kinesis Stream

Collect



AWS DMS

Generate



Amazon S3

Store

**Extract
Transform
Load**

Analyse

**Visualise/
Report**



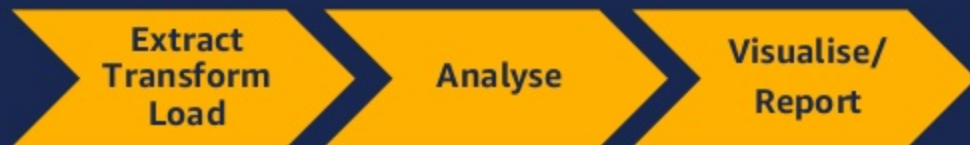
AWS Snowball



Amazon S3



Amazon
RDS



Database
on EC2





**Extract
Transform
Load**



Amazon EMR

Generate

Collect

Store



AWS
Lambda

Analyse

**Visualise/
Report**



Amazon
Kinesis Enabled
















No Problem...?
















Deal with these **Terabytes** and **Petabytes** of data








Simplify querying **disparate data** sets

Combine existing / **legacy data** with modern data sets

Prepare data for **machine learning**

<input type="checkbox"/>	Name 	Last modified 	Size 	Storage class 
<input type="checkbox"/>	 04dc3925_SO00016	--	--	--
<input type="checkbox"/>	 0608917d_SO0002	--	--	--
<input type="checkbox"/>	 07a46e08_SO0012	--	--	--
<input type="checkbox"/>	 14374589_SO0009	--	--	--
<input type="checkbox"/>	 1c1e778f_SO0008	--	--	--
<input type="checkbox"/>	 1f6c8c88_SO0007	--	--	--
<input type="checkbox"/>	 2fcb2661_SO0015	--	--	--
<input type="checkbox"/>	 3225cb4a_SO00010	--	--	--
<input type="checkbox"/>	 363eb7cb_SO0018	--	--	--
<input type="checkbox"/>	 5a6dfbe5_SO0009s	--	--	--
<input type="checkbox"/>	 75d6f6be_SO00001	--	--	--

<input type="checkbox"/>	Name 	Last modified 	Size 	Storage class 
<input type="checkbox"/>	 0051a423-8e40-406e-9db4-d1e88e496101	--	--	--
<input type="checkbox"/>	 02c70d2c-8ce4-4d51-be00-c8a5b8270b01	--	--	--
<input type="checkbox"/>	 16cee6b1-0eea-4042-9b9f-dcae989e2c1f	--	--	--
<input type="checkbox"/>	 16f23b83-95e4-40b4-927a-37713b907266	--	--	--
<input type="checkbox"/>	 195f54e8-5f84-44a3-8a9a-c3165211c5bb	--	--	--
<input type="checkbox"/>	 1e4d0fb1-0964-46b5-bf9b-0fd09b44c9e4	--	--	--
<input type="checkbox"/>	 1f3e86af-f3a6-4cb5-a3ec-a664b6d6c09f	--	--	--
<input type="checkbox"/>	 2203aebd-bd7f-42b9-b1e6-90ac11f8d030	--	--	--
<input type="checkbox"/>	 25eeff5c-af58-4d52-93f7-a83ab6f5fe80	--	--	--
<input type="checkbox"/>	 3334a568-af1a-4c65-a450-219f5282f673	--	--	--
<input type="checkbox"/>	 34805536-0d3a-4743-be5d-05776764522d	--	--	--

<input type="checkbox"/>	Name ↑☰	Last modified ↑☰	Size ↑☰	Storage class ↑☰
<input type="checkbox"/>	 0051a423-8e40-406e-9db4-d1e88e496101	--	--	--
<input type="checkbox"/>	 02c70d2c-8ce4-4d51-be00-c8a5b8270b01	--	--	--
<input type="checkbox"/>	<pre> 2 3 [{"ground_speed": "230", "spi": "0", "lastUpdated": 1510202706.596984, 4 "callsign": "QLK476D ", "is_on_ground": "0", "altitude": "3600", 5 "alert": "0", "emergency": "0", "message_type": "MSG", 6 "transmission_type": "5", "lat": "-34.16263", "hex_ident": "7C530C", 7 "lon": "151.12625", "vertical_rate": "-768", "track": "77", "squawk": "4263"}] 8 </pre>			
<input type="checkbox"/>	 1f3e86af-f3a6-4cb5-a3ec-a664b6d6c09f	--	--	--
<input type="checkbox"/>	 2203aebd-bd7f-42b9-b1e6-90ac11f8d030	--	--	--
<input type="checkbox"/>	 25eeff5c-af58-4d52-93f7-a83ab6f5fe80	--	--	--
<input type="checkbox"/>	 3334a568-af1a-4c65-a450-219f5282f673	--	--	--
<input type="checkbox"/>	 34805536-0d3a-4743-be5d-05776764522d	--	--	--

Some extra challenges..

Volumes will **grow** (the new oil)

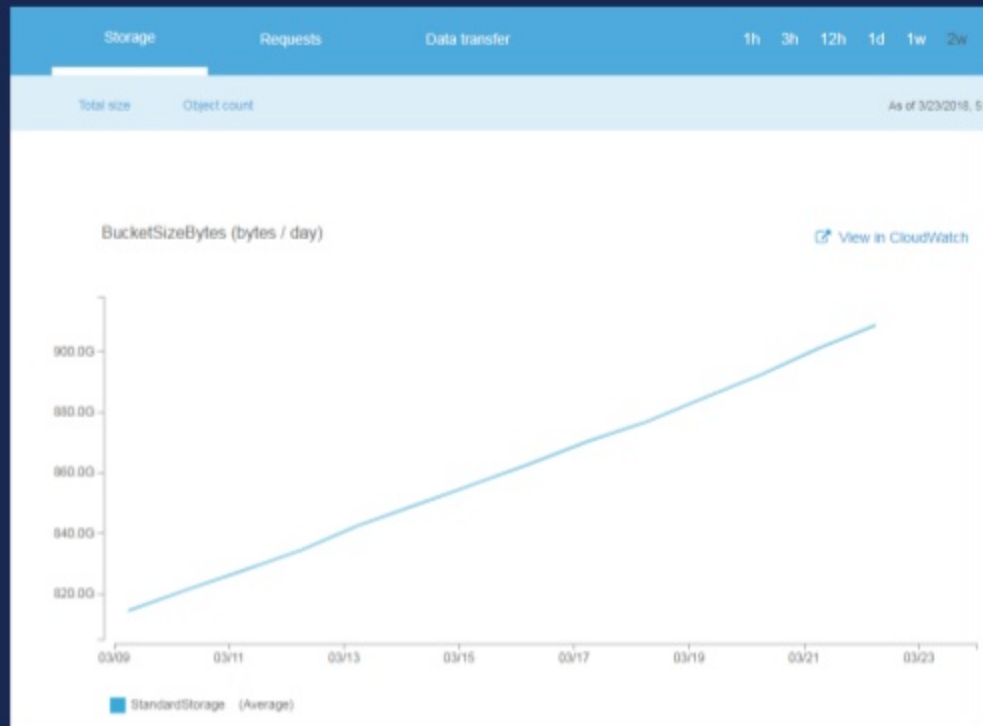
Adding data sources

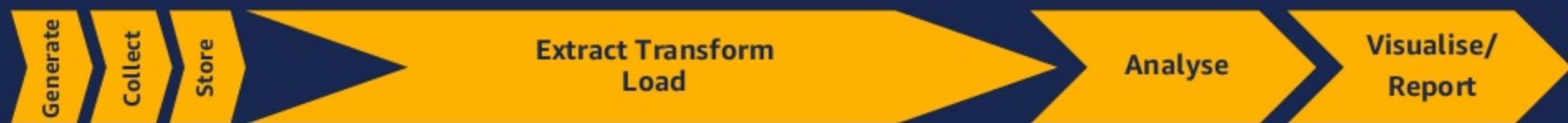
Large proportion of ETL is **hand coding**

Data **formats change** over time

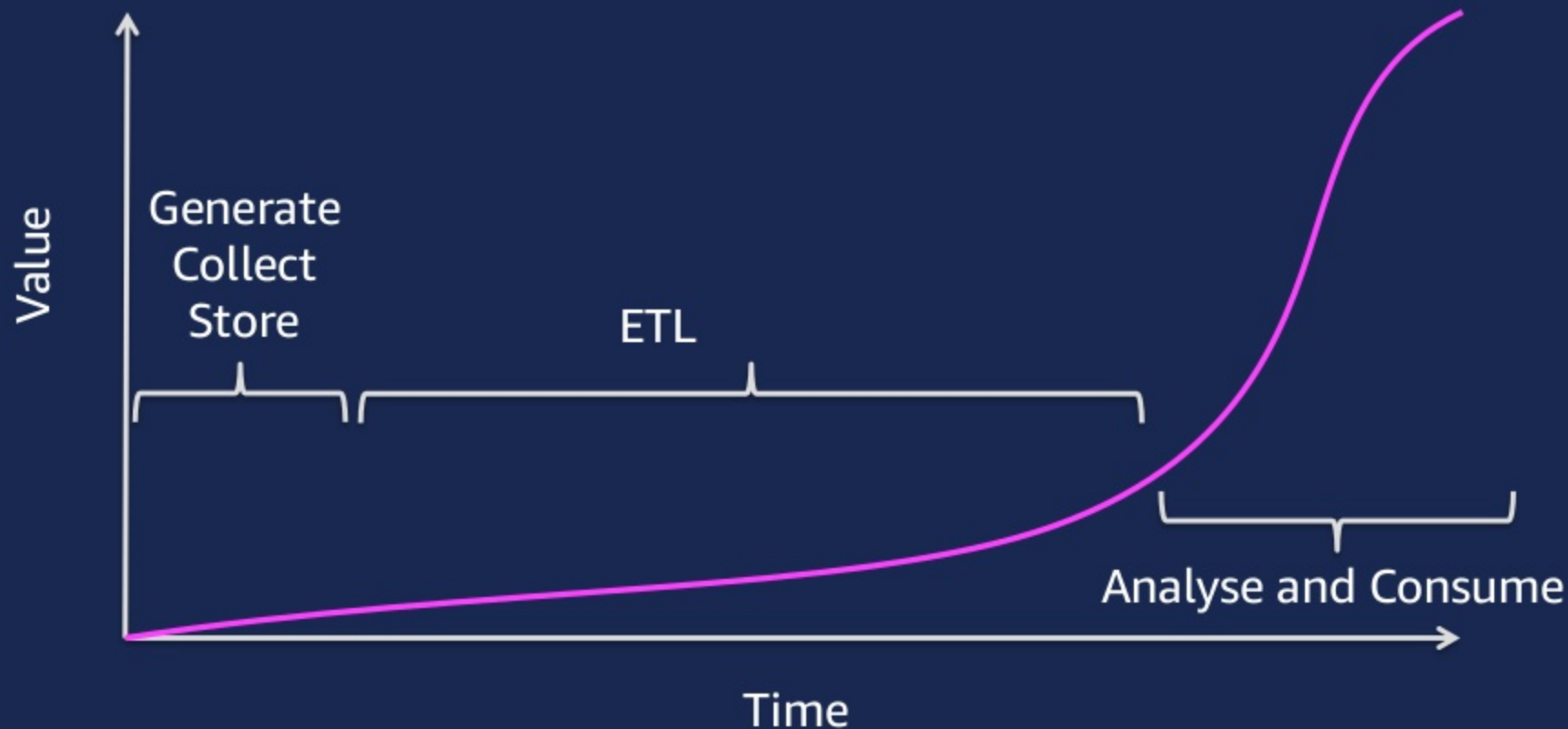
- Within data you already have
- Changes will be coming soon

Target **schemas change**





And.. ETL Is Not The Rewarding Part





Why AWS Glue?

Automate your ETL

Automatically **discover and categorise** your data

- Connect to your data sources
- Generate your Data Catalogue

Make it **immediately** searchable and **queryable**

- Athena
- Redshift
- EMR

Automate your ETL

Generates your ETL code

- Clean
- Enrich
- Move

Adaptable code

Extension to Spark in Python or Scala

Automate your ETL

Runs your ETL jobs **serverless**

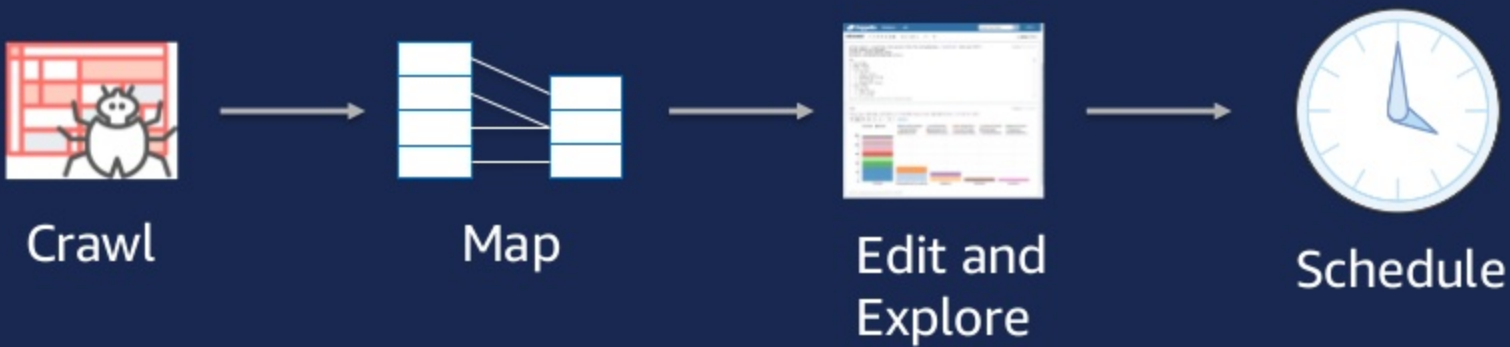
- Managed
- Control the amount of resources used
- Scales out automatically

Schedule or **trigger** jobs

Why Glue Examples

How do I ETL my data?

Four Steps



How Do I Discover My Data?



Glue Data Catalogue: Crawlers



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

Crawlers: Classifiers



IAM Role

Glue Crawler



JDBC Connection

Object Connection

Databases



Amazon
RDS

Data Warehouse



Amazon
Redshift

Data Lakes



Amazon S3

Create additional Custom
Classifiers with Grok!

Built-In Classifiers

MySQL
MariaDB
PostgreSQL
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)

Example Classifier

2018-03-18T01:44:19+00:00 [prefix-p-123-a-7z] WARN: There is a message

Grok expression example:

```
%{TIMESTAMP_ISO8601:timestamp} \[%{MESSAGEPREFIX:message_prefix}\] %{CRAWLERLOGLEVEL:loglevel} :  
%{GREEDYDATA:message}
```

Built in patterns:

```
TIMESTAMP_ISO8601 %{YEAR}-%{MONTHNUM}-%{MONTHDAY}[T ]%{HOUR}:%{MINUTE}(:%{SECOND})?%{ISO8601_TIMEZONE}?  
GREEDYDATA .*
```

Custom patterns

```
CRAWLERLOGLEVEL (BENCHMARK|ERROR|WARN|INFO|TRACE)  
MESSAGEPREFIX .*-.*-.*-.*-.*
```

Handy Grok debugger:
<https://grokdebug.herokuapp.com/>

Crawler: Detecting Partitions

S3 bucket hierarchy

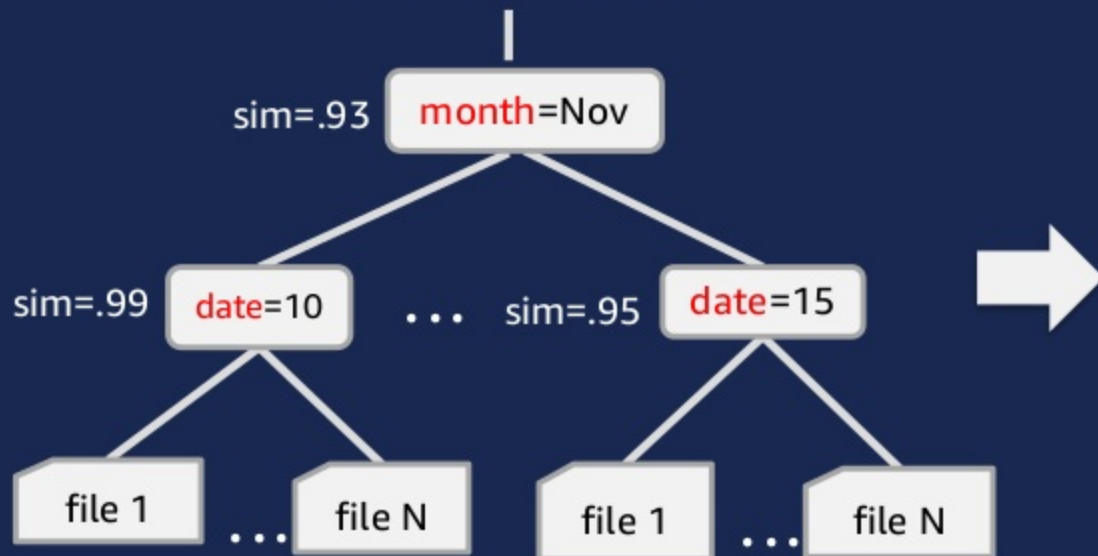


Table definition

Column	Type
<code>month</code>	str
<code>date</code>	str
col 1	int
col 2	float
⋮	⋮

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

Glue Data Catalog

The screenshot shows the AWS Glue Data Catalog console. The left sidebar contains navigation links for Databases, Tables (selected), Connections, Crawlers, Classifiers, ETL, Jobs, Triggers, Dev endpoints, Tutorials, Add crawler, Explore table, Add job, Resources, and What's new. The main content area is titled 'Tables' and includes a description: '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.' Below this is a search bar with the text 'Filter by attributes or search by keyword' and a 'Save view' button. A table lists various data sources with columns for Name, Database, Location, Classification, Last updated, and Deprecated. The table contains 12 rows of data, including tables like 'accountcost', 'amazonathena_test', 'data', 'etb_logs', and multiple instances of 'nyc_taxi_raw_tripdata'.

Name	Database	Location	Classification	Last updated	Deprecated
accountcost	amazonhydro	s3://logos-qa-qa-poc/	Unknown	3 August 2017 5:55 PM...	
amazonathena_test	amazonathena_test	s3://amazonathena-test-sydney-iot/	json	23 March 2018 6:55 P...	
data	nyctaxi	s3://aws-bigdata-blog/artifacts/glue-data-lake/data/	csv	11 March 2018 12:32 P...	
etb_logs	default	s3://athena-examples-us-west-2/etb/plaintext	Unknown		
etb_logs	sampledb	s3://athena-examples-us-west-2/etb/plaintext	Unknown		
nyc_taxi_raw_tripdata_2015-01.csv	nyc-taxi-raw	s3://nyc-tlc/trip data/tripdata_2015-01.csv	csv	18 March 2018 4:46 P...	
nyc_taxi_raw_tripdata_2015-02.csv	nyc-taxi-raw	s3://nyc-tlc/trip data/tripdata_2015-02.csv	csv	18 March 2018 4:46 P...	
nyc_taxi_raw_tripdata_2015-03.csv	nyc-taxi-raw	s3://nyc-tlc/trip data/tripdata_2015-03.csv	csv	18 March 2018 4:46 P...	
nyc_taxi_raw_tripdata_2015-04.csv	nyc-taxi-raw	s3://nyc-tlc/trip data/tripdata_2015-04.csv	csv	18 March 2018 4:46 P...	
nyc_taxi_raw_tripdata_2015-05.csv	nyc-taxi-raw	s3://nyc-tlc/trip data/tripdata_2015-05.csv	csv	18 March 2018 4:46 P...	
nyc_taxi_raw_tripdata_2015-06.csv	nyc-taxi-raw	s3://nyc-tlc/trip data/tripdata_2015-06.csv	csv	18 March 2018 4:46 P...	
nyc_taxi_raw_tripdata_2015-07.csv	nyc-taxi-raw	s3://nyc-tlc/trip data/tripdata_2015-07.csv	csv	18 March 2018 4:46 P...	
nyc_taxi_raw_tripdata_2015-08.csv	nyc-taxi-raw	s3://nyc-tlc/trip data/tripdata_2015-08.csv	csv	18 March 2018 4:46 P...	
nyc_taxi_raw_tripdata_2015-09.csv	nyc-taxi-raw	s3://nyc-tlc/trip data/tripdata_2015-09.csv	csv	18 March 2018 4:46 P...	
nyc_taxi_raw_tripdata_2015-10.csv	nyc-taxi-raw	s3://nyc-tlc/trip data/tripdata_2015-10.csv	csv	18 March 2018 4:46 P...	
nyc_taxi_raw_tripdata_2015-11.csv	nyc-taxi-raw	s3://nyc-tlc/trip data/tripdata_2015-11.csv	csv	18 March 2018 4:46 P...	

Glue Data Catalog: Table Properties

Table properties

Data statistics

Table schema

The screenshot displays the AWS Glue console interface for a table named 'simpletweets_json'. The left sidebar contains navigation links for 'Data catalog', 'Databases', 'Tables', 'Connections', 'Crawlers', 'Classifiers', 'ETL', 'Jobs', 'Triggers', 'Dev endpoints', 'Tutorials', 'Add crawler', 'Explore table', and 'Add job'. The main content area shows the table's metadata, including its name, description, database, classification, location, connection, deprecated status, and last updated time. Below this, the 'Properties' section lists various table properties such as 'sizeKey', 'objectCount', 'UPDATED_BY_CRAWLER', 'CrawlerSchemaSerializerVersion', 'recordCount', 'averageRecordSize', 'CrawlerSchemaDeserializerVersion', 'compressionType', and 'typeOfData'. The 'Schema' section at the bottom provides a table of column names and their data types. A modal window titled 'user schema details' is open, showing a JSON-like structure of the 'user' column's schema.

Table Properties:

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

Properties:

sizeKey	456580	objectCount	1	UPDATED_BY_CRAWLER	\$3Crawler	CrawlerSchemaSerializerVersion	1.0		
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_image BOOLEAN
```

Glue Data Catalog: Version control

Compare schema versions

List of table versions

The screenshot displays the AWS Glue Data Catalog interface for a table named 'simpletweets_json'. The interface is split into two panels: 'Version 0' on the left and 'Version 1 (Current version)' on the right. Both panels show the same metadata and properties, but the schema change log at the bottom highlights a change in the 'id' column between the two versions.

Table Metadata (Version 0 and Version 1):

- 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: 466580, objectCount: 1
- UPDATED_BY_CRAWLER: S3Crawler
- Properties: CrawlerSchemaSerializerVersion: 1.0, recordCount: 1001, averageRecordSize: 466, CrawlerSchemaDeserializerVersion: 1.0, compressionType: none, typeOfData: file

Schema Change Log:

Change	Column name	Data type	Key
	entities	struct	
Changed	id	bigint	
	retweeted	boolean	
	text	string	
	user	struct	

The 'id' column is highlighted in blue in the original image, indicating a change from 'bigint' in Version 0 to 'STRING' in Version 1.

How Do I Build The ETL?

Job Authoring: Automatic Code Generation

The screenshot displays the AWS Glue 'Add Job' interface. At the top, there's a navigation bar with 'Services', 'Resource Groups', and a user profile 'ben.thurgood'. Below this, the 'Add Job' header is visible. The main content area is titled 'Map the source columns to target columns.' and includes a sub-instruction: 'Verify the mappings created by AWS Glue. Change mappings by choosing other columns with Map to target. You can Clear all mappings and Reset to default AWS Glue mappings. AWS Glue generates your script with the defined mappings.' On the right side of this instruction are buttons for 'Add column', 'Clear', and 'Reset'. The interface is divided into two main sections: 'Source' and 'Target'. Each section contains a table with columns for 'Column name', 'Data type', and 'Map to target'. The 'Source' table lists columns like vendorid, trip_pickup_datetime, trip_dropoff_datetime, passenger_count, trip_distance, pickup_longitude, pickup_latitude, ratecodeid, store_and_fwd_flag, dropoff_longitude, dropoff_latitude, payment_type, fare_amount, extra, mta_tax, tip_amount, tolls_amount, improvement_surcharge, and total_amount. The 'Target' table lists corresponding columns, including vendorid, pickup_datetime, dropoff_datetime, passenger_count, trip_distance, pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude, amount_details (which is expanded to show fare_amount, extra, mta_tax, tip_amount, tolls_amount, improvement_surcharge, and total_amount). Arrows indicate the mapping from source columns to target columns. On the left side of the interface, there's a sidebar with a tree view showing 'Job properties', 'Data source', 'Data target', 'Schema', and 'Review'.

Map the source columns to target columns.

Verify the mappings created by AWS Glue. Change mappings by choosing other columns with **Map to target**. You can **Clear** all mappings and **Reset** to default AWS Glue mappings. AWS Glue generates your script with the defined mappings.

[Add column](#) [Clear](#) [Reset](#)

Source			Target		
Column name	Data type	Map to target	Column name	Data type	
vendorid	bigint	vendorid	vendorid	long	X ↓ ↑
trip_pickup_datetime	string	pickup_datetime	pickup_datetime	timestamp	X ↓ ↑
trip_dropoff_datetime	string	dropoff_datetime	dropoff_datetime	timestamp	X ↓ ↑
passenger_count	bigint	passenger_count	passenger_count	long	X ↓ ↑
trip_distance	double	trip_distance	trip_distance	double	X ↓ ↑
pickup_longitude	double	pickup_longitude	pickup_longitude	double	X ↓ ↑
pickup_latitude	double	pickup_latitude	pickup_latitude	double	X ↓ ↑
ratecodeid	bigint	-	dropoff_longitude	double	X ↓ ↑
store_and_fwd_flag	string	-	dropoff_latitude	double	X ↓ ↑
dropoff_longitude	double	dropoff_longitude	amount_details	struct	X
dropoff_latitude	double	dropoff_latitude	fare_amount	double	X
payment_type	bigint	-	extra	double	X
fare_amount	double	amount_details.fare_amount	mta_tax	double	X
extra	double	amount_details.extra	tip_amount	double	X
mta_tax	double	amount_details.mta_tax	tolls_amount	double	X
tip_amount	double	amount_details.tip_amount	improvement_surcharge	double	X
tolls_amount	double	amount_details.tolls_amount	total_amount	double	X ↓ ↑
improvement_surcharge	double	amount_details.improvement_surcharge			
total_amount	double	total_amount			

Job Authoring: Automatic Code Generation

[illegible]

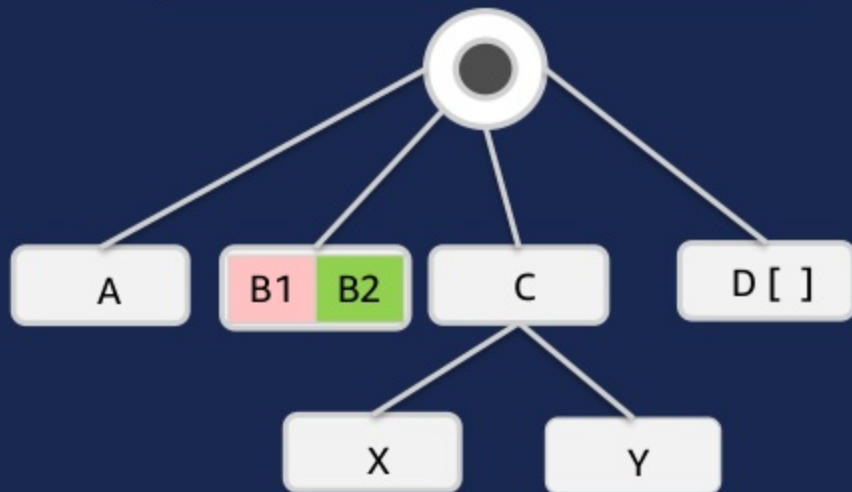
Job Authoring: ETL Code

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

```
15  sc = SparkContext()
16  glueContext = GlueContext(sc)
17  spark = glueContext.spark_session
18  job = Job(glueContext)
19  job.init(args['JOB_NAME'], args)
20  ## @type: DataSource
21  ## @args: [database = "snively-nyc-taxis-db", table_name = "snively_nyc_
22  ## @return: datasource0
23  ## @inputs: []
24  datasource0 = glueContext.create_dynamic_frame.from_catalog(database = '
```


Job Authoring: Glue Dynamic Frames

Dynamic frame schema



Like Apache 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

ResolveChoice()

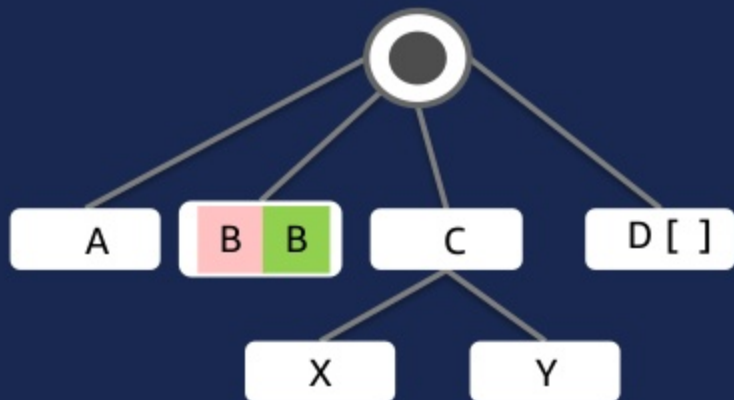


Apply Mapping()

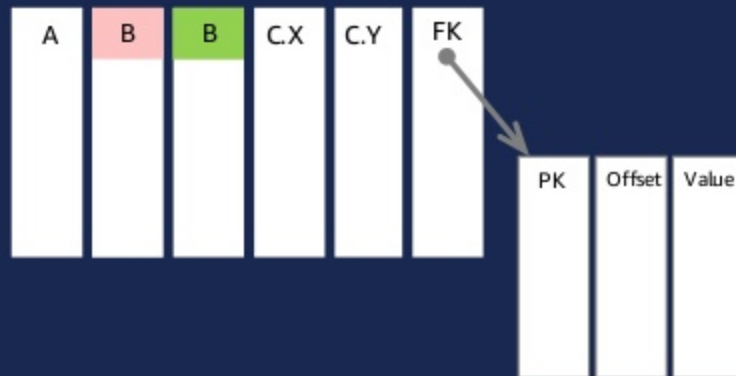


Job Authoring: Relationalize() Transform

Semi-structured schema



Relational schema

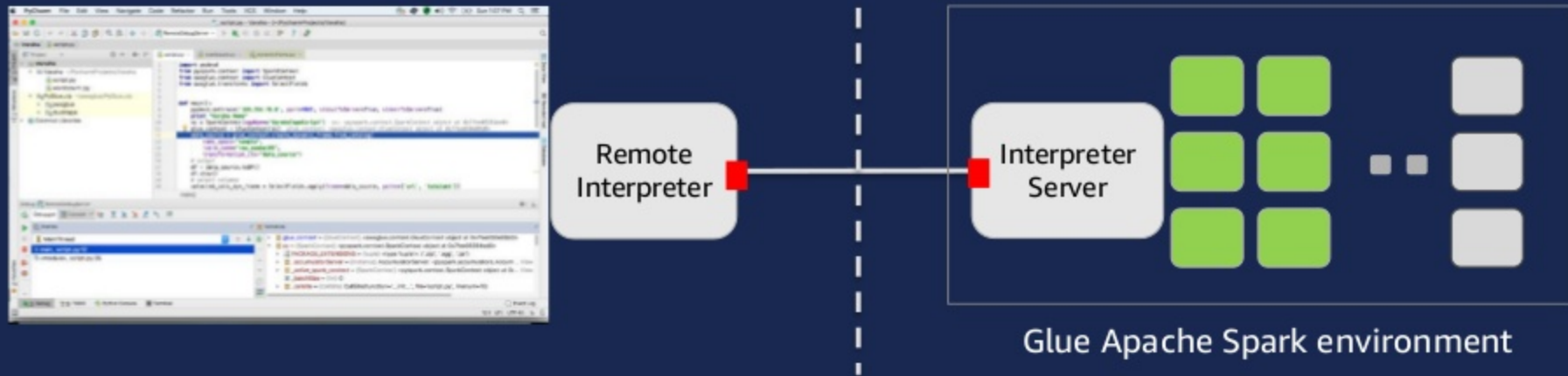


Job Authoring: ETL Code

- **Human-readable**, editable, and portable PySpark code
- **Flexible**: Glue's ETL library simplifies manipulating complex, semi-structured data
- **Customisable**: Use native PySpark, import custom libraries, and/or leverage Glue's libraries

```
42 #convert to a Spark DataFrame...
43 customDF = dropnullfields3.toDF()
44
45 #add a new column for "type"
46 customDF = customDF.withColumn("type", lit('yellow'))
47
48 # Convert back to a DynamicFrame for further processing.
49 customDynamicFrame = DynamicFrame.fromDF(customDF, glueContext, "customDF_df")
50 #-----
```

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: ETL Code

- **Human-readable**, editable, and portable PySpark code
- **Flexible**: Glue's ETL library simplifies manipulating complex, semi-structured data
- **Customisable**: Use native PySpark, import custom libraries, and/or leverage Glue's libraries
- **Collaborative**: share code snippets via GitHub, reuse code across jobs



How do I run ETL jobs?

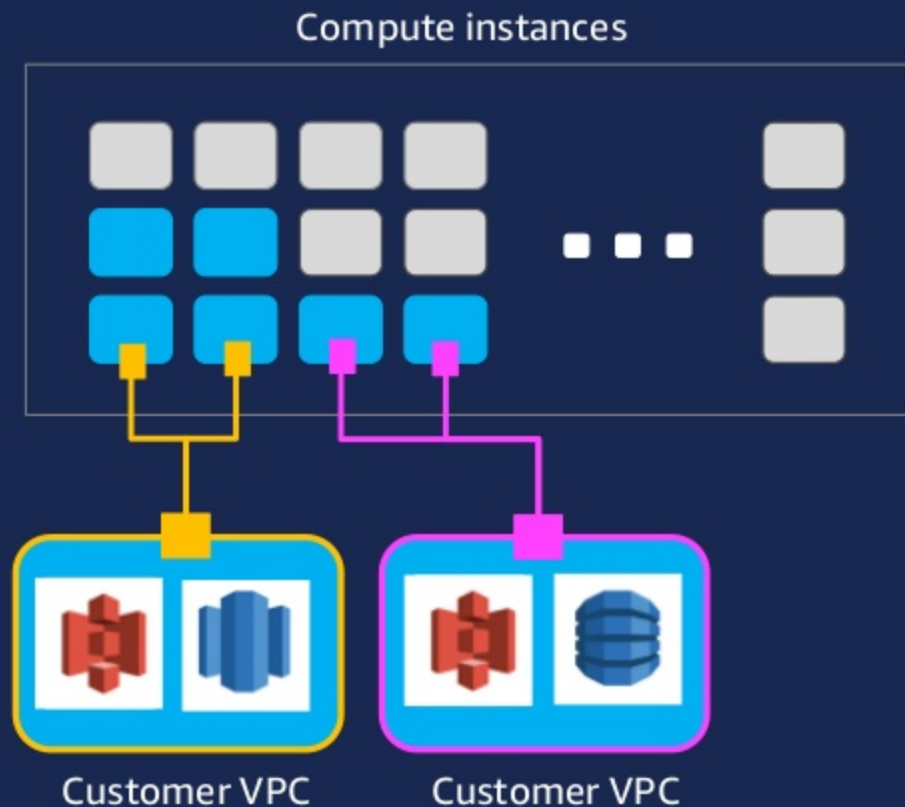
Serverless Job Execution

Auto-configure **VPC** & **role-based access**
security & isolation preserved

Customers can specify **job capacity**
using Data Processing Units (**DPU**)

Automatically scale resources

Only pay for the resources you consume
per-second billing (10-minute min)



Data Processing Units (DPUs)

1 **DPU** = 4 vCPU + 16GB RAM

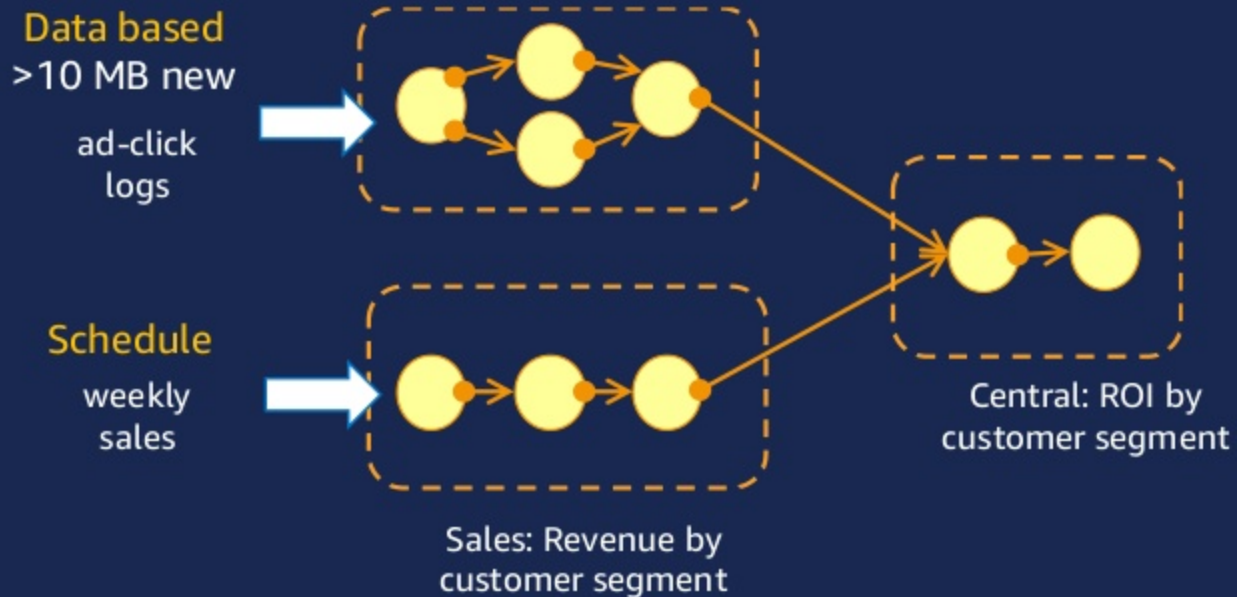
Storage:

- **Free** for the **first million** objects stored
- **\$1 per 100,000** objects stored above 1M, per month

Requests:

- **Free** for the **first million** requests per month
- **\$1 per million** requests above 1M in a month

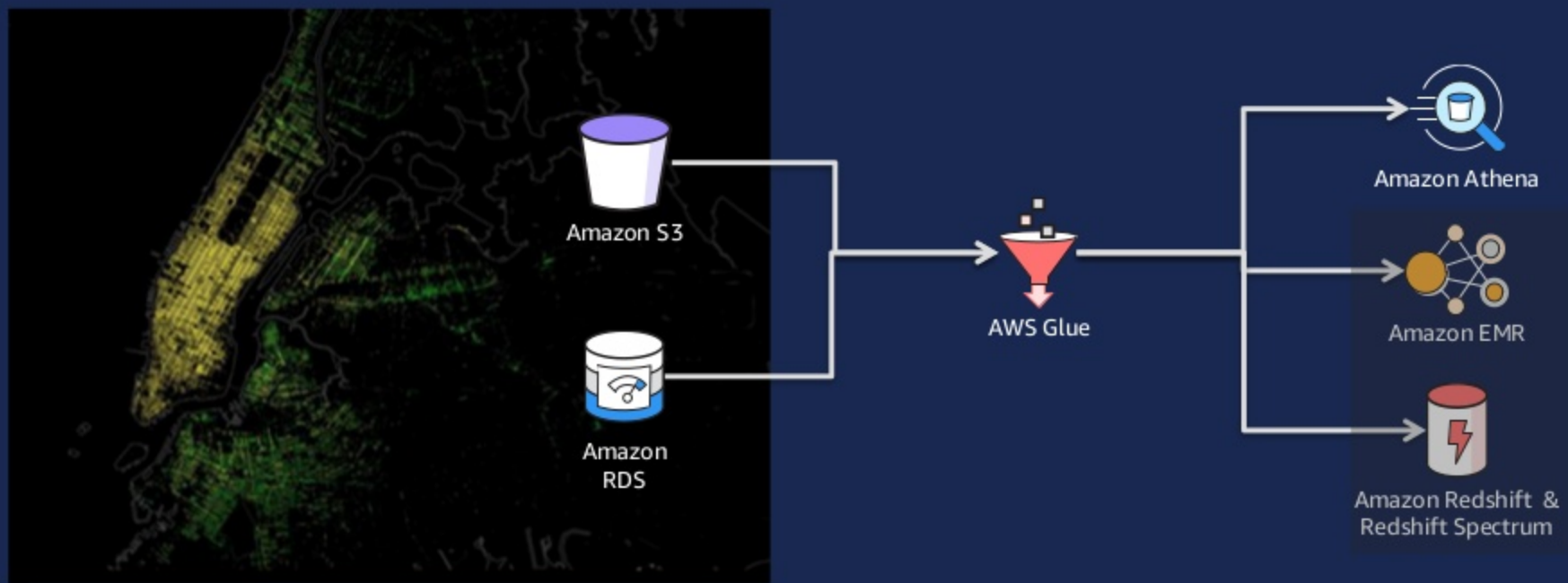
Job Composition: Example



Sounds Good In Theory...

What's It Really Like?

Demo context



How AWS Glue Helps with ETL

Automatically **discover** your data

Generate ETL code

Run your ETL jobs **serverless**



Speaker Contact

Ben Thurgood

Principal Solutions Architect

btgood@amazon.com

Thank You