



# Going Serverless - an Introduction to AWS Glue

Michael Rainey | Oaktable World 2018



© 2018 Snowflake Computing Inc. All Rights Reserved.

# **what is serverless**



**what is  
serverless ==  
function as a service**



**what is  
serverless ==  
fully managed**

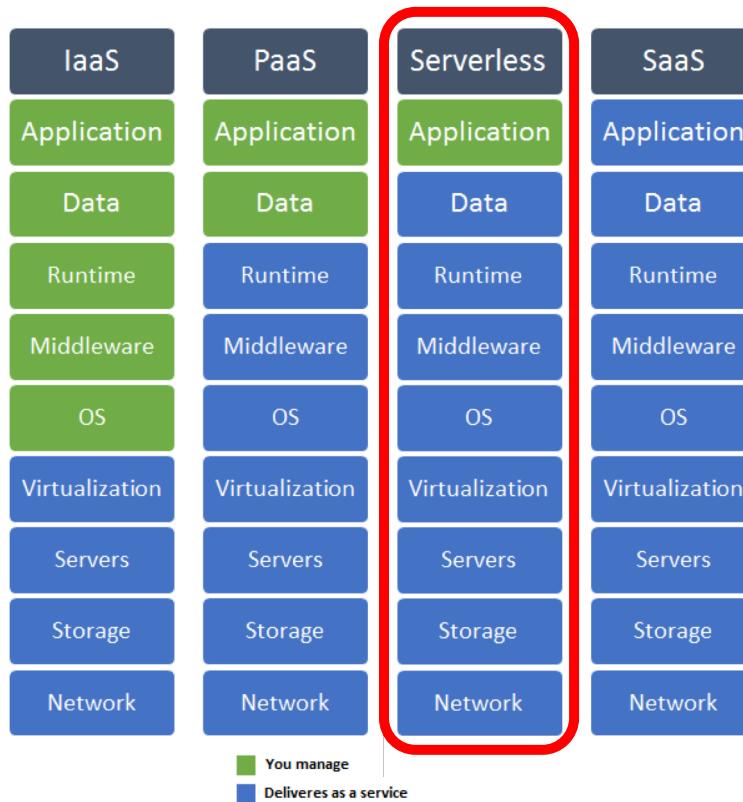


**what is  
serverless ==  
allowing developers to focus  
on ...development**

**“...any platform that provides utility while completely abstracting scaling and reliability”**

Nick Rockwell, NYTimes CTO

# Where serverless fits



Source: <http://qunnarpeipman.com/2018/03/serverless-architecture/>



# In the beginning...

- AWS Lambda in 2014
  - Function as a Service (FaaS)
- Others
  - Google Cloud Functions, Microsoft Azure Functions, The Fn Project (Oracle)
  - Amazon Athena, Google BigQuery, Snowflake, etc.
- Why serverless?
  - Pay as you go
  - Zero administration - “fully managed”
    - You’re not provisioning infrastructure, but you’re still in charge of the compute
  - Scale automatically
  - Ship code faster
- Serverless use cases
  - Websites
  - Post processing updates
  - Data ingestion pipelines
  - Backups/log analysis/etc
  - Capture customer social media data to store in your CRM



# Serverless: challenges / limitations

- Monitoring and debugging
- Integration testing can be difficult
- Maintaining state of application across multiple functions is tricky
- Startup latency
- Built for small, short processes  
(FaaS typically <5 min)
- Vendor lock-in
  - Access to multi-cloud data can be difficult



# AWS Glue

Serverless ETL



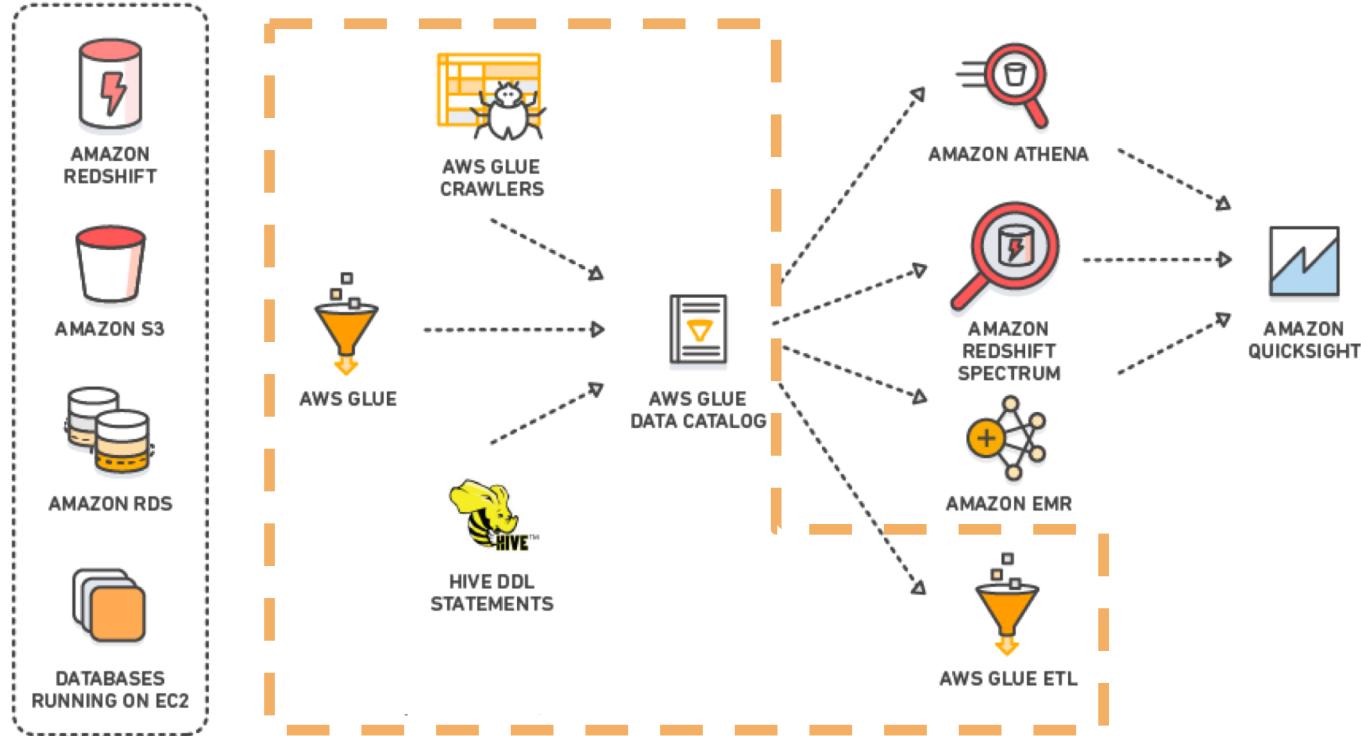
# AWS Glue

**“...a fully managed extract, transform, and load (ETL) service...”**

- Components:
  - Data Catalog
  - Crawlers
  - ETL jobs/scripts
  - Job scheduler
- Useful for...
  - ...running serverless queries against S3 buckets and relational data
  - ...creating event-driven ETL pipelines
  - ...automatically discovering and cataloging your enterprise data assets



# AWS Glue architecture



# Data catalog

**“...a central repository to store structural and operational metadata for all your data assets.”**

- Apache Hive Metastore compatible
  - Can act as metadata repository for EMR, Athena, Redshift Spectrum
- Tables
  - Organized in databases
  - Structure and metadata
  - Add using the wizard or run a crawler
- Data sources:
  - Amazon S3, Oracle, PostgreSQL, Amazon Redshift, Amazon Aurora, MySQL, MariaDB, MS SQL Server, custom JDBC

Name	Database	Location	Classification
10_45_0_23	int12102_sh	s3://gluent.backup/user/gluent/backup/10.45.0.23/	orc
10_45_1_88	int12102_sh	s3://gluent.backup/user/gluent/backup/10.45.1.88/	parquet
channels	int12102_sh	s3://gluent.backup/user/gluent/backup/sh.db/channels/	orc
channels_4179ce4c147...	int12102_sh	s3://gluent.backup/user/gluent/backup/sh_test.db/channels/	parquet
countries	int12102_sh	s3://gluent.backup/user/gluent/backup/internal_development/countries/	parquet
countries_8541194d2ae...	int12102_sh	s3://gluent.backup/user/gluent/backup/sh_test.db/countries/	parquet
customers	int12102_sh	s3://gluent.backup/user/gluent/backup/internal_development/customers/	parquet



# Crawlers

**“...connects to one or more data stores, determines the data structures, and writes tables into the Data Catalog.”**

- Include/exclude specific objects
- Detect Hive style partitions
- Detect schema changes as structures evolve
- Classifiers - automatic schema inference
  - Detects format of the data to generate the correct schema
  - Built-in and custom (written in Grok, JSON, or XML)
  - Can invoke list of classifiers

Cast a `num` field to an `int` data type

`%{NUMBER:num:int}`



# Jobs

**“A job is the business logic that performs the extract, transform, and load (ETL) work in AWS Glue.”**

- PySpark or Scala scripts, generated by AWS Glue
  - Use Glue generated scripts or provide your own
- Built-in transforms to process data
  - The data structure used, called a DynamicFrame, is an extension to an Apache Spark SQL DataFrame
- Visual dataflow can be generated
- Development endpoint available to write scripts in a notebook or IDE

ETL



# Jobs - built-in transforms

- **ApplyMapping**
  - Maps source and target columns
- **Filter**
  - Load new DynamicFrame based on filtered records
- **Join**
  - Joins two DynamicFrames
- **Map**
  - Applies a function to the records in a DynamicFrame
- **SelectFields**
  - Output selected fields to new DynamicFrame
- **Spigot**
  - Sampling of data written to S3
- **SplitFields**
  - Split fields into two new DynamicFrames
- **SplitRows**
  - Split rows into two new DynamicFrames based on a predicate

```
# Join the frames to create history
l_history = Join.apply(persons, memberships, 'id', 'person_id')
```



# Running a job in AWS Glue

**“With AWS Glue, you only pay for the time your ETL job takes to run.”**

- Fire off the ETL using the job scheduler, events, or manually invoke
- Data processing units (DPUs) used to calculate processing capacity & cost
  - A single DPU = 4 vCPUs compute and 16 GB of memory
  - Can be a custom set value from 2 - 100
  - Billed \$0.44 per DPU-Hour in increments of 1 second
  - 10-minute minimum duration for each job

## **ETL job example:**

Consider an ETL job that runs for 10 minutes and consumes 6 DPUs.

Since your job ran for 1/6th of an hour and consumed 6 DPUs, you will be billed  
**6 DPUs \* 1/6 hour** at \$0.44 per DPU-Hour or a total of **\$0.44**.



# Creating a job via the wizard

### Job properties

**Name**  
load\_sales\_target

**IAM role** AWSGlueServiceRoleDefault  
Ensure this role has permission to your Amazon S3 sources, targets, temporary directory, scripts, and any libraries used by the job. [Create IAM role](#).

**This job runs**  
 A proposed script generated by AWS Glue  
 An existing script that you provide  
 A new script to be authored by you

**ETL language**  
 Python     Scala

**Script file name**  
load\_sales\_target

**S3 path where the script is stored**  
s3://aws-glue-scripts-030461149623-us-east-1/mrainey

**Temporary directory**  
s3://aws-glue-temporary-030461149623-us-east-1/mrainey

**Advanced properties**

**Script libraries and job parameters (optional)**

**Next**

### Advanced properties

**Job bookmark** Disable

**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

**Concurrent DPU per job run** 10

**Max concurrency** 1

**Job timeout (min)** 2880

**Number of retries** 0

**Job parameters**

Key	Value
Type key...	Type value...



# Creating a job via the wizard

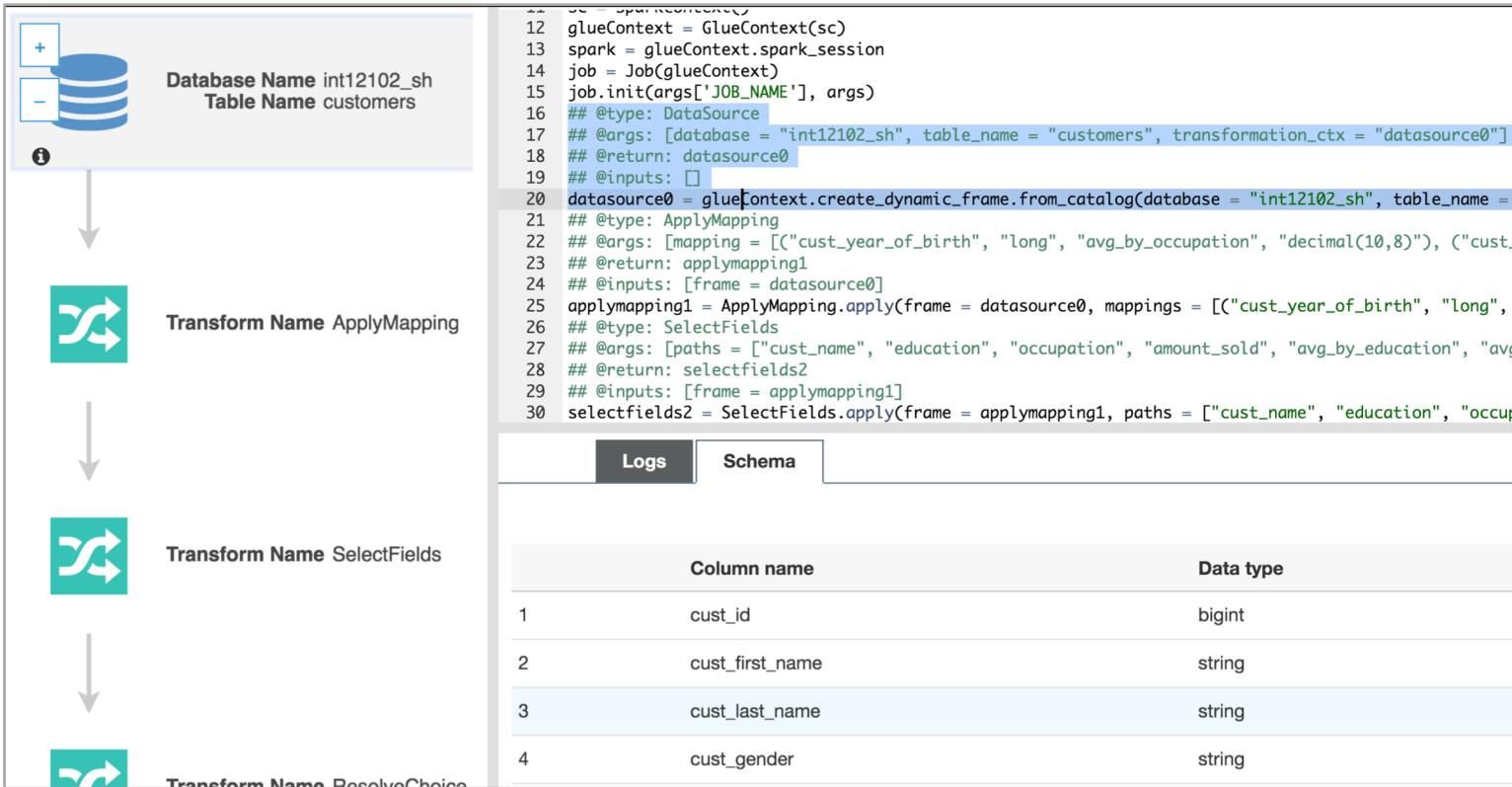
Map the source columns to target columns.

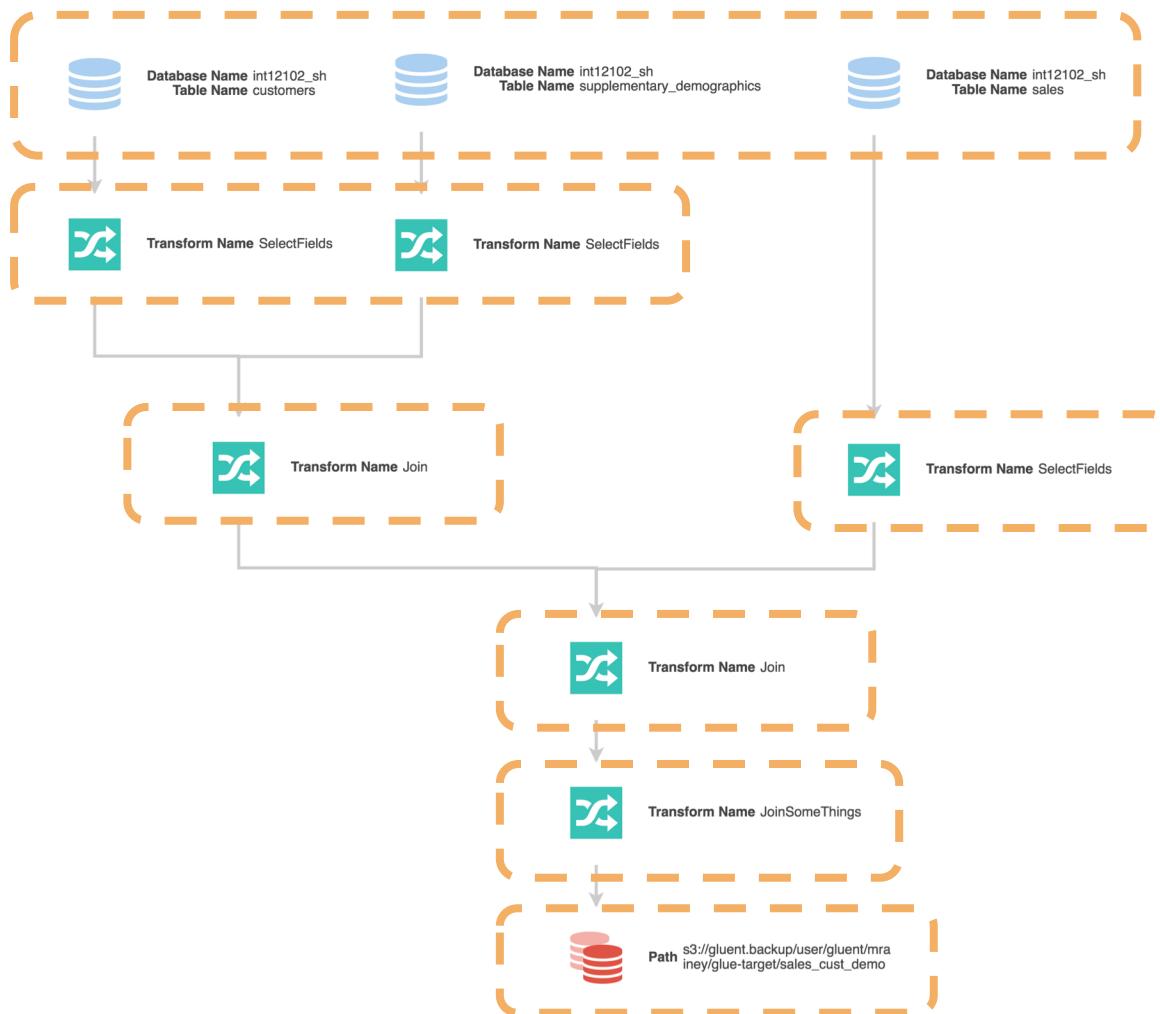
Verify the mappings created by AWS Glue. Change mappings by choosing other columns with **Map to target**. You can **Clear** all mappings and **Reset** to default AWS Glue mappings. AWS Glue generates your script with the defined mappings.

Source			Target	Add column	Clear	Reset
Column name	Data type	Map to target	Column name	Data type		
cust_id	bigint	-	avg_by_occupation	decimal(1...		
cust_first_narr	string	-	occupation	string		
cust_last_nam	string	-	amount_sold	decimal(6,4)		
cust_gender	string	-	avg_by_education	decimal(1...		
cust_year_of_l	bigint	avg_by_occupation	cust_name	string		
cust_marital_s	string	-	education	string		
cust_street_ac	string	-	avg_total	decimal(1...		
cust_postal_c	string	-				
cust_city	string	occupation				
cust_city_id	bigint	-				
cust_state_prc	string	-				
cust_state_prc	bigint	-				
country_id	bigint	amount_sold				

```
graph LR; S1[avg_by_occupation] --> T1[avg_by_occupation]; S2[occupation] --> T2[occupation]; S3[amount_sold] --> T3[amount_sold]; S4[avg_by_education] --> T4[avg_by_education]; S5[customer_name] --> T5[customer_name]; S6[education] --> T6[education]; S7[avg_total] --> T7[avg_total]; S8[cust_id] --> T8["-"]; S9[cust_first_narr] --> T9["-"]; S10[cust_last_nam] --> T10["-"]; S11[cust_gender] --> T11["-"]; S12[cust_year_of_l] --> T12["-"]; S13[cust_marital_s] --> T13["-"]; S14[cust_street_ac] --> T14["-"]; S15[cust_postal_c] --> T15["-"]; S16[cust_city] --> T16["-"]; S17[cust_city_id] --> T17["-"]; S18[cust_state_prc] --> T18["-"]; S19[cust_state_prc] --> T19["-"]; S20[country_id] --> T20["-"];
```

# Creating a job via the wizard





# Job script editing

```
cust_demo_sales_t = cust_demo_sales.toDF()
```

```
cust_demo_sales_sql = spark.sql("SELECT concat(cust_last_name, ' ',  
    cust_first_name) cust_name, education, occupation, amount_sold,  
    avg(amount_sold) OVER (partition by education) avg_by_education,  
    avg(amount_sold) OVER (partition by occupation) avg_by_occupation,  
    avg(amount_sold) OVER () avg_total  
FROM cust_demo_sales_tbl  
WHERE cast(cust_city as varchar) = 'Coventry'")
```

```
cust_demo_sales_final =  
    DynamicFr.fromDF(cust_demo_sales_sql,  
    glueContext, "cust_demo_sales_final")
```



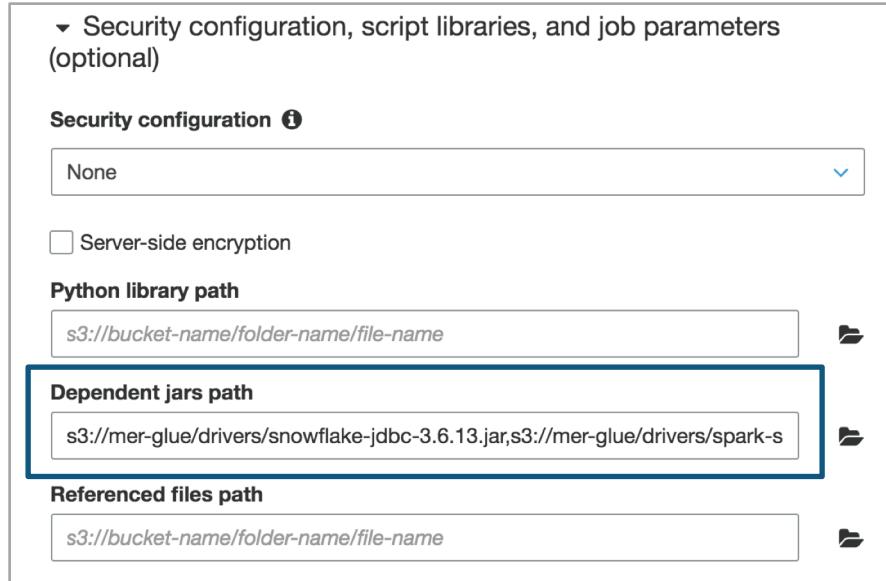
# AWS Glue in action

## Connecting to Snowflake



# Setup AWS Glue and Snowflake

- Drop the [Snowflake Spark Connector](#) and [Snowflake JDBC Driver](#) JAR files somewhere Glue can access them – like an S3 bucket
- Add path to “dependent jars” in job properties



# Setup

- Add parameters for connection to Snowflake

Job parameters	
Key	Value
--SCHEMA	PUBLIC
--URL	sfcsupport.snowflakecompu
--WAREHOUSE	MRAINEY_WH
--ACCOUNT	sfcsupport
--USERNAME	MRAINEY
--DB	MRAINEY_TEST
--PASSWORD	myCoolPassword



# Connecting to Snowflake

```
1 import sys
2 from awsglue.transforms import *
3 from awsglue.utils import getResolvedOptions
4 from pyspark.context import SparkContext
5 from awsglue.context import GlueContext
6 from awsglue.job import Job
7 from py4j.java_gateway import java_import
8 SNOWFLAKE_SOURCE_NAME = "net.snowflake.spark.snowflake"
9
10 ## @params: [JOB_NAME, URL, ACCOUNT, WAREHOUSE, DB, SCHEMA, USERNAME, PASSWORD]
11 args = getResolvedOptions(sys.argv, ['JOB_NAME', 'URL', 'ACCOUNT', 'WAREHOUSE', 'DB', 'SCHEMA', 'USERNAME', 'PASSWORD'])
12 sc = SparkContext()
13 glueContext = GlueContext(sc)
14 spark = glueContext.spark_session
15 job = Job(glueContext)
16 job.init(args['JOB_NAME'], args)
17 java_import(spark._jvm, "net.snowflake.spark.snowflake")
18
19 ## enable query pushdown to Snowflake
20 spark._jvm.net.snowflake.spark.snowflake.SnowflakeConnectorUtils.enablePushdownSession( \
21     spark._jvm.org.apache.spark.sql.SparkSession.builder().getOrCreate())
22
23 sfOptions = {
24     "sfURL" : args['URL'],
25     "sfAccount" : args['ACCOUNT'],
26     "sfUser" : args['USERNAME'],
27     "sfPassword" : args['PASSWORD'],
28     "sfDatabase" : args['DB'],
29     "sfSchema" : args['SCHEMA'],
30     "sfWarehouse" : args['WAREHOUSE'],
31 }
```



# Running a Glue job

```
34 ## Read from S3 file - employee.csv - into a Spark Data Frame
35 dfEmployee = glueContext.create_dynamic_frame \
36     .from_catalog(database = "oggdata", table_name = "employee").toDF()
37
38 ## Read from S3 file -蒙特羅斯公司員工銷售額
39 dfMontrose = glueContext.create_dynamic_frame \
40     .from_catalog(database = "oggdata", table_name = "montrose_sales").toDF()
41
42 ## Aggregate sales by employee
43 dfGrouped = dfMontrose.groupBy("EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "MONTH") \
44     .agg(F.sum("AMOUNT").alias("sum(AMOUNT)"))
45
46 ## Join employees and sales
47 dfEmplSales = dfEmployee.join(dfGrouped, "EMPLOYEE_ID")
48
49 ## Select columns
50 dfEmplSales = dfEmplSales.select("EMPLOYEE_ID", "FIRST_NAME", "LAST_NAME", "MONTH", "sum(AMOUNT)")
51
52 ## Write to Snowflake
53 dfEmplSales.write \
54     .options(**sfOptions) \
55     .option("dbtable", "MRAINEY_TEST.PUBLIC.EMPLOYEE_ANNUAL_SALES").mode("overwrite").save()
56 job.commit()
57
```



# Why use AWS Glue with Snowflake?

- Event Driven ETL Pipelines
  - Process and land data in S3
  - Trigger script to move data to Snowflake (COPY)
- ETL – S3 as a source, Snowflake as target
  - Process using Spark within Glue
- ETL – Snowflake as a source and target
  - Process using Spark within Glue or pushdown processing to Snowflake
- ETL – Multiple sources
  - Read data from Snowflake and join with Data from S3 (and/or other sources)
  - Write back to Snowflake (using pushdown)
- Understand data assets – Data Catalog

serverless  
fully managed  
**ETL**



**Jeff Hollan**

@jeffhollan



“Serverless computation is going to fundamentally not only change the economics of what is back-end computing, but it’s going to be the core of the future of distributed computing.” —

2:47 PM - Dec 23, 2017



66



41 people are talking



**Satya Nadella**

@satyanadella

CEO of Microsoft Corporation

Tweets

784

Following

213

Followers

1.66M





THANK YOU



# Going Serverless - an Introduction to AWS Glue



# Reach out for more info!

- Email: michael.rainey@snowflake.com
- Twitter: [@mRainey](https://twitter.com/mRainey)

## AWS Glue resources

- Docs:
  - <http://aws.amazon.com/documentation/glue>
- Forums:
  - <https://forums.aws.amazon.com/forum.jspa?forumID=262>
- Presentations:
  - [Serverless ETL with AWS Glue](#)
  - [How to Build a Data Lake with AWS Glue Data Catalog](#)
- Blogs
  - [How To Use AWS Glue With Snowflake](#)

