Advanced Serverless Pipeline Training

Data Team, AllCloud

January, 2020

Table Of Content

1. Overview **3**

2. Use Case **3**

3. Source Data **4**

hamustaTaxi_tripdata.csv 4

dayOfWeek_lookup_table.csv 5

Location_lookup_table.csv 5

4. Destination Data **5**

rideCount_Queens_per_dayOfWeek 5

5. System Architecture **6**

6. Start Building **6**

6.2 transformation processes 7

6.2.1 - Transformation using Athena (SQL) 7

6.2.2 - Transformation using Glue (PySpark) 8

1. Overview

In this workshop, you are going to implement a **serverless data pipeline** implemented using AWS services such as Glue, Athena, SQS, Lambda, SNS, and S3.

The goal of this workshop is to guide you through building a serverless decoupled solution that leverages each service's strengths. The solution you are about to implement is just one way to accomplish this task and is derived from the specific requirements of the client.

2. Use Case

Mr. Hamusta, the CTO of "Hamusta taxi" (a taxi company in NYC), asked allCloud's Data-Team to help to build a new data pipeline for the company. The company started using AWS services a few years ago, but still has some on-premise servers. Today all their data processes are running on self-managed Hadoop clusters on EC2 servers. The maintenance of such infrastructure is high but worked well until some of the ETL processes can't handle the amount of the data and in the last month start to frequently fail.

Last week you have met the client and gather the next requirements for the new data pipeline:

- The solution needs to be fully serverless. Mr. Hamusta is sick of spending company resources maintaining these processes on a weekly basis.
- The solution needs to be scalable, the company predicts a 10% per year increment of data.
- The data pipeline will be a nightly batch processing as it's working today.
- The data source will arrive daily to an S3 bucket as one zip file per table, inside each zip file there will be only one CSV file (not to overcomplicate the use case).
- In this POC project, Mr. Hamusta decided to focus only on the most complicated ETL process, "kuba-burgol-c2", and to address the others after finishing this POC successfully.
- The arrival of any zip file into the bucket will kick-off an ingestion pipeline followed by process pipeline and at result will create three new tables (ingestion section) for the sake of analysts/data engineers in the company.

3. Source Data

Since this POC will focus only on one ETL process, "cuba-burgol-c2", the source data that will be explained in this section will include the tables for this process only.

For a downloadable example, zip file click here.

For "cuba-burgol-c2" ETL process there are three zip files that contain three CSV files as follows:

1. hamustaTaxi_tripdata.csv
2. dayOfWeek_lookup_table.csv
3. Location_lookup_table.csv

What is the "cuba-burgol-c2" ETL process?

This process uses the above three table to create a new table that is the count of rides that picked up passengers from 'Queens' per day of week name

hamustaTaxi_tripdata.csv

Notice: Every row in the table is one taxi trip ride

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. 1= Creative Mobile Technologies, LLC; 2= VeriFone Inc.
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
PULocationID	TLC Taxi Zone in which the taximeter was engaged
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged
RateCodeID	The final rate code in effect at the end of the trip. 1= Standard rate 2=JFK 3=Newark 4=Nassau or Westchester 5=Negotiated fare 6=Group ride
Store_and_fwd_flag	This flag indicates whether the trip record was held in-vehicle memory before sending it to the vendor, aka "store and forward," because the vehicle did not have a connection to the server. Y= store and forward trip N= not a store and forward trip
Payment_type	A numeric code signifying how the passenger paid for the trip. 1= Credit card 2= Cash 3= No charge 4= Dispute 5= Unknown 6= Voided trip
Fare_amount	The time-and-distance fare calculated by the meter.

Extra	Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.
MTA_tax	$0.50 MTA tax that is automatically triggered based on the metered rate in use.
Improvement_surcharge	$0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.
Tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
Tolls_amount	The total amount of all tolls paid on the trip.
Total_amount	The total amount charged to passengers. It does not include cash tips.

dayOfWeek_lookup_table.csv

Field Name	Description
day_of_week_id	The id of the table ranges from 1 (Monday) to 7 (Sunday).
day_of_week	The day name

Location_lookup_table.csv

Field Name	Description
locationid	The id of the table, it's a foreign key for "hamustaTaxi_tripdata.csv" table for "PULocationID" and "DOLocationID" columns
borough	New York City encompasses five county-level administrative divisions called boroughs: The Bronx, Brooklyn, Manhattan, Queens, and Staten Island.
zone	The area inside each borough
service_zone	Not relevant to our use case

4. Destination Data

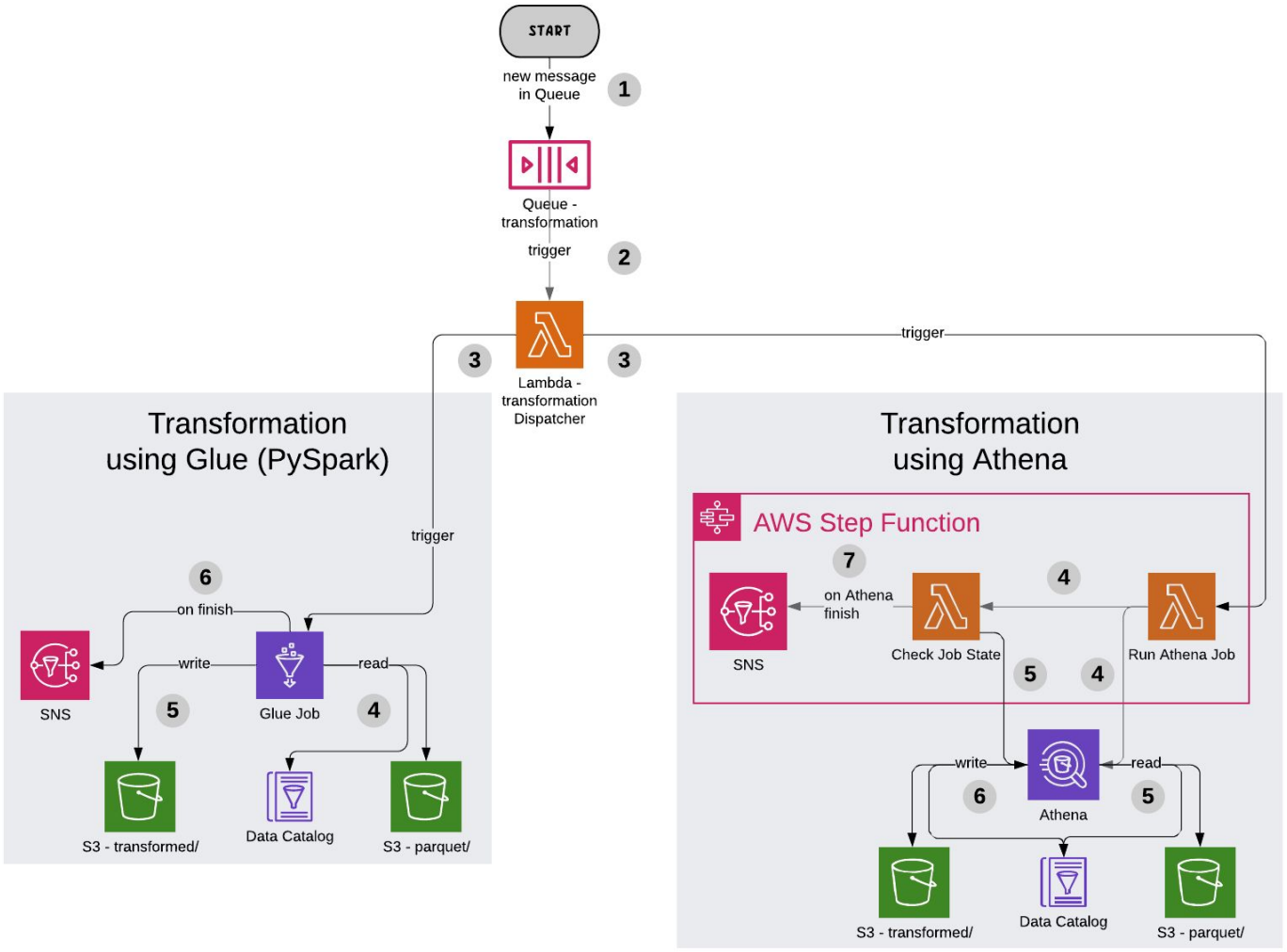
For the "cuba-burgol-c2" ETL process you will need to create the next destination table by performing transformations on source tables above.

This table represents a count of rides that picked up passengers from **'Queens'** per day of week name

rideCount_Queens_per_dayOfWeek

Field Name	Description
day_of_week	day name
ride_count	count of rides

5. System Architecture



Notice: in the diagram, there are two implementations of processes that will transform the data, the one is by using PySpark on Glue and the other is by using Step Function with Athena. **You will need to build both of the options and in the** "Lambda - transformation Dispatcher" **you will send a parameter (like boolean) that will decide what ETL process to use.**

6. Start Building

After understanding the use case, the data tables, and the ETL process that needs to be built, we arrive at the hands-on stage. This stage is divided into two main parts: ingesting the data and preparing it to the transformations and the transformations that will output the destination tables.

For this workshop using the same region for all your provisioned resources.

6.2 transformation processes

Prerequisites

- The data that will be transformed exists in Glue Data Catalog in the "parquet" database.
- The table in the "parquet" database exists in the "training_advanced_pipeline" S3 bucket in the "parquet" folder.
- Existed SQS queue "Transformation Queue" that was created in the "ingestion Phase"

6.2.1 - Transformation using Athena (SQL)

1. "Run Athena Job" Lambda

   Create a new **Lambda** (Python 3.8) with the IAM role with permissions to S3, Athena, and CloudWatch.

   Notice: increase the lambda's default timeout value from 3 sec.

   you will need to build this code by yourself by using boto3 inside the Lambda function to send a query to Athena service. (Python solution code here)

   recommendation: build and debug the SQL query in Athena until you reach the point you manage to create the destination table, and just then build the Lambda function.

2. "Check Job State" Lambda

   Create a new **Lambda** (Python 3.8) with the IAM role with permissions to S3, Athena, SNS, and CloudWatch.

   Notice: increase the lambda's default timeout value from 3 sec.

   you will need to build this code by yourself by using boto3 inside the Lambda function to check if the Athena service finishes executing the query using the "QueryExecutionId" from the previous Lambda, on finish publish a message to SNS topic. (Python solution code here)

3. SNS topic

   Create a new **SNS** topic to inform you when the ETL process was finished (successfully or not) subscribe to the topic via work email.

4. Step Function

   Create a new **standard Step Function** and use the next code for the definition.

   You will need to change the "`Resource`" field value for the Lambda functions and the SNS topic that you created.

5. "transformation Dispatcher" Lambda

Create a new **Lambda** (Python 3.8) with the IAM role with permissions to trigger Step Function, Glue Job, and CloudWatch.

Notice: increase the lambda's default timeout value from 3 sec.

you will need to build this code by yourself by using boto3 inside the Lambda function to invoke one of the ETL processes. (Python solution code here)

6. Modify "Transformation Queue"

   Modify existed **Queue** (SQS service) and Configure Incoming Messages to Trigger "transformation Dispatcher" Lambda Function you create the previous step.

6.2.2 - Transformation using Glue (PySpark)

1. SNS topic

   Create a new **SNS** topic to inform you when the ETL process was finished (successfully or not) subscribe to the topic via work email.

   recommendation: you can use the same topic you created in "6.2.1 - Transformation using Athena (SQL)" section.

2. Glue Job

   Create a new **Glue Job** (Spark and Python) with the IAM role with permissions to S3, Glue, SNS, and CloudWatch.

   you will need to build this code by yourself by using PySpark to perform the transformations, write the result to S3 and publish a message to SNS topic. (PySpark solution code here)

   recommendation:

   - build and debug the PySpark code using Glue dev endpoints notebooks until you reach the point you manage to perform the transformations, write the result to S3 and publish a message to SNS topic, and just then build the Glue job.
   - Use Spark documentation