Orchestration of SAS Data Integration Processes on AWS

ABSTRACT

This paper talks about the orchestration of SAS® data integration processes based on the arrival of the SAS data integration input files in Amazon Web Services (AWS) S3. Our client runs a daily process where they generate credit statements for their customers. Each customer receives their statement once in a month. Every day, around 200K customers are processed, eventually reaching out to their entire customer base of roughly 6 million in a month. The process starts in their on-premises datacenter, followed by the APR calculations in SAS data integration in AWS, and finally culminates with the generation of the statements in the on-premises datacenter. The entire architecture is built using microservices; small independent and highly decoupled components such as AWS Lambda, Simple Notification Service (SNS), Amazon Simple Storage Service (Amazon S3), and Amazon Elastic Compute Cloud (EC2). This makes it easier to troubleshoot any issues in the data pipeline. The choice of using a Lambda function for the orchestration adds a certain complexity to the process. However, it also provides the most stability, security, flexibility, and reliability for an enterprise architecture. There are simpler alternatives like S3Fs and CloudWatch SSM, but they do not fit well for an enterprise architecture.

INTRODUCTION

Orchestration describes automated arrangement, coordination, and management of complex computer systems, and services.

This paper talks about the Orchestration of SAS Data Integration Processes based on the arrival of the SAS DI input files in AWS S3.

Technical Stack:

SAS Version: 9.4 M3

SAS Product: SAS Data Integration Server

SAS DI Scheduler: Platform Process Manager

AWS S3: Amazon S3 is a cloud computing web service offered by Amazon Web Services. Amazon S3 provides object storage through web services interfaces

AWS Lambda: AWS Lambda is an event-driven, serverless computing platform provided by Amazon as a part of the Amazon Web Services. It is a computing service that runs code in response to events and automatically manages the computing resources required by that code.

ARCHITECTURE

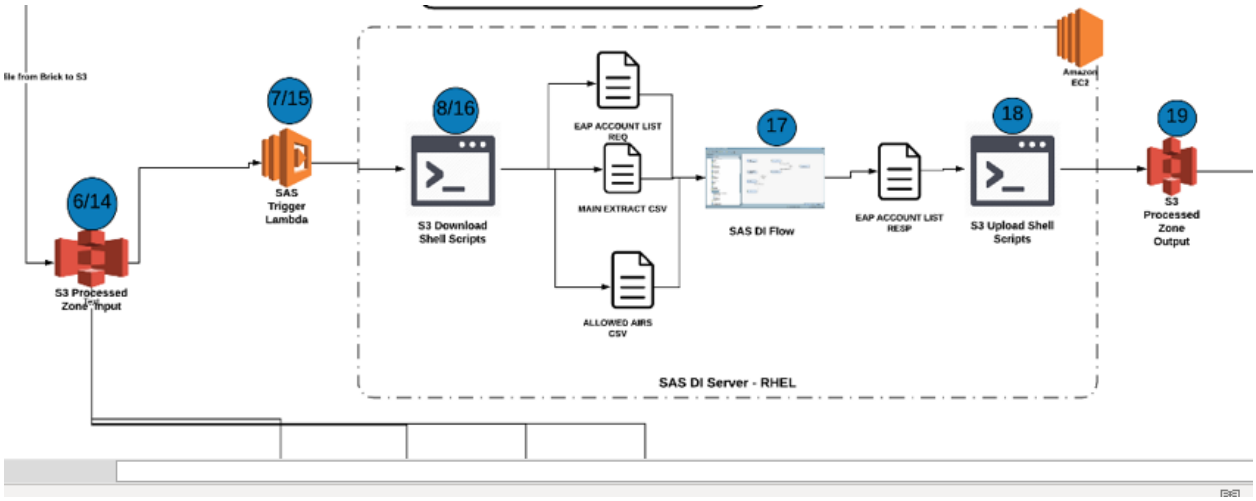

Figure 1: SAS Trigger Lambda

The architecture above is part of a more extensive data orchestration process. We have narrowed it down to the SAS processing. The process works as follows:

1) File Arrives in S3.

2) S3 Events are configured on the S3 bucket to trigger the "SAS Trigger Lambda"

3) The SAS Trigger Lambda is a Python function which parses the event and triggers a "S3 Download" Shell Script on the SAS DI Server.

4) The Shell Script downloads the file to from the S3 bucket to the SAS DI compute Server. The Shell Scripts also creates a marker file to denote the arrival of the file in the SAS DI Server

5) The SAS DI Job is scheduled using a file arrival event. The arrival of the file kicks of the SAS DI processing.

ARCHITECTURE RATIONALE

The entire architecture is built using Microservices; small independent and highly decoupled components such as AWS Lambda, SNS, S3, EC2. This makes it easier to troubleshoot any issues in the pipeline. The choice of using a Lambda function for the orchestration adds a certain complexity to the process. There are other simpler options which we have described at the end of the document. However, this option provides the most stability, security, flexibility and reliability for an enterprise architecture.

SAS TRIGGER LAMBDA MODULES

The following modules were developed to create the SAS Trigger functionality:

1) SAS Trigger Lambda Cloud Formation Template
2) SAS Trigger Lambda Python Code
3) S3 Bucket Events Cloud Formation Template

SAS TRIGGER LAMBDA CLOUD FORMATION TEMPLATE

This Cloud formation template (CFT) describes the various attributes for a lambda function such as VPC, security groups, runtime, etc. The json code for the CFT can be found here:

https://github.com/corecompete/sas-trigger-lambda/blob/master/hubs_data_publish_sasditrigger_lambda.json

SAS TRIGGER LAMBDA PYTHON CODE

The python code contains the triggering logic. It parses the S3 event and kicks off the shell script on the SAS Compute Server. The python code can be found here:

https://github.com/corecompete/sas-trigger-lambda/blob/master/lambda_function.py

S3 BUCKET EVENTS CLOUD FORMATION TEMPLATE

This CFT contains the attributes to create the S3 bucket and the associated event configuration for the bucket.

https://github.com/corecompete/sas-trigger-lambda/blob/master/hubs_data_processed_s3bucket.json

CONCLUSION

The orchestration approach described here uses lambda functions to use S3, a low cost and highly available storage solution. It also allowed us to keep the powerful scheduling and ETL workflow design in SAS.

This solution was developed in 9.4 M3 hence it required us to create a shell script which downloaded the file from S3 to the SAS Compute Server. Starting 9.4 M5, SAS provides a procedure called PROC S3 which allows us to download the file from S3 from a SAS session directly.

We could also containerize the SAS Compute Server so that we don't have a compute server waiting for DI schedules. The Compute server can be launched in a container when the file arrives.

There are other methods to achieve this kind of orchestration:

S3FS: S3FS is an open source Linux compatible filesystem implementation based on FUSE. It allows you the mount S3 buckets on Linux servers. This approach will make the files in S3 directly available on the SAS DI Server, effectively removing the Lambda Function and the File Download Shell Scripts. However, there are consistency and stability issues with the driver which makes it unreliable for an operational DI process.

Systems Manager (SSM) Run Command: The file arrival event in S3 can be also used to trigger an SSM Run Command. SSM RUN Commands is a service in AWS which allows running any command on EC2 servers, which means we can directly run commands on the SAS DI box. This approach will effectively remove the requirement of a Lambda function. However, S3 events offer less flexibility in the S3 folders path names, such as wildcard characters and special characters and were not suitable for our use case.

Lambda File Copy: In our architecture, we are using Lambda to trigger a S3 file downloader shell script. We could have coded the copying of the file from S3 to SAS DI server in the Lambda function itself. However, Lambda comes with a 5 min execution limit. The approach would have worked for smaller files, but for larger files, there is a risk of exceeding the execution limit.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author: Rohit Shetty

info@corecompete.com

https://corecompete.com/