SAS 1679-2018

Streaming ETL of High-Velocity Big Data Using SAS® Event Stream Processing and SAS® Viya®

Joydeep Bhattacharya and Manish Jhunjhunwala, SAS Institute Inc.

ABSTRACT

A typical ETL happens once in a day. How can you handle those use cases when ETL needs to happen in minutes? This paper presents a design using SAS® Viya® and SAS® Event Stream Processing to fetch large volumes of data from a data source, process that data in the flow itself, and load it into Hadoop files (SASHDAT format) at a frequency that can be as high as every 15 minutes. The data source used here is Amazon Elastic Compute Cloud (Amazon EC2) performance data, which is fetched hourly, and Amazon EC2 metadata, which is fetched every 15 minutes. The amount of data generated every hour by a medium to large Amazon Web Services (AWS) consumer environment can push the traditional ETL process to its limit. In this design, data is fetched by a distributed data collector that is running on SAS Viya microservices. The streaming data is transformed using SAS Event Stream Processing and is written to SAS® Cloud Analytic Services (CAS) tables temporarily, using the SAS Event Stream Processing CAS adapter. These tables are then integrated with existing Hadoop files using CAS actions. The design handles scaling and failover safety so that the data collection can be a long-running continuous process. The architecture is generic enough not only to collect the instance data of other cloud vendors but also data from any other similar distributed source.

INTRODUCTION

Traditionally, ETL is performed in batch jobs. These jobs take hours to complete, resulting in a huge latency between data generation and making the data available for actionable insights. In the current era of big data (where systems operate at a high speed), real-time or near real-time actionable insights are in continuous demand.

This paper describes an architecture for a streaming ETL pipeline that uses technologies that are based on SAS Viya. The architecture uses the following components of SAS Viya:

- Microservices that are based on SAS Viya
- SAS Event Stream Processing
- SAS Cloud Analytic Services

The architecture is implemented to collect the data in two different use cases. The data sources in these use cases were totally different.

The streaming ETL pipeline is used to collect data in a SAS® Research and Development (R&D) project. The project is one of the next generation SAS intelligence solutions. The SAS solution collects performance metrics (and other data including metadata, tag, cost and usage information, and so on) of public and private cloud resources and applies advanced analytics to it. These analytics will suggest the optimized usage of cloud resources with a goal of reducing cloud costs by 20% to 30%. As a solution to target cloud resources, the R&D project needs to collect data more frequently because of the way cloud resources are operated and billed. At the busiest time of the day, the data might consist of hundreds of thousands of observations. For an AWS account of SAS, a typical performance data can be more than three hundred thousand records in a single fetch. These records are processed as they are fetched. Thus, the temporary performance table contains more than 25 megabytes of data with around 27 thousand records in a single fetch. The complete data collection process takes approximately 15 minutes. This data collection process includes fetching data from AWS CloudWatch, processing it, writing it to CAS tables, and saving the CAS tables in the underlying Hadoop layer.

The same architecture is used to read multiple huge CSV zipped files and write the data to a single CAS table. The architecture completed reading all the data, processing it, and writing more than 15 million records to a single CAS table in less than two hours. The size of the saved corresponding SASHDAT file

was 45 gigabytes. Reading data from the distributed files simulate multiple streaming data sources. This shows that the architecture can be adjusted for the Internet of Things(IoT). In IoT, most of the data come from various sensors and need to be acted upon in real-time.

ARCHITECTURE

HIGH-LEVEL DESIGN

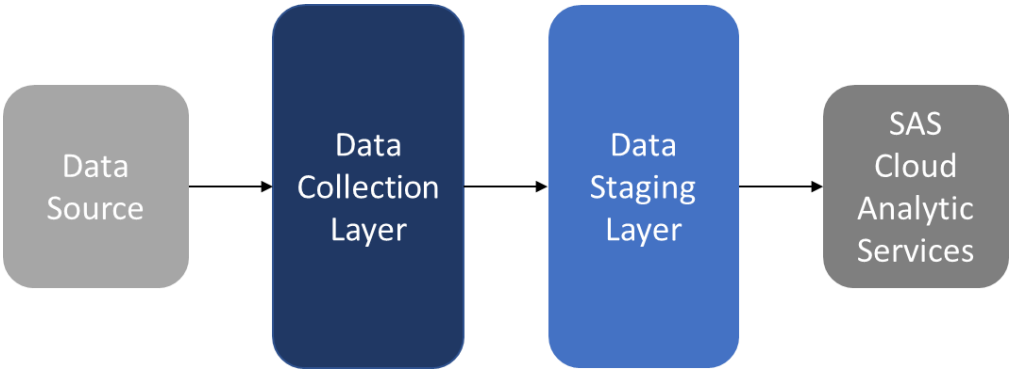

Figure 1. High-Level Architecture

Figure 1 represents a high-level architecture of the streaming ETL pipeline. The data collection layer and the data staging layer form the core of the ETL process.

The architecture handles several types of data sources. A data source can be, but is not limited to, one of the following sources:

- streaming data generated by various sensors
- multiple files distributed into different folders
- frequently gathered data through API calls

The data collection layer interacts with various data sources. The layer can collect data from distributed streaming or static sources. When the data is collected, it is sent to the data staging layer where it is transformed in the flow. The transformed data is written to CAS tables. These CAS tables can be used for further processing if required. Each of the components is described in detail below.

DATA COLLECTION LAYER

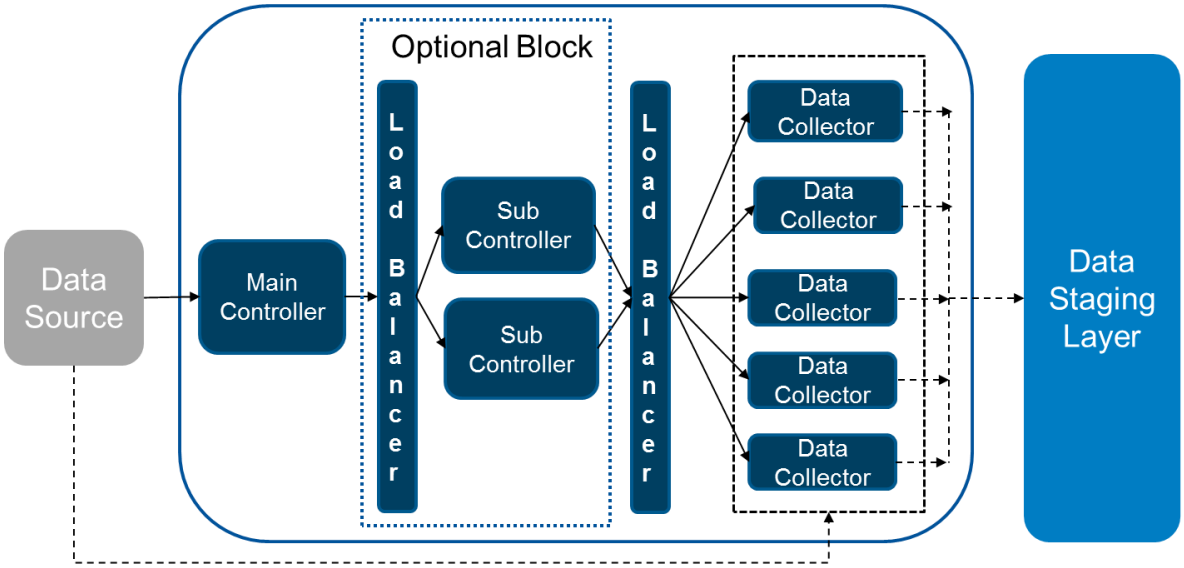

Figure 2. Data Collection Layer

Figure 2 represents the block diagram for the data collection layer. This layer is a distributed framework that consists of SAS Viya microservices. SAS Viya microservices include platform and custom microservices. The data collection layer consists of multiple logical sublayers. The data collection layer must contain a main controller sublayer and a data collector sublayer as separate microservices. A sublayer can be either a microservice or a part of a microservice.

Depending on the business requirement, an optional layer to break the data orchestration into further subgroups can be added. For example, suppose that the data is in multiple folders and each folder has multiple files. In such a case, the main controller can send a request to a sub-controller with the folder location as input. The sub-controller thus can handle all the files in that folder and send a request to the Data Collector accordingly.

Two main layers of the data collection layer are described below.

Main Controller

The Main Controller microservice orchestrates the data to be collected. This microservice is the entry point to the data collection process. The microservice performs various operations depending on the type of the data source. However, the main task of this microservice is to use REST calls to distribute the data collection to the next layer. This distribution depends on a suitable criterion that is related to the data source. After the data collection is complete, the main controller also performs CAS actions on the tables to get the required output. A CAS action can be a simple data step to merge recently collected data with the existing data.

The main controller calls various microservices of SAS Viya. The following list includes some of the SAS Viya microservices that the main controller uses:

- Authorization microservice: The authorization microservice provides a RESTful interface to create, read, update, and delete authorization rules for the new SAS Viya authorization system. The SAS Viya authorization microservice also provides a RESTful endpoint to evaluate authorization decisions that are based on an authorization context and the application of the rules in the system.
- Logon microservice: SAS Logon is a login service web application that also acts as an OAuth2

authorization server with OpenID Connect extensions.

- Configuration microservice: The Configuration Service enables independent services to fetch the configuration properties at runtime.
- CAS Management microservice: This service provides information on CAS servers, caslibs, sessions and tables. It supports definition of caslibs, creation of sessions, and tasks such as listing, loading, unloading, saving, and supporting tables.

In the SAS R&D project, the Main Controller fetches the AWS accounts from the configurations. To fetch data from an AWS account, an access key is required. The access key is generated by a custom written microservice. The Main Controller makes a REST call to AWS Security Token generator microservice which returns the generated key. Then, the Main Controller distributes the call to fetch the data for each individual account to the Data Collector service. The calls are made asynchronously through REST APIs. Each call passes the account information required to fetch the data for the account along with its AWS access key. After the data collection is complete, the Main Controller calls DATA Step code through CAS to merge the recently collected data with the existing data. The Main Controller then calls CAS Save action to save the merged table to the underlying persistent store in SASHDAT format.

For the second use case to read multiple zipped files, the Main Controller fetches the folder name that contains the zipped files from the configurations. It then fetches the filenames from the folder and divides the filenames into sub-lists. After this, the Main Controller makes asynchronous REST calls to the Data Collector microservice. Each call contains a sub-list of file names. This is how the main controller achieves the distribution of the data collection.

Data Collector

The Data Collector microservice fetches the data from the data source that is specified by Main Controller. This microservice implements calls that are based on the type of the data source for which the collection process is developed. The data, while being collected, is converted to a format that the data staging layer (which is the next layer in the data collection) understands. This converted data is passed to the data staging layer during the collection process itself by creating a stream of data.

The Main Controller can make a lot of requests to the Data Collector microservice during data fetch depending on the size of the data source. The volume of requests might increase the load on a single instance of the microservice. To handle this large volume of requests, the Data Collector microservice can be distributed by deploying it on multiple hosts. A load balancer can help in distributing the calls to the various hosts on which the Data Collector microservice is deployed. As Data Collector is stateless and independent, the architecture can be scaled up horizontally.

In the SAS R&D project, the Data Collector is divided into two sublayers. Each layer exposes a REST endpoint to call the corresponding functionality. A call to the Data Collector from the Main Controller contains an AWS AccountID to fetch the data. The Data Collector implements AWS SDK that is based on Java to communicate with AWS. The Data Collector first fetches the metadata for all the instances in the account by making an AWS API call. The metadata is passed to the data staging layer as soon as it is fetched. The Data Collector then creates sub-list of instanceIDs. Each sub-list contains a fixed number of instances. The data collector makes a call to the next sub-layer through REST calls passing a sub-list of instanceIDs. This technique helps in collecting data parallelly for accounts having a large number of instances. AWS Cloudwatch API allows fetching the data for a single metric at a time for a single instance. The Data Collector fetches the data for all instances in the sub-list one by one making a single call for every metric. The data collector parses the returned result and extracts the required values of the metrics. This data is passed to the Data Staging layer immediately.

In the second use case of reading zipped files from a folder, the Data Collector receives multiple REST calls. Each call contains a sub-list of filenames. The Data Collector reads the data of files one by one from the sub-list. Each line is passed to the data staging layer as and when it is read. The code reads data directly from the zipped files without extracting them. In this way, reading the data helps access a huge volume of data in zipped format. By distributing the file list into sub-lists, the whole folder is also read faster.

DATA STAGING LAYER

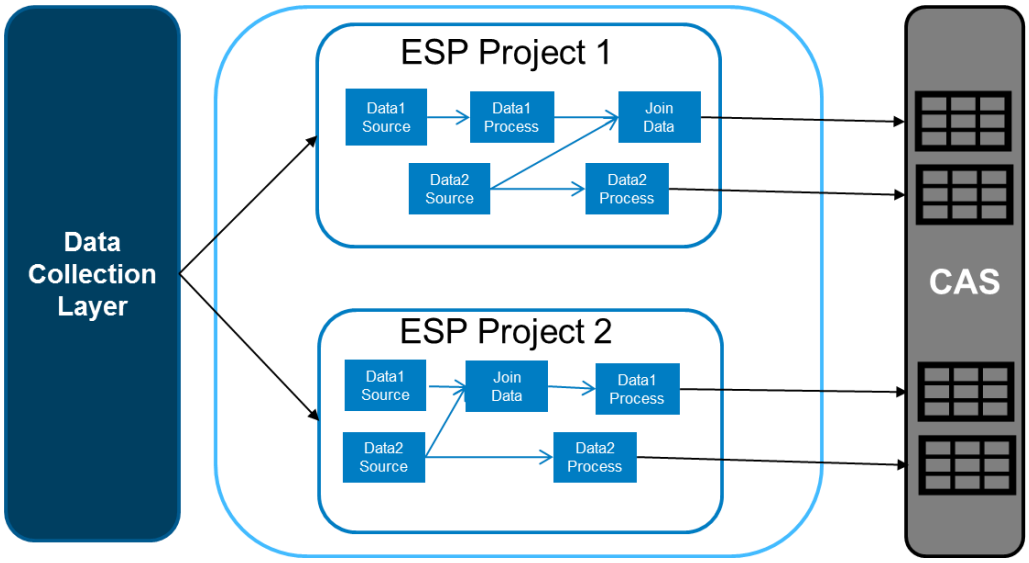

Figure 3. Data Staging Layer

Figure 3 represents the block diagram for the data staging layer. The layer consists of a SAS Event Stream Processing server where single or multiple SAS Event Stream Processing projects run. The purpose of this layer is to transform the raw data into a form that can be used for analysis and load the transformed data into CAS tables. If required further operation can happen on the CAS tables.

In the SAS R&D project, the data staging layer first transposes the metrics of the AWS EC2 instance performance data for each instance. The data staging layer adds the metadata information to the performance data. These operations are performed when the performance data is fed into the data staging layer.

The data collection layer uses the SAS Event Stream Processing publish-and-subscribe mechanism to feed the data to the SAS Event Stream Processing projects. Multiple instances of the Data Collector microservice write the data to the same SAS Event Stream Processing project. Thus, the SAS Event Stream Processing project can receive a huge volume of data quickly. SAS Event Stream Processing is capable of handling such a huge volume of fast-moving incoming data.

The CAS adapter of SAS Event Stream Processing writes the data from the data staging layer to CAS table. The CAS table is periodically saved as a SASHDAT file. The SASHDAT file becomes a persistent copy of the whole data.

After the data collection is completed, the transformed data is aggregated based on instances and is written to a PostgreSQL table. This table contains information about the timestamps for which the data is collected. This table also acts as a Control Table for the data collection. The Control Table helps recover gaps in the performance data.

In the use case of reading multiple zipped files from a folder, the SAS Event Stream Processing project doesn't do much processing of the data. The project acts as a single point of entry of the data into CAS tables.

SAS Event Stream Processing offers the following benefits for data collection:

- SAS Event Stream Processing writes the data to CAS, thereby enabling the developer to concentrate on collecting data from various sources.
- The integration between SAS Event Stream Processing and CAS helps write the data from multiple

sources to a single CAS table at a speed that matches the rate of data collection.

- The SAS Event Stream Processing failover safety feature ensures that data is not lost.
- SAS Event Stream Processing load balancing helps manage huge volumes of fast-moving, incoming data.
- If required, real-time analytics can be plugged into the same infrastructure.

CONCLUSION

The proposed flexible architecture handles many use cases of real-time data and distributed big data. The architecture can be modified to run on a single machine, multiple machines, and in the cloud. The architecture for data collection is useful for IoT scenarios. IoT scenarios contain streaming data sources and require responses to crucial analysis in real time.

The architecture can be enhanced further by incorporating the following features:

- Implement failover safety of SAS Event Stream Processing using Apache Kafka.
- Implement failover safety for the data collection microservices for a continuous collection process.
- Make the system future ready by using load balancing in SAS Event Stream Processing.

ACKNOWLEDGMENTS

- Joseph Hatcher, Terry Lewis, and Chris Weston for supporting the idea behind the paper and giving us a direction through various requirements for the data collection in the project.
- Manish Limaye, Suresh Koukuntla, Avinash Khairnar, Kritika Gupta, Milind Kale, Shagufta Sheikh, Girish Lolage, and Asmita Khedkar to help us in the refinement and development of the idea presented.
- Moti Thadani, Narender CV., Girish Hasabnis, and Joseph Hatcher for reviewing the paper and giving valuable feedback for improving the content of the paper.

RECOMMENDED READING

- *AWS Documentation (*https://aws.amazon.com/documentation/*)*
- *SAS® Event Stream Processing Fact Sheet (*https://www.sas.com/content/dam/SAS/en_us/doc/factsheet/sas-event-stream-processing-106151.pdf*)*
- *SAS® Viya® Solution Overview (*https://www.sas.com/content/dam/SAS/en_us/doc/overviewbrochure/sas-viya-108233.pdf*)*

Contact Information

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

Joydeep Bhattacharya
SAS Research and Development India Private Limited
joydeep.bhattacharya@sas.com

Manish Jhunjhunwala
SAS Research and Development India Private Limited
manish.jhunjhunwala@sas.com