Paper SAS4347-2020

SAS® Intelligent Decisioning: An Approach to High Availability for Real-Time Integration

Michael Goddard, SAS Institute Inc.

ABSTRACT

Are you thinking about real-time analytics—integrating your analytics with your business applications? Do you understand the deployment patterns for SAS® Micro Analytic Service? Do you understand high availability with SAS® Viya®?

This paper outlines two approaches to implementing high availability for SAS Viya, which is used for real-time analytics using SAS® Intelligent Decisioning. In order to implement high availability, you need to understand the deployment patterns and you need to understand that SAS Intelligent Decisioning can be used for batch and real-time processing.

Learn about the architecture considerations, deployment patterns, and an approach to high availability using a shared-nothing deployment architecture pattern.

INTRODUCTION

Architecting for real-time transactional systems is different from architecting for discovery analytics platforms. Once the analytics has been embedded into an operational system or business process, the stakes are raised! You need to be concerned with the business (calling application) service level agreements (SLAs), business continuity, high availability, and system performance.

The following aspects are important to consider when architecting the SAS platform:

- The availability and redundancy requirements. It is important to understand the end-to-end requirements.
- The integration pattern that will be used. For example, is integration via a service bus, message broker, or direct REST calls?
- System response time and latency become very important when integrating with on-line systems. The integration with the analytics platform should not make the calling application run slow.
- Are all the inputs for running the model or decision flow passed on the call or does the model need to reference other external data (reference data)?
- Does the model or decision flow execution need to write out data?
- What are the supporting batch run-time requirements versus the real-time run-time requirements?
- What are the Software Development Lifecycle (SDLC) environment requirements?

In addition to the above, there are also good practices for the model development that are beyond the scope of this paper, but it is sufficient to say that good model design is an essential aspect to achieving good system performance. For the purposes of this discussion

I use the term "model" to refer to an analytical model, decision flow, SAS® Event Stream Processing project or program that is being deployed to SAS® Micro Analytic Service.

SAS MICRO ANALYTIC SERVICE OVERVIEW

SAS Micro Analytic Service is available on both SAS® 9.4 and SAS® Viya®. This paper will only focus on SAS® Viya® 3.5 implementation.

SAS Micro Analytic Service is a memory-resident, high-performance program execution service that is included in selected SAS solutions. SAS Intelligent Decisioning, SAS Event Stream Processing and SAS® Model Manager include SAS Micro Analytic Service. Table 1 provides a high-level comparison, as of January 2020.

Function / Component	SAS Intelligent Decisioning	SAS Model Manager	SAS Event Stream Processing	SAS® Environment Manager
Provides SAS Micro Analytic Service	Yes	Yes	SAS Micro Analytic Service core only	-
Create decision flows	Yes	-	-	-
Create SAS Event Stream Processing projects	-	-	Yes	-
Manage models	-	Yes	-	-
Publish to SAS Micro Analytic Service	Yes	Yes	-	-
Create publishing destinations	-	-	-	Yes
Note: SAS Event Stream Processing contains only the SAS Micro Analytic Service core engine, not the SAS Micro Analytic Service REST microservice.				

Table 1. Functional Comparison

SAS Micro Analytic Service provides the capability to publish (or expose) SAS analytics, business rules, and user-written modules into operational environments. SAS Micro Analytic Service provides hosting for DS2 and Python programs and supports a "compile-once, execute-many-times" usage pattern.

SAS Micro Analytic Service has a layered architecture consisting of the SAS Micro Analytic Service core engine, a Java layer, and the REST layer (interface). See SAS® Micro Analytic Service 5.4: Programming and Administration Guide for more information.

When deployed as part of SAS Intelligent Decisioning, SAS Micro Analytic Service is called as a web application with a REST interface by both SAS Intelligent Decisioning and by other client applications. The REST interface (known as the SAS Micro Analytic Score service) provides easy integration with client applications and adds persistence and clustering for scalability and high availability.

PLANNING FOR AVAILABILITY AND RELATED TERMINOLOGY

Before getting into the detail of what a high-availability configuration might mean for a SAS Viya platform, let's first look at some definitions:

- Availability is a measure of a systems readiness for usage.
- Fail-tolerance is the ability of a system to continue to work in the event of the failure of some of its components. It does not imply continuous operations.
- Continuous operations means that the system is available at all times. It requires components to be fully redundant.

"High availability (HA) does NOT mean continuous (non-stop) operations"

Now that we have some baseline definitions, we need to consider the availability and recovery requirements. The two key requirements here are the Recovery Time Objective (RTO) and Recovery Point Objective (RPO). They are what we call "architecturally significant" as they will drive the deployment approach (architecture) that is taken.

When the real-time analytics is being integrated with your organization's core system(s), the SAS platform will (or should) inherit the availability and recovery requirements of the core system, the calling application. The system as a whole is only as good as its weakest link. This may mean that the availability requirement for SAS could be 4 nines (99.99%), or higher!

As an example, Table 2 illustrates the acceptable unscheduled outage for the different levels of availability. You can see how demanding the requirement quickly becomes.

Availability	Unscheduled Outage per Year
99%	3.6 days
99.9%	8.7 hours
99.99%	52.3 minutes
99.999%	5.2 minutes
Note, the availability calculation shown is based on 4 hours of scheduled maintenance per month.	

Table 2. Availability Calculation

Remember, availability is a measure of a system's readiness for usage. The availability calculation excludes the scheduled maintenance (outage) time.

But what does this mean for the SAS platform?

Most real-time analytics platforms also have an associated batch or scoring process to support the real-time decisioning. Therefore, a key question to ask is—does the availability requirement apply to both the batch and real-time processing?

"Does the availability requirement apply to both the batch and real-time processing?"

For both the batch and real-time processing, the RTO and RPO are key requirements. We need to consider this and not just focus on the real-time platform in isolation.

For SAS Intelligent Decisioning, batch and interactive processing uses the SAS Cloud Analytic Services (CAS) server, while real-time processing is supported by SAS Micro Analytic Service.

This is depicted in Figure 1 below.

Figure 1. Batch, Interactive, and Real-Time Processing

In this generic illustration, I use the term "batch processing" to refer to pre-processing of data to support the decisioning, the running of models in batch, and batch scoring. Interactive processing is referring to using the visual interfaces.

Figure 1 illustrates that typically SAS Micro Analytic Service is not used for batch processing, though it would be possible to call the SAS Micro Analytic Service REST API from a batch program.

On the right, the CAS server has a question mark as it is possible to call CAS using the SAS Viya REST APIs. However, from a SAS Intelligent Decisioning perspective, SAS Micro Analytic Service supports the real-time (REST) integration.

You need to understand these usage patterns when considering system sizing and the requirements for availability. Typically, the batch, interactive, and real-time processing may have different service level (availability) requirements.

Finally, the last bit of terminology that we need to discuss is "design" versus "production" environments. Typically, the development of decision flows and models occurs in a design environment. This may also be the analytics "discovery" environment that your analysts and data scientists use.

A key part of the planning is to determine if the design processing (activities) will be undertaken in the production environment or whether strict SDLC processes need to be followed, with separate platforms for development, test, and production. To implement real-time integration, a good practice is to have separate environments.

In the next sections we will focus on designing for availability of the SAS platform for the real-time analytics, supporting the real-time integration.

APPROACH 1: SAS VIYA HIGH-AVAILABILITY CONFIGURATION

There is an old saying in computing, that the last "9" of availability that you implement will be the most expensive IT spend. This is where the cost of an HA deployment is weighed against the business impact and costs.

Once you have established your availability requirements for the platform (batch, interactive, and/or real-time) you can start planning your deployment.

If your availability requirements aren't particularly high, they may be met by providing a level of redundancy within the platform. For example, for CAS this can be achieved using an MPP CAS server deployment with a secondary CAS controller and multiple workers. For SAS Micro Analytic Service, this would entail installing the MicroAnalyticService host group on multiple servers.

However, if high availability is required, we need to understand the SAS Viya software components and their dependencies. As SAS Micro Analytic Service has several dependencies on the wider SAS Viya platform, you can't just deploy the MicroAnalyticService host group as a standalone entity. You require a full SAS Viya platform deployment. Therefore, a SAS Micro Analytic Service HA deployment also needs a SAS Viya platform HA deployment.

In order to service the REST API calls, SAS Micro Analytic Service has dependencies on a number of core platform services, including: SAS® Configuration Server (Consul), SAS® Infrastructure Data Server (PostgreSQL), and SAS® Message Broker (RabbitMQ).

To provide HA for these core components, a minimum of three servers is required. Therefore, the absolute minimum HA configuration would be using three servers. However, additional servers may be required, depending on your requirements. For example, separation of CAS and SAS Micro Analytic Service workloads, and/or separation of the platform servers and services from CAS and SAS Micro Analytic Service components leads to the use of additional servers.

Focusing on the real-time transactions with a three-server deployment, an SMP CAS server can be used. In this pattern, the CAS server, the SAS Compute Server (the ComputeServer host group) and the Operations host group are only deployed on "Server 1". The three-server deployment is illustrated in Figure 2.

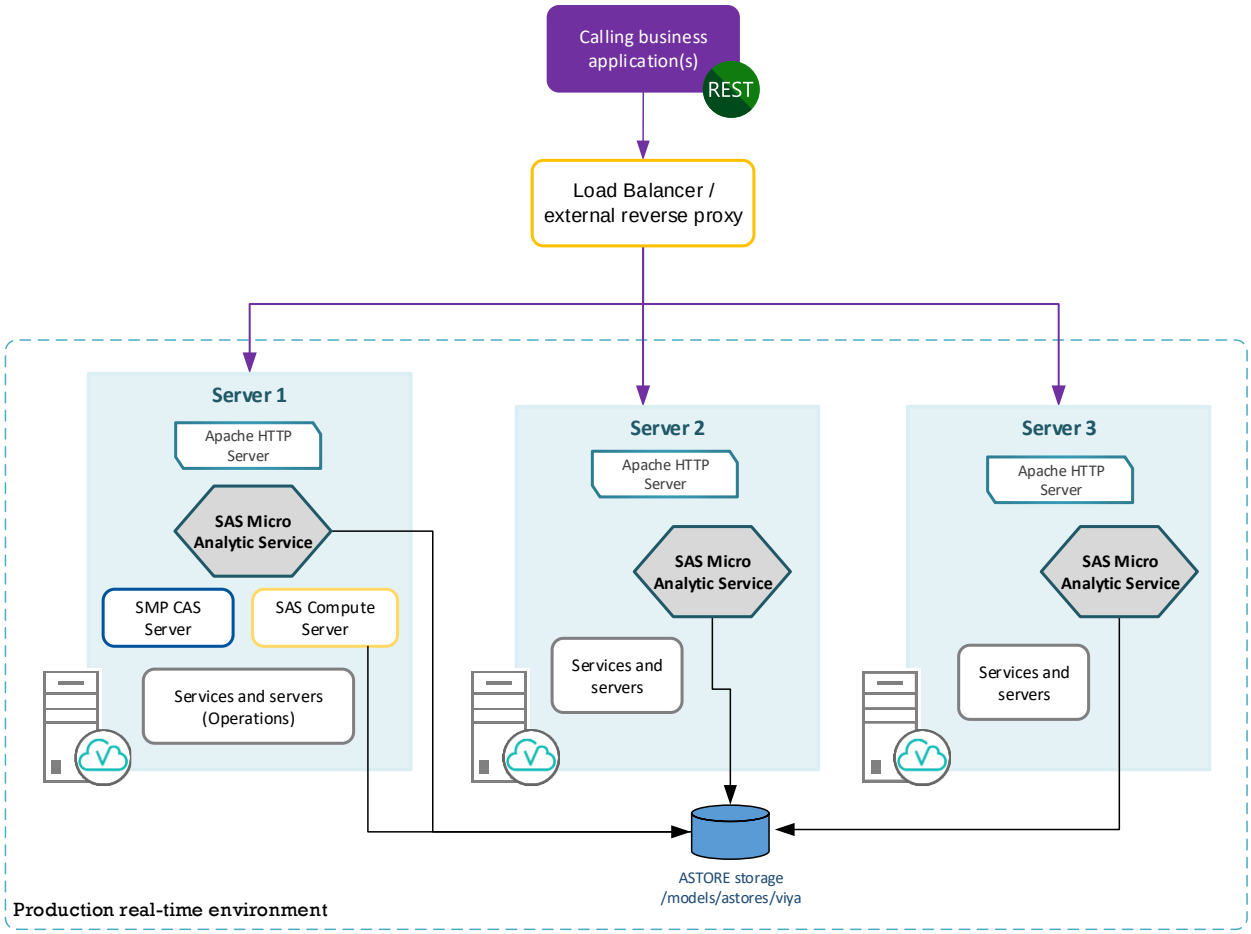

Figure 2. Minimum HA Deployment

Note the Operations host group is a singleton and there can only be one instance of this within the platform deployment. It should be noted that the loss of this function will not stop the platform from operating.

The figure also illustrates the need for shared storage for the analytic store (ASTORE) files. This is described in more detail later in this paper.

However, separation of SAS Micro Analytic Service from the other platform services and CAS processing provides a better model, as SAS Micro Analytic Service hosts can be optimized to support the real-time processing workloads. This is illustrated in Figure 3.

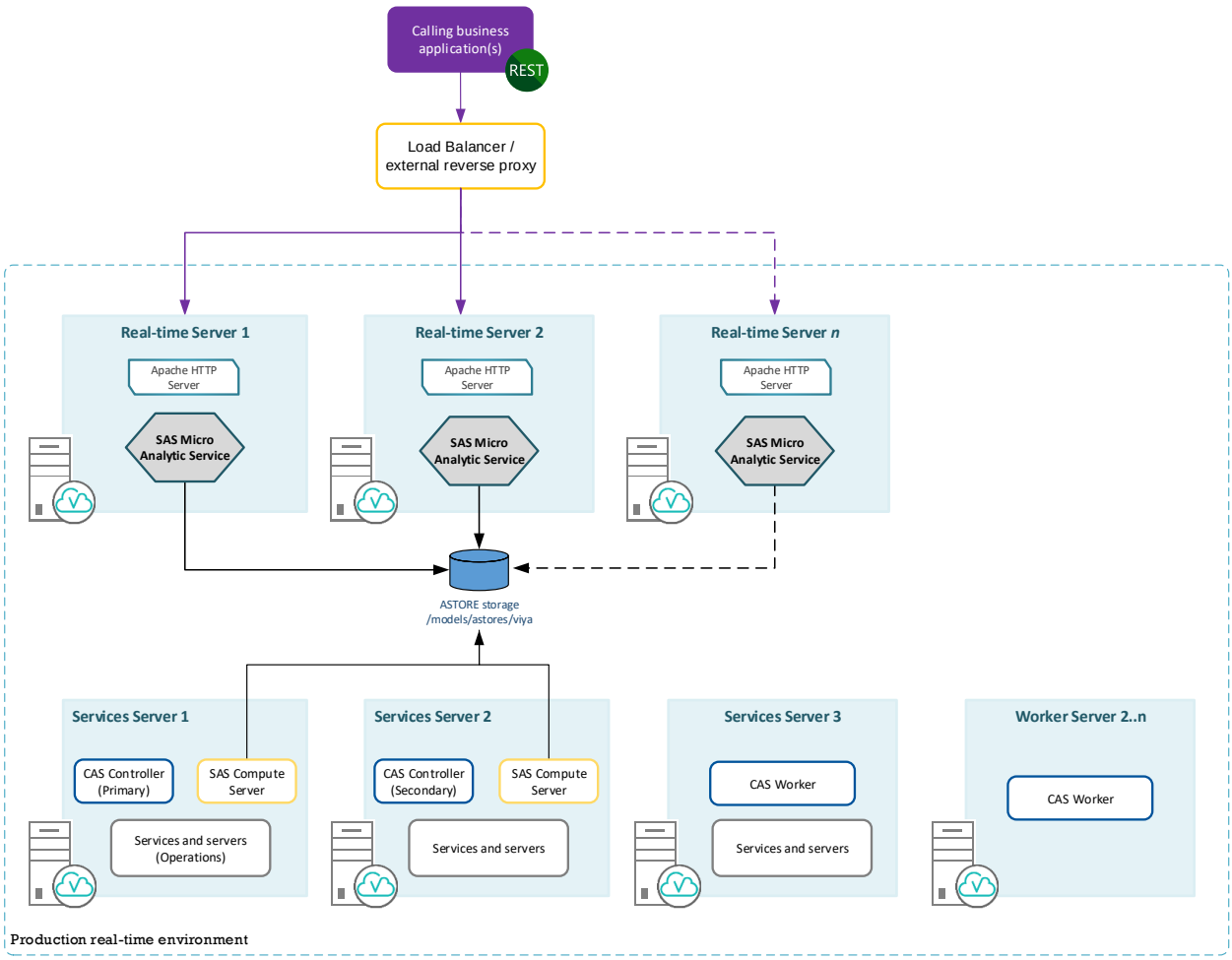

Figure 3. HA deployment with MPP CAS

This pattern can be implemented using five or more servers, depending on the number of SAS Micro Analytic Service servers required and whether you chose to implement an SMP CAS Server or an MPP CAS Server as illustrated in Figure 3.

Figure 4 provides a snippet of the inventory file that would be used to install the configuration shown in Figure 3, showing the key host groups illustrated in the figure.

```
# The httpproxy host group contains HTTP Proxy Server.
[httpproxy]
realtime-server01
realtime-server02

# The MicroAnalyticService host group provides a multi-threaded, low latency
# program execution service to support execution of decisions, business rules
# and scoring models.
[MicroAnalyticService]
realtime-server01
realtime-server02

# The consul host group contains the Consul server.
# A consul cluster must have an odd number of machines.
[consul]
services-server01
services-server02
services-server03

# A RabbitMQ cluster must have an odd number of machines.
[rabbitmq]
services-server01
services-server02
services-server03

[sas_casserver_primary]
services-server01

[sas_casserver_secondary]
services-server02

[sas_casserver_worker]
services-server03
worker-server02

[ComputeServer]
services-server01
services-server02
```

Figure 4. Inventory File Snippet

Note, the figures do not illustrate all the fine details for implementing a SAS Viya HA configuration. For example, shared storage is not only required for the ASTORE files. Please refer to the SAS documentation for these details.

In both examples, an external load balancer is required. This would also be configured as the reverse proxy for the platform. The load balancer is not a component provided by SAS.

ASTORE STORAGE

As briefly discussed above, there is a need for shared storage for the ASTORE files.

An Analytic Store, or ASTORE, represents the state of a trained predictive model that has been saved in a transportable form, a binary file. This enables it to subsequently be used to score new data in a variety of environments. Many SAS analytical procedures save the results from the training phase of model development as ASTORE models. A key feature of an ASTORE is that it can be easily transported from one platform to another. When an ASTORE is published to SAS Micro Analytic Service, the state of the predictive model is restored and is available for scoring new data.

The use of shared storage avoids having to manually copy files to each server running SAS Micro Analytic Service.

While there are numerous options for sharing storage, with a high-availability configuration, the best option is the use of highly available network attached storage, such as a NAS or a clustered file system.

What do you need to know?

- The SAS Compute Server extracts the ASTORE file from the analytic store's CAS table in the ModelStore caslib and copies it to the /opt/sas/viya/config/data/modelsvr/astore directory path.
  This happens when an ASTORE model is published or set as a champion with SAS Model Manager or when an ASTORE model-based decision flow is tested with SAS Intelligent Decisioning.
- At run time, SAS Micro Analytic Service requires the model's analytic store (ASTORE) file to be accessible from the /models/astores/viya directory path.
  Specifically, in order to publish analytic store models or decisions that use analytic store models to the maslocal (SAS Micro Analytic Service) publishing destination, the model's ASTORE file must be accessible from the /models/astores/viya directory path.

As can be seen, the SAS Compute Server and SAS Micro Analytic Service are using different locations, which has been done to allow for scaling SAS Micro Analytic Service.

Therefore, in order to make the model's ASTORE file accessible, you must map the /opt/sas/viya/config/data/modelsvr/astore directory on the SAS Compute Server to the /models/astores/viya directory on each server running SAS Micro Analytic Service.

This can be done in several ways. The simplest approach is to use a network share (NFS mount) to the host running the SAS Compute Server. However, this approach doesn't provide any HA. This is shown in Figure 5.

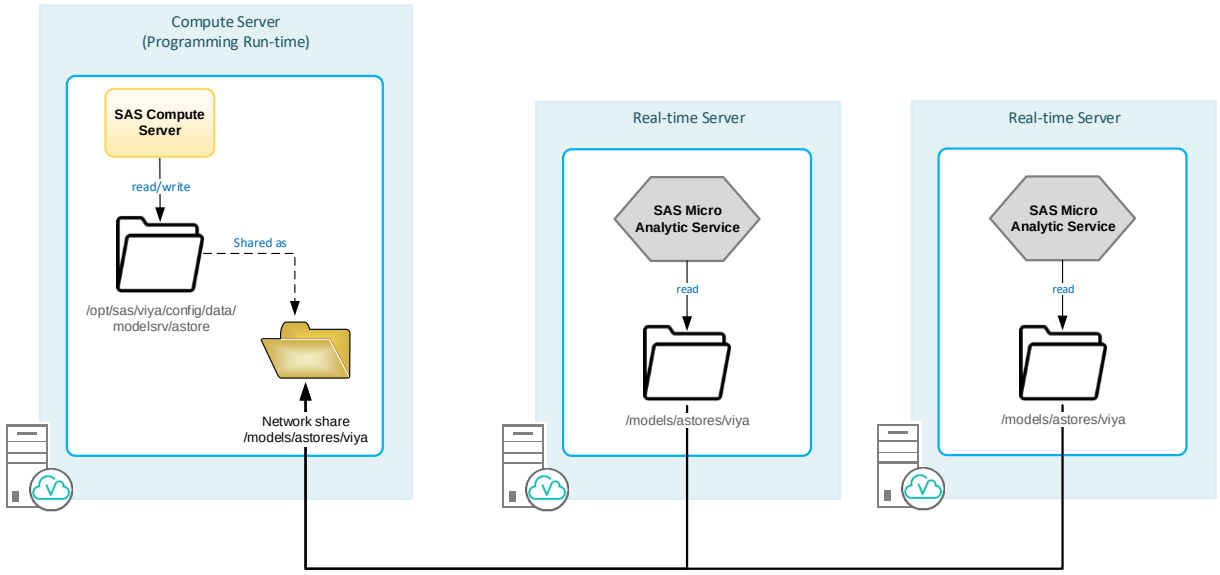

Figure 5. ASTORE Storage with Separate SAS Micro Analytic Service Servers

In this example, the /opt/sas/viya/config/data/modelsvr/astore directory on the SAS Compute Server is shared and has been directly NFS mounted as /models/astores/viya on each SAS Micro Analytic Service (real-time) servers.

A better approach for an HA configuration is to use a highly available network location. This is also the best option when implementing an HA deployment that is using multiple SAS Compute Servers and multiple SAS Micro Analytic Service servers. This is illustrated in Figure 6.

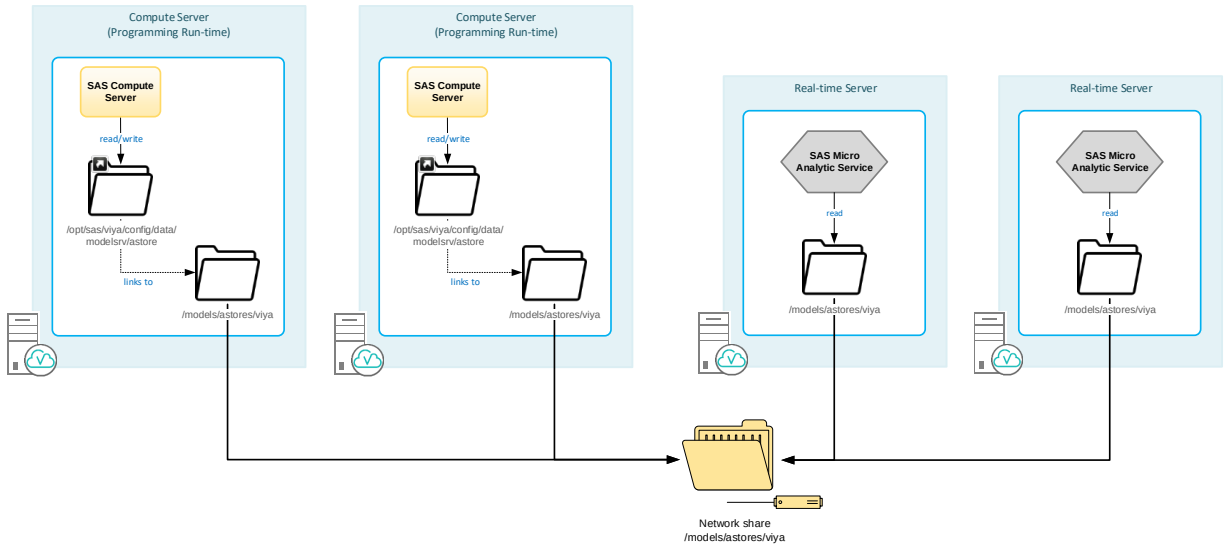

Figure 6. ASTORE Storage with Multiple Servers

In this example, the shared storage is external to the servers, network storage has been used. The shared storage has been NFS mounted as /models/astores/viya on each server, with a symlink being used on the machines that are running the SAS Compute Server. The symlink maps the /opt/sas/viya/config/data/modelsvr/astore path to the /models/astores/viya directory path.

APPROACH 2: SAS VIYA SHARED-NOTHING CONFIGURATION

An HA configuration adds complexity in configuration, administration, and operation. This complexity will have an impact on maintenance and upgrades. The advantages of an HA SAS Viya deployment outweigh the complexities, especially for real-time operation of critical production systems.

However, depending on your requirements, and if you truly have to run 24x7x365, an alternative approach is to use a shared-nothing architecture rather than a single platform with an HA configuration.

Definition:
A shared-nothing architecture (SN) is a distributed-computing architecture in which each node is independent and self-sufficient, and there is no sharing of components across the system.

The shared-nothing, or Active/Active, pattern involves implementing two or more SAS Viya servers to process the transactions, this is depicted as the "Run-time" servers in Figure 7.

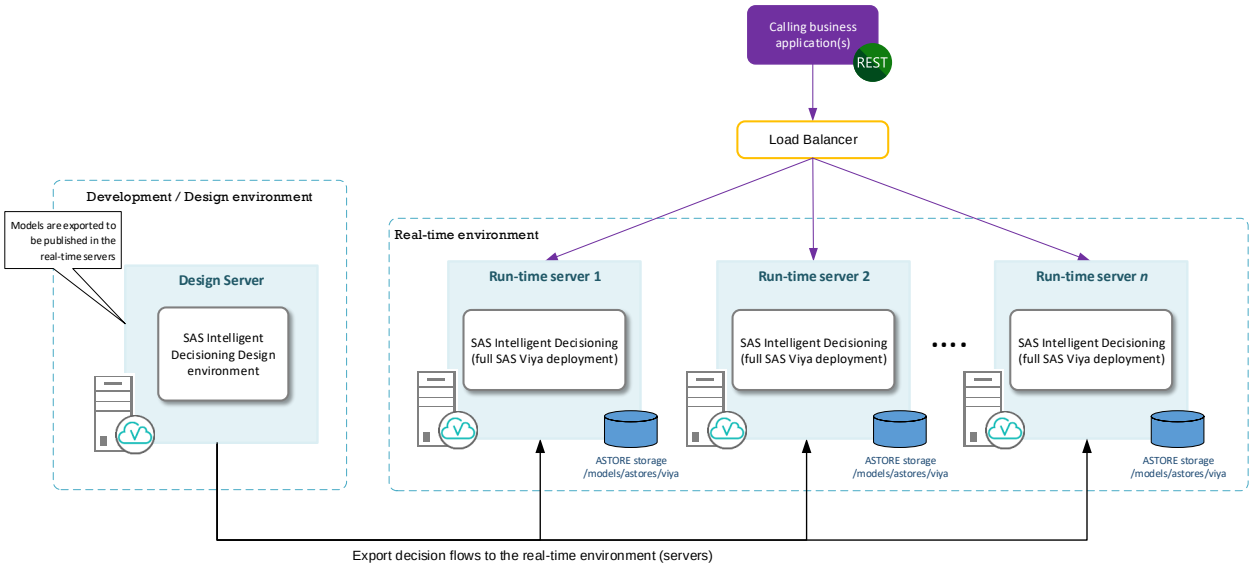

Figure 7. Real-Time Shared-Nothing Architecture

Each server is an independent (self-sufficient) full SAS Viya deployment. The decision flows are developed in a development or design environment, and are then imported and published on each SAS Micro Analytic Service run-time server. They can have their own instance of the ASTORE storage, as depicted, or they can make use of shared storage, which will eliminate file handling. See Figure 5 and Figure 6.

The shared-nothing (SN) architecture (as with an HA configuration) requires the use of an external load balancer in this case to direct the traffic to each SAS Micro Analytic Service run-time server. The smarts are in the configuration of the load balancer to detect an outage and to control (direct) the traffic to the remaining servers. Typically, this would be configured as an active-passive pair of load balancers.

The SAS Viya Transfer CLI would be used to deploy the decision flows to each server. This could be managed through Ansible to make the administration a little easier and to ensure (enforce) consistency of the commands being issued.

Depending on the SAS licensing, an SN deployment will most likely have an impact on the software costs. However, the additional cost of the extra licensing may even out when compared to the Total Cost of Ownership (TCO) of setting up and maintaining an HA environment. (*TCO is an estimate of all the direct and indirect costs involved in acquiring and operating a product or system over its lifetime.*) The extra cost should be traceable back to meeting the availability requirements.

Expanding on this concept, a hybrid approach would be to implement an Active/Passive configuration, where each SAS Viya deployment is a multi-server installation. This provides the option of scaling a SAS Micro Analytic Service deployment in each environment. You might also take this approach to implement an Active/Active configuration across two availability zones or data centers.

Some additional considerations are the contact history information and token management, which are discussed below.

IMPLEMENTATION APPROACH

To implement the shared-nothing pattern, as each SAS Viya deployment is independent, the calling application or "client" needs to be registered with each SAS Viya platform. Once the client has been registered, an access token and refresh token must be requested (generated). Therefore, you need a process to handle using the tokens and refreshing them. By default, the access token has a life of 10-hours.

As part of the client registration, it is possible to set the token validity, which is the time-to-live for the token. Depending on your security policies, it may be possible to set a longer token lifetime to help reduce the token management overhead. For example, you could set the token validity to expire after a month or longer.

Additionally, as with any application integration, there is more to the integration than just sending a request to a REST API. You need to implement the application (business) logic for error handling and transaction retries. Also, in our case, you need to implement the management of the access tokens.

The transaction load balancing process needs to include the logic to flag that a server is unusable (unavailable), and the ability to detect that a server is available again and has returned to normal service. Some form of "health-check" transaction may be required as part of this process.

Three integration patterns have been identified.

Direct application integration	This pattern is mentioned for completeness, but it is seen as the least desirable, as it tightly couples the business application(s) to the SAS platform. Using this approach, the error recovery and token management is handled by the calling application. It is the client to the SAS Viya platform. The business application needs to be aware of all the SAS servers, including which servers are available. The application is responsible for any load balancing or fail-over logic. This is not a good, sustainable approach to application integration.

Intelligent Proxy / Load balancer	This pattern isolates some or most of the integration complexity from the calling application. For example, with an intelligent load balancer, the server load balancing and access token injection is handled by the load balancer. The load balancer functions as a proxy, in that the calling business application does not specify which SAS Micro Analytic Service server to use. Rather, the request is directed to a virtual IP address or service host name for a SAS Micro Analytic Service cluster. However, the application still needs to handle the error recovery and transaction retries. This is illustrated in Figure 7. To assist with the error handling and token management, the client or load balancer needs to be able to identify the session or SAS Micro Analytic Service server using a header token. See an example configuration later. Some form of "health-check" transaction should still be implemented, and this could be run from the load balancer. There are many options for implementing this functionality depending on whether the platform is running in-house or in the cloud. For example, the F5 BIG-IP appliance or software could be used. This has a virtual server functionality, which would be the client to the SAS Viya platforms.
Broker / Middleware Integration	This pattern loosely couples (isolates) the business application from the SAS platform(s), with the middleware handling message delivery, the access tokens and their management. It also enables the implementation of a generic interface contract and message definition for integration with the business application. This is discussed in more detail below.

The "Broker / Middleware Integration" pattern is illustrated in Figure 8. There are many options for implementing this pattern, and this is one example. As can be seen, each SAS Micro Analytic Service run-time server is a full SAS Viya deployment, with its own set of SAS Viya services and servers.

As stated above, this pattern loosely couples (isolates) the business application from the SAS platform(s), with the middleware handling message delivery, the access tokens and their management. It also enables the implementation of a generic interface contract and message definition for integration with the business application.

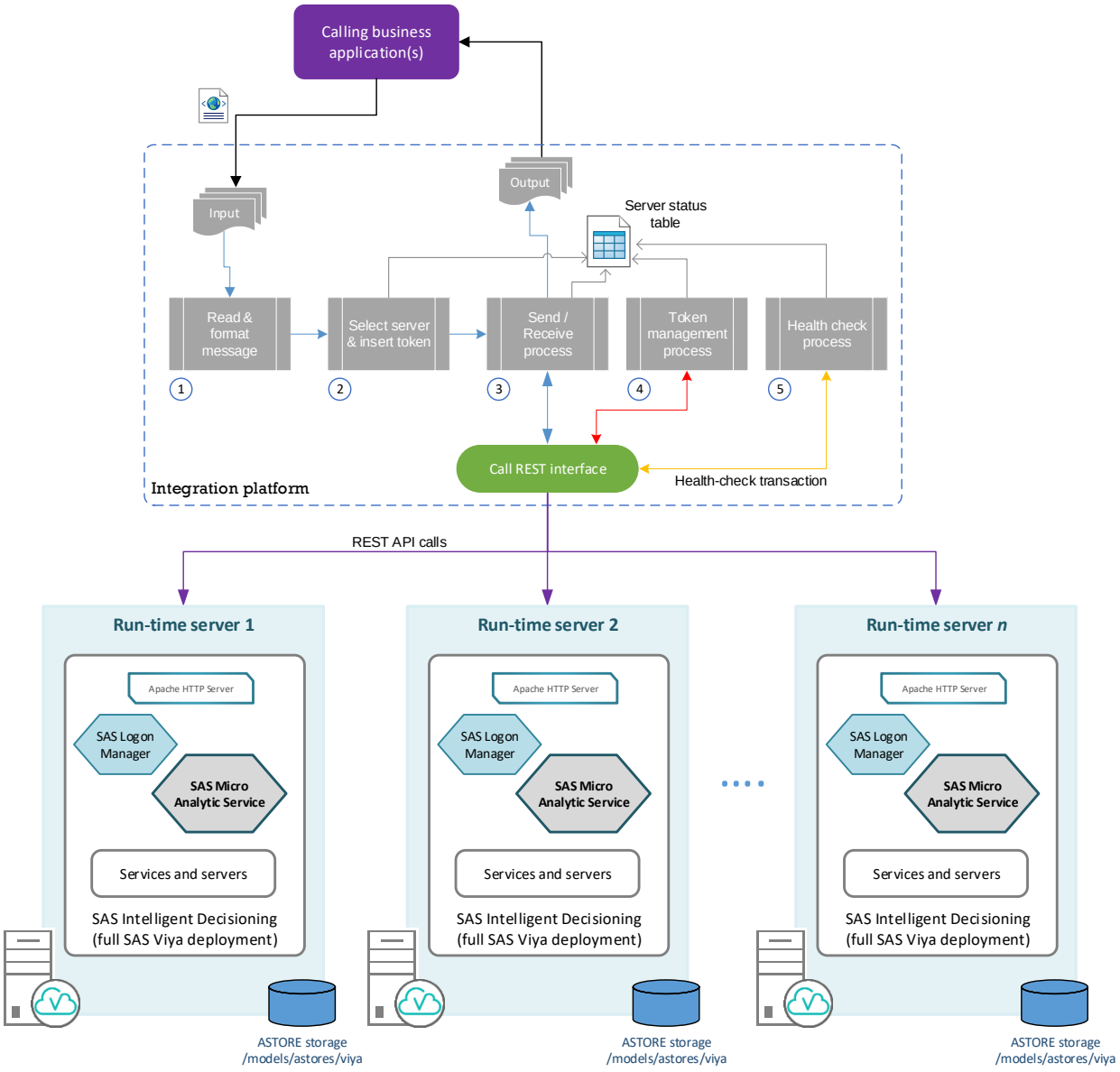

Figure 8. Broker / Middleware Integration Example

In this example, the business application "posts" the transactions to an input queue. The message format could include a sequence number to help with message delivery and sequencing. The responses from the "Run-time" servers are put in an output queue for delivery to the calling business application.

If there are multiple business applications, you would implement input and output queues for each application.

The major steps are illustrated in the diagram, namely:

1. Read and format message. The read message process reads a message from the input queue and formats the message for the SAS Micro Analytic Service REST API call.
2. Select server and insert token. This step selects an available SAS Micro Analytic Service ("run-time") server using the status information stored in the Server Status

Table and inserts the token for that run-time (SAS Micro Analytic Service) server. The message request is updated for the target server and the associated token.

3. Send / Receive process. This step implements the business logic to call the REST API and handle error retries. If the selected SAS Micro Analytic Service server is found to be unavailable, the Server Status Table is updated to reflect this status.

   An implementation choice is whether this process selects a new SAS Micro Analytic Service server and resends the transaction, or whether it returns a failed request (return code) and the business application is responsible for the message retry.

4. Token Management Process. The token management process handles the refresh of the tokens. It monitors the token expiry for each run-time (SAS Micro Analytic Service) server and requests a new token. It updates the Server Status Table with the new access token for the run-time server.

5. Health check process. The health check process is a side, or out-of-band process to monitor the availability of the run-time servers.

The Server Status Table would contain the following information: server name and status, the server access token, the Time-to-Live (TTL) for the access token and the server refresh token. For example.

ID	serverName	serverStatus	accessToken	accessTTL	refreshToken
1	mas01	1	...	...	...
2	mas02	0	...	...	...

To assist with the error handling and token management the integration platform may need to be able to identify the session or SAS Micro Analytic Service server using a header token. This can be achieved through the httpd configuration on each server.

Within the SAS platform the httpd configuration (/etc/httpd/conf/httpd.conf) can be updated to include a server identifier using the following statement. For example,

```
Header Set X-MAS-INSTANCE <server_ID>
```

Using this format, the httpd.conf file for the "Run-time server 2" (mas02) server would be updated with the following:

```
# Add Header Identifying Apache Instance
Header Set X-MAS-INSTANCE MAS02
```

Finally, it is good process to implement a health-check process to monitor the run-time server status and to flag the server as being available. It updates the Server Status Table based on the results of running the health-check transaction. This could be a decision flow that has been created for this purpose.

The scope of the health-check transaction also needs to be considered. Is it just that the REST API (SAS Micro Analytic Service) endpoint is contactable or does the dummy decision touch upstream resources such as databases, and so on?

The Spring Boot framework also provides a number of built-in endpoints that could be used. For example, the "health" endpoint provides basic application health information. There may

be other features provided by the chosen middleware that could be used to build out the health-check process.

Again, this is not the only way to implement this pattern. This is an example.

TOKEN MANAGEMENT PROCESS

An initial part of the application integration is the client registration and the process to obtain an access token. I won't go through this in detail as it is covered in the SAS manuals. Also see the SAS paper OpenID Connect Opens the Door to SAS® Viya® APIs for more information.

As previously stated, by default, SAS Logon Manager issues access tokens with a 10-hour expiration and refresh tokens with a 30-day expiration. When the access token has expired, a service may call the authorization server with the refresh token and obtain a new access token.

Note, you do not get a refresh token when using client_credentials as the authorization grant type.

When you register the client, it is possible to set the time-to-live (TTL) for the Access Token and Refresh Token using the access_token_validity and refresh_token_validity parameters. These should be set in accordance with your organization's security policies. Note, the parameter is specified in seconds.

For example, the following command can be used to register a client with a 24-hour (86,400 second) access token. The variable $CONSUL_TOKEN has been used to store the Consul access token that is used to register the client in Consul:

```
curl -X POST http://<your_server>/SASLogon/oauth/clients \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $CONSUL_TOKEN" \
  -d '{
      "client_id": "<client_ID>",
      "client_secret": "<my_client_secret>",
      "scope": ["openid", "*"],
      "resource_ids": "none",
      "authorities": ["uaa.none"],
      "authorized_grant_types": ["password"],
      "access_token_validity": 86400}'
```

The client registration is needed on each SAS Viya server (each of the "SAS Micro Analytic Service run-time" servers).

Using the example integration model (shown in Figure 8), the token management process should update the Server Status Table to mark the server as unavailable prior to refreshing the token. Therefore, the flow would be as follows:

1. Set server status to unavailable.
2. Refresh the server (client) Access Token.
3. Test the token using the health-check transaction.
4. On confirmation of a valid token, update the server status to available.

Ideally, this process should be run prior to token expiry to avoid failed transactions.

The token management approach illustrated in this paper is one approach. However, it is not the only way to implement the platform.

CONTACT HISTORY INFORMATION

As is often the case, the devil is in the detail, and the shared-nothing approach is no different. In addition to the token management, the contact history information is another area of complexity.

As each SAS Micro Analytic Service server is an independent SAS Viya platform, there isn't any consolidated view of the contact history information. A process would have to be developed to extract the information from each platform and to create a consolidated contact history database. You need to understand this if you are currently using or are planning to use the contact history information as an input.

If you are planning to make use of the contact history information, it is critical to understand that currently, with SAS Intelligent Decisioning 5.4, there is not any functionality for the contact information to be used to dynamically change the decisioning process. This is not a restriction of the SN architecture.

However, there are two macros that can be used to build a contact history database or analytical base table.

The %DCM_GET_SUBJECTCONTACT_HISTORY macro retrieves subject contact history information for the specified decision from the subjectContact microservice.

Then the %DCM_BUILD_SUBJECTCONTACT_SMP_ABT macro can be used to build an analytical base table (ABT).

The %DCM_BUILD_SUBJECTCONTACT_SMP_ABT macro reads the output tables that were produced by the %DCM_GET_SUBJECTCONTACT_HISTORY macro and builds an analytical base table (ABT) using the data in those output tables.

You can then consolidate the ABT tables from each platform in a shared contact history database, consolidated ABT. The consolidated ABT can be used as input for a modeling or reporting process in order to discover which attributes and variables are driving the decision process.

This functionality became available with SAS Intelligent Decisioning 5.3. See *SAS Intelligent Decisioning 5.4 Macro Guide* for more information on the %DCM_GET_SUBJECTCONTACT_HISTORY macro and the %DCM_BUILD_SUBJECTCONTACT_SMP_ABT macro.

Response Tracking Codes

Another consideration associated with the contact history is the Response Tracking Code (RTC) associated with a specific treatment or decision. This is returned to the calling application.

For example, when a customer service application or other calling application uses the subject ID to send a request to SAS Intelligent Decisioning that invokes a decision, a response tracking code and the set of offers for which the subject qualifies are returned to the calling application.

The calling application or a customer service representative can present the offers to the customer. The calling application can use the response tracking code to update the subject contact history data. For example, you could record which treatments are presented to the customer and the subject's response to the treatments.

The consideration here is that the RTC is specific to a SAS Viya platform. It is not shared across the SAS Viya platforms. However, as stated previously about the contact history, it is important to note that the RTC cannot be used to dynamically change the decision process.

If you need to keep track of the RTC and which platform issued it, then an HTTP header token (server identifier) could be used as described earlier. The calling application would need to capture the response and save this data for later processing or storing as part of the contact history ABT.

Note, the response including the responding SAS Micro Analytic Service server needs to be saved. This can be done using the HTTP header token identifying the SAS Micro Analytic Service server.

Summary

It is important to note that the use of these macros isn't the sole domain of using the shared-nothing architecture, but the need to consolidate the records is.

Currently, if you need contact history with response tracking, then the best option is the use of a single HA SAS Viya platform.

COMPARING THE TWO APPROACHES

Both approaches have their advantages and disadvantages. The key items are summarized in Table 3.

Approach	High-Availability Deployment	Shared-Nothing Deployment
Advantages	Decreased administration compared to the shared-nothing pattern, as it is a single platform.	No single points of failure.
	Single action to deploy a decision flow to the SAS Micro Analytic Service servers.	Each SAS Micro Analytic Service server is fully independent – failure of one server will not affect another SAS Micro Analytic Service server.
	There is a single management interface (SAS Environment Manager) for the platform.	Less complexity from a SAS server build perspective.
	Uses standard licensing, as it is a single platform.	Maintenance – each SAS Micro Analytic Service server can be patched independently.
Disadvantages	Increased build and configuration complexity due to clustering of components.	Increased administration and maintenance.
	Platform maintenance is more complex, especially if using the pgpool HA configuration.	More complex to deploy a flow and to synchronize ASTORE data across all servers.

Approach	High-Availability Deployment	Shared-Nothing Deployment
	Most maintenance will require a system outage. All SAS Micro Analytic Service servers need to be updated at the same time.	The contact history records need to be merged into a single database.
		There is no single management interface.

Table 3. Approach Comparison

CONCLUSION

This paper has presented two approaches for implementing high availability for a SAS Viya 3.5 platform to support SAS Micro Analytic Service and real-time transactions. We looked at deployment patterns for SAS Viya high availability and using a shared-nothing architecture pattern.

While the shared-nothing pattern adds complexity when it comes to token management and the creating a single contact history record, it has advantages in providing a true Active/Active configuration for the SAS Viya platform, which enables non-disruptive maintenance.

However, depending on your requirements, a single SAS Viya platform using a high-availability configuration may be the best approach, as it simplifies token management and creation of any contact history ABT.

As discussed, there is more to application integration than just calling a REST API. The end-to-end application business logic needs to be designed, including transaction retry and recovery. You will also need to have a process (manual or automated) for managing the tokens.

When dealing with critical business systems, the integration needs to be carefully planned and well designed.

Finally, the installation, configuration, and support of any load balancer or integration middleware is your responsibility. These are not components provided by SAS, but HA for these components also needs to be considered.

REFERENCES

Riva, Edoardo. 2019. "Proper Planning Prevents Possible Problems: SAS® Viya® High-Availability Considerations" *Proceedings of the SAS Global Forum 2019 Conference.* Cary, NC: SAS Institute Inc.
https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2019/3481-2019.pdf

Roda, Mike. 2018. "OpenID Connect Opens the Door to SAS® Viya® APIs" *Proceedings of the SAS Global Forum 2018 Conference.* Cary, NC: SAS Institute Inc.
https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/1737-2018.pdf

SAS Institute Inc. 2019. *SAS Intelligent Decisioning 5.4 Macro Guide.* Cary, NC: SAS Institute Inc. http://documentation.sas.com/?cdcId=edmcdc&cdcVersion=5.4&docsetId=edmmacro&docsetTarget=titlepage.htm&locale=en

SAS Institute Inc. 2019. *SAS Micro Analytic Service 5.4: Programming and Administration Guide*. Cary, NC: SAS Institute Inc. https://documentation.sas.com/?docsetId=masag&docsetTarget=masagwhatsnew.htm&docsetVersion=5.4&locale=en

SAS Institute Inc. 2019. *SAS® 9.4 and SAS® Viya® 3.5 Programming Documentation: DS2 Programmer's Guide*. Cary, NC: SAS Institute Inc. https://documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.5&docsetId=ds2pg&docsetTarget=titlepage.htm&locale=en

ACKNOWLEDGMENTS

I would like to thank the following people for taking the time to review and contribute to this paper:

- Chris Upton
- Glenn Clingroth
- David Duling
- Prasenjit Sen
- Mike Roda

CONTACT INFORMATION

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

Michael Goddard
SAS Institute Inc.
michael.goddard@sas.com