Paper SAS2272-2018

SAS® Viya® Service Layer Architecture Overview

Eric Bourn, SAS Institute Inc., Cary, NC

ABSTRACT

The SAS® service layer has been completely re-architected from the ground up in SAS® Viya®. The monolithic web application server used in SAS®9 has been replaced by modular microservices, which act as building blocks to help form a loosely coupled, scalable, and secure platform. This presentation focuses on the various technologies used within SAS Viya, from service registration to an event delivery architecture model that enables services to communicate with one another. Learn about a few of the key microservices included within SAS Viya and the benefits they provide.

INTRODUCTION

For nearly fifteen years, the architecture of a SAS®9 deployment has centered around two main components: the SAS® Metadata Server and the SAS® Web Application Server. These two components have served as the backbone of virtually every solution deployed on the SAS®9 middle-tier. Although these components have performed exceptionally well, they do have a few drawbacks that prevent them from being successful in today's modern cloud-centric environments.

The SAS Web Application Server is a monolithic application; it consists of a set of services deployed in a single unit. As new solutions and APIs are developed over time, the size of this monolithic application grows. As this happens, it becomes exceedingly difficult to maintain and introduce change. Deployment is more complex and overall start-up times continue to increase. Services contained within this application become more and more tightly coupled with one another. Because it is a tightly coupled system, the application now becomes a single point of failure. If a single service deployed within the application runs into an out-of-memory condition, it's conceivable that the error could bring down the entire application. Or if a patch needs to be applied to a service, the entire application must be stopped and restarted. That alone could result in significant downtime and loss of productivity.

In order to provide a scalable, high-performing system that is capable of running in any type of environment—cloud or on-premises—SAS realized that the architecture needed to be rebuilt from the ground up. It was time for a drastic change. The middle-tier platform that had served SAS so well for many years needed to be redesigned using the latest technologies and design patterns. This paper focuses on the architecture of it's replacement; the new SAS Viya service layer. It explores the benefits of the new components and how each of the components work together to form the core of SAS Viya.

MICROSERVICES

With more and more software being deployed to cloud environments, applications increasingly need to be more flexible and robust than in the past. They must be able to scale appropriately and expand to spikes in demand as needed. Doing so within a monolithic application just isn't possible. In order to properly scale, these applications must be broken down into smaller components, with each component isolated from the others and capable of operating on its own. SAS microservices are such components, and they are the building blocks of the SAS Viya architecture.

BENEFITS OF MICROSERVICES

The loose definition of a microservice is an application that does one thing and does it well. Each area of functionality is split into its own component, thus creating an extremely flexible environment where microservices can be deployed independently of one another. Many of the intra-service dependencies are removed so that these components are no longer tightly coupled. Without these dependencies in place, individual services can be stopped and started without affecting the state of other services or the overall system.

What makes this design better than the previous monolithic approach? In plain and simple terms, smaller units allow for more flexibility and control. Imagine the situation where a single microservice has run into an out-of-memory condition, or a critical security fix needs to be applied to patch a vulnerability. Restarting a single microservice affects only users of that particular service at a given time; it is no longer necessary to restart the entire application stack. Restarts for a monolithic application can require a significant amount of time and impact the entire system. For the SAS Web Application Server, a typical restart could take easily twenty or thirty minutes. Yet because microservices are smaller in size and their functionality is isolated, restarts for them generally take a couple of minutes in order for the service to become fully operational.

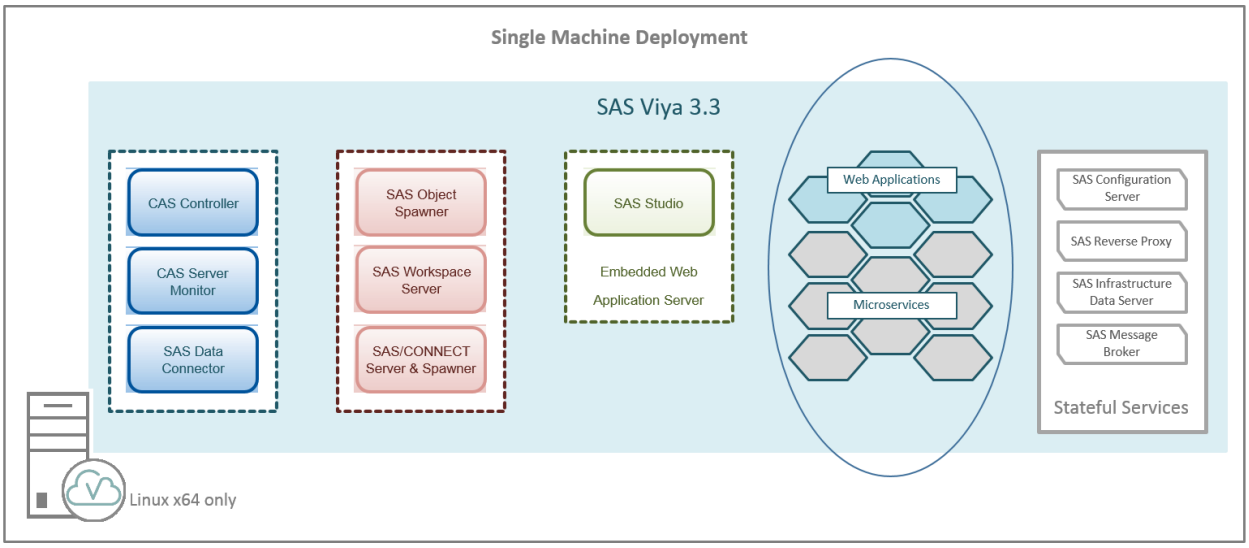
One primary goal of microservices is to reduce the possibility of a single point of failure. They are designed to be loosely coupled so that they continue to function at high capacity even when dependent services are unavailable. If a service is down, you lose only the functionality that service provides, while most other actions continue to function as normal. There are a few exceptions to this rule that will be discussed later, but in general, the impact of a down microservice does not affect the entire application stack. For example, if the microservice that supports delivering email messages to consumers is down, client applications continue to function as expected, with the exception of sending and receiving emails. The same cannot be said about the previous web application server. Because its services were tightly bundled together, any failures in a single component such as its mail delivery could have a significant impact on the entire system.

These concepts do not just apply to SAS microservices. Instead, they are part of a broad effort, known as “The Twelve Factor App,” which aims to develop applications that behave properly in a cloud environment. The idea in general is to create services that can start up quickly, can be started asynchronously from others, and can be gracefully shut down.

SAS VIYA PLATFORM

Now that we’ve discussed what microservices are and the benefits they provide, let’s dive into how they are bundled together with other components to form the SAS Viya platform.

At the heart of the SAS Viya architecture is a large collection of microservices. A single SAS Viya deployment can contain 50 or even 100 different microservices, each working together to form a cohesive system. Each service is designed to provide a specific set of functionality, and these smaller sets of work-units enable applications to pick and choose which services they should deploy. By supporting an à la carte methodology, applications can deploy the set of services that are applicable to them. For example, SAS® Visual Investigator can deploy a set of custom-designed fraud-related microservices that would not be applicable to any other application. Likewise, SAS® Visual Analytics can deploy a set of microservices related to data reporting that is specific to only that solution.



Display 1. SAS Viya Microservices

There are certainly too many microservices to document here, but there are a few services that are worth calling out. These core services provide a set of functionality that's applicable to nearly all SAS applications.

- *SAS Logon* is a web application that enables users to authenticate to a SAS Viya deployment, using features built around OAuth 2.0 and OpenID Connect.
- *Authorization* manages a set of authorization rules that grant users access to various resources in the system.
- *Identities* retrieves information about identities (users and groups) defined within an environment.
- *Configuration* maintains a set of configuration properties that can be modified by administrators in order to customize the behavior of a microservice.
- *Folders* provides a tree-based organizational structure for managing SAS and external content.
- *Preferences* provides storage and retrieval for application-based user preferences.

SERVICE REGISTRATION AND CONFIGURATION

In a cloud environment, the address of any microservice might vary. There's no guarantee that a service will reside on the same underlying physical hardware each time the service is started. In addition, in any given deployment, the set of available microservices might be spread out across multiple machines.

Given this setup, clients cannot possibly manage these addresses on their own. Instead, this burden falls on the SAS® Configuration Server. Based on the Consul product from HashiCorp, the SAS Configuration Server acts as a service registration and configuration repository as it maintains a registry of all known microservices defined in the system. When a microservice initializes, it is registered in a central location, which enables other services and clients to dynamically discover it at run time. The service registry's responsibility is to manage this information and provide the necessary information to determine where incoming requests should be routed.

In addition to address information, this registry also stores the current status of each microservice. The SAS Configuration Server is designed to send regular health checks to all available microservices at consistent intervals. These individual health checks over time give administrators an important view into

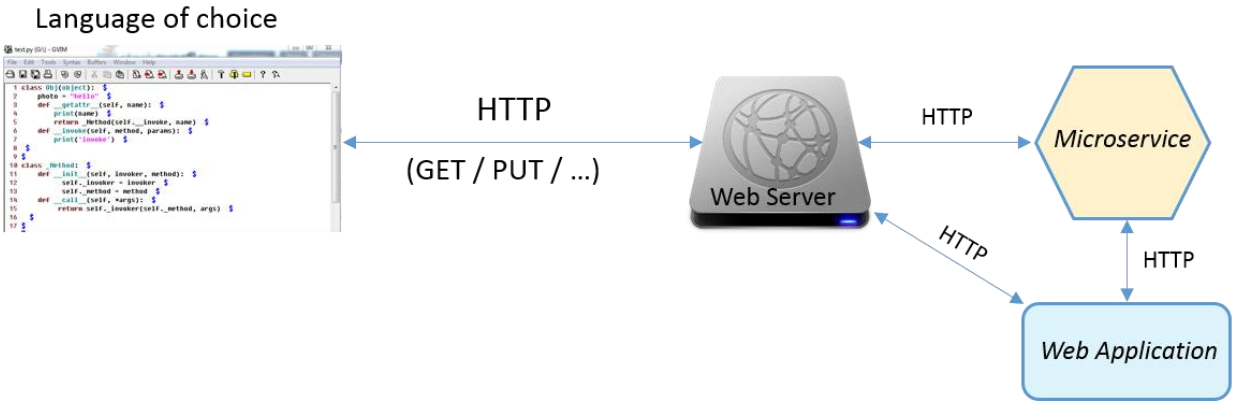
the overall state of the system. Any time a particular microservice instance becomes unavailable—whether it's because of network failure or a run-time error within the service itself—the change in state is detected by the health checks and broadcasted to the administration application, SAS® Environment Manager, which you will learn more about later.

Another key feature of this component is that it provides a central repository for storing configuration data for each microservice. Microservices are typically initialized along with their own set of default configuration settings that are persisted as key/value pairs. Many of these configuration settings can be customized as needed by administrators. Yet, by using newer technologies—including the Spring Framework and Spring Cloud projects—many of these configuration properties can be dynamically refreshed within a running microservice instance without requiring any manual restarts. As an example, suppose that the logging levels for a microservice need to be modified in order to determine the root cause of an unexpected error. Since the log levels for all microservices are managed via standard configuration properties, updating the value from INFO to DEBUG dynamically triggers a refresh within the service. From then on, subsequent requests to that service log all messages based on the updated settings. All this is accomplished without requiring any downtime from manually having to restart a service. Compare that to the SAS®9 approach, where any change to similar logging levels required an entire restart of the SAS Web Application Server.

COMMUNICATING WITH SAS VIYA MICROSERVICES

The SAS®9 architecture relies heavily on proprietary APIs. Developing a client application to interact with the SAS Metadata Server, for example, can only be done by using a collection of Java APIs, SAS macros, or DATA step functions. Writing a batch utility can involve spinning up a Java Virtual Machine instance, regardless of how simple or complex the operation is. This certainly is not very efficient or portable.

SAS Viya completely changes this model. Those proprietary APIs have all been replaced by industry-standard conventions, such as HTTP and REST. Regardless of whether the client is a web application or a batch script, communication with any SAS microservice is processed in a consistent manner. Clients needing to fetch information about the availability of a given resource on the system all use the same microservice to do so.



Display 2. HTTP Requests between Clients and Microservices

Why choose REST?

- HTTP is a standard protocol. A resource is accessible via a URI by using standard HTTP methods such as GET, PUT, POST, DELETE, and so on.
- RESTful services open up access to all clients, regardless of the language or platform involved. Client applications can be written in Java, Go, C, Python, and so on. The typical data transfer format

used by all microservices is Javascript Object Notation (JSON).

- Clients and services no longer have to stay in lock-step with one another. Services are free to version independently of clients, as long as they remain backward compatible. More on this will be discussed later, but services can even change their data model between versions without impacting clients.
- Each microservice is stateless; neither the client nor the server is required to store information across request boundaries.

Output 1 shows an example using the `curl` command line utility for submitting a request to the REST API for the SAS Identities service in order to retrieve information about the current authenticated user. The value of "TOKEN-VALUE" represents a valid OAuth2 token that enables a user to authenticate to the system.

```
$ curl http://viya.example.com/identities/users/@currentUser -X GET -H
"Accept:application/json" -H "Authorization: Bearer TOKEN-VALUE"

{
   "version": 1,
   "id": "demo_user",
   "name": "Demo User",
   "description": "User account for demos only",
   "creationTimeStamp": "2018-01-01T12:00:00.000Z",
   "modifiedTimeStamp": "2018-01-01T12:00:00.000Z",
   "providerId": "ldap",
   "state": "active",
   "links": [
      {
         "method": "GET",
         "rel": "self",
         "href": "/identities/users/demo_user",
         "uri": "/identities/users/demo_user",
         "type": "application/vnd.sas.identity.user"
      }
   ]
}
```

Output 1. Output from a Request to the Identities service

Acting as the front door processing each of these requests is the Apache HTTP Server. Each SAS Viya deployment consists by default of a single Apache HTTP Server that is capable of serving all dynamic and static content processed by each microservice in the stack. The web server's responsibility is to interact with the SAS Configuration Server in order to determine where (which machine and which microservice) to route all incoming requests to.

INTRA-SERVICE COMMUNICATION

As discussed earlier, microservices are designed to have as few dependencies on other microservices as possible. The motivation for this is to handle situations where if a particular service is down, it should not critically impact any other service or application that depends on it. While services are allowed to

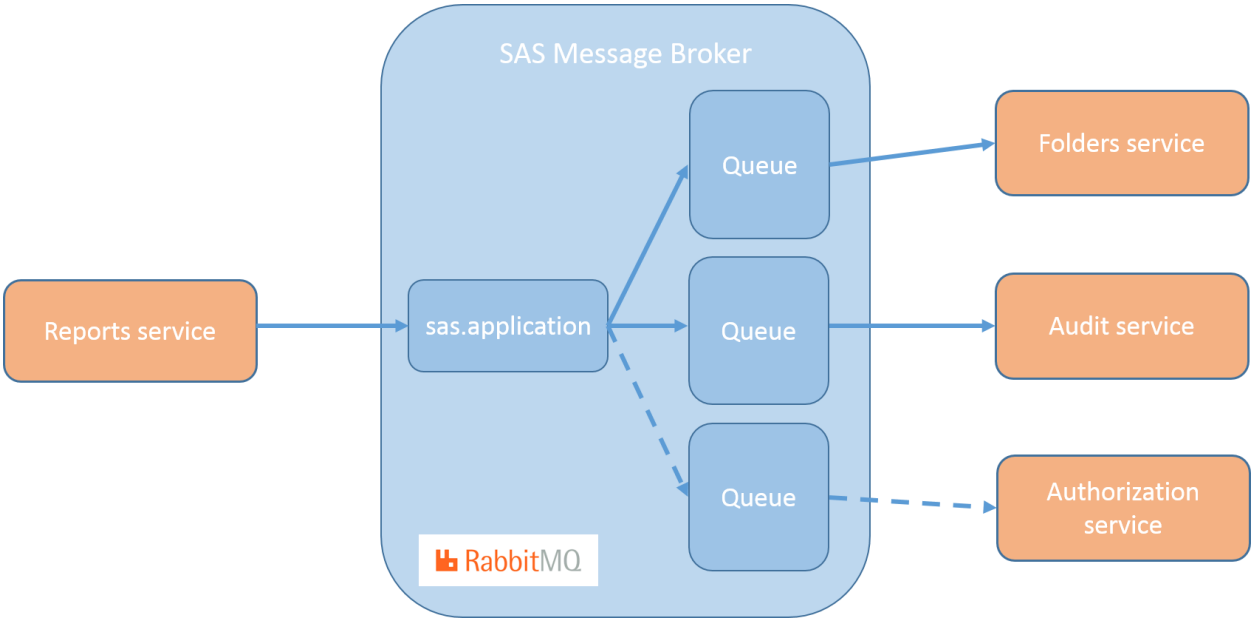
communicate with one another via standard HTTP requests, these requests are typically processed synchronously and end up tightly coupling the services together.

To facilitate the decoupling of services, SAS Viya introduces the SAS® Message Broker. The message broker—RabbitMQ—enables services to communicate with other services in an asynchronous manner through an event-driven architecture. A given service can publish messages to the message broker while other services, acting as consumers, can choose to react or ignore them. A service that publishes these messages does so in a "fire and forget" model in which they are not aware of or concerned about who is listening on the other side. Their goal is to publish a message that a particular event or action has occurred, but they do not need to worry about who the consumers of the message might be or what actions those consumers might take when the message is received.

In order to achieve this model, microservices communicate with the message broker via the Advanced Message Queuing Protocol (AMQP). Messages are published to one or more exchanges within the broker, and queues owned by various microservices listen for any incoming messages on an as-needed basis.

The *sas.application* exchange serves as the main bus within the message broker. Messages are sent to this exchange every time a resource is accessed or updated. In fact, each microservice is designed such that any HTTP request it receives automatically publishes an event indicating which resource was affected, who issued the request, what action occurred, and when the action took place. After they publish the message, their work is done. On the receiving side, however, consuming services can use a custom routing syntax to indicate which messages they need to listen to.

Display 3 illustrates what this flow looks like within some SAS microservices. The Folders service provides an organizational structure for content in which references to specific content resources are stored within folders. Any time one of these resources is updated, the Folders service must be notified in order for it to update its reference. If a report is deleted within the Reports service, the Folders service is notified of this action so that it can properly clean up any references it might have related to that report. Other services might even have their own need to listen to that same message about the reporting being deleted. For example, the Audit service must also consume these messages in order to properly record when resources are modified.



Display 3. The SAS Message Broker

A SAS Viya deployment consists of multiple exchanges that aid in delivering alert notifications and even monitoring metric information on the system. Regardless of which exchange is used to publish the message, the SAS Message Broker assists in enabling asynchronous intra-service communication while not forcing services to be coupled to others.

DATA PERSISTENCE

In a cloud environment, microservices can be started or stopped at unknown times. They must comply with the contract of being completely stateless in that nothing is shared. State information cannot be stored within a service at the risk of losing important information when a service becomes unavailable. Storing information within the file system itself isn't an option, as it's conceivable that when a service comes online, it could be located on a completely different machine. Therefore, it is imperative that services use a different approach for data persistence.

Microservices must also be designed to manage their own storage needs. SAS follows the paradigm of *polyglot persistence*, which means that each service can pick and choose its persistence model as it sees fit. Services can use the tools and components that are ideal for their usage. Technically, this means that a microservice can even change its underlying persistence model in future releases. And since services are bound to a backward-compatibility contract, any change like this would happen internally within the service without impacting any downstream consumers.

The SAS® Infrastructure Data Server serves as a central transactional storage mechanism for any microservice. The server is based on PostgreSQL, and although it is not used by every microservice in the SAS portfolio, it is used to store a wide variety of data, from authorization rules to audit records to content folders and their membership information. Data within the service is split into separate schemas per microservice, thus giving each service the necessary flexibility in how its data should be persisted.

Standing between the SAS Infrastructure Data Server and SAS microservices is the open-source software Pgpool-II. Pgpool-II acts as a proxy in order to determine how client requests are routed to the underlying database, while providing support for connection pooling, load balancing, and automatic failover. If a single server instance is brought down, it's the Pgpool-II proxy that determines how the request should be sent to the other set of remaining servers.

SCALING AND HIGH AVAILABILITY

What happens in production environments when an application becomes a bottleneck because demand has increased drastically? Or when the host machine that an application is running on experiences problems such that it's now a single point of failure? In the world of monolithic applications, scaling for performance or high availability can be achieved by replicating the entire monolith application. Given the sheer size of these applications and the resources they demand, scaling these can be an expensive task.

With microservices, however, this feat becomes much simpler. Individual services can scale independently of others on an as-needed basis. For example, if demand on the Authorization service continues to increase, scaling is as easy as deploying an additional instance of the service to another machine. After that service instance has been started successfully, it is registered within the SAS Configuration Server and becomes available for receiving requests. Simply by deploying individual services to other machines, without any additional configuration required, you now have a much more resilient environment.

In order to guard against unexpected issues, such as hardware, operating system, or network failures, it is recommended that instances be distributed across multiple machines. Deploying multiple instances of each service results in a highly available and more robust system that requires less attention in the event a failure occurs.

Now that we've discussed how to scale the individual microservices, what about the infrastructure components mentioned earlier? Most of these components have some support for high availability, and the concept is generally the same for each: Deploy multiple instances of the server as a cluster across different machines, whether it's the SAS Message Broker or the SAS Infrastructure Data Server.

We've learned already about the importance of a microservice being stateless. There are occasions, however, where cached data needs to be distributed across multiple service instances. For example, a single microservice might need to create an in-memory cache of artifacts in order to reduce the frequency that it needs to retrieve the information from a database. If this same service is deployed in a high availablility system, that same cache would need to be replicated across the other service instances. To solve that problem, SAS Viya deploys two servers, the Cache Locator and Cache Server. Both of these are microservices built on top of the Apache Geode project, where they provide a peer-to-peer or client/server model for replicating cached data. No matter which service instance processes an incoming request, it is guaranteed to access the same cached information across all instances.

ADMINISTRATION

With so many components and services involved in any given SAS Viya deployment, administrators need a consolidated view to manage and monitor the overall state of their environment as well as troubleshoot problems. The SAS Environment Manager provides this consolidated view. It offers a unified approach for customers needing to perform administrative tasks in their SAS environment. The following capabilities are supported by this application:

- providing a monitoring dashboard indicating the state and health of individual microservice instances
- managing and controlling authorization to various resources in the system, from folder content to CAS libraries
- viewing the set of identities defined on the system
- customizing the configuration settings for individual microservices

SAS Viya also provides a collection of command-line interfaces (CLIs) to perform many of these same administrative tasks. These utilities enable users to perform administrative functions without requiring access to a web-based application such as SAS Environment Manager. Each command is accessible via the sas-admin CLI as individual plugins, including the following:

- sas-audit
- sas-authorization
- sas-folders
- sas-identities

Similar to the REST API example earlier (Output 1), Output 2 below contains an example of how to use the sas-identities CLI in order to display information about the current authenticated user.

```
$ ./sas-admin identities whoami

Id                  demo_user
Name                Demo User
Title               Sales executive
EmailAddresses      []
PhoneNumbers        []
```

```
Addresses           []
State               active
ProviderId          ldap
CreationTimeStamp   2018-01-01T12:00:00.000Z
ModifiedTimeStamp   2018-01-01T12:00:00.000Z
```

Output 2. sas-identities CLI output

CONCLUSION

The architecture of SAS Viya is a drastic change compared to the monolithic approach used in SAS®9. By leveraging the microservices architectural pattern, the SAS Viya service layer becomes a more scalable and flexible system, one that can be deployed in any type of environment, including modern cloud-based systems. Newer technologies and design patterns help position the service layer to be successful for future generations of SAS solutions.

REFERENCES

SAS Institute Inc. 2018. *SAS® Viya® 3.3 Administration*. Cary, NC: SAS Institute Inc. Available: http://support.sas.com/documentation/onlinedoc/viya/index.html.

SAS Institute Inc. 2018. *SAS® Viya® 3.3 for Linux: Deployment Guide*. Cary, NC: SAS Institute Inc. Available: http://support.sas.com/documentation/onlinedoc/viya/index.html.

Wiggins, Adam. "The Twelve-Factor App." Available: https://12factor.net/. Last modified 2017. Accessed February 10, 2018.

ACKNOWLEDGMENTS

The author would like to thank Tony Dean and Joel Byrd from SAS Institute Inc. for the valuable comments they had to offer on this paper.

RECOMMENDED READING

- *Hamrick, Danny. 2018. "Command-Line Administration in SAS® Viya®." Proceedings of the SAS Global Forum 2018 Conference. Cary, NC: SAS Institute Inc.*

CONTACT INFORMATION

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

Eric Bourn
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
(919) 677-8000
Eric.Bourn@sas.com