Paper SAS3393-2019

Exploring the SAS® Viya® Operations Infrastructure

Bryan Ellington, SAS Institute Inc.

ABSTRACT

SAS® Viya® includes a new event-driven operations infrastructure for logs, metrics, and notifications. This paper explores the components, flows, and capabilities that give you powerful new insight into the operation of a SAS Viya deployment. With a particular focus on command-line tools, the paper teaches you how to view consolidated log and metric flows, check the status of services, and validate the deployment and its components. Various third-party and enterprise integration scenarios are also explored.

INTRODUCTION

The release of SAS Viya provided an opportunity to rethink and reimagine the operational architecture of a SAS deployment. The new approach provides a set of focused, decoupled components that function similarly to UNIX commands and that enable new levels of power and flexibility for administrators. This set of technologies and components is collectively referred to as the SAS Viya operations infrastructure.

This paper serves as an introduction and overview of SAS Viya operations infrastructure, including the architecture, components, and flows. Because the focus of this paper is on the underlying infrastructure, command-line utilities, and extensibility integration, the implementation of the operations infrastructure in SAS® Environment Manager is not covered in any detail.

You will gain a working knowledge of how the various services, tools, and technologies provide administrators deep visibility into the operational aspects of a SAS Viya deployment, including health, logs, metrics, notifications, and other events.

ARCHITECTURE

Goals

The SAS Viya operations infrastructure was designed with these goals in mind, all of which were improvements over SAS 9.4:

- **Minimal dependencies** – Monitoring the environment cannot depend on the environment itself, so components are designed “from the outside looking in.”
- **Discrete, composable components** – Individual components should be small, focused, and with clean interfaces that allow simple flows and composition.
- **Extensibility and integration** – Although a complete operational solution is included with SAS Viya, it is straightforward to integrate with existing enterprise systems and common industry tools and technologies.
- **Highly available** – The design is robust, scalable, and tolerant of point failures.
- **Cloud-friendly** – From streaming of logs and metrics to support of a dynamic service registry and multi-tenancy, the operations infrastructure is designed to support cloud deployments.

Event-Driven Architecture

The SAS Viya operations infrastructure, along with several other components of SAS Viya, is designed around an event-driven architecture. An **event** in this context is simply an

indication that something potentially interesting happened in the environment. These are some examples of events:

- a log message
- a service health check result
- a set of system metrics
- a user logoff
- the start or end of a scheduled job

An event-driven architecture logically separates, or decouples, **producers** of events from **consumers** of events. Events are **published** by producers and consumers **subscribe** to a subset of events that the consumers handle. The key component in the middle is the **message broker, which accepts** published events and sends them to the appropriate subscribers. If events are produced faster than they can be consumed, the events are queued on a per-subscriber basis.

In SAS Viya, events are published in a standardized JSON format. Although they share common wrapper fields, each event type (such as log, metric, notification, or resource), which is referred to as a 'payload,' has a unique specification that is customized for its purpose.

Nearly every service is an event producer. RabbitMQ serves as the message broker. There are few visible services that function primarily as consumers, although many services use events for internal communication and synchronization (such as when a tenant is on-boarded or a user logs out).

In the SAS Viya operations infrastructure, components collect, normalize, and publish logs, metrics, and notifications from all machines in a SAS Viya deployment. These events, along with others produced by the platform at large, are consumed, persisted into a data mart, and loaded into Cloud Analytic Services (CAS) for use by SAS Environment Manager or for direct exploration by administrators.

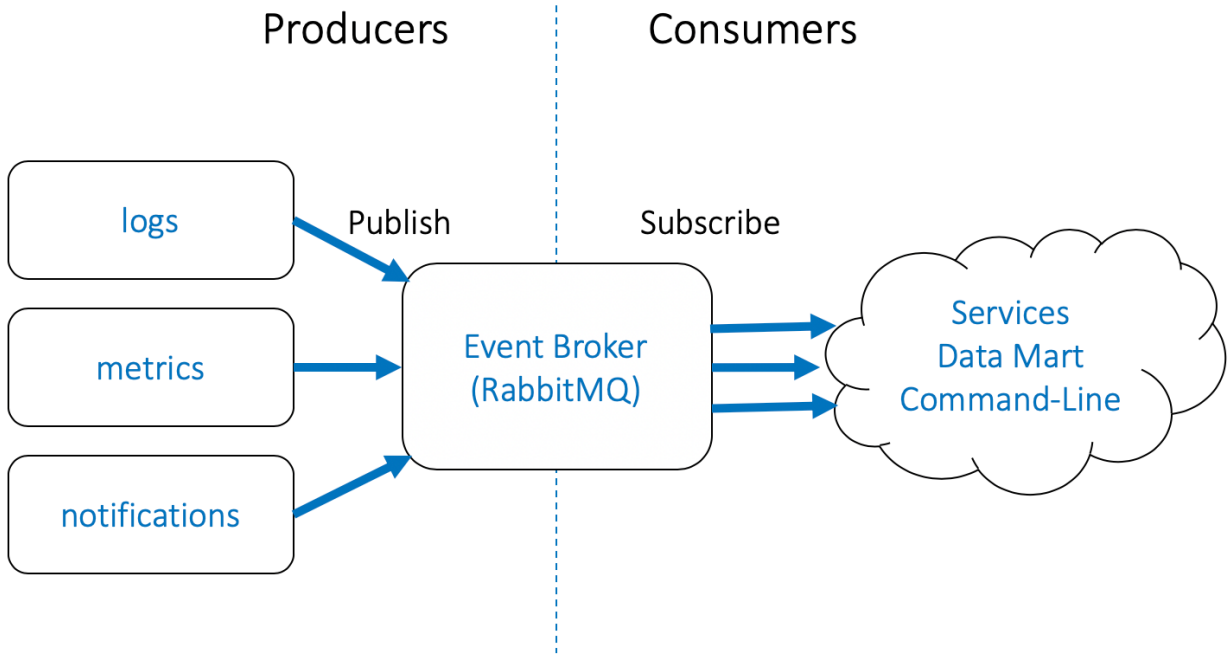

Figure 1. SAS Viya event-driven architecture

The components of the SAS Viya operations infrastructure were designed to have minimal dependencies on the rest of the platform. This allows the components to effectively monitor and validate the environment even when it is not completely healthy. In some cases (for example, system metric collection), a component does not have dependencies on other components, but does depend on the availability of Consul for its key/value store and service registry and RabbitMQ as the message broker.

COMPONENTS

SAS-OPS-AGENT

The sas-ops-agent is the operations agent deployed to each machine in a SAS Viya deployment. The `sas-ops-agent` command runs as a service and manages the execution a set of periodic tasks, many of which publish metric or notification events. It would take an entire paper to properly cover even a fraction of the capabilities of the sas-ops-agent in detail. For now, it's important to know that it manages a configurable set of tasks that perform functions such as these:

- Metric collection – usually performed once every 60 seconds, with one task per resource type
- System checks – checks for low memory, high disk space usage, and so on

The sas-ops-agent is also responsible for publishing events produced by the tasks that it executes. This capability allows the tasks to remain simple and have a single purpose. For example, metric collectors are required only to produce valid JSON on stdout, and they do not need to know how to create events or to connect and publish to RabbitMQ.

SAS-PEEK

The `sas-peek` command is a metric collection utility. It finds and queries various components of SAS Viya and produces JSON output containing the metric data. The metric data it produces includes metrics for the host system (such as CPU, memory, network, filesystems, and I/O), processes, CAS servers, SAS microservices, RabbitMQ, and Postgres.

Because the `sas-peek` command runs on every machine in a deployment, by default only local resources are reported to avoid duplication of metrics. Multiple levels of detail are supported for each type of metric collected. Although the `sas-peek` command is normally executed by the `sas-ops-agent` command, it can be run manually, as in this example:

```
$ sas-peek network --level 1 --format pretty
{
  "version": 1,
  "collectorName": "sas-peek-network",
  "collectorVersion": "1.4.59",
  "timeStamp": "2019-02-20T14:50:38.545856-05:00",
  "properties": {
    "consulNodeName": "ptnode20.ptest.sas.com",
    "hostname": "ptnode20.ptest.sas.com",
    "os": "linux_amd64"
  },
  "measurements": [
    {
      "resourceType": "system_network_interface",
      "resourceId": "FXb33IIfNYjjK2ZDMnEtsA==",
      "properties": {
        "address": "10.122.32.70",
        "broadcast": "10.122.35.255",
```

```
        "interfaceFlags": "up|broadcast|multicast",
        "interfaceName": "eth0",
        "linklocal6": "fe80::1ec1:deff:fe1d:4226",
        "mac": "1c:c1:de:1d:42:26",
        "mtu": "1500",
        "netmask": "fffffc00",
        "prefix6": "0",
        "resourceName": "ptnode20.ptest.sas.com|eth0"
      },
      "metrics": [
        {
          "name": "receiveBytes",
          "unit": "B",
          "type": "counter",
          "detailLevel": 1,
          "value": 8973264278618
        },
        {
          "name": "receivePackets",
          "unit": "count",
          "type": "counter",
          "detailLevel": 1,
          "value": 16443622557
        },
        {
          "name": "transmitBytes",
          "unit": "B",
          "type": "counter",
          "detailLevel": 1,
          "value": 6169640661219
        },
        {
          "name": "transmitPackets",
          "unit": "count",
          "type": "counter",
          "detailLevel": 1,
          "value": 15205215799
      ]
    }
```


SAS-WATCH

Sas-watch is a service that monitors a directory tree of `.log` files for changes, parses each line, and publishes structured log events to RabbitMQ. The `sas-watch` command runs on each machine and enables administrators to access a consolidated flow of logs from an entire SAS Viya deployment.

SAS-STREAM

The `sas-stream` command is a service that writes event data to the operations data mart. The service consumes most events that flow through a deployment and writes the data in tabular form in near real time to the data mart.

DATA MART EXTRACT, TRANSFORM, LOAD (ETL)

Periodically, data written by the `sas-stream` command is processed by a SAS ETL process. This process calculates derived data (for example, CPU % from raw counters), loads and

indexes logs into the CAS search facility, and loads metric, notifications, resource, and other events into CAS tables.

FLOWS

Using the components described in the previous topics, we can now fill in the generic event-driven architecture diagram from Figure 1 with SAS Viya operations infrastructure components.

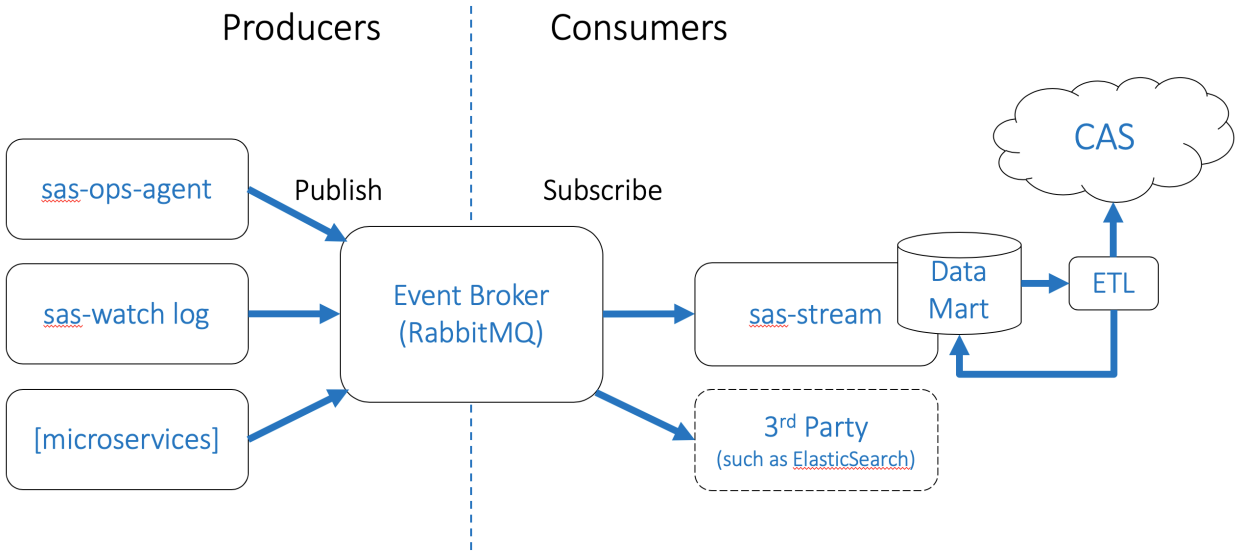

Figure 2. SAS Viya Operations Infrastructure Event Flows

SAS-OPS COMMAND-LINE INTERFACE

The `sas-ops` command is the primary command-line interface for the SAS Viya operations infrastructure. The `sas-ops` command is generally run from a terminal session connected to any machine in a SAS Viya deployment, although you can also run it remotely. In order to run the `sas-ops` command, you must either be `root` or the SAS install user, because the command requires access to some key files that are readable only by those users.

Covering each subcommand of the `sas-ops` command in detail is out of the scope of this paper, but the following topics describe a few key subcommands that are particularly useful to many administrators.

validate

The `validate` subcommand performs a detailed series of checks on a SAS Viya deployment and summarizes the results. Because the checks are deployment-wide, in most cases you can run the subcommand from any machine in a deployment and produce nearly identical results. This is an example of the validate subcommand.

```
$ ./sas-ops validate --level 3
Mon, 25 Feb 2019 11:01:13 EST - Validating deployment on
emishipped18w30.opsmonitor.sashq-d.openstack.sas.com...
  Level 1 - [consul] Verify Consul connectivity...
  Level 1 - [rabbit] Verify RabbitMQ exchanges...
  Level 2 - [consul] Verify Consul services...
  Level 2 - [datamart] Verify operations datamart ETL status...
  Level 2 - [local] Verify local system...
  Level 2 - [ops] Verify status of operations services...
```

```
  Level 2 - [tls] Verify TLS...
  Level 3 - [cas] Verify status of CAS servers...
  Level 3 - [http] Verify HTTP connectivity...
  Level 3 - [http] Verify OAuth...

Validation test(s) completed with 1 warnings(s) and 1 errors(s):
  WARN  evdm step audit status is [OK with warnings]
  ERROR evdm step resetdm_cas status is [Error]

Use --verbose for additional details
```

logs

You can use the `logs` subcommand to stream logs from a SAS Viya deployment to the terminal. It is similar in concept to the `docker-compose logs` command with which many administrators are familiar. Various filters (such as level, source, and message text) are supported, as well as color output and line formatting. Here is a limited example of the subcommand.

```
$ ./sas-ops logs
Listening for logs...CTRL+C to quit
INFO  2019-02-20 15:08:18.000 [sasdatasvrc] - child process with pid: 31587
exits with status 256 [pid:9327]
INFO  2019-02-20 15:08:18.000 [sasdatasvrc] - fork a new child process with
pid: 25705 [pid:9327]
INFO  2019-02-20 15:08:30.551 [reportalertseval] - service Iteration 11301
starts 0 evaluation task(s).
[classname:c.s.r.a.e.evaluation.EvaluationIterator pid:7145
threadname:Thread-16]
NONE  2019-02-20 15:08:45.000 [sasstudio] - GET /SASStudio/health HTTP/1.1
[size:53 status:200 url:/SASStudio/health user:- method:GET
protocol:HTTP/1.1 remote:10.122.32.70]
INFO  2019-02-20 15:08:55.692 [themes] - sasboot@@provider(739ee42f)
[56ad23432f0aeddf] [LOADING] Loading theme sas_corporate
[classname:c.s.themedesigner.PETCustomThemeService pid:15717 threadname:o-
auto-1-exec-4]
INFO  2019-02-20 15:08:55.707 [drive] - sasboot@@provider(739ee42f)
[2e72c95a261baa4d] SASDrive ApplicationSwitcher enabled=true
[classname:c.s.commons.html.taglib.ApplicationTag pid:9162 threadname:o-
auto-1-exec-7]
INFO  2019-02-20 15:08:55.725 [drive] - sasboot@@provider(739ee42f)
[2e72c95a261baa4d] SASDrive ApplicationSwitcher enabled=true
[classname:c.s.commons.html.taglib.ApplicationTag pid:9162 threadname:o-
auto-1-exec-7]
INFO  2019-02-20 15:09:00.643 [themes] - sasboot@@provider(739ee42f)
[1623aa651a994edb] [LOADING] Loading theme sas_corporate
[classname:c.s.themedesigner.PETCustomThemeService pid:15717 threadname:-
auto-1-exec-10]
INFO  2019-02-20 15:09:00.653 [drive] - sasboot@@provider(739ee42f)
[e72e688eb628a6ec] SASDrive ApplicationSwitcher enabled=true
[classname:c.s.commons.html.taglib.ApplicationTag pid:9162 threadname:o-
auto-1-exec-3]
INFO  2019-02-20 15:09:00.654 [drive] - sasboot@@provider(739ee42f)
[e72e688eb628a6ec] SASDrive ApplicationSwitcher enabled=true [threadname:o-
auto-1-exec-3 classname:c.s.commons.html.taglib.ApplicationTag pid:9162]
INFO  2019-02-20 15:09:02.000 [consul] - Synced check "service:postgres-
datanode0" [command:agent]
```

```
INFO  2019-02-20 15:09:05.189 [collections] - sasboot@@provider(739ee42f)
[44de500aba9c47fc] HHH000397: Using ASTQueryTranslatorFactory [pid:14363
threadname:o-auto-1-exec-3
classname:o.h.h.i.QueryTranslatorFactoryInitiator]
INFO  2019-02-20 15:09:11.188 [spawner] - The Bridge Protocol Engine Socket
Access Method lost contact with a peer (11482) during protocol recognition.
[user:sas index:00011357]
```

metrics

The `metrics` subcommand, streams events containing metric data to the terminal. The subcommand supports filtering by peek. Here is an example that displays system metrics in the default 'line' format.

```
./sas-ops metrics --key metric.sas-peek-system
Listening for metrics...CTRL+C to quit
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] totalCpu=3.03731631e+09 ms
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] userCpu=3.7259387e+08 ms
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] systemCpu=1.1840616e+08 ms
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] idleCpu=2.51775964e+09 ms
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] actualFreeMemory=78129 MB
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] usedMemory=85220 MB
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] freeMemory=43844 MB
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] actualUsedMemory=50935 MB
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] totalMemory=129063 MB
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] freeSwap=0 MB
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] uptime=385791 s
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] loadAverage1=0.61
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] ioWaitCpu=646040 ms
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] stolenCpu=2.382204e+07 ms
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] contextSwitches=6.862545502e+09
count
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] openFiles=17408 count
2019-02-24T22:29:00.079-05:00 [system:ops.sas.com] maximumOpenFiles=1.3134946e+07
count
```

notify

The `notify` subcommand sends out a simple message notification event. The `--level` argument controls the level of the notification (info, warn, or alert). Alert-level notifications are viewable by the `sas-ops alerts --last 10` command.

alerts

The `alerts` subcommand streams alert notification to the terminal. You can use the `--last [n]` argument to display a history of recent alerts (the default is 30). For example, this command displays the last three alerts.

```
$ ./sas-ops alerts --last 3
ALERT 2019-02-25T10:03:00.801-05:00 [ops] [ops.sas.com] Death Star has cleared the
planet
ALERT 2019-02-25T09:58:00.755-05:00 [ops] [ops.sas.com] Death Star will be in range
in 5 minutes
ALERT 2019-02-25T09:53:00.289-05:00 [ops] [ops.sas.com] Stand-by alert. Death Star
approaching. Estimated time to firing range, fifteen minutes
```

OPERATIONS COMMAND-LINE TIPS

You are encouraged to explore the `sas-ops` command-line interface, but some of its capabilities might not be obvious. Here are some tips that you might find useful under the right circumstances.

SEARCHING LOGS IN REAL-TIME

You can use the `sas-ops logs` command to search through logs in near real time. The more that you know about the source of the log, the more efficient the search is. For example, you can run this command to see error log messages from CAS that contain a particular user ID (`dvader` in this case):

```
./sas-ops logs --level error –source cas --match dvader
```

VIEWING SYSTEM INFORMATION

You can use the `sas-ops info` command to view some system details such as operating system, kernel version, user limits, and installed packages. By default, the output will be quite long, because the command reports data for every machine in the SAS Viya deployment. To limit the output, simply add a sub-path to the end that corresponds to the indented headers that label each section of the base output. For example, this command displays the the base properties of a machine (ops.sas.com in this case).

```
./sas-ops info ops.sas.com/common
common
   architecture : amd64
   boot-time : 2017-09-28T17:39:47.000000-04:00
   hostname-long : ops.sas.com
   hostname-short : ops
   ip-addrs : 10.1.2.34
   last-update : 2019-02-24T02:42:50.287681-05:00
   memory-total : 101172146176
   operating-system : linux
   timezone : EST
   timezone-offset : -05:00
```

THIRD-PARTY INTEGRATION

An entire paper could be written on third-party integration with the SAS Viya operations infrastructure, but these topics offer some ideas for integrating with external systems.

sas-ops

The command `sas-ops [logs|metrics|notifications] --format event` provides access to the stream of events coming from RabbitMQ. Events are sent to stdout in JSON format. You could then pipe the events to a query tool such as `jq` or a custom transformation tool to send metrics to a system such as Prometheus.

Direct to RabbitMQ

You can also connect to RabbitMQ and directly consume events using third-party software such as LogStash feeding ElasticSearch. A particular benefit is that such a system simply receives a copy of each event. Because of this, there is no reconfiguration needed on the SAS Viya deployment and no effect other that the minimal cost of sending copies.

CONCLUSION

The SAS Viya operations infrastructure is a large step forward in providing efficient, flexible, forward-thinking capabilities to administrators and operators. SAS Viya operations infrastructure works out of the box as well integrated into existing enterprise systems. This

paper provided a basic introduction and high-level overview, but there is much more to discover, with even more great capabilities in the pipeline.

RECOMMENDED READING

SAS Institute Inc. 2018. *SAS® Viya® 3.4 Administration: Operations Infrastructure.* Cary, NC: SAS Institute Inc.
https://go.documentation.sas.com/?cdcId=calcdc&cdcVersion=3.4&docsetId=calopsinf&docsetTarget=titlepage.htm&locale=en

CONTACT INFORMATION

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

Bryan Ellington
SAS Institute Inc.
Bryan.Ellington@sas.com