Paper SAS4214 -2020

SAS® Viya® Monitoring Using Open-Source Tools

Bryan Ellington, SAS Institute Inc.

ABSTRACT

Did you know that SAS® Viya® has an event-driven infrastructure that gives you access to a continuous flow of logs, metrics, and other activity? This paper discusses the event-driven architecture of SAS Viya and demonstrates how you can leverage it to send logs, metrics, and events to leading open-source tools. See how to export metrics to Prometheus and set up custom alerts that are triggered when specified thresholds are met or exceeded. Understand how to use Grafana to visualize metrics on dashboards that are customized to your needs. Learn how to send logs and events to Elasticsearch, and how to efficiently filter, search, and report on this data using Kibana.

INTRODUCTION

This paper builds on the introduction of the SAS Viya operations infrastructure in the paper *Exploring the SAS® Viya Operations Infrastructure*. The event and payload formats are described here in depth. In addition, this paper describes how metric and log events can be used to integrate with best-of-breed open-source monitoring and log aggregation projects such as:

- Prometheus - A metric collector, time-series database, and alerting system with a robust query language that has become a de facto standard for Kubernetes monitoring
- Grafana - Browser-based application that supports Prometheus with dashboards for metric visualization
- Logstash - A log processing pipeline that can push data to Elasticsearch
- Elasticsearch - Indexed document repository with powerful search and filtering capabilities
- Kibana - Provides dashboards and visualization for data in Elasticsearch

EVENT-DRIVEN ARCHITECTURE

The monitoring and logging infrastructure of SAS Viya is designed around an event-driven architecture. Events indicate that something potentially interesting happened in the environment. For example:

- a service writes a log message
- metric collection for a resource completes
- an admin sends a notification

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 JSON format. All events share a common wrapper format, while each event type (such as log, metric, or notification), defines its own JSON 'payload,' which is a specification very similar to a REST API media type.

For a more thorough description of the event-driven architecture and related tooling in SAS Viya, see the paper *Exploring the SAS® Viya Operations Infrastructure.*

MONITORING

Monitoring in SAS Viya uses a push model. In this model, metrics are collected locally on a schedule, then pushed out as events. This approach allows backend systems to easily collect all metrics by simply connecting to the message broker and listening for these events.

Each machine in a SAS Viya deployment runs an instance of the `sas-ops-agent` process, which serves as a task scheduler. Many of the recurring tasks run the `sas-peek` process to discover local resources, gather metrics, and provide output (using the JSON format for the metric payload described later in this paper.) The `sas-ops-agent` program takes this output, wraps it into an event, and publishes it to the message broker (the `sas.metric exchange` on RabbitMQ). At this point, all subscribers each receive their own copy of the event.

LOG AGGREGATION

Log aggregation is a natural fit for an event-driven architecture. In SAS Viya, .log files in the [*sasConfigRoot*]/var/log directory are actively watched (similar to the `tail` command) by the `sas-watch` process. When each line is appended to a watched file, it is parsed into the log payload JSON file described later. That payload file is then wrapped into an event and published to the message broker (the `sas.log exchange` on RabbitMQ).

EVENT FORMATS

EVENT WRAPPER

All events published by SAS Viya components use a common event wrapper that includes some key data about the event itself as well as information about the embedded payload data.

An abbreviated event (with the payload removed) looks like this:

```
{
   "version": 2,
   "id": "625d61a3-d378-43e9-a50c-27521357a270",
   "type": "metric",
   "payloadType": "application/vnd.sas.event.metric;version=1",
   "user": "sas.ops-agent",
   "headers": {
      "__tenant": "provider",
      "routing-key": "metric.sas-peek-system",
      "sas-content-type": "application/vnd.sas.event",
      "sas-event-source": "sas-event-pub",
      "sas-published-timestamp": "2020-02-19T08:10:00.668260-05:00"
   },
   "payload": {…},
   "timeStamp": "2020-02-19T08:10:00.065789-05:00"
}
```

Field	Type	Description
version	integer	Version number of the event wrapper. This is always 2 in SAS Viya 3.3 or later
id		A generated GUID that uniquely identifies this event.
type		The short payload type (such as “log”, ”metric”, “notification”, ”security”, or ”resource”)
payloadType		The full media type of the payload, Such as `application/vnd.sas.event.metric;version=1`

timeStamp	string	Time the payload was created. The format must be of the form:
		`yyyy-mm-ddThh:mm:ss.SSSSSS-[hh]:[mm]`
		For example:
		`2017-06-05T15:24:43.508084-04:00`

user	string	**Optional** - The identity of the entity that created the payload.
headers	map string:string	Event headers Not all headers are available for every event. `__tenant` - Identifies the tenant associated with this notification. A value of ”provider” is used if it is generated by the system or there is no tenant. `routing-key` - The routing key for this event. `sas-content-type` - The full event media type, including version. The value is `application/vnd.sas.event;version=2` for SAS Viya 3.3 or later. `sas-event-source` - The component or service that generated the event (such as ”folders”, ”comments”, ”sasstudio”, and so on) `sas-published-timestamp` - Indicates when the event was published. This timestamp might differ from the timestamp when it was generated. However, it must never be earlier. The format must be of the form: `2017-06-05T15:24:43.508084-04:00`
payload	custom	The event payload (see below)

Table 1 - SAS Viya Event Wrapper

METRIC EVENT

A full metric event published by the `sas-peek system` command is included here for reference:

```
{
   "version": 2,
   "id": "625d61a3-d378-43e9-a50c-27521357a270",
   "type": "metric",
   "payloadType": "application/vnd.sas.event.metric;version=1",
```

```
   "user": "sas.ops-agent",
   "headers": {
      "__tenant": "provider",
      "routing-key": "metric.sas-peek-system",
      "sas-content-type": "application/vnd.sas.event",
      "sas-event-source": "sas-event-pub",
      "sas-published-timestamp": "2020-02-19T08:10:00.668260-05:00"
   },
   "payload": {
      "version": 1,
      "collectorName": "sas-peek-system",
      "collectorVersion": "1.5.19+a510145",
      "properties": {
         "consulNodeName": "ops.sas.com",
         "hostname": "ops.sas.com",
         "os": "linux_amd64"
      },
      "measurements": [
         {
            "resourceType": "system",
            "resourceId": "j2y9Xi/Ime2SCOEONdzuVA==",
            "properties": {
               "cpucount": "24",
               "resourceName": "ops.sas.com",
               "uname": "Linux ops 3.10.0-514.el7.x86_64 #1 SMP Wed Oct 19
11:24:13 EDT 2016 x86_64 (none)"
            },
            "metrics": [
               {
                  "name": "totalCpu",
                  "unit": "ms",
                  "type": "counter",
                  "detailLevel": 1,
                  "value": 1802332229000
               },
               {
                  "name": "userCpu",
                  "unit": "ms",
                  "type": "counter",
                  "detailLevel": 2,
                  "value": 60593052040
               },
               {
                  "name": "systemCpu",
                  "unit": "ms",
                  "type": "counter",
                  "detailLevel": 2,
                  "value": 28030486090
               },
               {
                  "name": "idleCpu",
                  "unit": "ms",
                  "type": "counter",
                  "detailLevel": 2,
                  "value": 1712990923770
               },
               {
                  "name": "actualFreeMemory",
```

```
      "unit": "MB",
      "type": "gauge",
      "detailLevel": 1,
      "value": 42739
   },
   {
      "name": "usedMemory",
      "unit": "MB",
      "type": "gauge",
      "detailLevel": 2,
      "value": 85820
   },
   {
      "name": "freeMemory",
      "unit": "MB",
      "type": "gauge",
      "detailLevel": 2,
      "value": 10665
   },
   {
      "name": "actualUsedMemory",
      "unit": "MB",
      "type": "gauge",
      "detailLevel": 2,
      "value": 53746
   },
   {
      "name": "totalMemory",
      "unit": "MB",
      "type": "gauge",
      "detailLevel": 2,
      "value": 96485
   },
   {
      "name": "freeSwap",
      "unit": "MB",
      "type": "gauge",
      "detailLevel": 2,
      "value": 4063
   },
   {
      "name": "uptime",
      "unit": "s",
      "type": "counter",
      "detailLevel": 1,
      "value": 75483057
   },
   {
      "name": "loadAverage1",
      "unit": "none",
      "type": "gauge",
      "detailLevel": 1,
      "value": 2.54
   },
   {
      "name": "ioWaitCpu",
      "unit": "ms",
      "type": "counter",
```

```
                    "detailLevel": 2,
                    "value": 162469850
                },
                {
                    "name": "stolenCpu",
                    "unit": "ms",
                    "type": "counter",
                    "detailLevel": 2,
                    "value": 0
                },
                {
                    "name": "contextSwitches",
                    "unit": "count",
                    "type": "counter",
                    "detailLevel": 2,
                    "value": 1808713304515
                },
                {
                    "name": "openFiles",
                    "unit": "count",
                    "type": "counter",
                    "detailLevel": 2,
                    "value": 16992
                },
                {
                    "name": "maximumOpenFiles",
                    "unit": "count",
                    "type": "counter",
                    "detailLevel": 2,
                    "value": 9780545
                }
            ]
        }
    ],
    "timeStamp": "2020-02-19T08:10:00.065789-05:00"
},
"timeStamp": "2020-02-19T08:10:00.668254-05:00"
}
```


These are the metric payload fields for the previous example.

Field	Type	Description
version	integer	Version number of this payload format. This value is always 1 for SAS Viya 3.x

timeStamp	string	The time the payload was created. The format must be of the form:
		`yyyy-mm-ddThh:mm:ss.SSSSSS-[hh]:[mm]`
		For example:
		`2017-06-05T15:24:43.508084-04:00`

collectorName	string	The name of the collector that produced this metric payload.
collectorVersion	string	The semantic version number of the collector that produced this metric payload.

Field	Type	Description
properties	map string:string	This optional field is an open set of key/value pairs that specify additional details about the payload. Keys must be 32 characters or less and conform to these identifier restrictions: `^[_a-zA-Z][_a-zA-Z0-9\-]*$` Keys should use camelCase naming. Values are always strings. Properties might be helpful, but they are optional. Consumers should not fail if a desired property is not present.
measurements	array of measurement	Each measurement describes a single measured resource and its associated metrics.

Table 2. **Metric Event Payload – Top Level**

Field	Type	Description
resourceType		The type of resource for which metric data is being reported. The combination of resourceType and resourceId uniquely identify a resource within a deployment.
resourceId		The identifier of resource instance for which metric data is being reported. The value is an opaque string that should not be parsed. Use properties to distinguish instances in a human-readable way.
properties	map string:string	Key/value pairs that provide additional details about the payload, as described in Table 2
metrics	array of metric	Each metric describes one measurement of the resource.

Table 3. **Metric Event Payload - Measurement Object**

Field	Type	Description
name		The version number of this payload format. This value is always 1 for SAS Viya 3.x
type		Identifies the type of metric. If omitted a type of "gauge" is used. • `gauge` - A value that is absolute and may go up or down: free disk space, total memory, per second, percent, etc. • `counter` - A 64-bit integer value that always increases. Multiple values with a timespan are needed to make sense of the value to obtain value per unit time: total HTTP requests, total writes, bytes transferred, etc.

Field	Type	Description
unit	string	The unit of measure for the metric.
detailLevel	integer	The detail level of the metric: • 0 - None: Although this is not used in JSON, it can specify that a collector should not collect any metrics (to perform resource detection only, for example.) • 1 - Indicator: Specifies a common metric for a resource that best summarizes its performance or health. • 2 - Normal: Specifies a standard metric generally collected by default. • 3 - Fine: Specifies a detailed metric that is not normally collected by default, because it is too expensive to collect, it is not as well understood, or it is too detailed to collect by default.
value	float64	The value of the metric
properties	map string:string	Key/value pairs that provide additional details about the payload, as described in Table 2

Table 4. Metric Event Payload - Metric Object

LOG EVENTS

A full log event is included here for reference:


```
{
   "version": 2,
   "id": "b2e94cfb-01c1-4788-89c9-1f6ca061280c",
   "type": "log",
   "payloadType": "application/vnd.sas.event.log",
   "user": "sas.watch-log",
   "headers": {
      "__tenant": "provider",
      "routing-key": "log.cas.info",
      "sas-content-type": "application/vnd.sas.event",
      "sas-event-source": "sas-watch",
      "sas-published-timestamp": "2020-02-19T08:17:19.085995-05:00"
   },
   "payload": {
      "version": 1,
      "level": "info",
      "source": "cas",
      "messageParameters": {
         "index": "05017999"
      },
      "message": "MAIN sas.appRegistry 505410 [casgeneral.c:4823] -
Launched session controller. Process ID is 27999.",
      "properties": {
         "consulNodeName": "ptnode21.ptest.sas.com",
         "directory": "/home/opt/sas/viya/config/var/log",
```

```
            "hostname": "ptnode21.ptest.sas.com",
            "rawType": "SAS Server"
         },
         "timeStamp": "2020-02-19T08:17:19.083000-05:00"
      },
      "timeStamp": "2020-02-19T08:17:19.085974-05:00"
   }
```

5 describes the log payload.

Field	Type	Description
version	integer	Version number of this payload format. This value is always 1 for SAS Viya 3.x.

timeStamp	dateTime	The time the payload was created. The format must be of the form:
		`yyyy-mm-ddThh:mm:ss.SSSSSS-[hh]:[mm]`
		For example:
		`2017-06-05T15:24:43.508084-04:00`

level	string	The level of the log message. Must be one of these values: `trace`, `debug`, `info`, `warn`, `error`, `fatal`, or `none`.
source	string	The source of the log message. This is usually the short service name (`folders`, for example).
messageKey	string	This optional field is a language-independent message identifier. This field is not used in SAS Viya 3.x.
messageParameters	map string:string	This optional field is a string:string map of variable fields in a log message. The fields provided are highly dependent on both the source and the log message.
message	string	The text of the log message.
properties	map string:string	Key/value pairs that provide additional details about the payload, as described in Table 2 These are examples of meta content. These keys are examples, so these specific keys are not required or present in every payload. • thread • hostname • consulNodeName

Table 5. Log Event Payload Specification

PROMETHEUS AND GRAFANA

Prometheus is based on a pull model for metric collection, which means that it actively queries a list of targets. The targets may be statically configured, dynamically discovered, or a mix of both.

Although there is no direct way to push metrics to Prometheus, there is a solution available, called the Prometheus Pushgateway. The Pushgateway is a long-running service that accepts metric pushes, caches them, and responds to scrape requests from the Prometheus

server. The primary use case for the Pushgateway is for metric collection for jobs or scripts that execute and exit and that cannot respond to scrape requests.

For the purposes of this paper, metrics events are converted to Prometheus metrics and sent to the Prometheus Pushgateway as they are received from the message broker.

RECEIVING AND TRANSFORMING METRICS

There are several ways to consume metric events including these commands:

- `sas-event-sub --exchange sas.metric`
- `sas-ops metrics --format event` (returns the full event)
- `sas-ops metrics --format pretty` (returns just the metric payload)
- Write code in your favorite language to connect to RabbitMQ

Any of these methods will return events as JSON strings. You must then convert the metric payload into a format compatible with Prometheus. The transcoding has two parts - building a metric name in the snake-case style used by Prometheus and adding labels.

Perform these tasks for the metric name:

- Prefix all metric names with the value of the collectorName field plus an underscore. For example: 'sas_peek_system_"
- Add the metric name.
- Add the metric unit as a suffix (if it's not a simple count or 'none').
- Add a _total suffix if the metric type is 'counter'

Perform these tasks for labels:

- Include detail_level from the detailLevel field of the metric.
- Include resource_type from the resourceType filed of the measurement.
- Include instance from the resourceName property of the measurement.

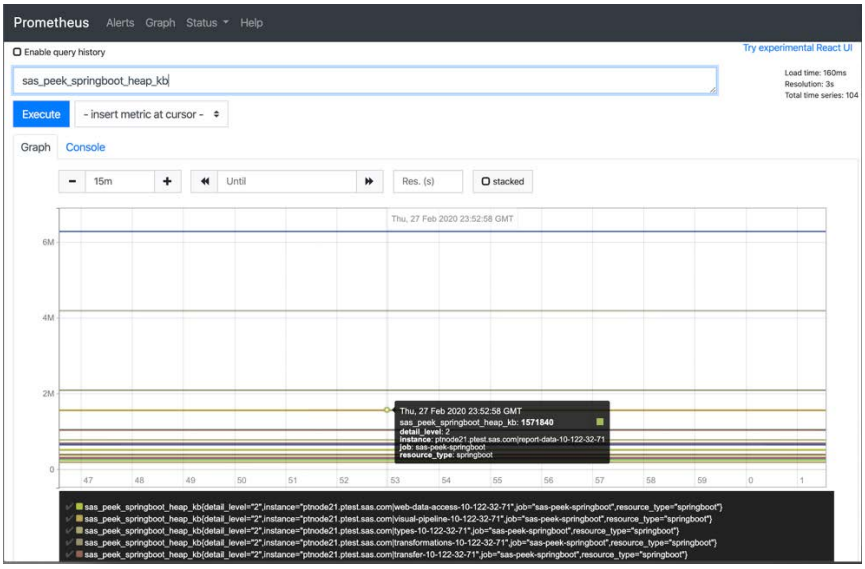
PROMETHEUS PUSHGATEWAY

The Prometheus client libraries are available for several programming languages. The library includes a 'push' package that sends metrics to the Prometheus Pushgateway.

The Pushgateway acts as a Prometheus exporter, which means that it exposes a `/metrics` HTTP endpoint that is suitable for scraping by Prometheus. There are several ways to get Prometheus to find and scrape that endpoint, but for SAS Viya 3.x, a static scrape configuration is the simplest way to collect the metrics. For example, if the Prometheus Pushgateway was started on port 9091 on the same host as Prometheus, the Prometheus configuration yaml file could be as simple as this:

```
scrape_configs:
- job_name: viya3-pushgateway

  honor_labels: true

  static_configs:
  - targets: ['localhost:9091']
    labels:
      viya_version: 3.x
```

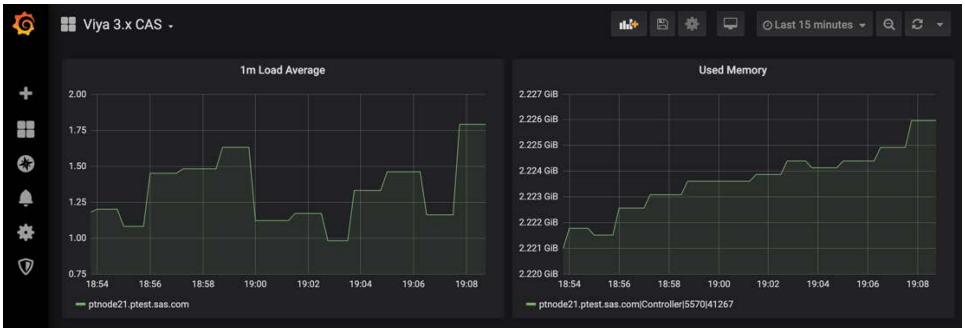
After Prometheus successfully scrapes the Pushgateway, you can use the Prometheus web application to see the time series and values:



Display 1. Testing queries in Prometheus

GRAFANA

After Prometheus is successfully scraping the Pushgateway, you can build Grafana dashboards to visualize the metrics. A simple dashboard might look like:



Display 2. Visualizing Viya 3.x metrics in Grafana

LOGSTASH AND ELASTICSEARCH

Elasticsearch is a perfect match for the standardized structured JSON logging in SAS Viya, because it includes excellent support for indexing JSON documents.

RECEIVING LOGS

Logstash includes native support for receiving events from RabbitMQ. A simple Logstash input configuration looks like the following:

```
input{
     rabbitmq {
        host => "${RABBIT_HOST:YOU_DID_NOT_SET_RABBIT_HOST}"
          port => "${RABBIT_PORT:5672}"
          exchange_type => "topic"
        exchange => "sas.log"
          durable => true
          ssl => true
        queue => "${RABBIT_LOGQ:sas.log.shared-queue}"
        auto_delete => true
        passive => false
```

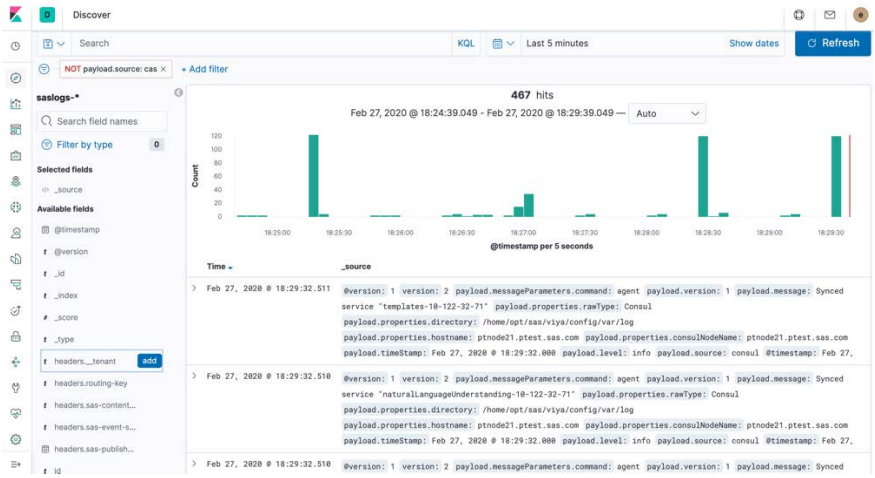
```
        user => "${RABBIT_USER:guest}"
        password => "${RABBIT_PASS:guest}"
        key => "#"

        tags => "sasviyalogs"
    }
}
```

The output configuration can be even simpler:

```
output {
 if ("sasviyalogs" in [tags]) {
   elasticsearch {
      hosts => ["localhost:9200"]
      index => "sas-log-${DEPLOYMENT_NAME:viya}-%{+YYYY.MM.dd}"
   }
 }
}
```

KIBANA

After logs are flowing to Elasticsearch, you can use Kibana to visualize, filter, and explore the logs. Kibana supports custom dashboards, full text search, UI-based filters, and a powerful query language for more advanced searching.



Display 3. Viewing Viya 3.x logs in Kibana

CONCLUSION

The event-driven architecture and JSON-based events of SAS Viya 3.x enables powerful and flexible integration with popular open-source tools for both monitoring and logging. This paper, while not a step-by-step guide, should demystify the event flows and help you get started connecting SAS Viya to popular open-source tools.

REFERENCES

Venkataramani, Meera. 2020. “Essential Performance Tips for SAS® Visual Analytics.” *Proceedings of the SAS Global Forum 2020 Conference*. Cary, NC: SAS Institute Inc. https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2020/4408-2020.pdf.

Crevar, Margaret. 2020. “Important Performance Considerations When Moving SAS® to a Public Cloud.” Proceedings of the SAS Global Forum 2020 Conference. Cary, NC: SAS Institute Inc. https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2020/4312-2020.pdf.

RECOMMENDED READING

Ellington, Bryan, 2019. “Exploring the SAS Viya Operations Infrastructure.” *Proceedings of the SAS Global Forum 2019 Conference*. Cary, NC: SAS Institute Inc. Available https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2019/3393-2019.pdf

CONTACT INFORMATION

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

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