SAS® Viya® reportImages Service: The Report Optimization Speedometer

Michael Drutar, SAS Institute Inc., Cary, NC

ABSTRACT

SAS® Viya® offers several techniques that can maximize the speed of SAS® Visual Analytics reporting: data partitioning, user-defined formats, and the use of aggregated data. However, every SAS Visual Analytics report can be different: different data, different graphs, and other differences in terms of filters, interactive widgets, and more. Testing how changes to individual reports affect speed can be laborious and might involve manually opening reports in the SAS® Report Viewer several times and meticulously reviewing each report’s diagnostics or microservice logs. Even with this information, external factors such as network performance can confound the diagnostics. This paper presents a programmatic way to call a SAS Visual Analytics report to quickly determine how long it takes the report to render using the reportImages service, available via the SAS Viya REST API. This paper provides all of the code for an automated, end-to-end process that leverages the SAS Viya REST API to retrieve the server-side render time of a SAS Visual Analytics report. Code is provided for testing an individual report on demand. This process can be repeated automatically while the report designer tests several versions of the report. Macro code demonstrates how to test a suite of reports for comprehensive A/B comparisons. Data gathered from these repeated API calls enables designers to quickly determinate the best performance techniques to meet their specific reporting needs.

INTRODUCTION

SAS Viya includes many REST APIs that enable developers to interact with the SAS® platform in new and exciting ways. Before using these APIs, developers should become familiar with basic REST concepts, such as endpoints, methods, headers, and content that can be contained within the body of a request. This paper assumes that you have a general understanding of using PROC HTTP to make REST API calls using these principles.

ACCESSING SAS VIYA SERVICES USING SAS® STUDIO 5.1

Before clients can leverage the many assets that SAS Viya REST APIs provide, they must first be authenticated using an access token. Methods for how to acquire access tokens vary, based on the language and method that clients use to interact with the SAS Viya REST APIs.

When logged on to the SAS Viya platform and submitting SAS code using SAS Studio 5.1, developers can connect to the SAS Viya services by placing the keyword sas_services in the PROC HTTP option oauth_bearer. This ensures that the access token for the current authenticated user is automatically obtained and attached to the request, thus enabling SAS programmers who are already logged on to the SAS Viya environment to seamlessly connect with the resources that are available in SAS Viya services. Details on the sas_services keyword and oauth_bearer option are beyond the scope of this paper; however, more information can be found in the SAS documentation in the References section of this paper.

The example SAS code for achieving this access is below:

```sas
* Base URI for the service call;
%let BASE_URI=%sysfunc(getoption(servicesbaseurl));
```
/*Assign filename for output*/
filename reports temp;

/*Call reports Service*/
proc http url="&BASE_URI/reports/"
    method='get'
    oauth_bearer=sas_services
    out=reports;
run;

This paper assumes that all code is submitted in a SAS Studio 5.1 session within a SAS Viya 3.4 (and beyond) environment that contains the SAS Viya services being called.

REPORTIMAGES SERVICE OVERVIEW

The reportImages service can be used by clients to create an SVG image of a SAS® Visual Analytics report. The process flow for using this service consists of two steps. Each step involves the client making a request to the service and the service sending a response. The requests and responses for these two steps are outlined below:

- **Step 1**
  - The client requests that a new job be created, which generates an SVG image of a specific SAS Visual Analytics report.
  - The service initiates the job and sends a response to the client containing information about the job (such as the job’s ID).

- **Step 2**
  - The client requests the status of the job using the job ID returned from Step 1.
  - The service sends a response to the client containing the current status of the job specified in the request.

When Step 2 returns a state of *completed* for the specified job, additional information is also contained in the response. Specifically, the response contains the duration the job took to generate the SVG image. After the service returns a *completed* status, the client can use other features of the reportImages service to retrieve the generated SVG image file itself.

MAKING REPORTIMAGES SERVICE REQUESTS USING SAS

**STEP 1: CREATING A JOB**

As described in the previous section, the client begins the process of using the reportImages service by first making a request to create a job to render a specific SAS Visual Analytics report as an SVG image. This request has the following components:

- **Endpoint**: https://www.example.com/reportImages/jobs
- **Method**: POST

The parameters of the API call are contained within the body of the request. For the purposes of leveraging the reportImages service as a speedometer, specific parameters and values must be included. The body should be composed of the following components:

- **reportUri**: /reports/reports/<report id>
- **layoutType**: entireSection
The **reportURI** parameter tells the service which SAS Visual Analytics report is to be rendered as an SVG. The **layoutType** parameter tells the service to create an SVG image of the entire report section that is specified in the **sectionIndex** parameter, which defines which report tab the service should render. The first tab in a report has an index value of zero, the second tab in the report has an index of 1, and so on. Similar to the **layoutType** parameter, the **selectionType** parameter tells the service to generate a single image, representing the entire report. The **size** parameter defines the desired width and height dimensions of the SVG image. Typically, these dimensions should be set so that they are similar to the size of the screen on which the SAS Visual Analytics Report is shown, when opened in the SAS® Report Viewer (for example, 1680x1050).

To retrieve the most accurate image generation data, it is important to set the **refresh** parameter to **true**. Ordinarily, when a request is made to generate an SVG of a specific report, the **reportImages** service first checks to see whether the requested image already exists within its cache. If it does, the service instantly returns the cached image, rather than rebuilding the image. However, this behavior is not desirable when using the **reportImages** service to gain insight into how long it takes the SAS Report Viewer to render a report in real time. Setting the **refresh** parameter to **true** forces the **reportImages** service to regenerate the SVG from scratch for each request. Therefore, the same report can be called several times and each resulting duration metric is much more accurate.

The following SAS code requests that a job be created to render a SAS Visual Analytics report:

```sas
* Base URI for the service call;
%let BASE_URI=%sysfunc(getoption(servicesbaseurl));

/* create filenames to hold responses*/
filename startjob temp;
filename resp_hdr temp;

/* Make request */
proc http
  method="POST"
  oauth_bearer=sas_services
  url="&BASE_URI/reportImages/jobs"
  ct="application/vnd.sas.report.images.job.request+json"
  in='
    "reportUri" : "/reports/reports/<- report id ->",
    "layoutType" : "entireSection",
    "selectionType" : "report",
    "refresh":true,
    "size" : "<Desired Image Size>",
    "sectionIndex" : 0
  ';
/* place response in filenames */
  out=startjob
  headerout=resp_hdr
  headerout_overwrite;
```
Submitting the code above results in successfully sending a request and the `reportImages` service sending a response. This response is captured in the `filename startjob`. The JSON LIBNAME engine can be used to read the contents of `startjob filename` as a SAS library. After this library has been assigned, the response from the `reportImages` service can be viewed as SAS tables. A subset of the contents of the `startjob.root` table is shown in Figure 1.

![Table: startjob.root](image)

**STEP 2: RETRIEVING JOB DURATION**

After the job has successfully started and the job ID has been stored in the macro variable `id`, the developer can move on to the second step of using the `reportImages` service: retrieving how long it takes to render the SAS Visual Analytics report as an SVG image. This is achieved by making a second request to the `reportImages` service for the status of a job.

This API request has the following components:

- **Endpoint**: `https://www.example.com/reportImages/jobs/<job id>`
- **Method**: GET

It is important to note the value at the end of the endpoint: `job id`. This must be the unique job ID that was stored in the macro variable `id` in Step 1. With a valid job ID value in the endpoint, the `reportImages` service successfully returns the status of the specified job. As in Step 1, the JSON LIBNAME engine is used to read the response in the form of a SAS table.

The SAS code that requests the status of a specific job ID and reads in the response via the JSON LIBNAME engine is below:

```sas
/* Base URI for the service call; */
%let BASE_URI=%sysfunc(getoption(servicesbaseurl));
```
The current status of the job specified in the request is contained as the variable `state` in the `j_status.root` table. If at the time of the request the `reportImages` service has not completed generating the SVG, the response has a value of `running`, as shown in Figure 2.

![Figure 2. Subset of Table: j_status.root](image)

If the service has completed generating the SVG image, the response has a value of `completed` and there is a new variable named `duration` in the `status.root` data set. (See Figure 3.) The `duration` variable contains the total time it has taken to successfully render the report as an SVG image. This value is the key to using the `reportImages` service as a speedometer for report rendering time.

![Figure 3. Subset of Table: j_status.root Including the Variable duration](image)

An example of such a macro is below:

```sas
/* Base URI for the service call;*/
%let BASE_URI=%sysfunc(getoption(servicesbaseurl));

/* initially set status */
%let status=0;
%macro jobstatus;
```

/* create filenames to hold responses*/
filename j_status temp;
filename res_hdr temp;
/*Make the request  */
proc http
  method="GET"
  oauth_bearer=sas_services
  url="&BASE_URI/reportImages/jobs/&job_id"
  out=j_status
  headerout=res_hdr
  headerout_overwrite;
run;

/* Use JSON LIBNAME engine to read in response */
libname j_status json;

---

5
The code above begins by creating a macro variable named `status` and setting its value to zero. After this, the definition for the jobstatus macro begins with creating a DO UNTIL loop that runs until the value of the macro variable `status` is not zero. Because this loop is expected to run several times, the `libref` and `filename` `j_status` are cleared of values that might have been assigned in the loop’s previous iteration.

After this setup, the request is made to get the status of the job that is specified in the `job_id` macro variable. Its response value is captured in the `j_status` filename and `libref`. A DATA step is then run to create a numeric variable named `status`, based on value of the `state` variable in the `j_status.root` table. If the value of `state` is `running`, the value of the `status` variable is created and set to zero. Alternatively, if the value of state is `completed`, the value is set to a quantity other than zero. The SYMPUTX function then reads the value of the `status` variable and uses it to reset the value of the `status` macro variable. Finally, the DO UNTIL loop is ended and macro definition is completed.

End to end, the macro `jobstatus` simplifies the process of retrieving the duration of a `reportImages` job by offering an automated way to make repeated requests until a status of `completed` is returned.

**COMPARING MULTIPLE REPORT RENDER DURATIONS**

The following example demonstrates how the `reportImages` service can be used as a speedometer for server-side report rendering time. This scenario consists of a SAS Visual
Analytics report that visualizes data containing information about the names of roads within the United States. Specifically, the report shows which states have roads that contain the word “Parkway” in their names. The data source of this report is a SAS data set named “us_roads” that contains 10 variables and 20 million observations. Figure 4 and Figure 5 show screenshots of the data source and report, respectively.

<table>
<thead>
<tr>
<th>DEGX</th>
<th>DEGY</th>
<th>SEGMENT</th>
<th>FULL_NAME</th>
<th>MTCCC</th>
<th>ROADFLG</th>
<th>TLD</th>
<th>STATE</th>
<th>COUNTY</th>
<th>ID</th>
<th>STATECODE</th>
<th>STATERENAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>-05.3046</td>
<td>34.1571</td>
<td>1</td>
<td>2nd Ave SW</td>
<td>1200</td>
<td>Y</td>
<td>100243255</td>
<td>01</td>
<td>040</td>
<td>000104</td>
<td>AL</td>
<td>Alabama</td>
</tr>
<tr>
<td>-05.2559</td>
<td>31.0599</td>
<td>1</td>
<td>East Bay Ave</td>
<td>1200</td>
<td>Y</td>
<td>10441314</td>
<td>01</td>
<td>041</td>
<td>0001041</td>
<td>AL</td>
<td>Alabama</td>
</tr>
<tr>
<td>-07.5294</td>
<td>34.9500</td>
<td>1</td>
<td>State Rte 157</td>
<td>1200</td>
<td>Y</td>
<td>40790754</td>
<td>01</td>
<td>003</td>
<td>00010032</td>
<td>AL</td>
<td>Alabama</td>
</tr>
<tr>
<td>-07.4256</td>
<td>32.4582</td>
<td>1</td>
<td>Jefferson Davis Hwy</td>
<td>1200</td>
<td>Y</td>
<td>1200420</td>
<td>01</td>
<td>047</td>
<td>0001047</td>
<td>AL</td>
<td>Alabama</td>
</tr>
<tr>
<td>-05.7691</td>
<td>31.1786</td>
<td>1</td>
<td>State Rte 127</td>
<td>1200</td>
<td>Y</td>
<td>10966942</td>
<td>01</td>
<td>009</td>
<td>0001009</td>
<td>AL</td>
<td>Alabama</td>
</tr>
<tr>
<td>-05.2211</td>
<td>31.2516</td>
<td>1</td>
<td>Plaza Dr</td>
<td>1200</td>
<td>Y</td>
<td>09554415</td>
<td>01</td>
<td>021</td>
<td>0001021</td>
<td>AL</td>
<td>Alabama</td>
</tr>
<tr>
<td>-05.2799</td>
<td>34.2116</td>
<td>1</td>
<td>I- 95</td>
<td>1100</td>
<td>Y</td>
<td>1002416522</td>
<td>01</td>
<td>003</td>
<td>0001043</td>
<td>AL</td>
<td>Alabama</td>
</tr>
<tr>
<td>-05.4593</td>
<td>33.7022</td>
<td>1</td>
<td>US Hwy 79</td>
<td>1200</td>
<td>Y</td>
<td>1020223771</td>
<td>01</td>
<td>020</td>
<td>0001020</td>
<td>AL</td>
<td>Alabama</td>
</tr>
<tr>
<td>-07.7598</td>
<td>31.1491</td>
<td>1</td>
<td>State Rte 27</td>
<td>1200</td>
<td>Y</td>
<td>09529542</td>
<td>01</td>
<td>045</td>
<td>0001045</td>
<td>AL</td>
<td>Alabama</td>
</tr>
<tr>
<td>-07.0594</td>
<td>31.8571</td>
<td>1</td>
<td>State Rte 92</td>
<td>1200</td>
<td>Y</td>
<td>20020951</td>
<td>01</td>
<td>006</td>
<td>0001006</td>
<td>AL</td>
<td>Alabama</td>
</tr>
</tbody>
</table>

Figure 4. Subset of the Source Table: us_roads

Figure 5. Visual Analytics Report Using the Source Table: us_roads

The data for this report is refreshed nightly. To ensure that a fast report load time occurs when the first user opens the report after the nightly data refresh, the developer is exploring various methods to maximize report speed.

One method is to create an aggregated version of the data source named us_roads_agg that contains only 3 variables and 12 observations. The report developer then creates a new version of the “Parkway Roads” report that uses us_roads_agg as its data source.
Screenshots of the *us_roads_agg* aggregated data source and its associated report appear in Figure 6 and Figure 7, respectively.

<table>
<thead>
<tr>
<th>STATENAME</th>
<th>STATECODE</th>
<th>Parkway_Roads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>AL</td>
<td>287</td>
</tr>
<tr>
<td>Arizona</td>
<td>AZ</td>
<td>109</td>
</tr>
<tr>
<td>Maryland</td>
<td>MD</td>
<td>2</td>
</tr>
<tr>
<td>Missouri</td>
<td>MO</td>
<td>33</td>
</tr>
<tr>
<td>New Hampshire</td>
<td>NH</td>
<td>2</td>
</tr>
<tr>
<td>New Jersey</td>
<td>NJ</td>
<td>2</td>
</tr>
<tr>
<td>Pennsylvania</td>
<td>PA</td>
<td>85</td>
</tr>
<tr>
<td>South Carolina</td>
<td>SC</td>
<td>81</td>
</tr>
<tr>
<td>Tennessee</td>
<td>TN</td>
<td>1,309</td>
</tr>
<tr>
<td>Texas</td>
<td>TX</td>
<td>308</td>
</tr>
<tr>
<td>Virginia</td>
<td>VA</td>
<td>827</td>
</tr>
<tr>
<td>Wisconsin</td>
<td>WI</td>
<td>6</td>
</tr>
</tbody>
</table>

**Figure 6. Aggregated Table: us_roads_agg**

Now that the report developer has created two different versions of the report, the *reportImages* service can be used to test the server-side rendering time for each of them.

Leveraging the previously discussed concepts of using the *reportImages* service to retrieve a job’s duration, the report developer writes one comprehensive macro to test these two examples reports and return their respective server-side rendering times. The macro is
called by using the parameters \texttt{datasource}, \texttt{report\_name}, and \texttt{report\_URI}. This macro’s definition is displayed below:

\begin{verbatim}
* Base URI for the service call;
%let BASE_URI=%sysfunc(getoption(servicesbaseurl));

%macro report_generation_duration(sourcedata,report_name,report_uri);
/* Refresh Data in CAS */
cas myses;
proc casutil;
droptable incaslib="PUBLIC" casdata="&sourcedata" quiet;
   load incaslib="PUBLIC" outcaslib="PUBLIC"
casdata="&sourcedata..sashdat" casout="&sourcedata" promote;
run;
/* create dynamic proc http 'in=' statement */
data create_params;
   request_params = "" || trim('"reportUri" : "") || "&report_uri" ||
      trim('","layoutType" : "entireSection","refresh":true,"selectionType" : "report","size" : "1680x1050","version" : 1)" || ");
call symput('request_params',request_params);
run;
/* create job and get response */
filename resp_hdr clear;
filename startjob clear;
libname startjob clear;
filename startjob temp;
filename resp_hdr temp;
proc http
   method="POST"
oauth_bearer=sas_services
   url="&BASE_URI/reportImages/jobs"
   ct="application/vnd.sas.report.images.job.request+json"
in=&request_params.
   out=startjob
   headerout=resp_hdr
   headerout_overwrite;
run;
libname startjob json;
/* capture job id into macro variable job_id */
data _NULL_; 
   set startjob.root;
      call symputx('job_id',id);
run;
/* Set initial &status to be zero */
%let status=0;
/* macro to check status until job is completed */
%macro jobstatus;
   %do %until (&status ne 0);
      filename res_hdr clear;
      filename j_status clear;
      %end;
%end;
\end{verbatim}
libname j_status clear;
filename j_status temp;
filename res_hdr temp;
/* Make API Call */
proc http
   method="GET"
oauth_bearer=sas_services
   url="&BASE_URI/reportImages/jobs/&job_id"
   out=j_status
   headerout=res_hdr
   headerout_overwrite;
run;

libname j_status json;
/* create &status macro variable */
data job_status;
   set j_status.root;
   if state = 'running' then status = 0;
   else if state = 'completed' then status = 1;
   call symputx('status',status);
run;

%end;
%mend jobstatus;
/* call macro %jobstatus */
%jobstatus;
/* create and print final data set */
data report;
   set j_status.root;
   reportName = "&report_name";
   reportURI = "&report_uri";
   label id = "reportImages Job ID"
   duration = "Job Duration"
   label state="Job Status";
run;
/* Print output */
title "reportImages Duration - Report: '&report_name'";
proc print data= report noobs label;
   var reportName reportURI id state duration;
run;
%mend report_generation_duration;

The macro begins by reloading the report data sources from sashdat files to simulate a nightly refresh of the data. This is achieved by using the PROC CASUTIL statement in conjunction with the sourcedata parameter. The next step is to dynamically generate the value for PROC HTTP's "in=" argument using the value contained the report_uri parameter. This value is placed in a macro variable named request_params via the SYMPUT function. The macro then uses the request_params value to make the needed requests to create a reportImages service job and repeatedly check its status until a value of completed is
returned. Finally, a data set that contains the final desired variables from the macro’s results is created and printed.

This macro can now be called with the values from the two reports in this example scenario:

```sas
/* call reports */
%report_generation_duration(us_roads,Parkway Roads - Source Data,/reports/reports/c199d225-a536-44c9-a9c3-5d9e19aac6cc);
%report_generation_duration(us_roads_agg,Parkway Roads - Aggregated Data,/reports/reports/e3704e78-2316-48b3-9f25-a056a2bccc3f);
```

The result of submitting these two macro calls (and their associated parameters) is the following PROC PRINT outputs of the report data set for each of the two example reports shown in Figure 8.

<table>
<thead>
<tr>
<th>reportName</th>
<th>reportURI</th>
<th>reportImages Job ID</th>
<th>Job Status</th>
<th>Job Duration Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parkway Roads - Source Data</td>
<td>/reports/reports/c199d225-a536-44c9-a9c3-5d9e19aac6cc</td>
<td>d50e520e-e0ba-4105-9834-a21054dacdbdb</td>
<td>completed</td>
<td>17.250</td>
</tr>
<tr>
<td>Parkway Roads - Aggregated Data</td>
<td>/reports/reports/e3704e78-2316-48b3-9f25-a056a2bccc3f</td>
<td>d50e520e-e0ba-4105-9834-a21054dacdbdb</td>
<td>completed</td>
<td>7.002</td>
</tr>
</tbody>
</table>

**Figure 8. Printed Output from the Macro report_generation_duration**

It is obvious from the results above that the report that uses the aggregated version of the data source renders much faster than the report that uses the larger source data.

**CONCLUSION**

Creating reports that render quickly is a key component in ensuring that report viewers are satisfied with their reporting systems. Though the true report rendering time for different users might be slowed by external factors, such as network performance, developers should ensure that their reports are as efficient as possible before the first customer opens the report using SAS Report Viewer. Using the reportImages service job duration time as a report optimization measuring stick greatly assists in this task.

As mentioned at the onset of this paper, there are several techniques that can maximize the speed of SAS Visual Analytics reporting: data partitioning, user-defined formats, and the use of aggregated data. Programmatically retrieving a report’s server-side rendering time empowers the report developer to quickly explore all options for ensuring that a customer’s data visualizations are returned in the shortest time possible. Armed with this new method of determining which report optimization is best for each specific report and data source, developers are ready to create their most optimized, fastest reports yet.

**REFERENCES**


**CONTACT INFORMATION**

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

- Michael Drutar
- SAS Institute Inc.
- Michael.Drutar@sas.com
- http://www.sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.