Paper 3727-2019

Take REST into Account: Account Level Profit with SAS® Viya APIs

Richard Carey, Demarq

ABSTRACT

The bottom line of a business is directly influenced by a great many factors, none more so than the profit generated or money lost from each customer. Accessing financial performance metrics at the account level is a crucial first step in optimising the performance of a company; from improving efficiency of marketing spend to identifying the most desirable customers to retain.

SAS® Viya's REST APIs provide the ideal route for a bespoke web application to access the power of CAS. In this paper, we explore the options available for manipulating, analysing and reporting on Account Level Profit using Viya REST API calls.

INTRODUCTION

SAS Viya has moved the game on when it comes to accessing SAS functionality from tools and languages outside the SAS ecosystem. These additional mechanisms allow and encourage the use of SAS Viya as a component of a larger solution. As a SAS partner, Demarq has done just that with its Account Level Profitability (Demarq-ALP) solution. Blending open source and SAS technology, Demarq-ALP allows allocation of income and costs down to the account level.

In this paper, we use Demarq-ALP as a case study to explore the options available when considering using SAS Viya in this manner. Firstly, we will define the fundamentals of an ALP system, focus in on a single use case, then explore how SAS Viya's various mechanisms for integration with third party software packages can help to meet those requirements.

SAS VIYA AND BESPOKE WEB APPLICATIONS

The mechanisms available for using SAS Viya as part of a web application are very different to those available in SAS 9. This is a reflection on the current state of web development. The current trend for stateless design and integration of disparate systems via REST APIs has been adopted by SAS, and a vast range of Viya functionality is presented as web services available to be consumed over HTTP.

The availability and support of a wide array of API's and programming libraries for SAS Viya encourage its use as a component of a larger solution. Python, Java, Lua and JavaScript (via restAF) are all catered for by SAS authored connectors; any other language with an HTTP library and the ability to process JSON can make use of the REST API's directly.

CASE STUDY: ACCOUNT LEVEL PROFITABILITY

This section introduces a case study as an example of how Viya can be used to fulfil reporting and data manipulation requirements of a larger solution. Firstly, we will consider the various aspects of the Demarq-ALP application, then focus on a single use case and discuss the options available to integrate with Viya to provide the functionality required.

WHAT IS DEMARQ-ALP?

Demarq-ALP is a data driven solution designed to give a view of profitability at the customer account level. Applicable to any business with customers, income and costs, the output allows the identification of profitable accounts along with identification of the factors that contribute to that profit. Of equal benefit is the identification of those accounts that are *not* profitable, allowing this activity to be eliminated and making an immediate impact to the bottom line. At a high level the solution consists of a data ingestion stage feeding a unified data warehouse that reconciles back to the general ledger, allocation and reconciliation logic rule engines, and a reporting layer. These areas are detailed in the following sections.

Data Ingestion

The first stage of the process is to get the required data into the system. For most customers, this will involve data from Accounts Receivable, General Ledger and the CRM, but could feasibly involve any data that you wish to use as input to allocating cost or income.

For example, you may wish to allocate the costs associated with a customer call centre in proportion with how often each account has used the service. In order to do so you need data that tracks who called and when.

Allocation

Allocation is the process of splitting costs and income between each account. Allocation rules fall into one of the following major types:

1. Allocate evenly between all accounts
2. Allocate evenly between accounts that meet certain criteria
3. Allocate in proportion to some metric (number of events, amount of credit used etc.)
4. A combination of the above

As an example, a credit card company may wish to allocate the cost of providing physical cards evenly between all accounts (1). Gold card customers pay a fee in order to have an account, so this income is split between those accounts only (2). The cost of sending letters to account holders should be allocated according to the amount of correspondence each account required (3). You could combine these rule types (4), for example allocating the cost of a Gold card concierge service to Gold card holders only (2) based on each account's usage of the service (3).

Reconciliation

Once financial data has been allocated between accounts using the allocation rules, the next step is to reconcile the figures back to the general ledger or other financial report. In accordance with double entry accounting, this step ensures that money has not been created or destroyed by a mistake in the allocation process and creates confidence in the output reports.

Reporting

Now allocated and reconciled, the data can be presented for use by the business. The reporting layer allows groups or "cohorts" of accounts to be selected based on certain criteria, and the profitability of that cohort to be displayed.

More than simple filtering, this "cohort reporting", allowing accounts to be selected based on their historical activity, is a key benefit of the solution and can show the profitability of accounts that have been through the same customer journey. For example, a cohort could contain all accounts that were active in January 2018, having been inactive in 2017.

SAS VIYA FUNCTIONALITY

In this section we explore the functionality SAS Viya can provide to a web-based application, and the options available for integration.

In our ALP example, we will focus on using SAS Viya in two ways; to subset and filter the allocated data set to report on, and to produce the reports themselves. This use case is described below.

DATA MANIPULATION

In order to achieve cohort reporting, the allocated data must first be subset to contain only the relevant accounts that meet our criteria.

Via web-based user interface, the criteria for account selection are entered and submitted to CAS to perform the query. The resulting data set is then used as the input to a pre-built Visual Analytics report.

REPORTING

Once subset, the solution presents a visualisation of the data via a report. To meet this goal we use SAS Visual Analytics. A series of static report images are generated and presented to the user. If desired, the user can then explore the data further interactively via the full SAS interface.

INTEGRATION USING THE REST API

How could we go about accessing the functionality required for the use case described above? In this section we discuss using the SAS Viya REST APIs to meet the requirements we have set out in the use case.

REST APIS

According to the documentation, the "SAS Viya REST APIs target enterprise application developers, who intend to build on the work of model builders and data scientists, to deliver apps based on SAS Viya technology" (SAS Viya REST documentation).

They do this by providing a mechanism to access Viya functionality over an HTTP interface. The loose coupling of server and client has meant REST API's have become an increasingly popular way to integrate disparate software into a single solution, especially in the realm of web development. Any programming language with an HTTP library can make use of a REST API; indeed, in many cases there will be a client library available for the specific API in question. This makes them an ideal fit for a wide range of software development needs, from full-blown local applications, to light touch web development, as well as the Internet of Things.

In general, REST APIs represent resources with URLs or endpoints, which can be operated on with HTTP requests. The various HTTP methods (GET, POST, DELETE etc.) along with information contained in the body of the request itself allow the same endpoint to be manipulated in different ways. For example, a resource might be created with a POST request, destroyed with a DELETE request, or information about the resource returned with a GET request.

SAS provides two REST API's for Viya; one covering Viya functionality, and a lower level CAS API. The main Viya API allows access to higher level functions of the system; load a table, generate a report, score a model. The CAS API provides low level access to the CAS grid itself, allowing direct access to manage sessions, run actions and gather information on and control the system itself.

For our ALP example, both the data manipulation and reporting use cases could be accomplished using the REST APIs. The documentation lists APIs in the following areas; Visualization, Compute, Text Analytics, Data Management, Decision Management, Core

Services. The software licenced at your site will of course limit the availability of functionality in these areas.

For our use case, both the Visualisation and Compute functions will be required. Compute offers functionality analogous to an application server context and workspace server in SAS 9, that is, the ability to start a session and run SAS code. Visualisation covers the reporting aspect - functions of SAS Visual Analytics.

Compute

Next, we look at the API calls required to meet our objectives. The following section is presented in a technology agnostic way; your choice of language or web framework will not impact the API calls required.

In our example, we wish to run a query against a pre-existing table and load the result to another table. We assume here that we already have an interface in place which allows the user to build the criteria with which to subset the allocated data.

The actual query and load to CAS can easily be accomplished with some SAS code, but how would we go about submitting that code using the API? The API calls required to submit are shown as a flowchart in Figure 1. The calls themselves are highlighted in yellow, the actual endpoints and HTTP verbs are shown as callouts.

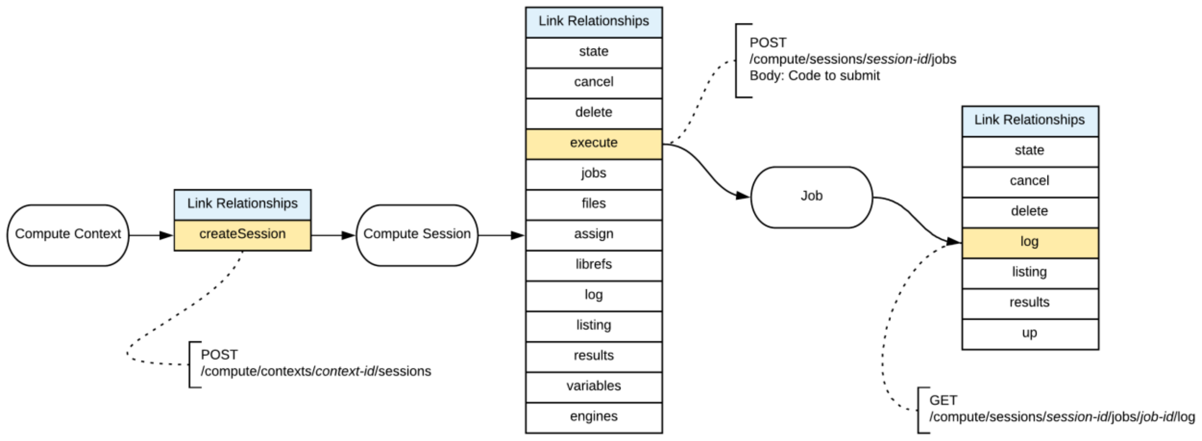

Figure 1 - Code submission API flow

Calls to the REST API return hypermedia, links that articulate in a programmatic way the possible interactions with the returned resource. Each link is described with a “rel” attribute, or “link relation”, giving options for where to go next, with a descriptive name. Along with this is the endpoint and method expected. The requests and responses themselves are encoded as JSON data, the most popular standard in use by REST APIs today.

As the application flow in Figure 1 shows, for our example from a compute context we create a compute session by making an HTTP POST request to the /compute/contexts/context-id/sessions endpoint; from the compute session we execute a job (a POST to /compute/sessions/session-id/jobs endpoint), and from the job itself return the log or results. This requires three separate calls to the API.

The same endpoint can be used to manipulate the resource in different ways depending on the HTTP method used. As an example, a GET request to the “compute/sessions/session-id/jobs” endpoint returns a listing of all jobs. A POST request to the same endpoint allows the submission of a new job.

Visualisation

To hit our reporting requirement, we need to display a visualisation to the user based on our newly filtered data set. We want to present the user a pre-built report, rather than a blank report linked to the filtered data set. In order to do so we must manipulate a templated SAS Visual Analytics report to point to the new table. This templated report must already exist, and a copy is created (or recreated) for each user when they log on.

The visualisation functions of the REST API enable us to do this. Firstly, we must check if the user already has a copy of the templated report. To do so, we make a GET request to the /reports/reports endpoint. By default this will return all reports the calling user has permission to see. In this example we make use of the filtering capability of the API; filter parameters are passed as a query parameter on the URL. Here, we are searching for a report with a specific name.

If the report already exists, we must delete it to enable a new copy with the correct data to be created in its place. The delete is an HTTP DELETE request to the same endpoint, identifying the report to delete using its ID.

The API calls needed to test for existence and delete the report if necessary are highlighted in yellow in Figure 2. The other available link relationships are also shown.

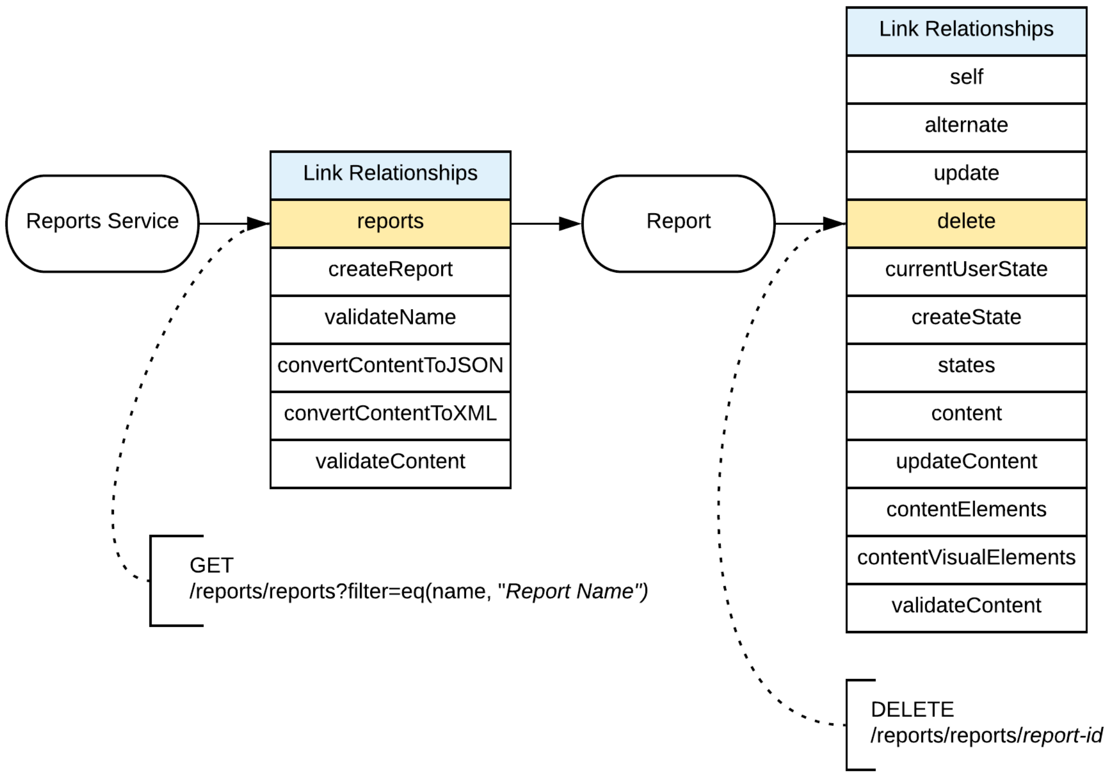

Figure 2 - Delete report if exists API flow

Once any existing report has been deleted, we can safely create a new copy of the templated report. This is done using the Report Transforms service, which allows modifications to be made to a copy of an existing report via a transformation specification. The specification is included in the body of the request, and details the changes required, in this case modifying the source data. The link relationships available from the Report Transforms service are shown in Figure 3. The required API call is highlighted in yellow.

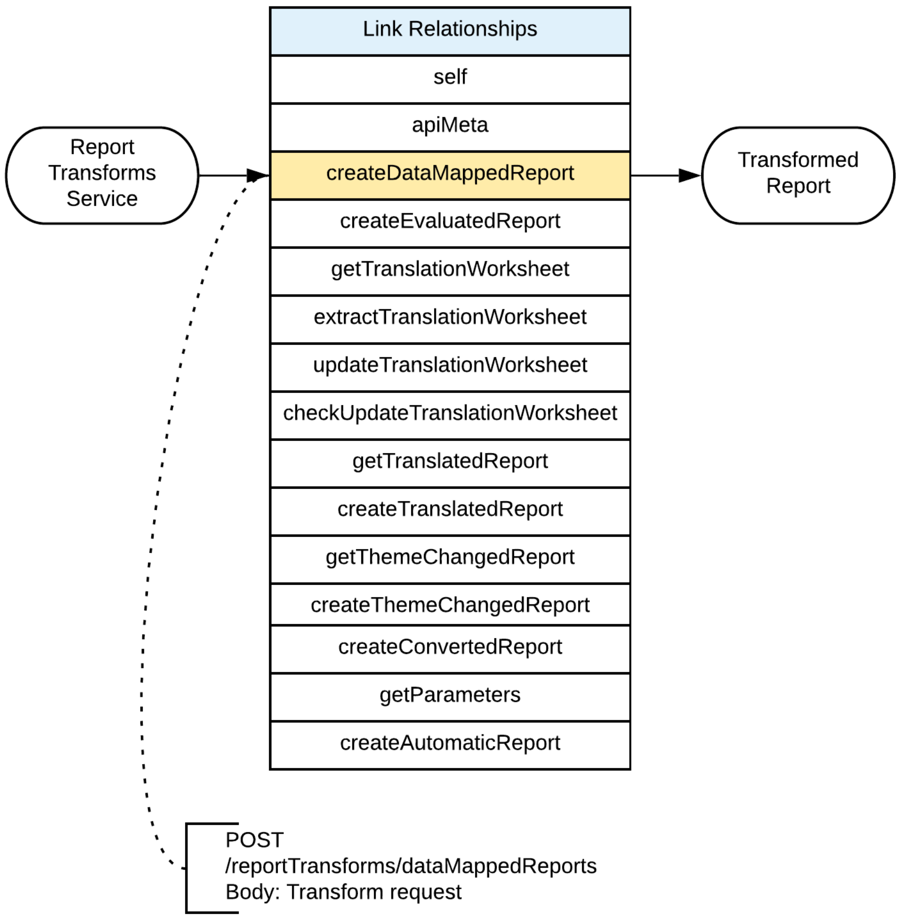

Figure 3 - Change report data source API flow

The final step is to generate a visualisation to display to the user. The Report Images service allows static images to be created from SAS Visual Analytics reports in several formats. An alternative approach could be to display the interactive report in the SAS Visual Analytics interface itself via an iframe or equivalent.

The link relationships available from the Report Images service are shown in Figure 4. The API calls required to generate a static image are highlighted in yellow.

First, the job is submitted with a POST call to the /reportImages/jobs endpoint, along with a JSON job specification in the body. This details various options for the creation of the image, such as the size, format and part of the report to generate. The state of the job must be polled until it is complete with GET calls to /reportImages/job/job-id/state. Once complete, a call can be made to retrieve the image itself, using a GET to /reportImages/images/image-id.

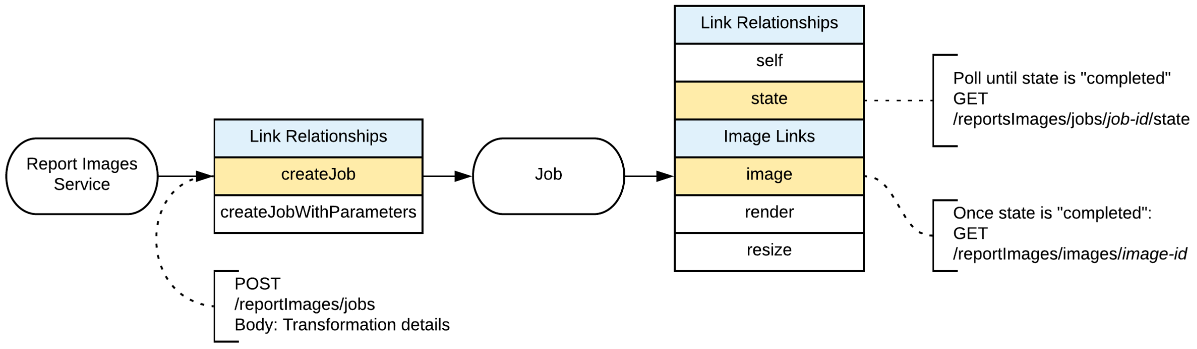

Figure 4 - Report image API flow

OTHER INTEGRATION OPTIONS

In addition to the REST APIs, SAS provides several other interfaces into Viya that can be used for application development.

PYTHON AND LUA SWAT LIBRARYS

If you are developing your web application in Python, perhaps using the Django framework or something like Flask, then the SAS Scripting Wrapper for Analytics Transfer (someone really wanted the acronym to spell SWAT!) package is ideal. Also available for the Lua language, SWAT allows access to commonly used CAS functionality such as control of sessions, submitting code and scoring models, and as such, would work well for the data manipulation requirements of ALP.

To satisfy the reporting requirement with SWAT alone would require the visualisations themselves to be generated in Python or Lua, rather than by SAS Visual Analytics. However, there is nothing to stop us using the REST APIs directly from Python in addition to SWAT to achieve this.

JAVA

The Java API offers classes corresponding to each CAS action available. Ideal for more heavyweight application development using frameworks such as Spring.

As with the Python SWAT library, our data manipulation requirements can be met using the Java APIs, and the visualisations would need to be generated locally, or by calling the REST APIs directly.

IOS AND ANDROID SDKS

For mobile app development, SAS provides software development kits for both iOS and Android. Currently they provide access to embed SAS Visual Analytics content within an app, so meet the ALP reporting goal, but do not provide access to lower level CAS functionality to handle data manipulation.

RESTAF

restAF is a Javascript library based on the SAS Viya REST APIs that abstracts away a lot of the complexity of working with the endpoints directly. As such, restAF offers identical capabilities to working with the APIs themselves, and therefore the data manipulation and reporting requirements of our use case can be met by following the same pattern as discussed above.

The library is available on the SAS GitHub page at https://github.com/sassoftware/restaf.

CONCLUSION

As we have seen, SAS Viya provides a myriad of functionality and options to access it from third party tools. This makes it easier than ever before to embed SAS into larger applications and spread advanced analytics capability throughout your organisation, through web applications, mobile apps and full client software.

REFERENCES

SAS Software. "REST APIs for SAS Viya and CAS", Accessed 15/01/2019. Available at https://developer.sas.com/guides/rest.html

CONTACT INFORMATION

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

Richard Carey
Demarq
richard.carey@demarq.uk
https://demarq.uk/services/pathway-to-account-level-profitability/

For further information on Demarq-ALP, please contact Demarq at:

info@demarq.uk