VIYA REPORT USAGE

OVERVIEW AND IMPLEMENTATION

Tommy Armstrong | December 9th 2020

TABLE OF CONTENTS

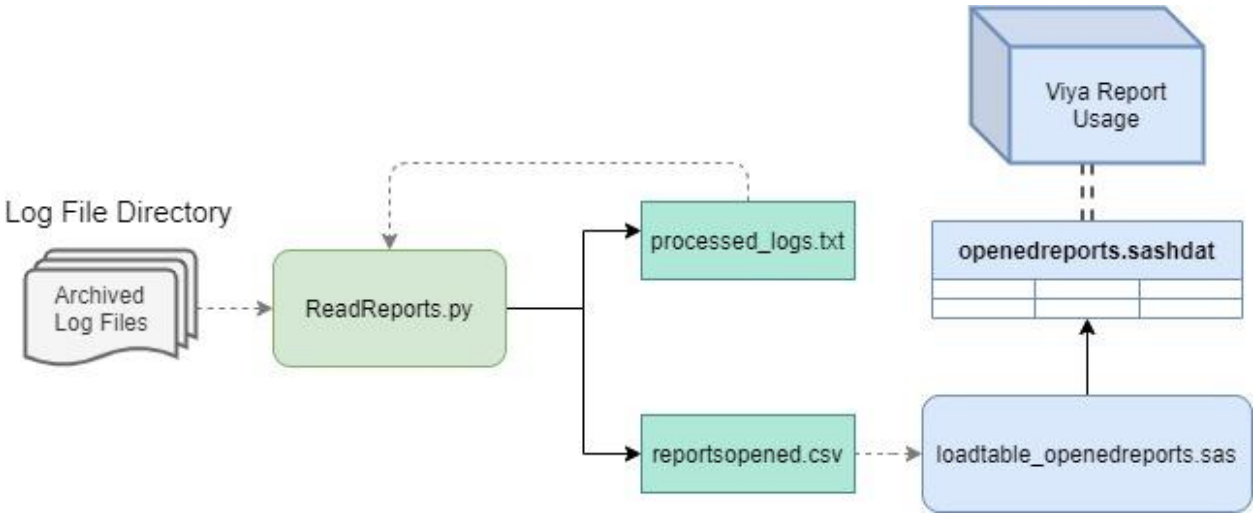
1 OVERVIEW

The **objective** of this project was to develop a method to parse through Viya logs in order to **identify** the event of a report being opened. Additionally, a report opening event is later *characterized with associated data* including the date and time the report was accessed, the report URI, the report name and the user's name who accessed the report. The resulting data allows for the measurement of report usage on a Viya environment.

This was achieved by implementing a **modular approach**. Briefly, a directory was created that housed all the Viya log files. Each log contains a day's worth of logging. A python script read in the log files and parsed them for report opening events. These events were written as output data to a CSV file. This CSV file was read in by a SAS code program that joined the data coming from the log files with additional descriptive data obtained from queries using Viya API calls. The resulting data table serves as the data source for the report to visualize report access on the Viya environment.

1 METHODOLOGY

Below is a high-level overview of the method implemented to achieve the goal of this project:



1.1 The Viya Logs

The first dependency of this method is the Viya logs. A **directory** that houses files containing the output of the Viya logging needs to be created and accessible for this method to be

successful. This also means that the Viya environment must be configured to periodically write log files to this directory. This directory will be referenced by the python script.

For the environment this project was developed for, the Viya Logs are located at:
/ndt/warehouse/default/ndtrun/audit
On both the NDT DEV and PROD environments.

1.2 Python Script

A python file called ReadReports.py was created with the purpose of parsing log files in order to identify report openings.

Syntax:

```
usage: readreports.py -l <logs directory path> -o <output directory path>
usage: readreports.py --logsDir <logs directory path> --outDir <output directory path>
```

The are **two parameters** passed from the command line:

1. --logsDir, -l : this parameter is the path to the directory where the log files reside as well as a file, processed_logs.txt, that records which log files have already been processed.
2. --outDir, -o: this parameter is the path where the CSV file, reportsopened.CSV, will be written to.

The script outputs **two files**:

1. reportsopened.csv : this file captures the report opening events as CSV formatted data.
   Example of file contents:

```
date,time,user,report_uri
2020-10-30,15:56:29.126,damccu,/reports/reports/2dee9f22-0adb-4f2b-8f9d-1da1ae73953b
2020-10-30,16:56:00.501,stsztu,/reports/reports/a76fd8aa-4076-482f-b472-1eb3da615bfc
```

2. processed_logs.txt : this file captures the filenames of each log that was successfully parsed. For each successful execution of the script, a line containing the date, time and the directory from which each log file was read is written to the file. Each line subsequently written to the file represents a processed log filename.

   The script also outputs general information regarding each processed log file: the log file name and the number of report openings identified in each.

The **execution** of the script proceeds in the following order:

1. A list of log files is created by reading from the directory passed as the log directory.

2. The processed_logs.txt file is read in to filter the list of logs to retain only those that haven't been processed. Thus, only log files from the last time of script execution should be processed.
3. The list containing only unprocessed log files is sorted chronologically from oldest to newest.
4. A loop is initiated for each unprocessed log file where the contents of each log is searched for the string pattern identifying a report opening.
5. Once found, a subsequent loop is initiated where preceding lines are searched for a report URI. Once the report URI is found, the loop is terminated. The loop continues until the report URI is found or the searched log lines have a timestamp greater than 1 second of difference with that of the log line identifying the report opening.
6. Once the report URI is resolved, it is output as an entry to the CSV file including the date, time, and user associated with the log line identifying the report opening.
7. After all the unprocessed logs have been evaluated in the manner above, the current datetime and the logs directory is appended to the processed_logs.txt. Each filename is then appended as a line to the processed_logs.txt file.

The ReadReports.py script can be accessed from the SVN link below:
https://svnhost.vsp.sas.com/viewvc/ndt/internal/trunk/python3/

1.3 SAS Code

After the ReadReports.py script generates the CSV file, it is enriched with additional descriptive and pertinent data such as the report name, the user's name and which workstream the report is associated with. This descriptive data is queried by utilizing the SAS Viya API with HTTP requests.

This is all accomplished with the loadtable_openedreports.sas program. The order of execution is as follows:

1. Folder objects are queried and converted to tabular data in order to determine workstreams.
2. Report objects are queried an converted to tabular data. This includes report names and the folder paths where the reports reside.
3. User objects are queried and converted to tabular data. This includes user names.
4. The CSV file is imported with a filename statement and a PROC IMPORT step.
5. A new column is added to the table containing the CSV data that stores the VA link to the report.

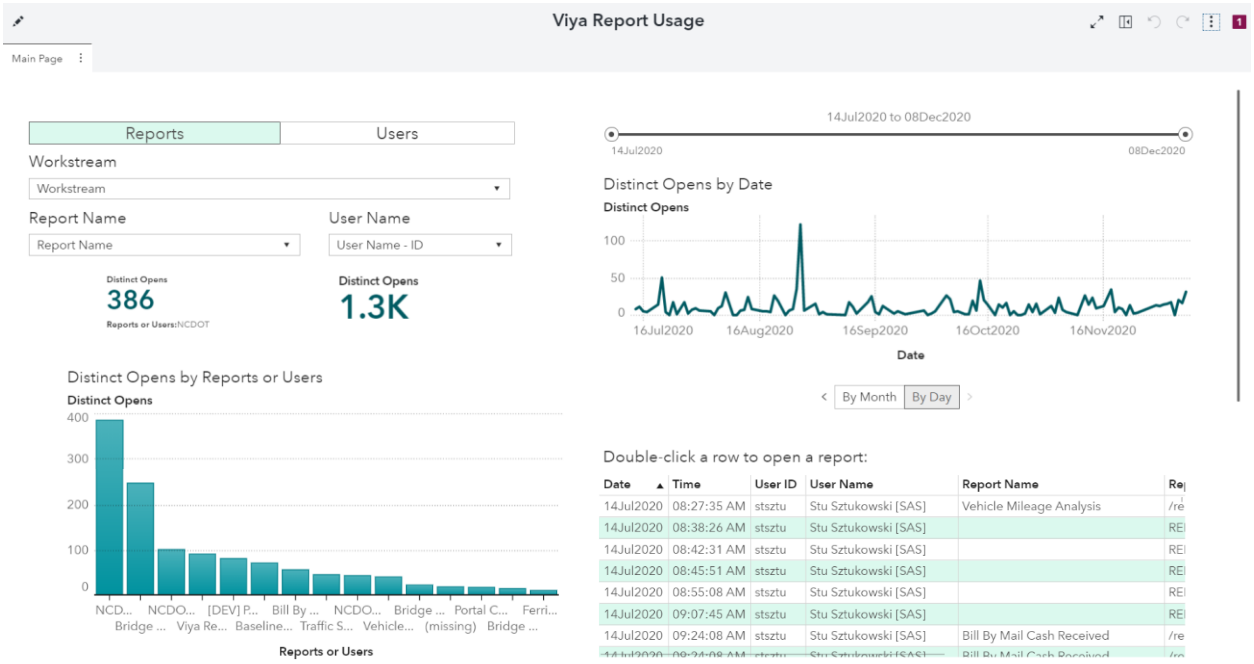
6. A PROC SQL step is used to join the CSV data with the queried Folder, Report and User data. The CSV data is now combined with the descriptive data of the queries to form the complete table.
7. The complete table is then appended to the existing openedreports table that resides in-memory under the Internal caslib.
8. This table is saved as a .sashdat CAS table and then dropped from memory. It is then reloaded as a global in-memory table so that all dependencies of the table now have access to the newly appended report opening event data.

NOTE: the object URI's that are queried in the API calls are relative to the Viya Environment used. If the environment is changed (e.g. from PROD to DEV) these URI's will need to be modified so that the objects being queried are the ones residing in the selected Viya environment.

The loadtable_openedreports.sas program can be accessed in SVN with the following link: https://svnhost.vsp.sas.com/viewvc/ndt/internal/trunk/sas/programs/loadtable_openedreports.sas?view=log

2 VIYA REPORT USAGE

The report used to visualize the report access data is title Viya Report Usage and can be found by a search in Visual Analytics or SAS Drive. It can also be accessed with the following link: https://ndtviyaprod.ondemand.sas.com/links/resources/report?uri=%2Freports%2Freports%2Fcc5a57f-4637-4f71-8629-830a4ab3bd49&page=vi281



The report allows for various views of the data including:

- Report access falling within a particular time period
- A specific report's activity by user
- A specific user's activity by report
- Report access by workstream

Additionally, reports can be opened in Visual Analytics by double-clicking the its corresponding row in the list table.