Paper 1041-2017

War and Peace: SAS® Platform Support—Can We Make It Easier?

Sergey Iglov, Royal Bank of Scotland

ABSTRACT

Over the years, the use of SAS® has grown immensely within Royal Bank of Scotland (RBS), making platform support and maintenance overly complicated and time consuming. At RBS, we realized that we have been living "war and peace" every day for many years and that the time has come to re-think how we support SAS® platforms. With our approach to rationalize and consolidate the ways our organization uses SAS® came the need to review and improve the processes and procedures we have in place. This paper explains why we did it, what we've changed or reinvented, and how all these have changed our way of operation by bringing us closer to DevOps and helping us to improve our relationship with our customers as well as building trust in the service we deliver.

INTRODUCTION

We all know how frustrating it can be when even a tiny bit of functionality or capability of an application is not working as intended or result that we are getting is not expected or we are not getting any result at all. It can take hours, days and even weeks to get to the bottom of a problem and often requires input from supporting teams (both application and infrastructure). This is getting to a whole new level when the solution to a problem is identified as a hotfix, patch or even an option that needs to be added to the configuration file and the only thing that delays resolution is the actual process of implementing the change.

Sound familiar? It surely does for us especially with the number of users reaching a thousand mark across the organization (counting only those using SAS® Desktop clients or SAS® Foundation) and versions of SAS® software varying from 9.2M3 to 9.4M3.

It gets even worse when the problem is reported by only one user or small group of users but resolution requires software re-packaging or implementing a solution that can potentially impact many if not all users. Supporting teams are faced with this difficult situation daily and coupled with complex change management process it forces them to say "no" to users which leaves them frustrated and disappointed.

At RBS, we have looked to augment the existing processes to allow our platform and application support teams to be more flexible and responsive to the requests from their user communities. To do that we developed two applications closely integrated with each other – SAS Clients Portal (SCP) and Software currency and Release management tool (SCARM).

THE PROBLEM

At RBS we have a large SAS® estate with an equally diverse range of users and uses. They have different technical abilities and understandings and set of SAS® clients they use on a daily basis is different. What they have in common though is the desire to have the most up to date, reliable software with reported problems fixed as quickly as possible. When it comes to software distribution all of them using the Virtual Client Services solution for the desktop and application virtualization provided by the third party vendor.

Over the years topic of software currency and maintenance releases was omitted due to the fact that focus was put on stability and reliability leaving supporting teams with "don't touch it unless broken" approach towards SAS® platforms support and maintenance. The same applied to SAS® clients re-packaging that was deemed an overly complex, long, costly and error prone process what puts additional blocker on the hotfixes and maintenance releases deployment due to the requirement to have server and client components "in sync". In the end it left us with some platforms being seriously outdated and problems not fixed and pilling up.

We wanted to change this. We wanted to be able to keep our platforms up to date and be flexible and responsive enough to deliver the latest version of SAS® clients to our customers without complicating the process and making it unsupportable.

Our primary development focus was on SAS® 9.4 based platforms with the plan to extend the same approach to all existing platforms across the SAS® estate.

OUR SOLUTION

As mentioned in the introduction, what we came up with to solve this problem was two applications – SCP and SCARM. Both are a PHP based web applications that can be deployed to a Windows server where SAS® clients are installed. As this is standalone server, with no SAS® configured on it, it requires Apache to deliver the front-end interface. At the back-end it uses windows batch scripts to connect web GUI with SAS® installation(s) in case of SAS® Client Portal as well as SAS® platform(s) that Software currency and release management tool monitors.

SCP application allows users to start SAS® Java-based clients and re-direct to the web applications from the browser without using V-App, Citrix or any other virtualization technology. It also doesn't require local administrator permissions or Java installation.

At the moment the following SAS® applications can be delivered through SCP:

- SAS Management Console
- SAS Data Integration Studio
- SAS Information Map Studio
- SAS OLAP Cube Studio
- SAS Enterprise Miner (Java Web Start)
- Flow Manager
- Calendar Editor

The list above is not conclusive and can be extended. For example, we are working on including SAS® Studio (local) and SAS® Foundation in the list.

SCARM application uses custom developed PHP based API to interact with the SAS® platforms. API coupled with the set of scripts is deployed on the mid-tier server(s) that allows the application to collect deployment information across all servers via passwordless ssh, aggregate it and send to the version control repository (Atlassian Bitbucket in our case).

Let's look at each of the applications more closely.

SAS® CLIENTS PORTAL (SCP)

THE HOME PAGE

As SAS® Clients Portal is the customer facing application we aimed to make it simple and intuitive only exposing information that individuals should be seeing and hiding everything else. Our main goal was to make transition from existing V-app packages based way of distributing SAS® clients to the new solution as transparent as possible. We began by gathering requirements and polling various groups of users trying to identify the scope, positive and negative experiences and what users would like to change. Based on that we found that users:

- Do no care about version of clients they use as long as they can connect to the application/platform they need and do the job
- Do not care about compatibility of clients used with applications/platforms they connect to
- Remember application names but do not remember what platform they are deployed on

- Do not like a lot of buttons/information thrown at them

Considering all the above we split all SAS® Clients on per application basis and put authentication framework in place that is linked to the Active Directory. As a result users can migrate from packaged SAS® clients to the new solution without requesting additional access. When they login to the application they can straightaway see those application and associated SAS® clients that they are already authorised to access – nothing more.

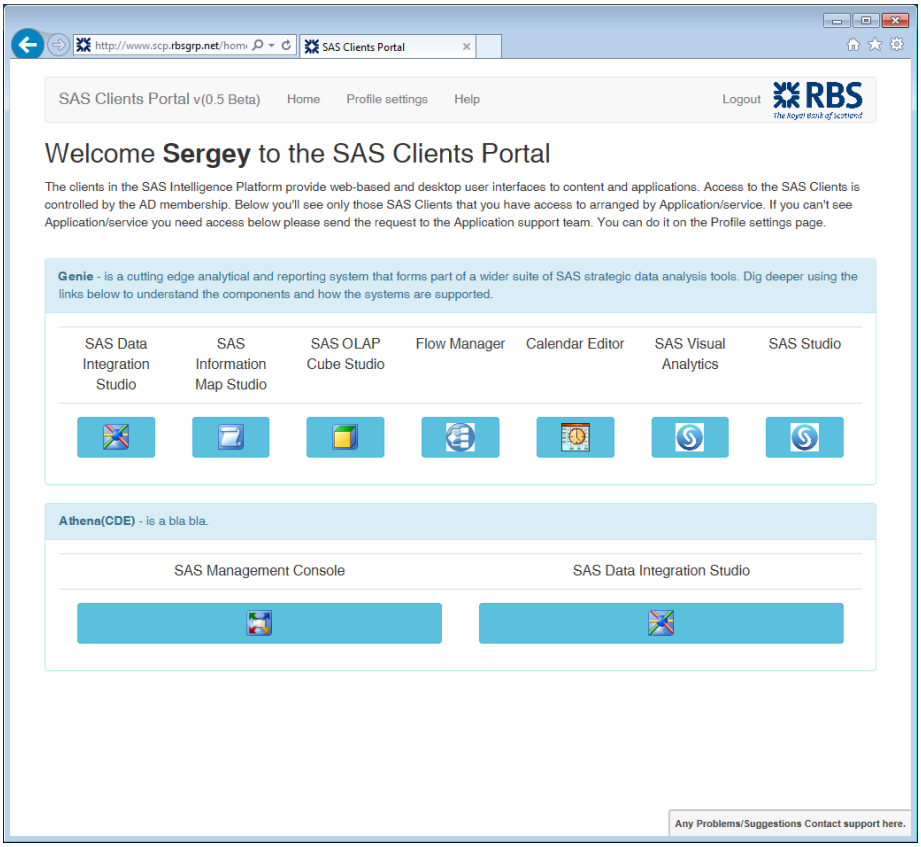

Figure 1. The home page of the SAS® Clients Portal

THE PROFILE SETTINGS

During the logon process each user gets individual space allocated on the server that is called – user profile. User profile is read-only to everyone except for a profile owner. A profile can be completely locked down to an individual user should it be a deployment requirement. User profile is used to store application specific information, connection profiles, application log files and configuration files. User profile is populated automatically with no intervention required from a user except for checking the logs and optionally, though it is not recommended, updating connection profiles.

To protect user experience and make user profile resistant to mistakes, typos and other changes that curious users can make we made it "refreshable" without impacting application functionality or other users. Even if a user decides to wipe it clean and delete all folder and files in it they will be automatically recreated next time SAS® client is started and the only inconvenience that a user will experience will be related to the loss of connection profiles and customizations made via web interface.

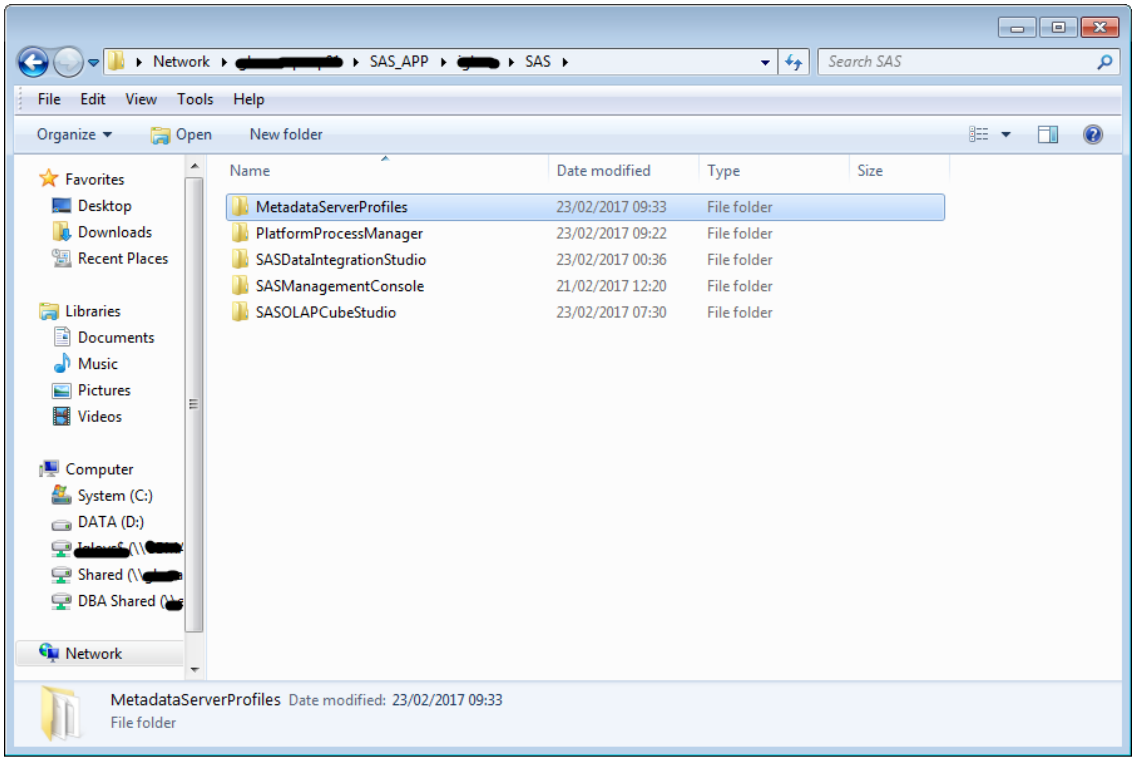

Figure 2. User profile folder of the SAS® Clients Portal application

While all files are stored in the user profile folder on the server there is no need for a user to go there unless specifically requested by a supporting team as the web interface of the Profile settings page provide functionality required to request access to an application, hide/unhide application icons on the front page, manage connection profiles and access the profile folder to investigate log files.

One of the main comments we get when we asked user community – what they want to get out of this tool, was – simplicity and ability to start using it without spending time finding out what package to request or connection details for each application so we added "get default connection profiles" button that downloads pre-defined connection profiles based on the access level also allowing users to upload their own profiles should they want to do it. We even went further and made sure that when a SAS® client is started only relevant connection profiles are presented. For example, if a user has access to both applications deployed on SAS® 9.2M3 and SAS® 9.4M3 and starts, say, SAS® Data Integration studio 4.2 only connection profiles that allows connection to SAS® 9.2M3 based application will be presented so user can't accidently connect to SAS® 9.4M3 based application instead.

Figure 1 above shows home page where a user has access to only one application and it keeps it nice and simple but when a user has access to two or more applications it can get messy so we added functionality to hide application icons from the home page so users can keep only those used daily shown there. It also serves the purpose of controlling application access (this functionality is coming in the future releases) when application support teams will be able to define what SAS® clients users can see for each application they have access to.

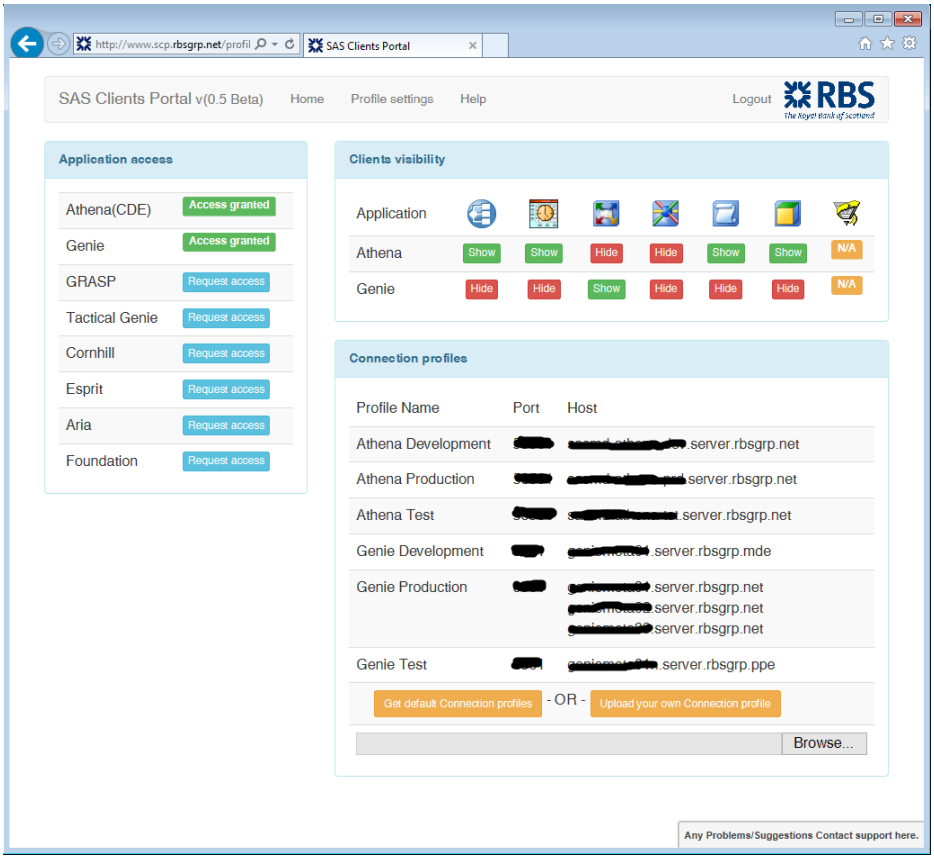

Figure 3. The Profile settings page of the SAS® Clients Portal application

GENERAL APPLICATION SETTINGS

This part of the application that is accessible only by application support teams is used to configure global settings within the application. This includes things such as the Active Directory related settings: connection details (server name/port for the ldap_bind function), Active Directory groups that users should be a member of to be able to login to the application, Active Directory groups that governs access to the specific applications as well as the "request access" messages content, root folder for the users' profiles and contact details for a supporting teams. We also plan to introduce clients specific configuration options in the next version of the SAS® Client Portal application that will allow administrators to control scripts that are used to launch SAS® clients, SAS®Home directories for different versions of SAS®.

Another requested functionality that are in the pipeline for the future release is the control over logging level both on individual and global levels, capability to hide SAS® clients for individual users and notify users of the upcoming maintenance that can impact application availability.

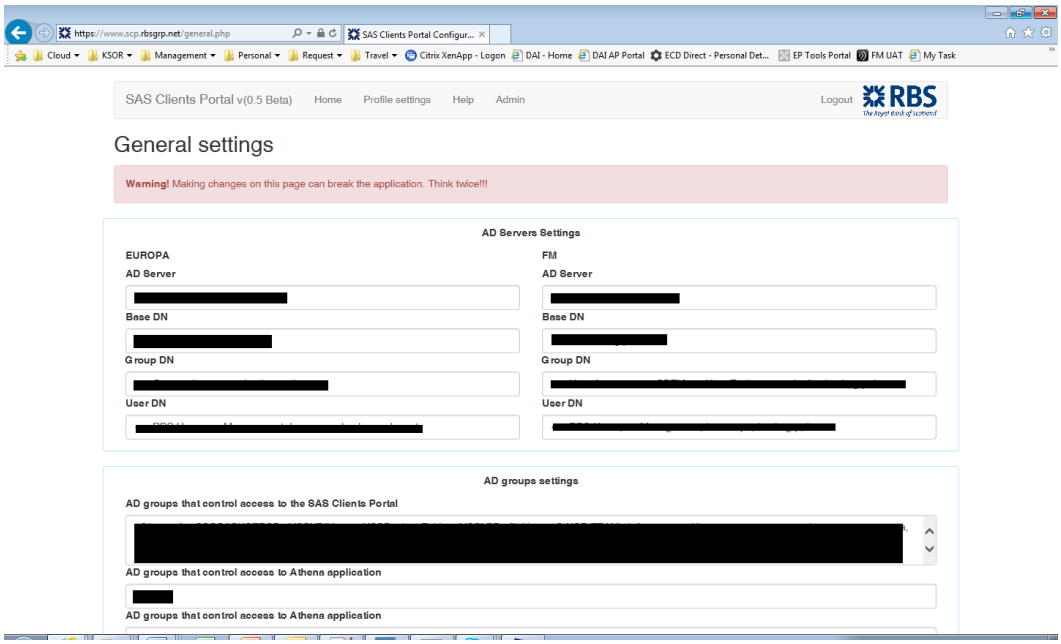

Figure 4. The general settings where global variables are configured

BEHIND THE SCENES

Let's now look at what's happening behind the scenes when users click a button on the home page to start an application.

All scripts used to launch SAS® clients are written using windows batch scripting that we wrapped up into only 2 files: config.cmd and launcher.cmd.

Launcher.cmd

```
138  :smc
139  if %2 == 92 (
140  echo "%username% start %1 %time%" >> %profile_root%\activity.log
141  start/b "SAS Management Console" "%java%" -Xmx512m -Djava.system.class.loader=com.sas.app.AppClassLoader
     -Dsas.app.class.dirs="%launch_path%\SASManagementConsole\9.2" -Dsas.app.class.path=.;"%launch_path%
     \SASManagementConsole\9.2\build" -Dsas.ext.config="%launch_path%
     \SASManagementConsole\9.2\sas.java.ext.config" -Dsas.app.launch.config="%launch_path%
     \SASManagementConsole\9.2\picklist" -Djava.security.policy="%launch_path%
     \SASManagementConsole\9.2\Config\security\java.policy" -Djava.security.auth.login.config="%AppData%
     \SAS\login.config" -Dcache.auth.policy=true -Dsas.app.splash.path="%launch_path%
     \SASManagementConsole\9.2\splash.gif" -Dlog4j.configuration="file:\%launch_path%
     \SASManagementConsole\9.2\log4j.properties"  -Dsas.app.repository.path="%launch_path%
     \SASVersionedJarRepository\9.2\eclipse" -Dsas.services.information.types.path="%launch_path%
     \SASPlatformObjectFramework\9.2\plugins" -Dsas.appdatapath="%profile_path%" -cp "%launch_path%
     \SASVersionedJarRepository\9.2\eclipse\plugins\sas.launcher.jar" com.sas.console.visuals.MainConsole
     -pluginsDir "%launch_path%\SASManagementConsole\9.2\plugins"
142  echo "%username% finish %1 %time%" >> %profile_root%\activity.log
143  goto exit
144  )
145
146  if %2 == 94 (
147      if %3 == M2 (
148  echo "%username% start %1 %time%" >> %profile_root%\activity.log
149  start/b "SAS Management Console" "%java%" -Xmx1024m -XX:MaxPermSize=128m -Djava.security.policy="
     %launch_path%\SASManagementConsole\9.4\Config\security\java.policy" -Djava.security.auth.login.config="
     %launch_path%\SASManagementConsole\9.4\Config\security\login.config" -Dcache.auth.policy=true
     -Dsas.app.splash.path="%launch_path%\SASManagementConsole\9.4\splash.gif" -Dlog4j.configurationfile:"
```

Figure 5. Snippet from the Launcher.cmd file showing start-up command for the SAS® Management console 9.2

Each button on the front page has a corresponding script that calls for launcher.cmd file passing number of parameters that determine what application was triggered. This approach allowed us to reduce maintenance by reducing the number of changes.

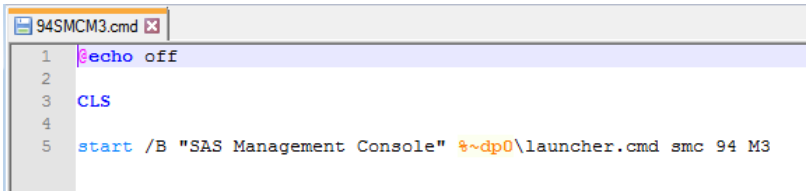

Figure 6. Snippet from the client specific batch file that is used to start SAS® Management console 9.2

To start an application JRE (we use SAS® Private Java Runtime Environment in case of SAS® 9.4 and recommended JRE version for SAS® 9.3 or SAS® 9.2) stored on the server is used that is triggered through launcher.cmd script with number of options that we took from ini files for each SAS® client. This approach allowed us to use UNC paths and keep everything needed for SAS® clients' execution on the server without any need to copy, install or store anything on the clients' machines. We aimed to move all files that are automatically generated when SAS® clients are started from the users' machines though we found that some files are still created and we plan to address this in the future releases of the application.

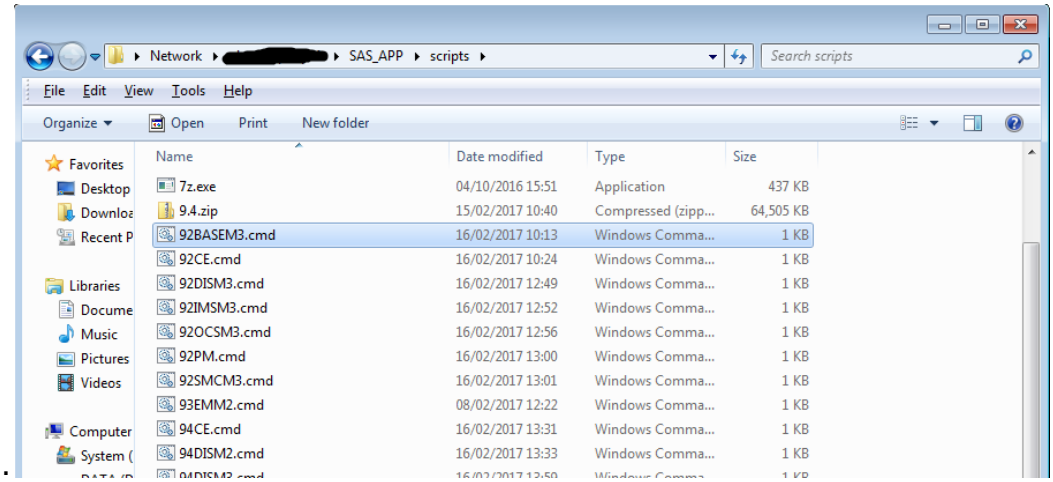

Figure 7. List of Windows batch scripts used to start SAS® clients

CHALLENGES AND LIMITATIONS

As we develop SAS® Clients Portal application we faced a number of challenges that we either resolved or acknowledged as limitations of the current approach. The biggest one is related to the fact that we can't deliver .NET based SAS® clients this way. This rules out SAS® Enterprise Guide and SAS® Add-ins for MS Office that we package and deliver using V-Apps.

We also found that Java Web Start based applications, like SAS® Enterprise Miner, has performance impact if no "direct connection" option is used when application is triggered. To resolve this problem we, as the exceptional measure, used Java 8 and deployment.properties file that are copied to the clients' machines.

Working through the list of SAS® clients we found that SAS® Management Console and specifically GRID Manager plugin will not work unless login.config file is copied and stored locally in the users windows profile so we and to implement this as part of the application start up process.

Performance, no doubt was on the top of the list of requirements that we gathered and while we haven't identified any runtime issues with SAS® clients delivered through the SAS® Clients Portal we observed start time fluctuations depending on the SAS® client triggered. For example, SAS® Data Integration Studio takes the longest to start among all clients we have and we are still going through the testing cycles that involves users groups to make sure that we meet requirements for the application startup (between 30 and 60 s).

SOFTWARE CURRENCY AND RELEASE MANAGEMENT TOOL (SCARM)

One of the biggest problems for supporting teams, as mentioned earlier was, and to the certain extent is, the fact that part of the platforms support and maintenance process related to the software currency that includes hot fixes, maintenance releases and bug fixes is too complex and time-taking. Main contributing factors are:

- Different versions of SAS® software used across existing and newly built platforms and applications
- Different build standards used to deliver applications and platform in the past
- Different processes for hot fixes installation across different SAS® version
- Different release cycles for hot fixes, maintenance releases across different SAS® versions

As the result we had platforms with a backlog of hotfixes 100+ items long and number of long-lasting issues that either require SAS® clients re-packaging or were not implemented as being deemed "impactful".

With the SCARM tool we set our goal to change this situation. We started with SAS® 9.4 based platforms/applications as we have full control of SAS® deployment there and it is easier for us to implement custom-built tools that require web server and/or php being available. Also as per the build standards passwordless ssh is enabled between al servers within the specific environment which also made easier for us to deploy and run the tool we've developed.

As was mentioned before SCARM doesn't have a front-end interface as its main purpose is to collect the deployment data from existing SAS® platforms, analyze it and produce the output that is split into 2 sections:

- deployment registry files that are stored in the version control repository and used to download required hotfixes;
- set of JIRA tickets (one per each hotfix/bundle) that is used to review, approve and plan hotfixes reported;

It uses a mix of php and command line scripts (windows batch scripts that are execute on the Windows server and shell scripts for Linux servers). Let's not look closely at how it works.

DATA COLLECTION

SCARM tool uses api calls to trigger a script that iterates through all servers collecting information about SAS® deployment. We use SAS®.tools.viewregistry.jar to produce the DeploymentRegistry.txt file for each server and then aggregate collected data on per platform basis. One of the biggest challenges for us was the fact that while servers within a tier have identical (or nearly identical) set of binaries deployed collecting deployment data across tiers and then environments and finally platforms meant that we ended up with the high number of DeploymentRegistry.txt files.

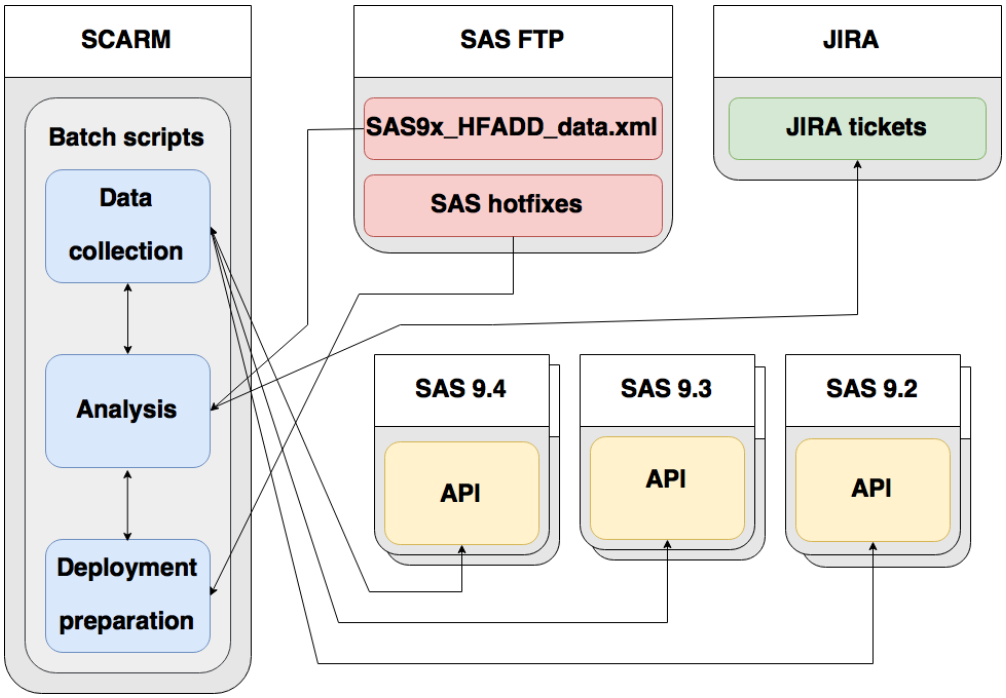

Figure 8. The SCARM tool design diagram

The question was how can we reduce the number of files and ideally limit it to one deployment registry per platform?

To solve the problem above we came out with the solution that includes parsing content of each deployment registry file collected across one tier, comparing components, removing duplicates and merging unique components together. We then repeat the same process for tier specific files for each environment. End result was one deployment registry file per environment that has all components deployed across all servers. Last step of the data collection process was to upload resulted files to the version control repository so they can be used at the next step – information analysis.

INFORMATION ANALYSIS

Once we got deployment information collected and stored in the version control repository we needed to analyze the data. We use SAS® Hot Fix Analysis, Download and Deployment Tool to obtain information about SAS® hotfixes available so we run it a number of time (as many deployment registry files we've collected) and as the output we got number of JIRA tickets created that have information about environment and platform it should be installed on, what issues it fixes and priority level (Alert or not).

JIRA tickets are created in case hot fix is available for one of the SAS® components we have deployed and it is not already installed. SCARM tool checks for new hot fixes every week at midnight on Sun. Deployment registry information is updated once a month after the scheduled maintenance window. Hot fixes review meeting with supporting teams is scheduled 3 weeks before the planned maintenance window to allow reported hot fixes to be reviewed and approved by all stakeholders.

Filter Results: SAS 9.4 hotfixes

Key	P	Summary	Assignee	Status	Labels	Fix Version/s
CBISSBF-580	↑	Hotfix T04009	*Unassigned*	OPEN	94 GenieP GeniePP Linux SAS_Enterprise_GRC_6.1 x64	SAS 9.4 Aug 16
CBISSBF-566	↑	Hotfix N49010	Sirshendu Ghosh	OPEN	94 GenieP GeniePP Linux SAS_Scalable_Performance_Data_Server_5.1 x64	SAS 9.4 Aug 16, SAS 9.4 Feb 17
CBISSBF-565	↑	Hotfix S19009	Sirshendu Ghosh	OPEN	94 GenieP GeniePP Linux SAS_Visual_Analytics_7.1 x64	SAS 9.4 Aug 16, SAS 9.4 Feb 17
CBISSBF-564	↑	Hotfix R94004	Sirshendu Ghosh	OPEN	94 AriaBSE AriaNFTE AriaPROD GenieP GeniePP Linux SAS_Web_Application_Server_9.41 x64	SAS 9.4 Aug 16, SAS 9.4 Feb 17
CBISSBF-563	↑	Hotfix R76013	Sirshendu Ghosh	OPEN	94 AriaBSE AriaNFTE AriaPROD GenieP GeniePP Linux SAS_Management_Console_9.4_M2 x64	SAS 9.4 Aug 16, SAS 9.4 Feb 17
CBISSBF-562	↑	Hotfix R75013	Sirshendu Ghosh	OPEN	94 AriaBSE AriaNFTE AriaPROD GenieP GeniePP Linux SAS_Middle_Tier_9.4_M2 x64	SAS 9.4 Aug 16, SAS 9.4 Feb 17
CBISSBF-561	↑	Hotfix S47008	Sirshendu Ghosh	OPEN	94 AriaBSE AriaNFTE AriaPROD GenieP GeniePP Linux SAS_Web_Server_9.4_M2 x64	SAS 9.4 Aug 16, SAS 9.4 Feb 17
CBISSBF-547	↑	Hotfix R94003	Sirshendu Ghosh	OPEN	94 AriaBSE AriaNFTE AriaPROD GenieP GeniePP Linux SAS_Web_Application_Server_9.41 x64	SAS 9.4 Aug 16, SAS 9.4 Feb 17
CBISSBF-531	↑	Hotfix S28007	Sirshendu Ghosh	OPEN	94 AriaBSE AriaNFTE AriaPROD GenieP GeniePP Linux SAS/Secure_SSL_9.4_M2 x64	SAS 9.4 Feb 17
CBISSBF-524	↑	Hotfix R94002	Sirshendu Ghosh	OPEN	94 AriaBSE AriaNFTE AriaPROD GenieP GeniePP Linux SAS_Web_Application_Server_9.41 x64	SAS 9.4 Feb 17

Figure 9. The list of JIRA tickets produced by the SCARM tool

SCARM tool also performs a check on the existing JIRA tickets to avoid duplication during the weekly run and filters out existing tickets.

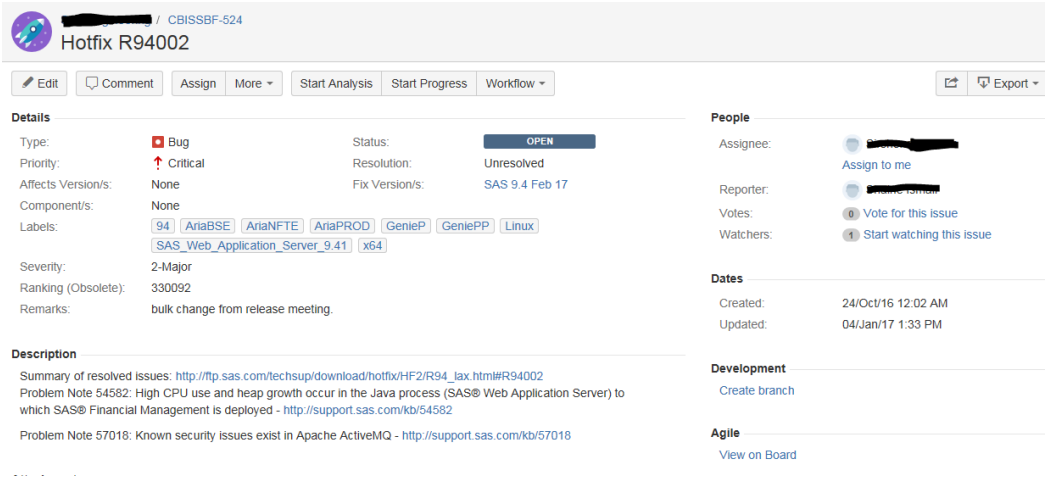

Figure 10. The JIRA ticket produced by the SCARM tool

As the bonus by-product of SCARM development we were able to implement checks that ensure that all servers within a tier are “in sync” i.e. have the same and expected number of SAS® components deployed. If any issue is identified SCARM will send an email to a supporting team notifying of the problem and giving details on what server is “out of sync” and what components are missing or not expected there.

CHALLENGES AND LIMITATIONS

The main challenge that we faced was to adopt SCARM tool to the SAS® platforms built on SAS® 9.2M3 and SAS® 9.3M2 where Web Server wasn’t part of the deployment. In RBS we have mid-tier deployed on

IBM Websphere that is supported by a separate team and either IIS or IHS used as the Web Server/load balancer. This presented us with the problem – how to deploy api scripts that we require to facilitate connection between a platform and SCARM tool? Solution that we found was to adopt and package Apache web server with php module enabled on Master GRID node and use it as the entry point.

Another challenge that we face arises from the fact that we collect data from many servers and then merge it together what "masks" the origin of it and makes it difficult to develop automated hot fixes deployment process and pushes it still to be the manual process when it comes to the hot fixes installation and configuration.

FUTURE WORK

Both SAS® Clients Portal and Software currency and release management applications have plenty of development opportunities.

For SCP they are:

1. SAS® Studio (single user) adoption
2. SAS® Foundation adoption
3. Full Firefox and Chrome browsers support
4. Further enhancements to the SAS® clients performance (especially related to the start-up time)

For SCARM tool they are:

1. Automated deployment of SAS® hot fixes using Puppet automation framework
2. Deployment information collection improvement (making components server-specific when reported).

CONCLUSION

Existing situation around SAS® platforms related to the software currency and release management as well as SAS® clients distribution do not promote or encourage a strong or engaging user experience. With the development of SCP and SCARM applications, RBS has tried to alleviate some of that pain as we as introduce additional features that can help make the life of a SAS® platform administrator looking after a system more bearable.

We have seen that it is possible to create an application that allows a platform administrator to be more responsive and flexible when it comes to the problem resolutions when it involves hot fixes or maintenance releases deployment. We have also seen this as the possibility to re-think, re-design and as the result improve the way SAS® clients are delivered to end-user improving user experience.

We are living "war and peace" every day though we believe that while the grand battle isn't yet won we managed to significantly improve our chances of winning.

ACKNOWLEDGMENTS

The author would like to call out the following people at RBS for their support and encouragement with the publication of this paper: Chris Waite, Shaine Ismail.

RECOMMENDED READING

- *http://getbootstrap.com/*
- *SAS® 9.4 Intelligence Platform: Desktop Application Administration Guide, Eighth Edition*

- *SAS® 9.4 Intelligence Platform: System Administration Guide, Fourth Edition*

CONTACT INFORMATION

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

Sergey Iglov
2 ½ Devonshire Sq, London, EC2M 4BA, United Kingdom
+44 7500 982 852
Sergey.Iglov@rbs.co.uk