MWSUG 2019 - Paper AL-41

No Wrong Door – A case study

Jack N Shoemaker, Greensboro, NC

ABSTRACT

This is a case study of new implementation of SAS® on a Linux platform and the migration and transition of the SAS® ecosystem to an open and flexible topology anchored by the new platform.

INTRODUCTION

The work described in this paper is based on work done at Medical Home Network (MHN) which is a Chicago-based accountable care organization (ACO). The SAS® migration was done in the context of a broader company initiative to broaden interoperability among various players. Briefly stated, the MHN business model is to get pertinent and current information about health-care 'events' to the appropriate health-care providers so that whatever clinical workflow applies can happen. A simple example is an emergency-room visit which may trigger a follow-up from a care manager. The MHN model connects over a dozen federally-qualified health clinics (FQHCs) and nearly two score associated hospitals in Cook County. Given the heterogeneity of the transaction systems feeding the MHN ecosystem, the only way to execute on the business model described previously is to take a 'no-wrong-door' approach to the data. This means accepting data from whomever in whatever format is available. This approach has allowed the MHN model to consume not only standard administrative and observational data, but also what are very non-standardized sources of data for social determinants of health and other sorts of information available in the broader health-care delivery environment. That there are no data strangers and there is no wrong way for data users to request and analyze data was a guiding principle for this implementation and migration.

PRIOR STATE

The prior-state configuration was simple. SAS® was installed on a Windows server. SAS® Enterprise Guide® was installed on this same server for a defined set of users. No metadata server was created. As shown in the diagram below, the set of defined users (noted as MDAT in the diagram) would establish a remote desktop connection to the SAS® Windows server and do their work on that machine. There were three ways to run SAS® programs – using the default SAS® display-manger system, using SAS® Enterprise Guide®, or editing source code in an editor like Notepad++ or Emacs and then executing that code in batch mode from a Windows command prompt.

When run without a metadata server running, SAS® Enterprise Guide® works like an integrated development environment. That is, you get an editor and a way to submit code for execution. You need to be logged into the local Windows server for this to occur because only the 'localhost' machine will be available to SAS® Enterprise Guide®. Running SAS® Enterprise Guide® has several advantages not the least of which is that there is no metadata server to administer and maintain. What you don't get is the ability to fire up SAS® Enterprise Guide® on any network-connected device and then connect to a server running SAS® elsewhere on the network. For simple stand-alone implementations like the prior state at MHN, this approach has much to recommend it as the burden of maintaining the metadata server outweighs the benefit. SAS® Enterprise Guide® is just another local Windows application in this configuration.

The SAS® server in the prior state was not completely isolated. As shown in the diagram below, the server had connections to two servers hosted by a vendor shown as 'Texture' in the diagram. The vendor maintained an MS\SQL-based data warehouse that provided much data to MHN. In addition, the vendor maintained a file server that contained primary data upstream from the data warehouse. The Texture vendor had a connection to a second vendor labeled 'SNC' on the diagram. Primary data from SNC was made available to MHN by way of the Windows file server managed and maintained by the Texture vendor.

Information requests in the prior state usually started with a walk-by or email request to one of the MDAT team. They would RDP to the Windows server; do the work; and then email the results back to the requestor. The work product was often an MS\Excel spreadsheet created by an ODS EXCEL destination. Using the EMAIL device on a FILENAME statement allowed for direct delivery of the work product to the requestor. This arrangement worked well but did not reflect the no-wrong-door philosophy. In fact, there was really no door at all for the business users as they did not have (nor desire) RDP access to the SAS® server.

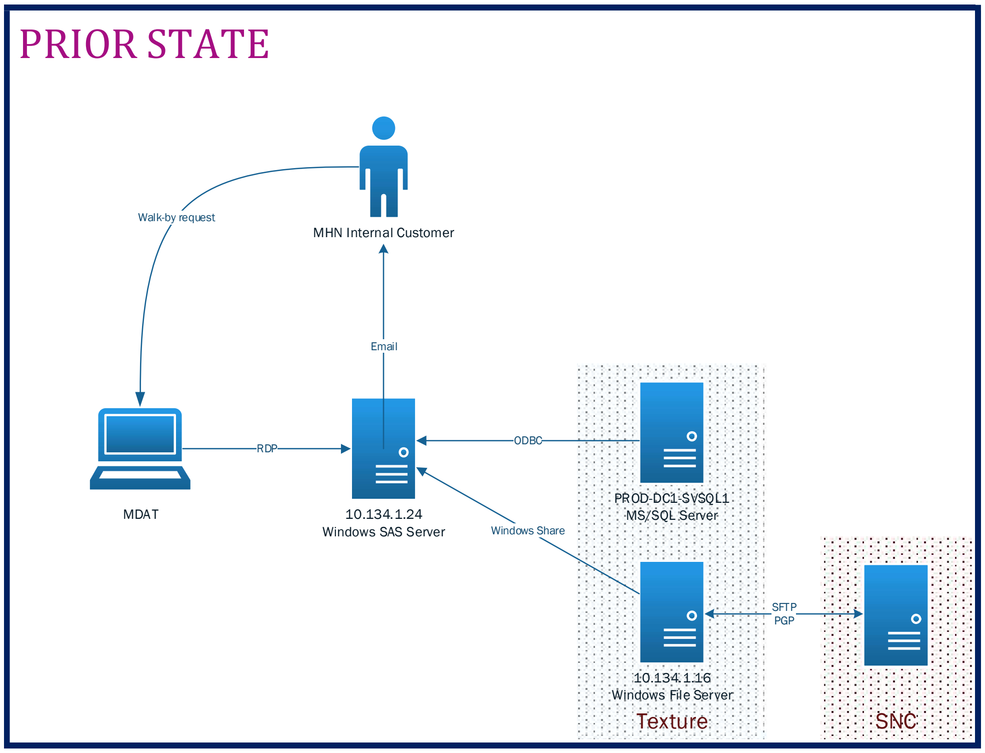

Exhibit 1. Prior-state SAS® environment

CURRENT STATE

For current state a full-blown Office Analytics solution was deployed as shown in Exhibit 2. To simplify administration the metadata server runs on the same physical Linux server as the SAS® application server and object spawner. To simplify the network configuration, Linux-based authentication (what SAS® calls DefaultAuth) was chosen in favor of integrated Windows authentication (IWA). So far, the user burden of remembering a second set of credentials for data access has been outweighed by the benefit of enhanced data access and visibility into the important data that the business users need to run their operation. The spokes of the wheel are described below starting at Noon and proceeding counterclockwise.

MHN DATA SCIENTIST

For the data scientists on staff we installed Anaconda which provides a Python kernel and an R kernel as well as a Jupyter Notebook implementation. As part of the Anaconda installation on individual work stations, we also installed a SAS kernel which allows for the execution of SAS jobs from within a Juypter Notebook. Given that users have a variety of ways to run SAS®, rarely is a Jupyter Notebook used if that's all there is to do. More typically, users will import the SASpy package which allows the user to reference native SAS® data sets as Python pandas. That is, these users use Anaconda to run Python applications that use SAS® data sets.

Although not mentioned above, Jupyter Notebooks worked in the prior-state environment, though only on the local server. That means running Anaconda from the server as opposed to the user workstation. Once the effort was made to make a remote desktop connection to the server, there was little point in using Anaconda except for the coolness factor. It is quite cool!

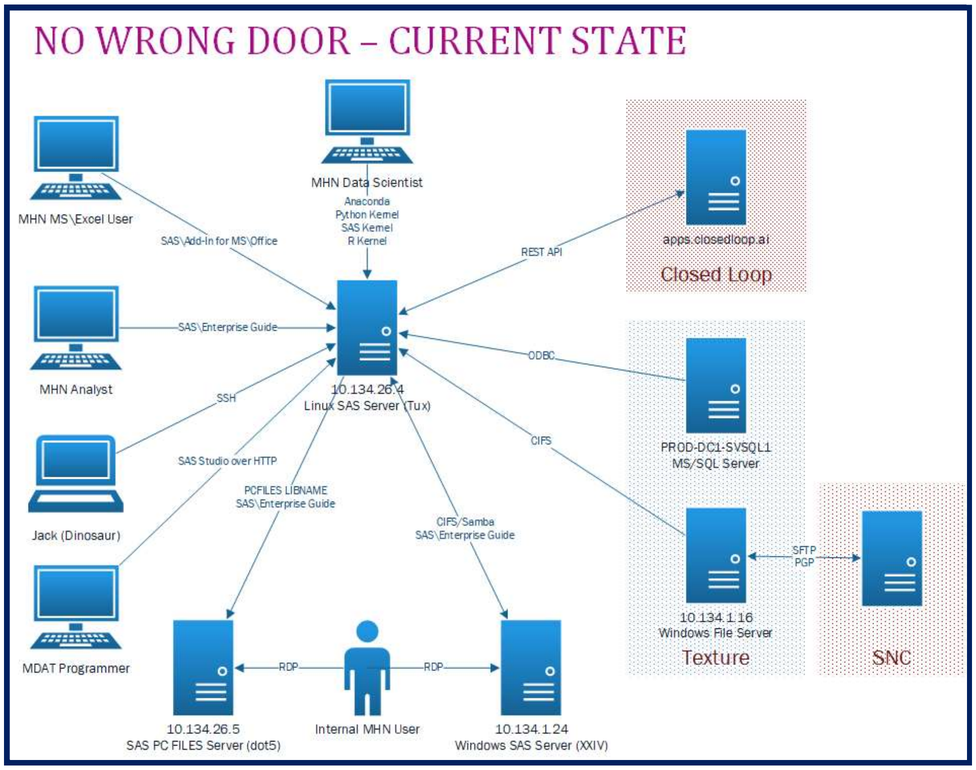

Exhibit 2. Current-state SAS® environment

MHN MS\EXCEL USER

Most of the business users fall into this category. That is, MHN is an MS\Office shop and MS\Excel is the tool of choice for many users. The SAS Add-In for Microsoft Office (SASAMO) provides the connection. SASAMO is a standard MS\Office add-in package, that is, it is a native Windows application that users must install in order to make the SAS® add-in available. Once the add-in is available, users have three main pathways to the data. A popular and bare-bones method is to access a SAS® data set from the 'Data' button on the SASAMO menu bar. Those users that are comfortable with Pivot tables often choose to access data in that manner. The no-extra-fee-required nature of the Pivot drill-down is quite popular as it allows the business user to grab the detail for a specific cell in the Pivot report. Most of the available data sets contain columns suitable for classification and aggregation which fits nicely with how Pivot tables function.

Some users prefer to load the data set details directly into a spreadsheet by applying filtering rules at the point of import. In either case, providing this door for the business users has given the business users a self-service capacity that they did not have in the prior state. This has allowed the MDAT team – the SAS® experts – to focus on creating useful information rather than acting as extract clerks which was too often the case in the prior-state model.

These two methods provide the user raw and unadorned data which is appropriate for those users with a just-give-me-the-data attitude. These users prefer touching and handling the data directly and will summarize and format those data as they see fit. Other users would prefer a more polished result or a more guided filtering path. For these users, reports generated by a stored process are the answer. Creating a stored process from an existing SAS® program is usually as simple as wrapping the source code inside the STP begin and end macros supplied by SAS®. The facility also provides for creating prompts and other sorts of user interaction as needed.

Once you have created a stored process you have the choice of storing the stored process in the metadata server or storing the code as a file on the native file system. Storing the stored process in the metadata server means that the stored process becomes a metadata object just like any other metadata object in the metadata server. That means you can apply the granular security restrictions to the object if there is a business need to do so. If you don't have a business need for the type of security available through the metadata server it is probably best to store the stored-procedure source code as an operating-system file.

MHN ANALYST

This door is like current state with a couple of notable exceptions. The most important is that SAS® Enterprise Guide® can run directly from a user's workstation. As a native Windows application, SAS® Enterprise Guide® can only run from a Windows device. That is, SAS® Enterprise Guide® does not work on the Linux platform because it is a Windows application. Although SAS® Enterprise Guide® works fine from user workstations, some users still prefer to make an RDP connection to a Windows server and then use SAS® Enterprise Guide® from there. The main reason that users do this is that allows them to leave their work running on the server while closing and logging off their desktop device at the end of the day.

DINOSAUR

What could be better than shelling into a Linux server, starting Emacs, and then using that environment to write, modify, and execute SAS® programs? Although the author would answer is rhetorical question, "Nothing", no one else sees it this way. Notwithstanding, there are three score production jobs that run through the cron scheduler at various points during the day, week, and month. These scheduled production jobs are managed, monitored, and maintained from the Linux server which necessitates doing some work from the shell prompt. (You can't do everything in SAS®!)

A suitable workaround for many users is to use Notepad++ to edit source-code files – especially non-SAS® source-code files like shell, Perl, and Awk scripts. Following the no-wrong-door philosophy, the filesystems containing data and source code on the Linux server are exposed as Samba shares which allows users to map a Windows drive letter to these shares and then reference files stored on Linux through that mapped drive. This allows users to use Windows applications like Notepad++ on files stored on the Linux server without the need to log into the server shell. Of course, the Samba share does authenticate the inbound users, but this is transparent to the end user who just needs to know the share name.

MDAT PROGRAMMER

SAS® Studio has been a welcome addition to the environment and has caught on as the user interface of choice for the SAS® programmers. Somewhat surprisingly, SAS® Studio is also popular with some of the business users who just want to see some data. For these users, pointing to a data set and applying a filter gives them just what they need. It matters not that these business users are in a full-blown programming environment. For whatever reason, the guided analysis available in SAS® Studio has not taken root.

Unlike SAS® Enterprise Guide, SAS® Studio does not require any additional software to work. All one needs is a browser on a device that can connect over the network to the Linux server. And that browser need not be running on a Windows platform. This user interface is the future of SAS® due to the platform-independent nature of the application.

SAS® Studio was not available in prior state because it connects to SAS® through the object spawner which does not run in the stand-alone, local-server, prior-state environment. User authentication occurs through the Linux operating system. Since DefaultAuth is used for the metadata server as well, this means that the users have just one 'SAS' password to remember although that password is not synchronized with their Windows credentials.

WINDOWS SERVERS

There are still Windows servers in the ecosystem as shown by the two servers at the bottom of the diagram. One of these servers continues to run stand-alone, local-server SAS®. This was kept in place mainly as a transition issue. That is, while the organization was migrating to a Linux-based ecology, it helped to have the legacy Windows-only environment available. The use of SAS® on this sever has vanished. The other Windows runs the PC Files Server service as well as some other useful non-SAS® applications like Subversion for version control. The PC Files Server service allows SAS® applications running on Linux to establish LIBNAME connections using the PCFILES engine. In the event, this LIBNAME engine is rarely used in day-to-day practice. It is there if needed; however, the XLSX engine provides cleaner and more direct access to MS\Excel spreadsheets.

Storage locations on these Windows servers are available on the Linux platform by way of CIFS mounts. This is the reciprocal action of the Samba shares which provide Windows access to the Linux filesystems. With these two sets of shares in place users and programmers may choose to work on whichever environment suits their prejudices while retaining access to both platforms. As a matter of policy, MS\Office files like MS\Excel spreadsheets are stored on one of the Windows servers while SAS® binary files are stored on the Linux platform. Given the breath of sharing available between the two operating systems, files and data sets may be stored in either location. Setting forth guidelines on what should be stored where helps keeping things orderly though there was some confusion during the migration period when users would typically use both platforms for SAS® during the day.

LEGACY CONNECTIONS TO VENDORS

The existing connections to the Texture and SNC vendors remain in place. The main difference is that the new Linux server accesses data on the Texture file server by way of a CIFS mount instead of a Windows share. More precisely, the Windows share is mounted to a Linux file system using the Linux mount command and a CIFS connection type. Authentication is done using a non-user service account with credentials stored in a hidden file that only the root account can see. Here is what a typical mount command looks like.

```
mount -v -t cifs //10.134.1.16/FileServer /Texture/FileServer -o \
ro,credentials=/etc/.cifsrc.texture
```

Exhibit 3. Sample mount command using CIFS and hidden credentials

RESTFUL API CALLS

The last door to mention is the use of RESTful API calls to push data, to pull data, and to execute programs on remote servers. Even in the prior-state environment there were API calls using PROC HTTP to retrieve data from external sources. For example, here is a code fragment that pulls drug data from the FDA.

```
%let FDA = www.accessdata.fda.gov/cder;

proc http
  method = “get”
  url =http://&FDA./ndctext.zip
  out = filename
;
run;
```

Exhibit 4. Using PROC HTTP to retrieve data from external sources

In addition to pushing and pulling data, RESTful API calls can execute programs on remote servers and return results usually as JSON payloads. The box labeled ‘Closed Loop’ represents a new vendor that provisions just such API calls. (MHN runs data through various models to score members and apply appropriate clinical workflows as part of their primary mission.) What’s most interesting about this vendor is that they would rather their customers run these models using the APIs that they have developed. To that end, the full suite of Python-based APIs is installed in the ecosystem. This allows a data scientist to run models directly by using the SASpy mechanism described at the beginning of this section. Thus, we have come full circle in providing doorways into and out of the ecosystem.

GLOBAL CONFIGURATION

A key design consideration was that all SAS® execution environments – batch, EG, AMO, Studio – use the same global configuration so that LIBNAMES, user-defined formats, macros, and macro symbols are common across all.

Both the metadata server and SAS® Studio have their own autoexec files. This allows sites to customize their environment as they see fit. For this implementation we chose to standardize everything. To do this, the 'usermods' versions of the autoexec files for these two servers contains a single %include sstatement that includes the global autoexec file. That means that there is just one configuration file to manage and maintain. Here are some considerations on what to include in this common autoexec file.

GLOBAL SYMBOLS

It is often handy to have today's date and other 'special' dates available as common macro symbols. This encourages uniformity in coding and application. Here are some from the current-state environment.

```
/* current date as macro symbols */
%let _ = %sysfunc( date(), yymmdd10. );
%let __ = %sysfunc( date(), yymmddn8. );
%let RUNDATE = &__;
%let MMDDYY = %sysfunc( date(), mmddyy6. );
%let YYYYMM = %sysfunc( date(), yymmn6. );
%let MMDDYYYY = %sysfunc( date(), mmddyyn8. );

/* data and time constants */
%let EOT = %sysfunc( mdy( 12, 31, 2299 ) );
%let EODT = %sysfunc( dhms( &EOT., 0, 0, 0 ) );
%let EXCEL_DATE_OFFSET = 21916; /* 01JAN1960 in Excel */
```

Exhibit 5. Date symbols in the common autoexec file

That the current date is expressed in six different formats just underscores how difficult standardization can be. The first symbol, &_, expresses the current date in YYYY-MM-DD format which is ideal for naming files, but no suitable if you want the date as part of a SAS® data set name. For that, the &__ is used. The other renditions of the current date just reflect the reality of the existing code base.

ODBC DEFAULTS

ODBC connections are made through a common macro called %ODBCLibDef. When called without any parameters, this macro creates a LIBNAME called DATAMART which points to a specific schema on the MS\SQL server. The default values for these server details looks like this.

```
/* ODBC defaults */
%let SERVER = PROD-DC1-SVSQL1;
%let ODBCDSN = PROD;
%let DB = CAPDB;
%let SCHEMA = DATAMART;
```

Exhibit 6. ODBC default values in the common autoexec file

LIBNAMES

Defining a set of common LIBNAMES in a common autoexec file promotes standardization of data set references. Here is a small sample from the current environment.

```
    libname auth     "/proj2/prod/Auth/sasdata" access = readonly;
    libname claims   "/proj2/prod/claims/sasdata" access = readonly;
    libname clin     "/proj2/prod/Clinical/sasdata" access = readonly;
    libname elig     "/proj2/prod/elig/sasdata" access = readonly;
```

Exhibit 7. Common LIBNAME definitions in the common autoexec file

GLOBAL OPTIONS

The OPTIONS statement allows you to customize MANY parts of the SAS® environment. Global options should be just that – global and applicable to all programs. The SASAUTOS statement prepends a directory contain user-written macros to the default SASAUTOS path. The FMTSEARCH option creates a search path for user-defined formats in the COMMON.FORMATS and COMMON.EXTRAS catalogs.

```
/* standard options */
options
    sasautos = ( "&CMN./programs", !SASAUTOS )
    fmtsearch = ( work, common, library, common.extras )
    formchar = "|----|+|---+=|-/\<>*"
    missing = ''
    emailhost = mail.texturehealth.com
    emailport = 25
    ;
run;
```

Exhibit 8. Global OPTIONS in the common autoexec file

DEFAULT EMAIL ADDRESSES

There are still plenty of applications that use email to deliver files and advices. The most basic case is that a programmer will send a file or email advice to themselves. To facilitate that common occurrence, macro symbols for the to: and from: email fields are generated using the name of the current user. Note that this works because as a matter of policy the user names are the same in the Windows and Linux environments.

```
/* default SENDER and NOTIFY options */
%let SENDER = %str( sender = %( "&SYSUSERID.@MHNChicago.org" %)
                    from = %( "&SYSUSERID.@MHNChicago.org" %)
                    replyto = %( "&SYSUSERID.@MHNChicago.org" %) );
%let NOTIFY = %str( to = %( "&SYSUSERID.@MHNChicago.org" %) );
```

Exhibit 9. Default email addresses in the common autoexec file

The &SENDER and &NOTIFY symbols are used on a FILE statement associated with a FILENAME reference using the EMAIL device type.

USER-SPECIFIC MODIFICATIONS

Users must be able to override the common autoexec as needed. To accomplish this, the last line of the common autoexec file searches a user's home directory for a file called 'autoexec.sas' and includes it if it were found.

```
/* load user-specific autoexec if found */
%if %sysfunc( fileexist( '$HOME/autoexec.sas' ) ) %then %do;
    %put MHN_NOTE: Now loading user autoexec [%sysget(HOME)/autoexec.sas];
    %include '$HOME/autoexec.sas';
    %end;
```

Exhibit 10. Enabling user-specific modifications in the common autoexec file

CONCLUSION

Over the past three score decades, information-processing technology has become more powerful and ubiquitous. This has changed not only how quickly we can turn around data-analysis tasks but also how we think about approaching those tasks in the first place. User acceptance to any tool or technique is inversely proportional to the amount of re-learning or change is involved. As we become ever more interconnected, the data-sharing demands increase. Given that there are multiple ways for data publishers to surface data and that there are multiple ways that business users want to use and manipulate those data, a no-wrong-door approach ensures that there are no data strangers in the ecosystem. This short essay describes one case example of making this a reality at an organization that embraces interoperability as a necessary and core competency.

CONTACT INFORMATION

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

Jack N Shoemaker
Greensboro, NC
jack.shoemaker@gmail.com