Paper TT11

Building a Scalable Utility Service to Make Multi-lingual Applications Available to the Masses!

Aksel Thomsen, Ferring Pharmaceuticals, Copenhagen, Denmark

ABSTRACT

To reduce the workload from traditionally manual tasks in our clinical trials, we have developed a language independent utility service with a webpage, that integrates with our clinical data repository (SAS® Life Science Analytics Framework).

The service enables the developer to build smaller containerized applications to process these tasks in their favourite language (R, Python, or any other), and make them readily available to their colleagues, without them being required to have any knowledge of the inner workings or language of the application.

The service is hosted in MS Azure and consists of several container apps. The primary being a Shiny app that provides a webpage which works as a graphical user interface and distributor. Each of the individual services are hosted as separate orchestrated app containers.

INTRODUCTION

In the last couple of years, we have seen a push to embrace a rapidly evolving technological landscape and use a more diverse tech stack in the pharma world. Traditionally most have embraced a one-size fits all statistical computing environment (SCE) where SAS® has been the default programming environment.

Moving towards a SCE future with several co-existing programming languages opens new possibilities, but also new challenges concerning how to share newly developed tools and process improvements with your colleagues. The main challenge is, that it is no longer a certainty that you are proficient in the same programming language that your colleague has developed a new the tool in, and you are likely not supposed to become it either.

This paper describes how we at Ferring, has developed a generic utility service to overcome this challenge, and made it possible to share newly developed tools with a colleague despite proficiencies in different programming languages.

TECHNOLOGICAL LANDSCAPE

Our primary SCE is SAS Life Science Analytics Framework (LSAF), that also functions as our clinical data repository. In here most of the trial work is programmed in SAS, although there is also the possibility to code in R in the latest versions. The latest release (version 5.4.1) introduced a REST API feature enabling users to access content from other applications in a safe and robust way.

Besides LSAF we are also using Microsoft Azure as our data science platform when we have the need to analyse unstructured data, such as images, use machine learning libraries, or develop applications. On the platform we develop solutions primarily in Python and R.

THE UTILITY SERVICE TERMINAL

While initially working on a single new tool in Python, a validation application for End-Of-Text packages, we anticipated the need for a more generic system, so that we did not have to develop the infrastructure around a new tool again for all future applications.

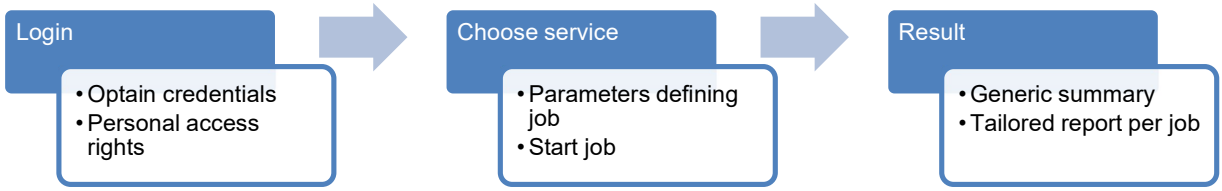
This led to the creation of the Biometrics Utility Service Terminal (BUSTER), which enables the user to use new tools, hosted in Azure, on data from our clinical data repository (LSAF) from a generic graphical user interface (GUI). This ensures efficient development of future applications since we can reuse existing infrastructure with minimal overhead. It also ensures an efficient sharing of new applications by collecting them the same place and having the same interface.

The scope of tools in BUSTER is tools for ad hoc analysis or processes, since running them require manual input, and it is then not suitable for your daily tasks. Instead, the focus will be on developing automation tools around currently heavy or time-consuming processes.

USER EXPERIENCE

The general user experience can be summarised by the figure below (example screenshots can be found in the accompanying presentation [1]):

Figure 1: Flow chart of the user experience in the GUI



The user accesses the utility service by going to the internally hosted webpage. Initially the user will be asked the login using their credentials from our data repository (LSAF). This ensures that the user only have access to relevant data.

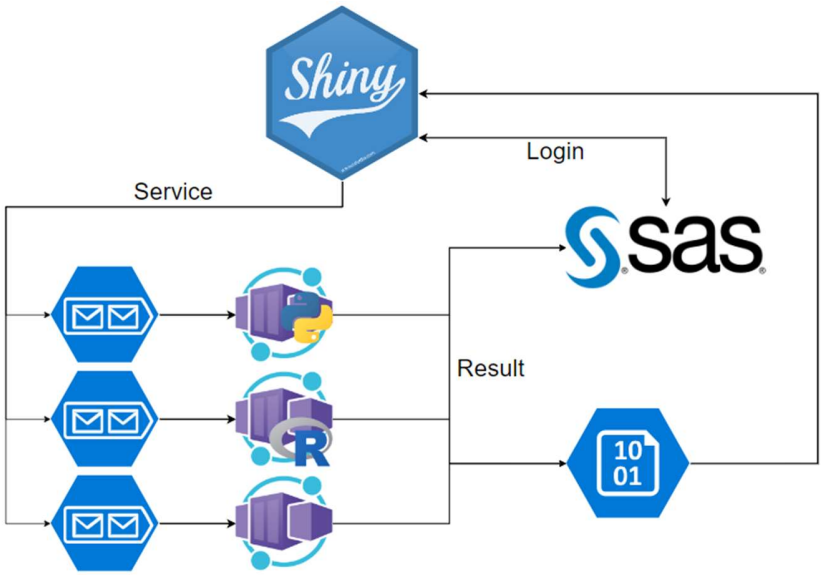
Hereafter the user must choose which new service, or tool, to use, and give a series of input parameters, before starting the job. As an example, the user can select the service to validate a new End-Of-Text package against a previous version. In this service the user must provide the path to each version of the package in LSAF as parameters.

When the job has completed the user will be presented with a summary report providing the status of the job (completed, failed) and a few key metrics defined by the individual service. For the validation example this could be the number of changes, additions, and removals. The user also receives a link to a more detailed report, again defined by the service, that has been uploaded to LSAF.

ARCHITECTURE

The user facing interface is a R Shiny application, that also serves as the distributor of the individual services, that are hosted as separate orchestrated container applications. All components are hosted on the Azure platform, and using the REST API feature they can access LSAF. The general architecture is described below:

Figure 2: Utility service architecture



When a user accesses the Shiny application and starts a new session the following happens:

1. The user enters their username and password in the login page of the application.
2. The Shiny session retrieves a temporary access token from LSAF using the REST API. The token ensures that the applications only have access to same files as the user.
3. The user selects a service, provides the appropriate parameters, and starts a new job.
4. The Shiny application formats the parameters (into JSON) and sends the specification to an Azure Storage Queue. Each service has their own queue, and the Shiny application distributes them accordingly.
5. When a new entry is found in a queue a new container of the corresponding service is started. This container then reads the specifications from the queue and executes the service. At the end each service is writing a summary result to a common blob storage container, and the more detailed report into LSAF.
6. While 5. Is running the Shiny application is checking the blob container for new output to determine that the job has finished. When new output is found, the job is reported as complete, and summary results are shown to the user.

The entire architecture around the individual services is made generic, and we can therefore deploy a new service in a containerised application with a relatively low overhead.

CONCLUSION

Building a general utility service has enabled us to gain the best of both worlds. We are now capable of developing and hosting smaller containerised applications in any programming language and making them available to users in a simple and unified manner, while still enabling the appropriate secure access to data from our clinical data repository.

The BUSTER project is still in its initial phase, but having the developed the critical architecture, we hope to develop many useful applications in the future!

REFERENCES

[1] Presentation PRE_TT1, PHUSE EU Connect 2023, PHUSE Archive.

ACKNOWLEDGEMENTS

I would like to thank my colleagues, and co-developers on the project, Mads Raad and Katrine Facius, for their great collaboration and ideas enabling us to create BUSTER. Without them it would not have been possible.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the author at:

Aksel Thomsen
Ferring Pharmaceuticals
Amager Strandvej 405
2770 Kastrup
Denmark

Work Phone: +45 28 78 75 13
Email: aksel.thomsen@ferring.com