Data Lake as a Service for Open Science

June - September 2021

AUTHOR(S):
Muhammad Aditya Hilmy
CERN Openlab Summer Student 2021

SUPERVISOR(S):
Riccardo Di Maria

CO-SUPERVISOR(S):
ESCAPE CERN team

CERN openlab

PROJECT SPECIFICATION

The project is aimed at deploying the ESCAPE DataLake-as-a-Service over the course of the summer. The DataLake-as-a-Service is a service that allows end-users to interact with the ESCAPE Data Lake in an easily-understandable and user-friendly way. It is based on the JupyterLab and JupyterHub software packages. The Rucio JupyterLab Extension[1] software package shall be used to integrate the service with the ESCAPE Data Lake Rucio instance. The service should be integrated with ESCAPE IAM as the Authentication, Authorization, and Identity (AAI) provider using the OpenID Connect protocol. As a stretch goal, an integration with a latency-hiding layer based on XCache shall be evaluated.

[1] https://github.com/rucio/jupyterlab-extension

ABSTRACT

ESCAPE Data Lake is a collection of interconnected services and tools designed to store and orchestrate huge amounts of data produced by multiple experiments and sciences, which would be especially challenging in the coming years as they will need to deal with a volume of data at the exabyte scale. The Data Lake has a lot of moving parts, and in order to hide the complexities of the Data Lake from the end-users, the DataLake-as-a-Service (DLaaS) is developed. The DLaaS was built on top of JupyterHub running in CERN Openstack, with several additional components to allow it to be integrated with the ESCAPE Data Lake. Some of the features of the DLaaS includes OpenID Connect authentication to ESCAPE IAM, ESCAPE Data Lake browser, file upload, EOS-backed scratch space, multiple environment options, and XCache integration.

Keywords: *ESCAPE Data Lake, Jupyter, Rucio*

TABLE OF CONTENTS

1. INTRODUCTION

In the coming years, we will have new devices and experiments coming online. These new experiments are expected to produce an enormous amount of data, possibly at the scale of exabytes. The European Science Cluster of Astronomy & Particle Physics ESFRI Research Infrastructures (ESCAPE) project is tasked with addressing the challenges of Open Science, one of which is big data organisation, management, and access.

Within the ESCAPE project, the Work Package 2 (Data Infrastructure for Open Science) is responsible for developing, operating, and maintaining a prototype of the Data Lake. The Data Lake is a modular ecosystem of services capable of handling exabytes of data in a scalable way. It consists of multiple daemons, orchestrators, and storage endpoints that work together to store and manage data.

However, the gap between big data management and user analysis is wide, mainly due to the complexity of the Data Lake. In layman terms, it means that it takes quite a lot of steps to get from storing the data, to retrieving them, and to finally interact with the data in a useful manner. Hence, it is necessary to have tools and services that can abstract the complexity of the Data Lake from the end users, so that they can focus less on how to get the data and more on doing science on the data.

This project aims to develop a service to bridge the gap between big data management and user analysis. The service is called “Data Lake as a Service”.

2. ACTIVITIES

The work done during the summer includes several activities, which mostly revolve around the development of the Data Lake as a Service.

a. SETTING UP SINGLE-USER JUPYTERLAB INSTANCE

My first task was to deploy a single-user JupyterLab server instance on top of CERN Openstack. This task was primarily used as some kind of 'training wheel' to familiarise with the environment and make sure everything was ready for the next tasks. This task includes several steps:

1. Provisioning a server node using CERN Openstack

   A node of size m2.medium was provisioned on the CERN Openstack service that is only accessible from within the CERN computer network. The node was installed with CentOS 8 operating system.

2. Installing Docker Engine

   Docker Engine was installed according to the steps outlined in the guide provided by Docker, Inc[2].

3. Evaluating the Rucio JupyterLab Extension for use with Rucio server version 1.25.6

   When the extension was first developed almost a year earlier, ESCAPE Data Lake was running a different version of Rucio server. Therefore, in this subtask, the extension was evaluated to see if it would work with version 1.25.6. It was concluded that the extension worked fine with version 1.25.6.

[2] https://docs.docker.com/engine/install/centos/

b. SETTING UP MULTI-USER JUPYTERHUB INSTANCE

The next task was to deploy a more production-appropriate instance of the Data Lake as a Service using JupyterHub, which supports multi-user configuration. To do this, we used the Zero-to-JupyterHub Helm chart[3] that was supported by the Jupyter maintainers. The instance was running on a Kubernetes cluster on top of CERN Openstack.

Provisioning a Kubernetes Cluster

The first step of this task was to provision a new Kubernetes cluster, separated from the cluster where the ESCAPE Rucio instance and services were running. This was to ensure that any mishaps during the development of the service wouldn't disrupt the operation of the Rucio server. We used CERN Openstack service to create the cluster. As a test cluster, the servers were only accessible from within the CERN network.

Setting up GitOps with Flux v2

To ease the management of the resource manifests, we used the GitOps deployment model. In short, the resources running on the cluster will be declaratively defined as files in a Git repository, and a daemon will fetch the repository to add or delete resources according to the definition files. This allowed us to track any changes happening in the cluster more easily.

To set up the GitOps system, Flux CLI was used to install the daemons in the cluster. The daemons would fetch the resource manifests from a repository we set up using CERN Gitlab[4].

Writing a custom Docker image for the Data Lake as a Service

There are two custom Docker images that were needed for the Data Lake as a Service: one for the single-user container, another one for the JupyterHub. The single-user container is where the user files reside and codes execute. It is only accessed by a single user. JupyterHub, on the other hand, is responsible for handling authentication and dynamically provisioning the single-user containers.

For the single-user container, we used jupyter/scipy-notebook[5] as the base image. We then installed the Rucio JupyterLab Extension and configured the environment variable to point the extension towards the ESCAPE Data Lake Rucio instance.

For the JupyterHub image, we added a modified version of CERN SWAN's KeyCloakAuthenticator[6] to allow the service to authenticate against ESCAPE IAM using the OpenID Connect standard. Additionally, we modified the user endpoint of the JupyterHub API to allow automatic token refresh.

The source code for both images are hosted at CERN Gitlab, and the compiled images are hosted on CERN Gitlab Container Registry[7].

[3] https://zero-to-jupyterhub.readthedocs.io/en/latest/index.html
[4] https://gitlab.cern.ch/escape-wp2/flux-rucio
[5] https://hub.docker.com/r/jupyter/scipy-notebook/
[6] https://github.com/swan-cern/jupyterhub-extensions
[7] https://gitlab.cern.ch/escape-wp2/docker-images

Writing Helm values for Zero-to-JupyterHub Helm Chart

The Zero-to-JupyterHub Helm chart comes with some degree of configuration. For details of what can be configured, refer to the Zero-to-JupyterHub Customization Guide[8].

The first configuration item that needed to be set was the authentication method. The service needed to be configured to use ESCAPE IAM as the identity provider. For this, initially the `GenericOAuthenticator` plugin was used. However, this was later changed to a modified version of CERN SWAN's `KeyCloakAuthenticator` for reasons that will be discussed later.

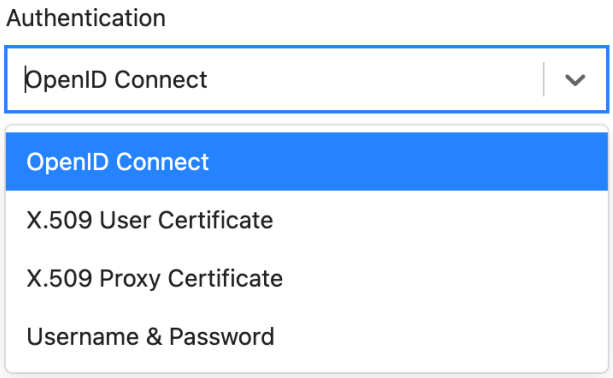
The second configuration item that needed to be set was about how users' home directories would be provisioned. By default, JupyterHub will create one `PersistentVolume` for each user for better isolation. In CERN Openstack, a `PersistentVolume` is implemented as a CephFS Share. The number of Shares that we could create was limited. Hence, it would be challenging to have one Share for each user. The solution was to use a single Share with a large capacity (500GB) for all the users. However, the storage quota would be pooled for all users, instead of each user having its own individual quota.

The third configuration item that needed to be set was the image name of the customized single-user and JupyterHub container.

c. ADDING OPENID CONNECT AUTHENTICATION

Once JupyterHub was configured to authenticate against ESCAPE IAM and the Rucio JupyterLab Extension[9] was installed on the single-user container, users authenticated by ESCAPE IAM would be able to use the service. Ideally, once the users log on using OpenID Connect (OIDC), the extension should be automatically configured with the user credentials. However, the extension only had support for X509 and Userpass authentication (deprecated). Hence, users would need to upload their X509 certificate to the home directory before interacting with Rucio through the extension.

To solve the issue, a new capability must be developed in the Rucio JupyterLab Extension. Pull Request #7[10] to the Rucio JupyterLab Extension repository added OpenID Connect support for authentication. When OpenID Connect authentication is configured, the extension will read the access token from either a file, or an environment variable.



One challenge that we faced was that the access token returned by ESCAPE IAM didn't have an audience claim (`aud`) that was needed by the Rucio instance. Therefore, the token exchange mechanism was used

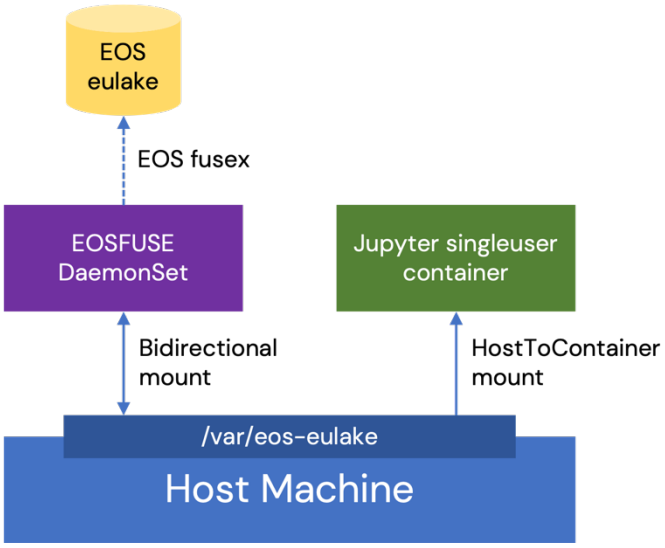
[8] https://zero-to-jupyterhub.readthedocs.io/en/latest/jupyterhub/customization.html
[9] https://github.com/rucio/jupyterlab-extension
[10] https://github.com/rucio/jupyterlab-extension/pull/7

to get a new access token with the correct audience claim. In the DataLake-as-a-Service, this is handled by the authenticator plugin running on JupyterHub. This is the reason why we used a modified version of CERN SWAN's KeyCloakAuthenticator instead of the generic one, because it provided a convenient way of doing the token exchanges.

d. CONFIGURING FUSE MOUNT TO EOS EULAKE

In the Data Lake as a Service, Rucio JupyterLab Extension should operate in Replica mode. In short, Replica mode works by having a Rucio Storage Element (RSE) mounted using FUSE to the single-user container. A FUSE mount allows the files inside the storage element to be accessed by the container as if the files were local, hence allowing the files to be read in a standard way. The RSE that was FUSE-mounted was EULAKE-1, which ran on the EOS file system[11].

To do a FUSE mount to EOS EULAKE, we used EOS FUSEx client that was running in a separate container as a Kubernetes `DaemonSet`. That container is mounted to the host using bidirectional mount propagation and an elevated privilege. The single-user container will then mount to the same folder in the host, but with Host-to-Container mount propagation. An overview of how the mount works can be seen in the image below.



Once the FUSE mount was set up, the next step to take care of was authentication. Since the rest of the service used OIDC tokens as authentication, it is possible to authenticate EOS using tokens as well. EOS comes with support for OIDC tokens. The steps to enable OIDC authentication on EOS EULAKE are described as follows.

First, the userinfo endpoint of ESCAPE IAM needs to be configured in the EOS MGM instance. Also, the tokens would need to have a specific audience claim. Hence, a token exchange is necessary.

Second, the FUSE `DaemonSet` container needs to be configured for SSS authentication. In hindsight, using SSS seems dangerous. However, the SSS keytab is mapped to user `nobody`, which has no privileges. Thus, the SSS keytab can be made public since it would be useless to have only the SSS keytab without a valid OIDC token.

[11] https://eos.web.cern.ch

Third, in the single-user container, the OIDC token must be stored in a file inside the container. That file must have at most 0600 permission. Furthermore, an environment variable (`OAUTH2_TOKEN`) needs to be set to let the FUSEx client know where the token file is located. More details on how to configure EOS for OIDC authentication can be found in the official documentation website[12].

e. CONFIGURING JUPYTERHUB FOR MULTIPLE NOTEBOOK ENVIRONMENT PROFILE

After presenting the preliminary version of the DataLake-as-a-Service to the ESCAPE WP2 community, it was suggested that different sciences could have different requirements in terms of what software packages to install in the single-user image. Thus, it's important to have multiple options of installed packages to accommodate for those different needs.

To do this, a minimal base image[13] was developed and published in the container registry. That image contains all the necessary scripts and packages to properly interact with different parts of the service. Another image[14], that is extended from said base image, was also created with ROOT[15] installed. To let JupyterHub know about those two images, a `ConfigMap` is used to define two environment profiles corresponding to the two images.

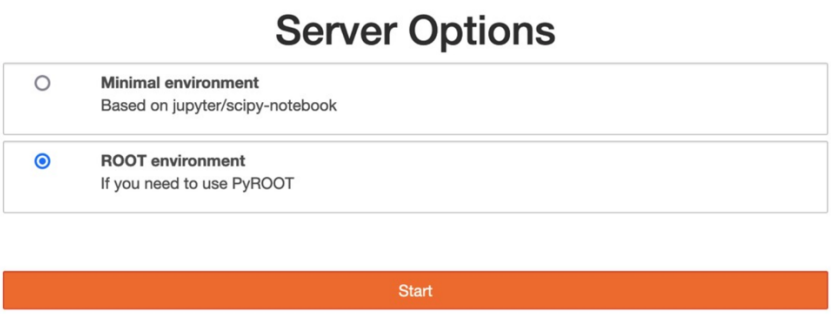
environments.yaml 508 Bytes Edit Web IDE

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: jupyter-environments
  namespace: jupyterhub
data:
  values.yaml: |
    singleuser:
      profileList:
        - display_name: "Minimal environment"
          description: "Based on jupyter/scipy-notebook"
          default: true
        - display_name: "ROOT environment"
          description: "If you need to use PyROOT"
          kubespawner_override:
            image: gitlab-registry.cern.ch/escape-wp2/docker-images/datalake-singleuser-root:fbe8c51b

```

With this configuration in place, the user could then pick which environment to use for the notebook, as seen in the screenshot below.

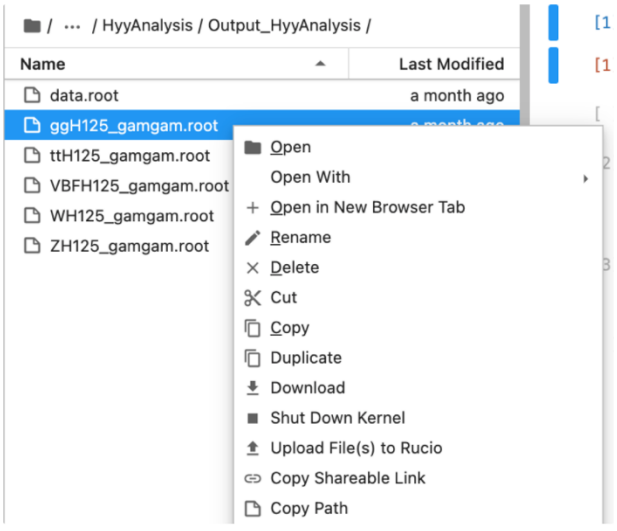
[12] https://eos-docs.web.cern.ch/using/oauth2.html
[13] https://gitlab.cern.ch/escape-wp2/docker-images/-/tree/master/datalake-singleuser
[14] https://gitlab.cern.ch/escape-wp2/docker-images/-/tree/master/datalake-singleuser-root
[15] https://root.cern.ch/

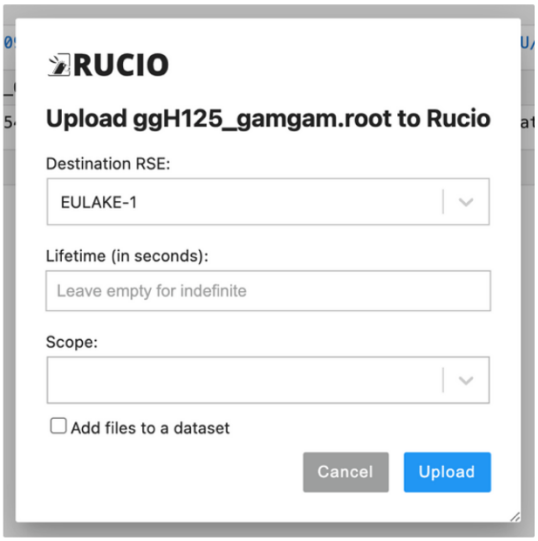


f. ADDING UPLOAD FUNCTIONALITY

Another idea for improvement that was suggested by the ESCAPE WP2 community was to have an upload functionality embedded in the DataLake-as-a-Service. This is useful when the DLaaS is used to generate certain files that need to be uploaded back to the Data Lake. For example, a user might generate ROOT files of some plots, and wants to upload them back to the Data Lake.

Rucio Client included an API to upload files directly to the storage elements. Hence, the upload functionality implemented on the Rucio JupyterLab Extension uses the said API. To allow an easy interaction with the user, a graphical user interface (as seen below) was developed. With the upload functionality implemented, users wouldn't need to upload files using the Rucio CLI. Instead, they could use the JupyterLab file browser to do that. The upload functionality was implemented on Pull Request #11 to the Rucio JupyterLab Extension repository[16].





g. SETTING UP SCRATCH SPACE AND UPLOAD BOOTSTRAP SCRIPT

When a file is uploaded using the Rucio client, the Grid File Access Library (GFAL)[17] is used internally to do the file transfer, and it is running inside the single-user container. By design, the single-user container has limited resources and is short-lived. Thus, it is not suitable for uploading large files which might be a

[16] https://github.com/rucio/jupyterlab-extension/pull/11
[17] https://github.com/cern-fts/gfal2

long-running task. To address this challenge, an alternative means of uploading large files was put into place.

EOS scratch space

We need a place to put large files that are generated by scripts running in the notebook. It is done by mounting another EOS storage to the single-user container, similarly to what has been discussed previously. It is mounted to the /scratch path. This mount path, however, is writeable and designated as a scratch space for temporarily storing files to be uploaded. A cron job will automatically delete files and folders older than two days old to free up space.

Uploading large files inside the scratch space

We then need a way to upload the large files inside the scratch space to a destination storage *without* using the single-user container. Rucio's function as a data orchestrator can be used for the upload process.

As an EOS storage, the scratch space can be registered as a Rucio Storage Element (RSE). In the case of the DataLake-as-a-Service, it is registered as RSE `JUPYTER-SCRATCH-EULAKE`. Then, Rucio can be used to move the files from the scratch space to another storage element, effectively uploading the files.

The reason why Rucio is used for the upload is to avoid the files from being streamed from EOS to the single-user container via FUSEx, and then streamed back to EOS by GFAL. With Rucio (and thus FTS), the files will be streamed using Third Party Copy instead.

After the file upload is complete, the file inside the scratch space is no longer necessary and should be deleted immediately to free up space. This is done by setting the scratch RSE as a greedy RSE[18] and adding a short-lived Replication Rule for the uploaded files inside the scratch space. As the files are stored in a greedy RSE, Rucio will delete the files immediately once the short-lived Replication Rule expires.

To upload files present in the scratch space, the process is as follows:

1. The local path of the file is translated into a full network-accessible physical file name (PFN). For example, file `/scratch/data.root` is translated into `root://eoseulake.cern.ch:1094//eos/ /scratch/data.root`
2. A Replica is added on the Rucio catalogue, to let Rucio know that the PFN exists on RSE `JUPYTER-SCRATCH-EULAKE`.
3. A short-lived Replication Rule is added to keep the Replica inside the scratch RSE during the duration of the upload.
4. Another Replication Rule is added to move the files to the destination storage. The lifetime of this rule is set by the user when uploading the files.
5. Rucio will execute a file transfer to move the files from scratch RSE to a destination storage. The file transfer will be coordinated by File Transfer Service instead of the single-user container.
6. Once the file transfer is complete and the short-lived Replication Rule has expired, Rucio will delete the Replica in the scratch RSE.

To simplify the upload process, an upload bootstrap script[19] was developed.

[18] https://rucio.readthedocs.io/en/latest/overview_Replica_management.html
[19] https://gitlab.cern.ch/escape-wp2/flux-rucio/-/blob/master/escape/jupyterhub/configmap/datalakectl-script.yaml

h. MOVING TO THE PRODUCTION CLUSTER

The JupyterHub deployment should be migrated to the production cluster, so that it's accessible from the outside world. There was one challenge, though: the production cluster uses Flux v1, and the JupyterHub Helm chart did not support Flux v1. Therefore, before migrating the JupyterHub deployment, the production cluster must be migrated to Flux v2 beforehand.

Around the time of this task, the CERN ESCAPE WP2 team was upgrading the Rucio server on the production cluster to a newer version, and it was decided that the service was shut down completely during the upgrade process. Thus, during that period of downtime, we took the opportunity to also migrate the cluster to use Flux v2.

The first step was to create a subfolder inside the production cluster GitOps repo. We then used Flux CLI to configure Flux v2 to fetch the manifests from that subfolder. After the Flux v2 daemons were configured, we then migrated the Helm release YAML files according to the official Flux migration guide[20]. Finally, we copied the manifests for the JupyterHub deployment to the new folder and restarted the cluster.

i. LATENCY HIDING LAYER INTEGRATION

As a part of the work, a latency-hiding layer based on XCache was integrated with the DataLake-as-a-Service. Due to the time constraint of the project, the integration and testing of XCache was conducted at a small scale.

To integrate XCache with the DataLake-as-a-Service, several steps need to be done. First, the XCache instance must be configured to accept a Data Lake RSE as an origin. For the initial testing, EOS EULAKE instance was configured as an origin. Second, the XCache prefix URL must be known to Rucio by registering it as a XRootD proxy of a specific site name. Finally, the Rucio JupyterLab Extension must be configured to present the same site name when making requests to Rucio.

The initial testing was a success. Rucio prepended the XRootD path to the full file Physical File Name (PFN) when the client is configured for the specified site name, as shown below. Reading the XCache logs, it was verified that the file transfer went through the cache, and subsequent requests were served from the cache instead of fetching the files from an origin storage.

```
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| RSE: REPLICA                                                                                                                                                                  |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| EULAKE-1: root://xcache-redirector.cern.ch//root://eoseulake.cern.ch:1094//eos/eulake/tests/rucio_test/eulake_1/ATLAS_LAPP_JEZEQUEL/bd/8f/data.root |
| ALPAMED-DPM: gsiftp://lapp-testse01.in2p3.fr:2811/dpm/in2p3.fr/home/escape/rucio/lapp_dpm/ATLAS_LAPP_JEZEQUEL/bd/8f/data.root                 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

There was another challenge, however. XCache must be configured to accept all the storages in the Data Lake as an origin. The list of storages in the Data Lake is managed by the ESCAPE CRIC and can change at any moment. To overcome this challenge, a cron job was developed to periodically check the storage catalogue and update the XCache configuration automatically.

[20] https://fluxcd.io/docs/migration/flux-v1-migration/

3. CONCLUSION

Over the course of the summer, the DataLake-as-a-Service has successfully been deployed according to plan. An integration with a latency-hiding layer based on XCache was also evaluated successfully. Additionally, some new features were implemented based on feedback from the ESCAPE community.

The DataLake-as-a-Service is expected to help bridge the gap between big data management and end-user analysis, by providing a simple way for end-users to interact with the Data Lake. The DLaaS abstracts the complexities of the Data Lake from the end-users, allowing them to focus on doing science instead of data procurement, thus increasing productivity.

In addition, the development of the DLaaS is driven by the needs of different sciences and experiments. Therefore, the DLaaS attempts to accommodate different requirements and computing models and could potentially be useful for both novice and expert users.

4. FUTURE IMPROVEMENTS

To further improve the usefulness of the DataLake-as-a-Service, new features and functionalities can be implemented as follows:

1. Add support for more Jupyter kernels. As of the time of writing of this report, the Rucio JupyterLab Extension only supports Python kernel.
2. More variety of base images. At the end of the summer project, there is only one base image that is based on Ubuntu. Inputs from the ESCAPE community suggest that having another base image based on CentOS could be useful.
3. Larger-scale integration with XCache.

5. ACKNOWLEDGEMENTS

The DataLake-as-a-Service is made possible with the support from multiple people from various teams from the IT and EP department.