D2.4.2: Lessons Learned on Setup and Operation of a Proto-DAC for Commissioning and Data Preview

WP2.4: Provision of the DAC Platform

Project Acronym	LUSC-B
Project Title	UK Involvement in the Large Synoptic Survey Telescope
Document Number	LUSC-B-26

Submission date	25/NOV/2022
Version	1.0
Status	Release version
Author(s) inc. institutional affiliation	Stelios Voutsinas; Gareth Francis; George Beckett; Bob Mann
Reviewer(s)	Chris Frohmaier (Southampton), Ken Smith (QUB)

Dissemination level	
Public	

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	9/FEB/22	Document structure drafted	MGB
0.9	23/SEP/22	Updated draft	STV, RGM
0.95	6/OCT/22	Internal review and resolution of outstanding comments	MGB
1.0	25/NOV/22	Updated considering reviewer feedback	MGB

Table of Contents

Index of Figures

1 Executive Summary

This document describes a series of experiments to prototype an LSST Data Access Centre in the UK based on deployment of the components of the Rubin Science Platform together with simulated data from Rubin Data Preview 0.1; within the context of LSST:UK Science Centre planning, this prototype is known as UKDAC2.

Working closely with the Rubin Science Quality and Reliability Engineering team - who develop the Rubin Science Platform – the UK DAC team deployed, on UK infrastructure, software components implementing a range of Data Access Centre services using the same set of Kubernetes-based tools as used by the Rubin team for their own prototype Data Access Centre in the US.

Both the US and UK teams are deploying their prototype Data Access Centres in cloud environments, but they are using different flavours of cloud, and this caused some problems within this prototyping activity – e.g., when the Rubin team had taken advantage of vendor-specific features of the particular cloud system within which they had developed the Rubin Science Platform – while some further complications resulted from some design decisions in the software that did not readily support the deployment of multiple instances of it. The positive nature of the collaboration between the US and UK teams led to a number of these issues being acknowledged and addressed, and future releases of the software by the Rubin team should be better matched to the requirements of deploying the Rubin Science Platform in a range of differing environments, as will be needed once the network of Independent Data Access Centres comes online.

2 Introduction

Astronomers will access Rubin Observatory Data Products through a Data Access Centre (DAC). The Observatory will establish at least two such DACs – one in USA and one in Chile. However, there is also an opportunity for an international contributor to operate and provide a DAC – commonly referred to as an Independent Data Access Centre (IDAC) – for the community.

The LSST:UK Consortium plans to operate an IDAC serving both the needs of the UK astronomy community as well as a subset of the international community. This commitment is a key element of LSST:UK's in-kind contribution to the Rubin Observatory.

The Rubin Observatory have defined several different kinds (effectively, scales) of DAC in RTN-003 [1]. LSST:UK has proposed to operate a *Full IDAC* hosting processed images and catalogues, with an indicative storage requirement of 6 Petabytes per year.

A key element of an IDAC service is the user interface – the way in which users interact with the Observatory Data Products. Given the volume of data involved, users are expected to *bring their code to the data*. To support this model, each DAC or IDAC offers an allocation of compute time and working storage alongside the Data Products, with a Full IDAC committing to running the Rubin Science Platform (RSP [2]), which comprises several software models to support use of the Data Products:

- A web portal, called Firefly, which provides a set of common query and analysis views for catalogue data and images.
- A Jupyter Notebooks interface, called Nublado, which allows a user to script more complex analysis workflows using standard community tools and libraries.
- A programmatic API, which would allow large-scale processing of Data Products. The details of the API are, at the time of writing, still to be confirmed.

LSST:UK's strategy has been to reuse Observatory software, tools, and models as much as possible, in the expectation that:

- This would ensure a consistent set of interfaces for the community, potentially moving between different DAC/ IDAC instances.
- This would maximise the capability of the IDAC, for science, given that the Rubin Observatory budget for developing DAC technologies far outweighs what is available in LSST:UK.
- This would allow LSST:UK to exploit other services within the Rubin Observatory, such as the Community Forum.

LSST:UK do plan to extend and adapt the IDAC technologies to enable additional capabilities and datasets within the UK, mostly produced within the Science Centre DEV programme.

Extensions would typically involve, for example:

- Hosting additional data products within the UK IDAC.
- Offering additional software within the Nublado service,
- Extending the Firefly web interface – e.g., adding further portlets for UK-priority functionalities.

2.1 Purpose

In 2019, LSST:UK created a DAC Roadmap [3], which described a sequence of increasingly complex deployments of a proto-UKDAC, through which the UK DAC team would develop the experience needed to support users during Rubin operations.

At the time of writing, the third iteration of the UK IDAC, labelled UKDAC2, has been deployed onto cloud-based infrastructure hosted at the University of Edinburgh. This was initially intended to support UK researchers contributing to Rubin Commissioning (hence the title of this document), but, with the decision that commissioning data will remain in a Rubin-operated DAC in the US, its scope has been restricted to prototyping of RSP functionality with the Data Preview 0.1 (DP0.1 [4]) dataset.

This report documents the experience of setting up UKDAC2, lessons learned and how these will influence plans for further iterations in the roadmap, towards having a mature candidate Full IDAC in time for telescope operations, which are due to begin in mid-2024.

2.2 Glossary of Acronyms

ACF	Advanced Computing Facility
DAC	Data Access Center
DP	Data Preview
DRP	Data Release Processing
EIDF	Edinburgh International Data Facility
HPC	High Performance Computing
IDAC	Independent Data Access Center
LUSC	LSST:UK Science Centre
SQuaRE	Science Quality and Reliability Engineering
RAL	Rutherford Appleton Laboratory
RSP	Rubin Science Platform
TAP	Table Access Protocol

3 DAC Architecture

As illustrated schematically in Figure 1 below, the UK DAC comprises both user-facing and supporting services, which currently exist at different levels of maturity. The astronomers' view of the UK DAC centres on the RSP, likely supplemented by additional user-interface functionality for the *Lasair* alert broker [5] and, possibly, for services developed by other UK in-kind contributions to Rubin operations. Astronomers will use the RSP to access LSST catalogues held in a local instance of the Qserv database system, together with images accessed through the Data Butler [7] and any data within their own allocation of User Storage. The LSST data products will be ingested into the UK DAC following their transfer from the three LSST Data Facilities (in the UK, US and France) and all access will be subject to appropriate authentication and authorisation protocols to enforce LSST data rights. The underlying storage and compute resources will be provided through the STFC-funded IRIS [8] project, and the UK DAC will run accounting and monitoring services to ensure adherence to IRIS policies and procedures.

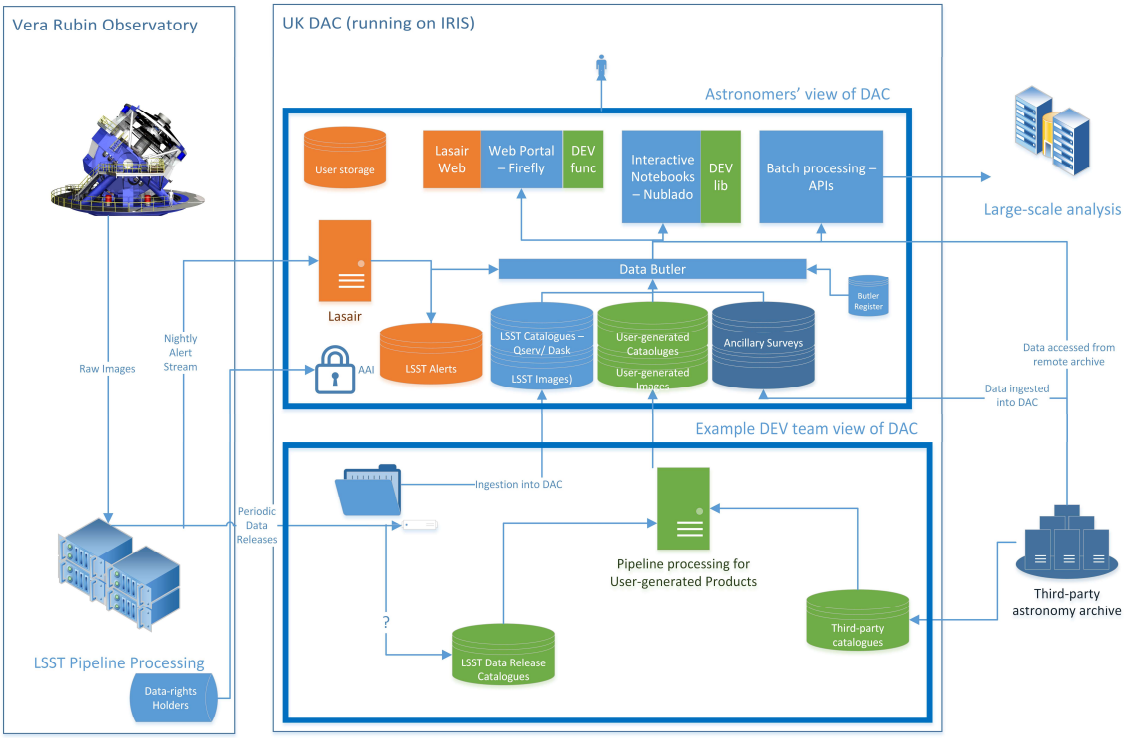

Figure 1: A schematic of the UK DAC architecture.

4 LSST:UK DAC Infrastructure Plans

The architecture outlined in Figure 1 will be implemented on hardware resources provided through the IRIS[1] project, which is providing the infrastructure for STFC-funded activities that have significant computational requirements. To meet these requirements, IRIS provides projects with access to a range of resource types – for example, cloud, HPC, and Grid for compute, plus various classes of storage. Key principles for IRIS are that workloads should be run on shared resources whenever possible, but that projects should have dedicated resources if that is necessary for their science.

The overall LSST:UK Science Centre (LUSC) programme is expected to exercise the full range of IRIS resources. The UK DAC will primarily be deployed on cloud resources but may require the ability to launch large-scale data processing jobs on HPC resources.

Initial DAC deployment experiments were conducted on a range of resources, from dedicated testbed machines located at the Royal Observatory Edinburgh to IRIS cloud resources at RAL, but attention is focussing now on the development of the Somerville cloud, located at the University of Edinburgh's Advanced Computing Facility (ACF). The ACF houses several major research computing facilities, including ARCHER2, the UK's National High Performance Computing system, and the Edinburgh International Data Facility (EIDF), which provides a suite of DAC-like services for a wide range of user communities across many domains. Locating Somerville within the ACF allows the UK DAC to benefit from both its existing physical infrastructure and the relevant expertise within the ACF team. Currently, Somerville is primarily running services for the UK DAC, but the intention is that it will be able to offer data management and analysis services for other STFC projects, as part of the IRIS shared infrastructure.

While Somerville is expected to host most of the DAC components included in Figure 1, the interface between the UK DAC and Rubin Data Release Processing (DRP)—which will be shared between Data Facilities in the UK, US and France—has yet to be defined in detail and that will determine the location of bulk image storage in the UK; it may be possible that image files less regularly required by end-users (e.g., raw and single-visit reduced images) are best stored away from the DAC in a location determined by DRP requirements, with only the most-used images (such as, deep image stacks) stored in Somerville.

[1] IRIS web site: www.iris.ac.uk

5 Experience of Deploying UKDAC2

5.1 Working closely with Rubin SQuaRE team

As our developments have been based on work being conducted from the Rubin SQuaRE team, it has been crucial to create and maintain a close relationship with the core development team of the RSP. This was important for getting guidance for how to get our first version of the RSP up and running, as well as for getting continuing support for handling issues, bugs, and requests for changes. The main interaction avenue has been the LSSTC Slack channel and in particular a dedicated channel named “rubinobs-df-support”. Members of this channel include the core RSP development team, as well as a select number of external teams who are planning to deploy an instance of the RSP, including UKDAC. A second avenue of communication has been occasional telecons, which are held every few months, where all parties provide updates on developments, discuss changes and additions to the RSP itself, and raise issues and difficulties encountered during development and operation.

Throughout our interaction with the Rubin SQuaRE team, the UK DAC team has very much benefited from this close working relationship, as it has made the process of overcoming any hurdles much easier than it would have been without the help of the SQuaRE team. At the same time, as noted by our Rubin colleagues, this relationship has also been beneficial to them, as we have acted as a beta-testing platform, which has led to us discovering and at times providing solutions to various technical issues. Our contributions have also been in the form of providing feedback from the perspective of an external DAC who are also hosting an instance of the RSP, but on a different environment than what is being tested by the Project team. This feedback has been positively received and helped guide the development of the RSP codebase into a more portable and modular form.

5.2 Phalanx repository & custom UKDAC environment

The RSP consists of various components and services, which are brought together and launched using ArgoCD [9] from a central configuration repository named *phalanx* (https://github.com/lsst-sqre/phalanx). The phalanx repository, among other scripts and configuration files contains a “services” directory, which is organized into subdirectories, one for each service. Each service consists of a custom configuration file for each environment, or in other words, for each instance of the RSP deployment (at the time of writing, the repository includes instances such as “*idftest*”, “*idfprod*”, “*tuscon*”, “*summit*”).

Any potential DAC that wishes to run a local instance of the RSP, would have to create a replica of this repository (fork in GitHub terms), and create a custom configuration file for each of these services. The configuration would obviously have to be tailored to the environment on which this instance of the RSP is being deployed, which could include, for example, changes to the host URL, authentication tokens, storage info, etc.

This is exactly what we have done in the UKDAC, and this custom phalanx environment can be found at: https://github.com/LSP-UK/phalanx.

There were a few customizations that had to be made to our environment files:

a) Different hostname – rsp.lsst.ac.uk. (This obviously differs among all environments).
b) Using GitHub as our main OAuth service (Changes included adding our OAuth token and the GitHub teams that would have access to our service).
c) Changing the Vault path (also differs between environments).

d) Customize the Ingress controller (which is deployed on the Kubernetes master node only).

We have also made some additional modifications to our environment, which include customizing to use our own Qserv instance, and modifying components to use NFS storage (these are in development branches, at the time of writing).

While deploying from a forked version worked for us and we were able to get our customized RSP prototype service deployed and tested, it subsequently become more difficult to keep up to date with the main Rubin:SQuaRE phalanx repository, which was fast moving, and had almost daily pushes to it. This required us to run frequent merge processes, which was complicated due to having to trace all changes made to the relevant environment files and locate the changes that we would have to apply to our environment, including bugfixes but also version upgrades of the different components. The merge process often introduced technical issues in the following deploy and test procedures, which in turn required interaction with the SQuaRE team to debug and effort on both sides to be spent on fixing the issues.

Following our close working relationship with the SquaRE team, it was decided that the best route for both us as well as the SQuaRE team in terms of providing support to external DACs, would be to include the customized environment files into the main repository. The UKDAC still maintains a forked version of the phalanx code, from which we deploy our version of the RSP. However, with our configuration files now being part of the main repository, any major changes to versions, or bugfixes would also be applied to our configuration files, thus making it much easier to keep track of changes and stay up to date.

The reason for still running from a forked version is to avoid automatic deployment upgrades from ArgoCD, which could be triggered unexpectedly and without being tested, if we were running off the SQuaRE phalanx repository.

We now have a monthly task of pulling the latest changes into our forked repository, and redeploying, which has become relatively straightforward over time.

5.3 Deployment Tools (ArgoCD, Helm, Vault)

ArgoCD is a core tool used to deploy the RSP. ArgoCD is a GitOps tool which is used within a Kubernetes[12] ecosystem to deploy applications from Git repository. Specifically, ArgoCD monitors the Git repository which contains within configuration files the versions of the applications that are deployed, and can automatically change the state, i.e., by updating the applications that are deployed following a change to the Git repository.

Two other technologies that are relevant to deploy the RSP are Helm and Vault.

Helm [10] is a tool used to manage the installation of Kubernetes applications, similar to how apt/yum is used to install packages on Linux machines. Rubin have created Helm charts for each of the services that the platform consists of, and ArgoCD installs the packages using Helm (https://github.com/lsst-sqre/charts). It is worth noting that upgrading the version of an RSP package means creating a new Docker Image, and then updating the Helm chart associated with that package, and finally updating the ArgoCD configuration for that package.

The other tool worth mentioning is Vault [11], which is used to store Kubernetes secrets. An RSP deployment expects various secrets to exist, including SSL certificates, GitHub

tokens, and ArgoCD credentials. The SQuaRE team has an in-house Vault service running which they use for their deployments, and the UKDAC has been using this in our installations as well. However, it will be necessary to run a local UK-based Vault service, and experiments have begun towards this end.

The ecosystem of Kubenetes, ArgoCD, Helm, Docker, and Vault is designed to produce a deployment process that minimizes operations costs and helps reproducibility. It does come however with a steep learning-curve plus introduces a high complexity when it comes to external developers who may wish to extend any of the RSP components. This has been the case with a few of the components that we would like to customize to UKDAC specifications.

For example, in Firefly there is a list of TAP (Table Access Protocol) services that a user encounters when navigating to the main query page. This is sourced from a hard-coded list in the Firefly code base: modifying this list currently involves modifying the Firefly code, building, and generating, and pushing a new Docker image to DockerHub, as well upgrading the Helm package, and modifying the ArgoCD environment to use our own forked Helm chart repository.

Another example where we have met challenges is with the TAP services. Again, creating a UKDAC-specific version is quite complicated, and an easier route would be developing the changes in a way that results in a more generic codebase, where the features that we would like modified are moved into the configuration itself, thus becoming more extensible & modular.

5.4 Challenges

In producing a UKDAC deployment of the RSP there have been additional challenges aside from customizing the platform components that are briefly described here.

The main source of the challenges has been the differences in the cloud resources on which the RSP is being deployed. The UKDAC Kubernetes cluster is created on an OpenStack service (Somerville). This differs from the Rubin SquaRE deployments which are created on Google Cloud resources. A simple example of a challenge this introduced earlier on, was a requirement for setting up an OpenStack-specific storage provisioner, for which we selected OpenStack Cinder. This required creating a Storage Class in our Kubernetes cluster before running the deployment process.

Furthermore, the Rubin SquaRE deployments use commercial cloud technologies for various aspects of the platform. For example, for storing temporary results of TAP asynchronous queries, Google Cloud Storage is used. As using commercial cloud services is currently out of scope for the UKDAC and there are currently no alternative options in the platform, this limits the capabilities of our TAP services to just synchronous queries. This subsequently restricts Firefly from using the RSP TAP Service and thus our QServ instance, as Firefly uses the asynchronous mode when interacting with TAP services. Work is on-going, at the time of writing, to generalise the temporary storage to any S3-compatible store, allowing us to use OpenStack Rados Gateway storage.

Another area where we have encountered difficulties has been in the process of setting up multiple different environments for various developers wanting to experiment with the RSP. What we have found is that the process of creating a copy of the phalanx repository and having to manually alter most of the individual service configurations to make repeated changes—for example, setting the hostname and vault path--is burdensome and could be streamlined if there was a way to contain these changes in a single configuration file which would propagate them to all the services.

6 Experience of Ingesting Data Products into UKDAC2

6.1 Data Butler

One of the recent developments in the UKDAC and the RSP has been:

a) setting up a Data Butler repository, and
b) using a Butler repository via Nublado notebooks.

The Butler is a data-access framework which consists of a user-facing interface used to access Datastores (Butler Client library) and a Data repository that has been set up using Butler.

As described in the official documentation [13], Butler is a high-level object that provides read access to the Datasets in a single Collection and write access to a single run.

A Butler repository will consist of a configuration file (butler.yaml) a registry (gen3.sqlite3), and a datastore, which will most likely look like one or more directories containing the processed data (such as, FITS images). Note that Butler repositories can be local or remote.

6.2 Preparing a Butler Repository

As described in the Rubin Pipeline documentation (https://pipelines.lsst.io/), the process of creating a Butler repository for a given dataset involves:

1. Install the Rubin Science pipeline software stack, remembering to activate the installation and set up the package stack in each new shell session.
2. Fetch/ download the data (images) to the directory where the Butler will be configured.
3. Run the setup script (https://pipelines.lsst.io/getting-started/data-setup.html)

6.3 Accessing a Butler Repository

On a Python environment such as Nublado, which has the Rubin pipeline software installed, one can then query the Butler repository using the Butler client library.

A quick set of examples of how to do this in Python is shown below:

1. import the butler Class

```
import lsst.daf.butler as dafButler
```

2. Initialize a Data Butler object:

```
butler = dafButler.Butler(repo)
```

3. Navigate available data:

```
registry_list = butler.registry.queryCollections()
```

4. Query the registry for specific entries matching a set of criteria:

```
datasetRefs                                                    =
registry.queryDatasets(datasetType='calexp',dataId={'ban
d': 'g'}, where='visit > 700000', collections=collection)
```

6.4 Open Questions

During this initial evaluation of the Data Butler and following discussions with the Rubin SQuaRE team, one of the takeaways is that the Butler is still relatively new and a work-in-progress.

There was one area where this was relevant and potentially meant that we may have to wait for future developments, and it is regarding creating and storing a public/shared Butler repository.

One of the initial use cases that we are planning to use the Butler for, was creating a VISTA Data repository, and allowing early adopters of the UK RSP to access and query it via Nublado Notebooks.

By creating a Butler repository for VISTA data on our shared NFS node, it meant that all Nublado users would be expected to point the Butler client to this repository and query it. This raises several questions about shared access and permissions.

For any given Nublado user to be able to query the repository, Butler expects the user to have read and write access to the directory, otherwise the client will produce an error. This consequently means that any user would in theory be able to modify files of that directory, which is a potential threat to the stability of the Butler repository. In addition, it is not yet clear to us whether running commands with the Butler client generates any changes to the repository files, for example in the metadata files, configuration, or the registry SQLite database.

Furthermore, an area where our initial evaluation has raised questions is regarding ingesting external data into the Qserv and making it available via the RSP. This requires metadata for the data to be generated and added to the TAP service under the TAP_SCHEMA database. However, currently, the process is complex and requires a manual process of producing metadata files in a format and using a schema (YAML files) described in the RSP documentation. There are a few more steps involved, which include generating custom Docker images and modifying the environment configuration to use those. Overall, there seems to be room for automating some of these steps, and a more integrated solution may involve merging catalogue metadata files we may generate into the main metadata repositories used by the Rubin team.

There is one more area of the RSP that requires manual intervention which we would like to address, and this is the handling of certificates (SSL) for the RSP. Currently, this process involves generating a certificate using a University of Edinburgh service, editing the Vault secrets to include the new certificate, and restarting several Kubernetes pods so that the new certificate is installed. Ideally, we would like this process to be automated, and this is possible if we switch to using a “Let’s Encrypt” [14] certificate and the certificate manager service that comes with the RSP, which would handle the automatic renewals.

7 Lessons Learned and Future Plans

Several useful lessons have been learnt from this exercise:

- Close collaboration with the Rubin SQuaRE team will be vital for effective development and operation of the UK DAC. This collaboration also benefits the SQuaRE team, by providing them with informed feedback on the applicability of the RSP codebase in environments different to that in the Project.
- Any DAC team wishing to deploy its own RSP instance will need to create a set of custom configuration files within a phalanx repository. This could be a separate, forked replica of the Rubin phalanx repository or these configuration files could be kept within the main Rubin phalanx repository. There are pros and cons to both approaches: storing the files in the Rubin phalanx directory allows updates to be made to them by the SQuaRE team, when they make the corresponding updates to their own files, but use of a forked repository ensures that untested revisions are not automatically deployed by ArgoCD.
- The configuration model of the current RSP services is not well suited to the deployment of multiple (e.g., test and development) environments.
- The SQuaRE team use the Kubernetes-based ecosystem of ArgoCD, Helm, Docker and Vault to automate deployment of RSP services. This automation comes at the cost of a steep initial learning-curve for external teams wishing to deploy the RSP and adds complexity to the process of customising RSP components. Some customisations – e.g., deployment of a local Vault service to store deployment-specific secrets – will be inevitable.
- While the use of tools like Kubernetes within cloud environments is intended to simplify the deployment of software on different hardware, the tendency to take advantage of vendor-specific features within one cloud system can complicate its deployment within a different cloud.

Near-future plans will address some of the issues noted in this report:

- The UK team will test a generalised version of the Rubin TAP services, able to exploit a range of temporary storage options, to expose the full TAP functionality through the RSP.
- The UK team will set up a UK-based Vault service in which to host UKDAC-specific secrets.
- Early science users, who will validate fused VISTA-HSC datasets created by the WP3.5 DEV team, will be invited to test the UK DAC and provide feedback on its performance and functionality.
- The UK team will deploy a second, UK-based RSP instance aimed at test and development. This will allow the current RSP to be exposed to early users, with less risk of interruption or downtime.

8 References

[1] O'Mullane W., et al, Guidelines for Rubin Independent Data Access Centers, https://rtn-003.lsst.io/, (revision of 2021-08-07)

[2] Dubois-Felsmann G., et al, Science Platform Design, https://ldm-542.lsst.io/, (revision of 2019-01-29)

[3] UK DAC Roadmap, https://lsst-uk.atlassian.net/wiki/spaces/LUSC/pages/921337857/Phase+B+DAC+Roadmap

[4] Vera C. Rubin Observatory Documentation for Data Preview 0.1, https://dp0-1.lsst.io/

[5] Lasair-ZTF prototype, https://lasair-ztf.lsst.ac.uk

[6] Qserv documentation, https://qserv.lsst.io/, (revision of 2022-09-22)

[7] Jenness T., et al., 2022, The Vera C. Rubin Observatory Data Butler and Pipeline Execution System, to appear in Proc SPIE 12189, "Software and Cyberinfrastructure for Astronomy VII", Montreal, CA, July 2022 (https://arxiv.org/abs/2206.14941)

[8] IRIS, (www.iris.ac.uk)

[9] ArgoCD, (https://argo-cd.readthedocs.io/en/stable/)

[10] Helm, (https://helm.sh/)

[11] Vault, (https://www.vaultproject.io/docs/platform/k8s)

[12] Kubernetes, (https://kubernetes.io/)

[13] Butler Design, (https://dmtn-056.lsst.io/index.html)

[14] Let's Encrypt, (https://letsencrypt.org/)