Converting Rubin Observatory's Data Butler to a client/server architecture

Tim Jenness[1], David H. Irving[2], James F. Bosch[3], Andrei Salnikov[4], Nate B. Lust[3], and Russ Allbery[1]

[1]Vera C. Rubin Observatory Project Office, 950 N. Cherry Ave., Tucson, AZ 85719, USA
[2]Vera C. Rubin Observatory/NSF NOIRLab, 950 N. Cherry Ave., Tucson, AZ 85719, USA
[3]Department of Astrophysical Sciences, Princeton University, Princeton, NJ 08544, USA
[4]SLAC National Accelerator Laboratory, 2575 Sand Hill Rd., Menlo Park, CA 94025, USA

ABSTRACT

The Vera C. Rubin Observatory's Data Butler provides a way for science users to retrieve data without knowing where or how it is stored. In order to support the 10,000 science users in a hybrid cloud environment, we have to modify the Data Butler to use a client/server architecture such that we can share authentication and authorization controls with the Rubin Science Platform and more easily support standard tooling for scaling up backend services. In this paper we describe the changes being made to support this and some of the difficulties that are being encountered.

Keywords: Vera C. Rubin Observatory, Data Management, Web Services, LSST

1. INTRODUCTION

The Vera C. Rubin Observatory is located on Cerro Pachón in Chile and consists of the 8.4 m Simonyi Survey Telescope and the 1.2 m Rubin Auxiliary Telescope. Over the course of 10 years the Rubin Observatory will take data for the Legacy Survey of Space and Time (LSST) and deliver 11 data releases.[1] The Rubin Observatory generates a data volume of approximately 20 TB per night and this data is handled using a Data Management System[2] that transfers the files from the summit to the US Data Facility hosted at the SLAC National Accelerator Laboratory (SLAC), triggers the pipeline processing, and makes the data available to the data access centers. The LSST Science Pipelines[3] will be used to process the data and are designed such that neither the pipeline algorithmic code nor the scientist inspecting the data know where the data are stored or what data format they are stored in. The software abstraction layer we use for this is called the Data Butler.[4,5]

2. THE DATA BUTLER

The Rubin Observatory Data Butler concept has been in the data management plan from the very early days of the project.[6,7] In 2017[8] we decided to rewrite the Butler library from scratch to take into account new technologies, clarified requirements,[9] and lessons learned from previous implementations. This version, colloquially known as the "generation 3 middleware", was formally accepted by the project in 2022.[10]

The Data Butler has been described in a previous paper[5] and a simple block diagram is shown in Fig. 1, but the important concepts of dimension records, data coordinates, collections, dataset types, and formatters are summarized here.

2.1 Dimension Records and Coordinates

The Butler registry assigns datasets specific coordinates within a dimensional space (what we call a "dimension universe"). These dimensions generally refer to scientifically useful concepts such as the instrument, an observation on the sky, or a patch on the sky. Some common examples of dimensions used at the Rubin Observatory are listed in Table 1. These dimensions and associated coordinates (which can only be strings or integers) are created when new observations are ingested or when new instruments or skymaps are defined.

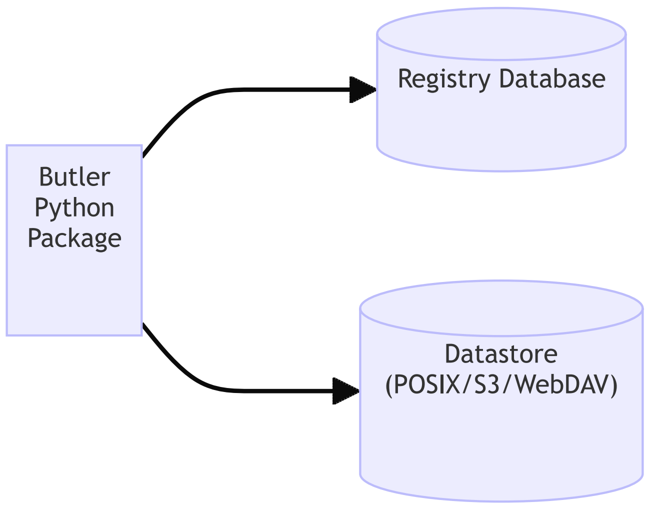

Figure 1. The direct butler architecture. A client connects directly to a database and a datastore. The datastore can be a POSIX file system, WebDAV server, or an S3 object store. The database registry can either be SQLite for testing or PostgreSQL.

Table 1. Common dimensions present in the default dimension universe.

`instrument`	The instrument that generated this data.
`band`	Waveband of interest.
`detector`	A specific detector within the instrument.
`physical_filter`	Filter used for the exposure.
`day_obs`	The observing day.
`group`	Grouping identifier.
`exposure`	Individual exposure.
`visit`	Up to 2 exposures taken sequentially at the same sky position.
`tract`	Tesselation of the sky.
`patch`	Patch within a tract.

2.2 Dataset Types

A “dataset type” describes an input or an output from a data processing task in a general way. The name of the dataset type is meaningful to the pipeline user or scientist with examples such as “raw” (a file that comes straight from the instrument), “calexp” (a calibrated single exposure), or “coadd” (a generic stack of multiple images). The dataset type also specifies the relevant dimensions – a “raw” might be described by `exposure`, `instrument`, `detector` but an output co-add might be described by `tract`, `patch`, `skymap`, and `band`.

The final part of a dataset type is the “storage class”. The storage class is a proxy for the Python type that will be used. Along with the Python type it can also describe individual components (such as metadata or image, variance, or mask) which are declared to be retrievable independently, and also specify conversion methods to handle cases where a pipeline task returns something that is compatible with the underlying storage class but is not identical.

2.3 Collections

Every dataset in a Butler repository must be stored in a `RUN` collection. This collection can be thought of as representing a folder or directory, although there is no requirement that a file will be written somewhere. A single `RUN` collection can contain any number of datasets so long as a dataset does not already exist in that collection with an identical dataset type and data coordinate.

A `CHAINED` collection consists of a list of collections that will be searched in order. These collections can contain more `CHAINED` collections. A `CHAINED` collection is usually created during a data processing campaign and generally consists of all the input collections and a timestamped collection. This allows a single output collection to be used to find all the inputs and outputs and the timestamped name allows resubmissions to be stacked using additional timestamped names.

A `TAGGED` collection is used to create a bespoke collection containing a curated list of datasets. The only requirement is that there are no duplicate dataset type / data coordinate combinations.

A `CALIBRATION` collection is a special type of collection that does allow duplicated dataset type and data coordinate combinations but uses validity timespans to break any ambiguity. This allows a raw exposure, which is implicitly anchored in time, to be associated with the relevant processed calibration datasets.

2.4 Formatters

A Butler datastore provides an additional abstraction layer within Butler that decouples the serialization and storage of a dataset from knowledge of its existence. Datastores exist that can write directly to the Rubin Observatory metrics database[11] and talk to multiple datastores as if they were one datastore, but the most commonly-used datastore implementation is one that reads and writes files. When a file-backed datastore is given a Python object it uses a Python formatter class to serialize it to bytes and, conversely, when someone requests a dataset the datastore reads the bytes and uses the formatter to convert it to a Python object. The choice of formatter class is controlled by datastore configuration and is selected by comparing with the storage class or dataset type.

3. MOVING TO THE CLOUD

The original operating plan for the LSST was to host the data at the National Center for Supercomputing Applications in Champaign-Urbana.[12] In this scenario the 10,000 science users that the archive was sized to support would all be given accounts directly at the archive center and access would be controlled directly on the file system using ACLs and on the databases using database accounts. With the move to SLAC as the archive center[13] this approach was no longer feasible given that SLAC is a Department of Energy facility which has much tighter controls over who can be issued computer accounts and a much more detailed background check. Ten thousand accounts would take a significant amount of time to be processed and it was unacceptable to the project that some science users with data rights might not be able to acquire accounts at all.

To solve this problem the project adopted a "hybrid cloud" solution[14] where all the science users will be using the Rubin Science Platform[15] and be hosted on a commercial cloud where they will be given access via their educational instititution's accounts, but the data would be stored at the US Data Facility at SLAC. In this way the DOE would not need to vet 10,000 users and we would proxy access through our infrastructure. We prototyped cloud hosting on Google using one of our Data Previews[16] and demonstrated that the deployment, data access, and user support would work correctly with a few hundred science users. The Data Previews, though, stored the data in the cloud and used shared authentication credentials for data access – for LSST Data Releases the data volume will be too large to host on the cloud, with our current budget profile, and we are required to provide per-user and per-group access controls.

To support this hybrid cloud approach and to provide scalability for 10,000 users on day one of a data release, we were required to rethink our approach to the Butler and re-engineer it to use a client/server architecture. In this way the server can use standard Rubin Science Platform authentication,[17] utilize signed URLs to access the data at the USDF without requiring SLAC accounts,[18] and allow for a backend infrastructure that can make use of cloud resources to scale with spikes in load.

4. MIGRATING TO A CLIENT/SERVER ARCHITECTURE

The delivered Butler uses a direct connection to a database (generally PostgreSQL) and direct access to either a POSIX file system, an object store (supporting either Google Cloud Storage or Amazon S3), or a WebDAV server. There is no fine-grained access control restricting database access and object store access is generally all or nothing. For a POSIX file system datastore it is possible to restrict file system access based on ACLs. We call this implementation the "direct butler" to distinguish it from the "remote butler" that connects to a butler server.

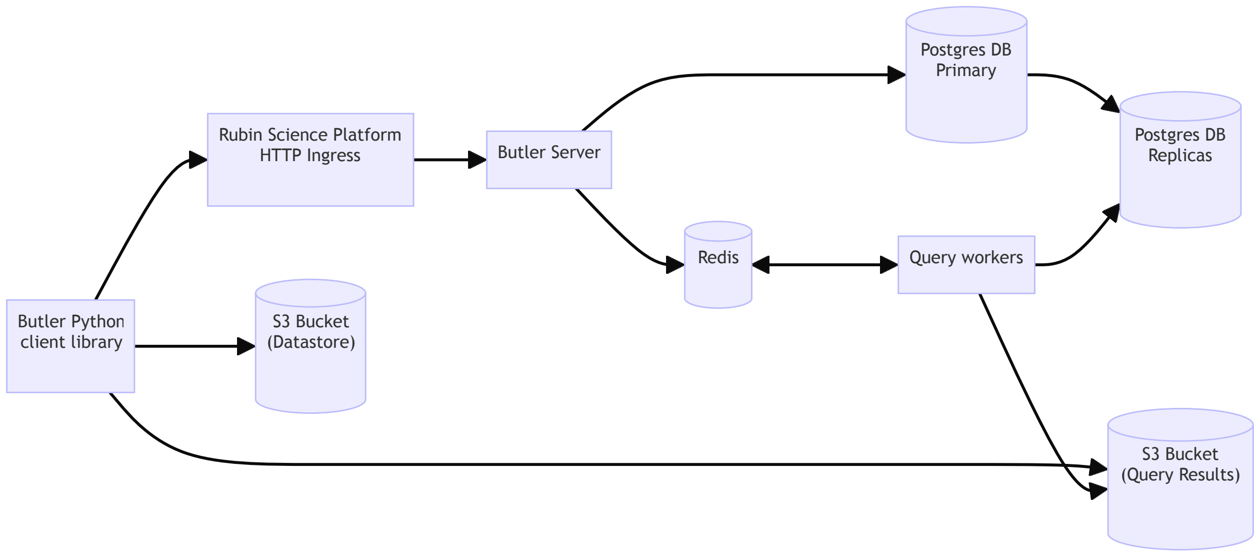

Figure 2. The client/server butler architecture. A client connects directly to a service which mediates the database connection and provides signed URLs to the S3 buckets.

For Data Preview 0.2[19] this was the system we used – anybody could read and write anything with no group or user restrictions. For a deployment that was mainly used as a test bed and which was using previously published simulated data this was not an issue. This approach to authentication and authorization is acceptable when running batch processing at the USDF where the campaign processing team and developers are Rubin Observatory staff, but a new approach is needed for the hybrid cloud scenario with external science users for formal data releases.

For a formal data release or access to the prompt products database, the requirements become stricter:

- Data products are only visible to data rights holders until the two-year proprietary period is exceeded.
- Data rights holders can all see the formal data products but can not delete or modify them and can not write to any of the data release collections.
- Data rights holders should be allowed to store their own derived products and no-one else should be able to access them or even know they exist.
- Data rights holders should be allowed to collaborate with others and make use of shared collections that are private to their group.
- File quotas will be used to ensure fairness, although a petition can be made to increase the default quota.

To satisfy these requirements the only option is to move to a web services architecture. We refer to this implementation as the “remote butler” where the client sends messages to a server, the server checks the authentication tokens, decides whether the user has access to the dataset or collections, and then uses the “direct butler” to process the request. The server can also sign URLs and return them to the client which is then responsible for retrieving the dataset and reading it into memory. A block diagram with the architecture is shown in Fig. 2.

Replacing direct connections to databases and file systems with a web service does complicate some of the Butler design and we have been forced to make some fairly major internal changes. The “direct butler” implementation must still work (that is how the server communicates with the registry) but certain assumptions are no longer valid.

For example, during graph-building, the process whereby we take a pipeline definition and combine it with a butler query to generate an execution workflow,[20] the direct butler makes use of temporary tables in the database. In a stateless client/server with load balancing there is no mechanism to ensure that the client is talking to the same web service or the same database server and so long-lived temporary tables may no longer be viable without significant re-architecting. In the following sections some of these difficulties are discussed and solutions proposed.

4.1 Configuration

The direct Butler supports configuration via a YAML or JSON file. The system was designed from the beginning to allow a butler to be constructed from a single configuration file which could be augmented by the user to override any of the settings including output file formatters, file templating, addition of chained datastores, file-writing compression parameters, and even storage class definitions. If they inadvertently modified the registry or datastore definitions in an incompatible way the worst that would happen is that they wouldn't be able to connect to the butler or they might write a file to a datastore that no-one else could read. Furthermore, many of the configurations allowed a full Python class name to be specified, for example in the definition of a storage class, or controlling which datastore was loaded.

In a client/server environment this sort of configuration flexibility is not at all desirable for two important reasons. Firstly, a server process should not trust anything it gets from a client that can not be vetted, and secondly, a client should not load arbitrary Python code based on a message it gets from a, potentially compromised, server.

This realization is requiring us to fundamentally adjust how we treat configuration in the butler:

- The ability for a client to inject arbitrary configuration overrides should be reassessed and possibly be replaced by an explicit system for allowing configuration of formatters and write recipes (which are important abilities during testing and development).
- Storage class definitions, the underlying Python types of datasets stored in a Butler, must solely be read from the client system. The server should never need to load any Python code associated with the datasets being stored, and in particular, a server deployment should not need to have access to the full suite of LSST Science Pipelines Python code.
- Storage classes must be versioned for each data release (Python types might evolve over the years even if the storage class name remains the same).
- Formatters must now be stored in the server as a label rather than a Python class and configuration in the client must map that label to a Python class. A client can not import a fully-qualified Python class name coming from the server.
- The server issues signed URLs for file upload and therefore the client can not be responsible for defining the file naming template.
- The client should not ever see registry database configuration.

4.2 Registry Database Queries

With potentially 10,000 simultaneous users running queries for datasets and dimension records, it is impossible for one server to be able to handle that load with one database. To solve this problem there will be multiple strategies:

- Servers will pre-fetch some database dimension records and collection information.[21]
- Multiple server instances will be deployed with load balancing.
- Multiple database server instances will be deployed.
- Servers will not run database queries directly, instead they will spawn workers that will run the queries.
- The number of queries a single user can execute will be limited and a queuing system employed.

4.2.1 Regular Expressions

Some of the existing Butler APIs allow a user to specify a search using a regular expression or a glob. Regular expressions are very useful but they are a very bad idea when the regular expression is coming from a potentially untrusted source and can even be a bad idea if a valid user is trying to do something clever and ends up with something that is very complicated. It is easy to create a regular expression that can turn into a denial-of-service attack, and we are therefore proposing that regular expressions be removed from the public APIs and that we only allow globs.

4.2.2 Dimension Record Pre-fetching

Dimension records are used extensively and in many cases the associated record metadata is required. If many clients all request datasets that contain coordinates `(instrument="LSSTCam", detector=4)` the server should not query the database backend every single time to look up the definitions of those IDs. Direct Butler does pre-fetch many common dimension records but the situation in the server is much more complicated where synchronizing a shared cache between many threads becomes the primary concern, and the client code likely doesn't need to cache anything because the definitions will always be sent over the wire from the server.

For a static data release the dimension coordinate space is pre-defined and no new records will be added. Even for a live data repository, such as the Prompt Processing repository, certain record categories are only rarely augmented and are relatively small.

For example, the number of `instrument` records is of order 10, the number of `physical_filter` records is of order 1,000, and the number of `detector` records is of order 1,000. These are very common dimensions and pre-fetching them into the server is very important to reduce database load. If a new filter is added, this event can be planned for and a mechanism can be in place to invalidate the server copy to allow it to be recreated.

The dimensions relating to how we break up the sky into tracts and patches are also something that can be considered for pre-fetch.

Even for `day_obs` the total number of records is only about 4,000 for LSSTCam over the ten year survey (twice that if LATISS is included) with limited metadata attached to those records, and for a static data release we could consider pre-fetching those.

The situation is different for `exposure`, `visit` and `group` where there can be of order 1,000 of each of those created per night (millions of records). Those records (including metadata fields) can not be pre-fetched into the server and must be obtained from the database every time they are needed.

4.2.3 Collection Caching

In a full data release where there can be many chained collections as well as tens of thousands of user collections, it is critical that a client request for a collection definition doesn't result in multiple database queries every single time. A user will only be allowed to see their own user collections and any group collections they are part of but the total number of collections to manage will be very large.

Caching all the collection definitions in the server might be possible, so long as the cache is shared between server threads, but the issue is knowing when to invalidate the cache. A data release will have a large number of unchanging collection definitions combined with user and group collections that can be created or disappear at any time. A prompt processing repository will be getting new collections on a daily basis in addition to the user and group collections. It might be necessary for specific collections to be marked as permanent to allow the servers to know that those will not be modified, but require database queries for the dynamic collections. The approaches to collection caching are still to be worked out and we are investigating reorganizing how collections are stored in the database to make queries on them more efficient without the recursion currently used to support chains of datastores with chains of datastores.

It might be simpler for the client to never cache collection information if the server is caching and the client will only ever see a small subset of the total number of collections.

4.2.4 Multiple Servers

A single server, however large, can not reliably handle thousands of simultaneous requests. This is not helped by the butler APIs being designed before `async` was considered stable, requiring the server to create threads for all incoming requests and Python not currently being a language that supports good thread performance and FastAPI being optimized for async.

To allow arbitrary scaling we will deploy multiple server instances with some form of load balancing. We are designing the server to be stateless such that any client can talk to any server at any time without having a client pinned to a specific server.

4.2.5 Multiple Database Servers

Deploying multiple Butler services allows client requests to be handled at scale but that needs to be matched by database capacity. Multiple read-only clones of a data release database can scale relatively easily. There is a complication when clients are allowed to store data in the butler in terms of replication to replicas, and this is discussed later.

4.2.6 Query System

Many Butler queries can take minutes or even longer, and it is not desirable to lock up resources in the server process whilst waiting for these queries to complete. The solution is to use a system where queries are sent to workers in an execution pool. The client will be issued a job ID and can query the server to obtain the worker status. The client will initially use polling to determine when a query has completed and we will investigate whether UWS[22] is a reasonable approach. Once a job completes the client will receive a URI to the results in either JSON or parquet format. These results will be written with an expiry date to allow for automatic cleanup of results that are no longer needed. Additionally, context managers will be supported to allow the client to send a message to the server when it no longer needs the results. If the query resulted in many results they can be written as multiple files and multiple URIs can be returned to allow the client to paginate.

Additionally, a queue system makes it possible for individual users who are doing many queries simultaneously to be rate-limited. The system should not look like it is down for people if a single user is submitting hundreds of queries and is ahead of everyone else in the queue.

4.3 Reading Datasets

The direct Butler interface supports a generic "datastore" abstraction layer with multiple pluggable options. Supporting arbitrary datastore configurations allows a single butler to store datasets as files and to define datastores that use databases such as the Sasquatch system.[11] It is even possible for users to override datastore configurations to add their own additional datastores into the chain. For client/server Butler this flexibility is an unwanted complication since it weakens the security model and adds unnecesary complexity. The client/server design therefore enforces support solely for file-backed datastores where the server process issues signed URLs to allow the client to download only the datasets they have been approved to see. The client is responsible for interpreting the formatter name specified by the server to allow it to use the correct Python code for converting the file back into a Python object.

4.4 Writing to a Butler

The biggest complication in implementing client/server Butler is handling writes. Clients will rarely need to update dimension records (those are created by instrument registration, raw data ingest, or visit definitions, which are generally thought of as administrative tasks that can, for now, use direct butler connections, although it might be necessary for users to create their own sky map definitions) but Data Rights holders will want to read datasets, construct new datasets derived from them, and store them back in a butler.

We assume that there will be far fewer writes than there will be reads and the majority of the writes will involve multi-dataset transfers from personal workspaces or from user-batch outputs[23] rather than people writing individual datasets from Jupyter notebooks.

In the baseline plan user data will be written to a SLAC object store with the server issuing a signed URL to the client to allow the upload of the file. Once the file has been uploaded the client will inform the server to allow the registry database to be updated.

Quotas will be needed and will likely require the butler to keep track of per-user quota usage – if a user happens to do multiple writes in parallel across multiple butlers there is a chance that they could exceed their quota but we would run daily quota verification jobs to ensure that the butler accounting is accurate.

With multiple replicas of a read-only data release registry database server we can horizontally scale and support many user queries. Once writes are allowed it is necessary for each replica to be notified of that modification and there will be a time where the user who wrote to the registry would not see that update if they were to immediately do a get or a query. To simplify the replication and backup situation we are considering using read-only data release registry databases and a separate user registry database – the user registry should be significantly smaller than the data release registry.

We have not yet decided how many user databases to create: a single user database server for all user data with replicas is one option, but it is also feasible to consider giving each user their own self-contained user registry as a namespace within a larger database server. There is also an intermediate strategy where users are assigned specific user databases, although there is a risk that you could end up with some databases having no user activity and others having all the power users; something that would require migrating users to different servers to balance load. Regardless, multiple registries is a complication for butler where the system has been designed to be able to do joins of user collections with data release collections using the database directly and supporting two or more registry databases in a single query would lead to more work having to be done by the query worker process, using more memory, in order to do the combination from multiple registry databases.

This is particularly problematic when we come to do our user-batch processing where we build up workflow execution graphs.[20,24] These workflow graphs require large queries and do joins across all collections and also utilize temporary tables, and this will need to be significantly redesigned if the user collections and the data release collections are in distinct databases.

5. CONCLUSIONS

Converting an existing library that uses direct database and object store connections to one which uses modern web service paradigms, including supporting scalability and security, is a challenging exercise. Whilst we realized early on that such a change was needed[25,26] other priorities and staff resourcing issues meant that we could not really begin to look seriously at the implementation until more recently.[27,28]

We have made significant progress in the implementation of the client/server Butler this year and it is already deployed with core functionality and used by the Rubin Science Platform Virtual Observatory services.[29] A system will be deployed in time for Data Preview 1 in early 2025.[30]

ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation through Cooperative Agreement AST-1258333 and Cooperative Support Agreement AST-1202910 managed by the Association of Universities for Research in Astronomy (AURA), and the Department of Energy under Contract No. DE-AC02-76SF00515 with the SLAC National Accelerator Laboratory managed by Stanford University. Additional Rubin Observatory funding comes from private donations, grants to universities, and in-kind support from LSST-DA Institutional Members.

REFERENCES

[1] Ivezić, Ž., Kahn, S. M., Tyson, J. A., Abel, B., Acosta, E., Allsman, R., Alonso, D., AlSayyad, Y., Anderson, S. F., Andrew, J., Angel, J. R. P., Angeli, G. Z., Ansari, R., Antilogus, P., Araujo, C., Armstrong, R., Arndt, K. T., Astier, P., Aubourg, É., Auza, N., Axelrod, T. S., Bard, D. J., Barr, J. D., Barrau, A., Bartlett, J. G., Bauer, A. E., Bauman, B. J., Baumont, S., Bechtol, E., Bechtol, K., Becker, A. C., Becla, J., Beldica, C., Bellavia, S., Bianco, F. B., Biswas, R., Blanc, G., Blazek, J., Blandford, R. D., Bloom,

J. S., Bogart, J., Bond, T. W., Booth, M. T., Borgland, A. W., Borne, K., Bosch, J. F., Boutigny, D., Brackett, C. A., Bradshaw, A., Brandt, W. N., Brown, M. E., Bullock, J. S., Burchat, P., Burke, D. L., Cagnoli, G., Calabrese, D., Callahan, S., Callen, A. L., Carlin, J. L., Carlson, E. L., Chandrasekharan, S., Charles-Emerson, G., Chesley, S., Cheu, E. C., Chiang, H.-F., Chiang, J., Chirino, C., Chow, D., Ciardi, D. R., Claver, C. F., Cohen-Tanugi, J., Cockrum, J. J., Coles, R., Connolly, A. J., Cook, K. H., Cooray, A., Covey, K. R., Cribbs, C., Cui, W., Cutri, R., Daly, P. N., Daniel, S. F., Daruich, F., Daubard, G., Daues, G., Dawson, W., Delgado, F., Dellapenna, A., de Peyster, R., de Val-Borro, M., Digel, S. W., Doherty, P., Dubois, R., Dubois-Felsmann, G. P., Durech, J., Economou, F., Eifler, T., Eracleous, M., Emmons, B. L., Fausti Neto, A., Ferguson, H., Figueroa, E., Fisher-Levine, M., Focke, W., Foss, M. D., Frank, J., Freemon, M. D., Gangler, E., Gawiser, E., Geary, J. C., Gee, P., Geha, M., Gessner, C. J. B., Gibson, R. R., Gilmore, D. K., Glanzman, T., Glick, W., Goldina, T., Goldstein, D. A., Goodenow, I., Graham, M. L., Gressler, W. J., Gris, P., Guy, L. P., Guyonnet, A., Haller, G., Harris, R., Hascall, P. A., Haupt, J., Hernandez, F., Herrmann, S., Hileman, E., Hoblitt, J., Hodgson, J. A., Hogan, C., Howard, J. D., Huang, D., Huffer, M. E., Ingraham, P., Innes, W. R., Jacoby, S. H., Jain, B., Jammes, F., Jee, M. J., Jenness, T., Jernigan, G., Jevremović, D., Johns, K., Johnson, A. S., Johnson, M. W. G., Jones, R. L., Juramy-Gilles, C., Jurić, M., Kalirai, J. S., Kallivayalil, N. J., Kalmbach, B., Kantor, J. P., Karst, P., Kasliwal, M. M., Kelly, H., Kessler, R., Kinnison, V., Kirkby, D., Knox, L., Kotov, I. V., Krabbendam, V. L., Krughoff, K. S., Kubánek, P., Kuczewski, J., Kulkarni, S., Ku, J., Kurita, N. R., Lage, C. S., Lambert, R., Lange, T., Langton, J. B., Le Guillou, L., Levine, D., Liang, M., Lim, K.-T., Lintott, C. J., Long, K. E., Lopez, M., Lotz, P. J., Lupton, R. H., Lust, N. B., MacArthur, L. A., Mahabal, A., Mandelbaum, R., Markiewicz, T. W., Marsh, D. S., Marshall, P. J., Marshall, S., May, M., McKercher, R., McQueen, M., Meyers, J., Migliore, M., Miller, M., Mills, D. J., Miraval, C., Moeyens, J., Moolekamp, F. E., Monet, D. G., Moniez, M., Monkewitz, S., Montgomery, C., Morrison, C. B., Mueller, F., Muller, G. P., Muñoz Arancibia, F., Neill, D. R., Newbry, S. P., Nief, J.-Y., Nomerotski, A., Nordby, M., O'Connor, P., Oliver, J., Olivier, S. S., Olsen, K., O'Mullane, W., Ortiz, S., Osier, S., Owen, R. E., Pain, R., Palecek, P. E., Parejko, J. K., Parsons, J. B., Pease, N. M., Peterson, J. M., Peterson, J. R., Petravick, D. L., Libby Petrick, M. E., Petry, C. E., Pierfederici, F., Pietrowicz, S., Pike, R., Pinto, P. A., Plante, R., Plate, S., Plutchak, J. P., Price, P. A., Prouza, M., Radeka, V., Rajagopal, J., Rasmussen, A. P., Regnault, N., Reil, K. A., Reiss, D. J., Reuter, M. A., Ridgway, S. T., Riot, V. J., Ritz, S., Robinson, S., Roby, W., Roodman, A., Rosing, W., Roucelle, C., Rumore, M. R., Russo, S., Saha, A., Sassolas, B., Schalk, T. L., Schellart, P., Schindler, R. H., Schmidt, S., Schneider, D. P., Schneider, M. D., Schoening, W., Schumacher, G., Schwamb, M. E., Sebag, J., Selvy, B., Sembroski, G. H., Seppala, L. G., Serio, A., Serrano, E., Shaw, R. A., Shipsey, I., Sick, J., Silvestri, N., Slater, C. T., Smith, J. A., Smith, R. C., Sobhani, S., Soldahl, C., Storrie-Lombardi, L., Stover, E., Strauss, M. A., Street, R. A., Stubbs, C. W., Sullivan, I. S., Sweeney, D., Swinbank, J. D., Szalay, A., Takacs, P., Tether, S. A., Thaler, J. J., Thayer, J. G., Thomas, S., Thornton, A. J., Thukral, V., Tice, J., Trilling, D. E., Turri, M., Van Berg, R., Vanden Berk, D., Vetter, K., Virieux, F., Vucina, T., Wahl, W., Walkowicz, L., Walsh, B., Walter, C. W., Wang, D. L., Wang, S.-Y., Warner, M., Wiecha, O., Willman, B., Winters, S. E., Wittman, D., Wolff, S. C., Wood-Vasey, W. M., Wu, X., Xin, B., Yoachim, P., and Zhan, H., "LSST: From Science Drivers to Reference Design and Anticipated Data Products," *ApJ* **873**, 111 (Mar 2019). DOI: https://doi.org/10.3847/1538-4357/ab042c.

[2] O'Mullane, W., Economou, F., Lim, K.-T., Mueller, F., Jenness, T., Dubois-Felsmann, G. P., Guy, L. P., Sullivan, I. S., AlSayyad, Y., Swinbank, J. D., and Krughoff, K. S., "Software Architecture and System Design of Rubin Observatory," *arXiv e-prints* , arXiv:2211.13611 (Nov. 2022). DOI: https://doi.org/10.48550/arXiv.2211.13611.

[3] Bosch, J., AlSayyad, Y., Armstrong, R., Bellm, E., Chiang, H.-F., Eggl, S., Findeisen, K., Fisher-Levine, M., Guy, L. P., Guyonnet, A., Ivezić, Ž., Jenness, T., Kovács, G., Krughoff, K. S., Lupton, R. H., Lust, N. B., MacArthur, L. A., Meyers, J., Moolekamp, F., Morrison, C. B., Morton, T. D., O'Mullane, W., Parejko, J. K., Plazas, A. A., Price, P. A., Rawls, M. L., Reed, S. L., Schellart, P., Slater, C. T., Sullivan, I., Swinbank, J. D., Taranu, D., Waters, C. Z., and Wood-Vasey, W. M., "An Overview of the LSST Image Processing Pipelines," in [*Astronomical Data Analysis Software and Systems XXVII*], Teuben, P. J., Pound, M. W., Thomas, B. A., and Warner, E. M., eds., *Astronomical Society of the Pacific Conference Series* **523**, 521 (2019). DOI: https://doi.org/10.48550/arXiv.1812.03248.

[4] Jenness, T., Bosch, J., Schellart, P., Lim, K.-T., Salnikov, A., and Gower, M., "Abstracting the Storage and Retrieval of Image Data at the LSST," in [*Astronomical Data Analysis Software and Systems XXVII*], Teuben, P. J., Pound, M. W., Thomas, B. A., and Warner, E. M., eds., *Astronomical Society of the Pacific Conference Series* **523**, 653 (Oct. 2019). DOI: https://doi.org/10.48550/arXiv.1812.08085.

[5] Jenness, T., Bosch, J. F., Salnikov, A., Lust, N. B., Pease, N. M., Gower, M., Kowalik, M., Dubois-Felsmann, G. P., Mueller, F., and Schellart, P., "The Vera C. Rubin Observatory Data Butler and pipeline execution system," in [*Software and Cyberinfrastructure for Astronomy VII*], *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **12189**, 1218911 (Aug. 2022). DOI: https://doi.org/10.1117/12.2629569.

[6] Kantor, J., Axelrod, T., Becla, J., Cook, K., Nikolaev, S., Gray, J., Plante, R., Nieto-Santisteban, M., Szalay, A., and Thakar, A., "Designing for peta-scale in the lsst database," in [*Astronomical Data Analysis Software and Systems XVI*], Shaw, R. A., Hill, F., and Bell, D. J., eds., *Astronomical Society of the Pacific Conference Series* **376**, 3–+ (Oct. 2007).

[7] Axelrod, T., Kantor, J., Lupton, R. H., and Pierfederici, F., "An open source application framework for astronomical imaging pipelines," in [*Software and Cyberinfrastructure for Astronomy*], Radziwill, N. M. and Bridger, A., eds., *Proc. SPIE* **7740**, 15 (July 2010). DOI: https://doi.org/10.1117/12.857297.

[8] O'Mullane, W. and Jenness, T., "Butler Working Group Charge," (August 2017). Vera C. Rubin Observatory LDM-563, https://ls.st/LDM-563.

[9] Dubois-Felsmann, G., Jenness, T., Bosch, J., Gower, M., Krughoff, S., Owen, R., Schellart, P., and van Klaveren, B., "Data Management Middleware Requirements," (May 2018). Vera C. Rubin Observatory Data Management Controlled Document LDM-556, https://ldm-556.lsst.io/.

[10] Carlin, J., "LDM-GEN3: Gen 3 Butler Acceptance Testing Test Plan and Report," (July 2022). Vera C. Rubin Observatory Data Management Test Report DMTR-271, https://dmtr-271.lsst.io/.

[11] Fausti Neto, A., Thornton, A., Reuter, M., and Economou, F., "Sasquatch: Rubin Observatory metrics and telemetry service," in [*Software and Cyberinfrastructure for Astronomy VIII*], Ibsen, J. and Chiozzi, G., eds., *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **13101**, 13101–59, in press (2024). https://dx.doi.org/doi#here.

[12] Freemon, D. M., Lim, K.-T., Becla, J., Dubois-Felsman, G. P., and Kantor, J., "Data management cyberinfrastructure for the Large Synoptic Survey Telescope," in [*Software and Cyberinfrastructure for Astronomy II*], Radziwill, N. M. and Chiozzi, G., eds., *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **8451**, 84510V (Sept. 2012). DOI: https://doi.org/10.1117/12.926596.

[13] Dubois, R. and O'Mullane, W., "Data Facilities Transition Plan," (September 2022). Vera C. Rubin Observatory Technical Note RTN-021, https://rtn-021.lsst.io/.

[14] O'Mullane, W., Dubois, R., AlSayyad, Y., Economou, F., Speck, D., Huang, F., Li, Y.-T., Mueller, F., Yang, W., Jenness, T., and Chiang, J., "Rubin's Hybrid On Premises-Cloud Data Access Center," in [*Software and Cyberinfrastructure for Astronomy VIII*], Ibsen, J. and Chiozzi, G., eds., *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **13101**, 13101–86, in press (2024). https://dx.doi.org/doi#here.

[15] Dubois-Felsmann, G., Economou, F., Lim, K.-T., Mueller, F., Pietrowicz, S. R., and Wu, X., "Science Platform Design," (January 2019). Vera C. Rubin Observatory Data Management Controlled Document LDM-542, https://ldm-542.lsst.io/.

[16] O'Mullane, W., Economou, F., Huang, F., Speck, D., Chiang, H.-F., Graham, M. L., Allbery, R., Banek, C., Sick, J., Thornton, A. J., Masciarelli, J., Lim, K.-T., Mueller, F., Padolski, S., Jenness, T., Krughoff, K. S., Gower, M., Guy, L. P., and Dubois-Felsmann, G. P., "Rubin Science Platform on Google: the story so far," *arXiv e-prints* , arXiv:2111.15030 (Nov. 2021). DOI: https://doi.org/10.48550/arXiv.2111.15030.

[17] Allbery, R., "Possible authorization approaches for Butler," (March 2021). Vera C. Rubin Observatory Data Management Technical Note DMTN-182, https://dmtn-182.lsst.io/.

[18] Lim, K.-T., "Signed URLs for Data Releases at the USDF," (April 2024). Vera C. Rubin Observatory Data Management Technical Note DMTN-284, https://dmtn-284.lsst.io/.

[19] O'Mullane, W., Alsayyad, Y., Chiang, H.-F., Economou, F., Graham, M., Guy, L., Lin, H., Mueller, F., Jenness, T., Slater, C., and Dubois-Felsmann, G., "Data Preview 0.2 and Operations rehearsal for DRP.," (July 2023). Vera C. Rubin Observatory Technical Note RTN-041, https://rtn-041.lsst.io/.

[20] Lust, N. B., Jenness, T., Bosch, J. F., Salnikov, A., Pease, N. M., Gower, M., Kowalik, M., Dubois-Felsmann, G. P., Mueller, F., and Schellart, P., "Data management and execution systems for the Rubin Observatory Science Pipelines," *arXiv e-prints* , arXiv:2303.03313 (Mar. 2023). DOI: https://doi.org/10.48550/arXiv.2303.03313.

[21] Bosch, J. F., "Caching Database Content in Butler," (April 2024). Vera C. Rubin Observatory Data Management Technical Note DMTN-289, https://dmtn-289.lsst.io/.

[22] Harrison, P. A. and Rixon, G., "Universal Worker Service Pattern Version 1.1." IVOA Recommendation 24 October 2016 (Oct. 2016). DOI: https://doi.org/10.5479/ADS/bib/2016ivoa.spec.1024H.

[23] O'Mullane, W., "User batch - possibilities and plans.," (July 2023). Vera C. Rubin Observatory Data Management Technical Note DMTN-223, https://dmtn-223.lsst.io/.

[24] Gower, M., Kowalik, M., Lust, N. B., Bosch, J. F., and Jenness, T., "Adding Workflow Management Flexibility to LSST Pipelines Execution," *arXiv e-prints* , arXiv:2211.15795 (Nov. 2022). DOI: https://doi.org/10.48550/arXiv.2211.15795.

[25] Allbery, R., "A model for Butler registry access control," (November 2020). Vera C. Rubin Observatory Data Management Technical Note DMTN-169, https://dmtn-169.lsst.io/.

[26] Jenness, T., "A client/server Butler," (March 2021). Vera C. Rubin Observatory Data Management Technical Note DMTN-176, https://dmtn-176.lsst.io/.

[27] Bosch, J. F., Jenness, T., Salnikov, A., Lust, N. B., and Allbery, R., "Butler Client/Server Design Meeting October 2023," (February 2024). Vera C. Rubin Observatory Data Management Technical Note DMTN-282, https://dmtn-282.lsst.io/.

[28] Jenness, T. and Irving, D. H., "Butler Client/Server Implementation and Deployment Strategies," (April 2024). Vera C. Rubin Observatory Data Management Technical Note DMTN-283, https://dmtn-283.lsst.io/.

[29] Allbery, R., "RSP image cutout service implementation strategy," (June 2022). Vera C. Rubin Observatory Data Management Technical Note DMTN-208, https://dmtn-208.lsst.io/.

[30] Guy, L. P., Bechtol, K., Bellm, E., Blum, B., Dubois-Felsmann, G. P., Graham, M. L., Ivezić, Ž., Lupton, R. H., Marshall, P., Slater, C. T., and Strauss., M., "Rubin Observatory Plans for an Early Science Program," (October 2023). Vera C. Rubin Observatory Technical Note RTN-011, https://rtn-011.lsst.io/.