Services

Types of Services

The following services will be provided:

- Metadata Service
  - Provides efficient, flexible querying capabilities to all stored metadata information by querying Metadata Store, or by extracting information from the data
- Query Service (Qserv)
  - Provides access to Database repositories
- Image Service (ImgServ)
  - Cutout Service: provides access to individual images / cutouts of images
  - Mosaic Service: handles complex operations that involve multiple images
- Web Data Access Service
  - Main point of entry for SUI. Parses incoming commands (RESTful format), channels to the appropriate service.

Querying Through Services

Allow users to browse and search repositories (these queries are against Metadata Service). Query by:

- intention
- user
- tag(s)
- date
- task (and options)
- config value
- area on the sky

Allow users to retrieve image datasets (this goes to Image Cutout Service or Mosaic Service):

- Images
- Image cutouts
- Example queries:
  - return a list of images (note these would be unique image ids, not physical locations), or actual images that:
    - include a given point
    - contain a given area (entire area falls inside image)
    - are within a given area (entire image fall inside the area)
    - overlap with a given area (at least one pixel). For pixels that fall outside, fill them with NaN or user-provided value
  - return the best (most centered) image (or image id) for the usecases listed above
  - return cutout of a given size centered around a given point
  - for the above further filtering might be involves, e.g., images owned by certain user, from certain release, inside certain time window, images of certain type, etc

Allow users to retrieve catalog information (this goes to Qserv):

- SELECT and WHERE clauses for database table datasets
- Optionally full SQL query against databases

Allow users to browse history of queries they run in the past, rerun queries from that list.

Allow users to generate output (images, tables) and treat it as input for next query.

Allow users to upload a list of "things" to workspace, then request "repeat a given query for each "thing" from the uploaded list". Example of "things": ra/dec points, ra/dec points + distances, object ids, object names, bounding boxes, etc. In IPAC terminology, this is called "table upload queries". Note, we will need to assign some sort of unique id to such list. Results must contain information which result correspond to which "thing". E.g, if user asks for neighbors near (1,1), (4,4), it is not enough to just return list of neighbors, we need to tell which neighbor is for which point. (related epics: DM-1556 for Qserv, DM-1557 for ImgServ)

Submitting query from SUI – typical scenario:

- run "*explain <query>*"
- "*explain <query>*" returns cost estimate (based on #chunks accessed, # joins, math complexity), and query type (interactive / scan, if scan: scan type, scan speed)
- submit query (interactive or in background depending on what "explain" suggested)

- if interactive and query takes too long, kill and resubmit as background query.

Note that SUI will facilitate rerunning past queries easily. Metadata Store will contain information about availability of results from queries recently executed, which can be used by SUI to return result without rerunning the query, if results are available. Note, it can get tricky with L1, as "latest" is a moving target.

Returning Results from Services

- Streaming, not stored anywhere
- Persisting in a user workspace (to database table, or a file or a set of files), return "location").
- All background queries use "persisting" mode. Interactive can do either one.

Output should be delivered in a variety of data formats (even in the short term):

- FITS images
- FITS tables
- Text files (e.g. pex_config or serialized metadata)
- JSON
- CSV
- In the long term, we'll also need to support VOTables (TAP), but that may be solely via the DB.

Authentication / Authorization

Access to all services will be controlled. Envisioned modes:

- who: owner only / group only / open to everyone
- access type: read / write

Authentication will be handled through Data Center-provided mechanisms (e.g., provided by NCSA for the main Data Archive Center).

Single-sign-on system would be ideal, especially for users coming from SUI: upon successful authentication (through a cookie or whatever) appropriate mysql-credentials associated with given user will be fetched and used. Note, it can be hard to implement for direct database API connections, it is more feasible if access occurs through wrapped, LSST-provided APIs.

Every science user should have MySQL credentials.

Relevant Off-the-shelf Systems

IRODS

- http://irods.org/
- scalability
    - today's large-scale installations: o(100) million files, o(100) simultaneous users
    - Federated Icat System used to distribute load on metadata for very large installations
- useful features: automatically (dynamically) applying rules and triggers, enforcing policies
    - for example: generate and store checksum, verify checksum on read, automatically group (tar) small files before storing in MSS
- APIs
    - command line shell
    - java, php, python
- metadata
    - options for catalog backend: PostgreSQL, Oracle, MySQL
    - based on AVU (attribute-value-unit) triplets. (This might impair performance - structured metadata would be faster)
- few observations from Andy Salnikov based on experience at LCLS:
    - building iRODS is complicated, mostly interactive process, though I managed to wrap it into RPM script
    - client APIs for iRODS look cumbersome, especially C API
    - building Python wrappers (which is poorly-supported third-party stuff) needs some non-trivial steps like rebuilding iRODS libraries with -fPIC
    - iRODS works best if all data is accessed through iRODS only, without direct file-system level access to data
    - iRODS only knows one checksum type (md5, I believe, which is a bit CPU-intensive)
    - for some things we had to mess with ICAT database directly
    - MySQL backend may not be well supported, Postgres is their standard option (I wrote MySQL backend myself, and it is a bit messy)

FERMI Data Catalog

- https://github.com/brianv0/datacat-doc/blob/master/LSST-Datacat-overview.md
- http://docs.datacatalog.apiary.io (rest api, some out of date: there is no “children” resource anymore, the blurb about 302/303 codes is misleading, field names have changed slightly. Will be revisited shortly)
- catalog for image data, loosely coupled with the data
- originally developed for Fermi Gamma-ray, used in production since 2007
- key features: metadata (blended mix of structured and unstructured), crawlers with pluggable project-specific components REST api, efficient searching, handles ACLs
- managing 23 millions files for Fermi
- spacial optimizations: currently as a separate application built on top of the catalog, but could be done inside the dataCat if needed
- backend: originally Oracle, now supports MySQL (porting almost finished, undergoing final tests)