A Web Based Data Catalog for Data Access and Analysis

An iteration over the Fermi Gamma-Ray Space
Telescope Data Catalog developed by
The Scientific Computing Applications Group

Brian Van Klaveren

with Tony Johnson

bvan@slac.stanford.edu

tonyj@slac.stanford.edu

CHEP2013 2013-10-04

U.S. DEPARTMENT OF **ENERGY**
Office of Science

SLAC NATIONAL
ACCELERATOR
LABORATORY

This Talk

- Overview of the Data Catalog
- Updating the Data Catalog for use by different experiments
- Moving to RESTful APIs for the Data Catalog
- Leveraging HTML5 for a modern user experience

Data Catalog Overview

- Initially developed for Fermi Gamma-Ray Space Telescope (Fermi-LAT)
- Designed to be experiment independent
- Development started 2007, in use since launch 2008
- Written in Java (JSP for web)
- Now in use by other experiments (EXO, LSST)

SLAC

It is a metadata database for your files.

And that metadata is stored in a virtual hierarchy.

It is not tied to any processing system.
It is not tied to any file system or protocol.
It needs only an RDBMS and servlet container.

More detail

Comprised of several software components:

- APIs for creation, reading, modification, and searching
- A web application for user browsing and interaction
- A daemon which checks file integrity and automating metadata extraction from files
  - Plugins are written for supporting new file types, access protocols, and metadata extraction
- A download manager for reliable download of files and file collections which fetches underlying files

Data Catalog Primitives

The Data Catalog consists of three primitives, all of which can be tagged with user-defined metadata.

- A Dataset is a logical set of data (run, MC, calibration, e.g. file)
  - Can have different versions
  - Can have multiple physical locations
- A Folder is a container. It can contain other folders, groups, or datasets.
- A Group is a Dataset container, wherein the Datasets contained are restricted to a common semantic type. It's basically a macro-Dataset.

Current web application

Brian Van Klaveren - A Modern Web Based Data Catalog - CHEP2013

Modernizing and Generalizing the Data Catalog

Why?

- So we can reuse it and better maintain it
- Better API and Web App performance

What do we need to support going forward?

- Reduce the Data Catalog to core features

  - Add plugins for experiment-specific features

- Create language-agnostic APIs (we use REST)
- Provide secure access to the Data Catalog over the web

  - Again: <u>Not the underlying files</u>, just info about them

- Modernize the web application (HTML5, AJAX)

REST
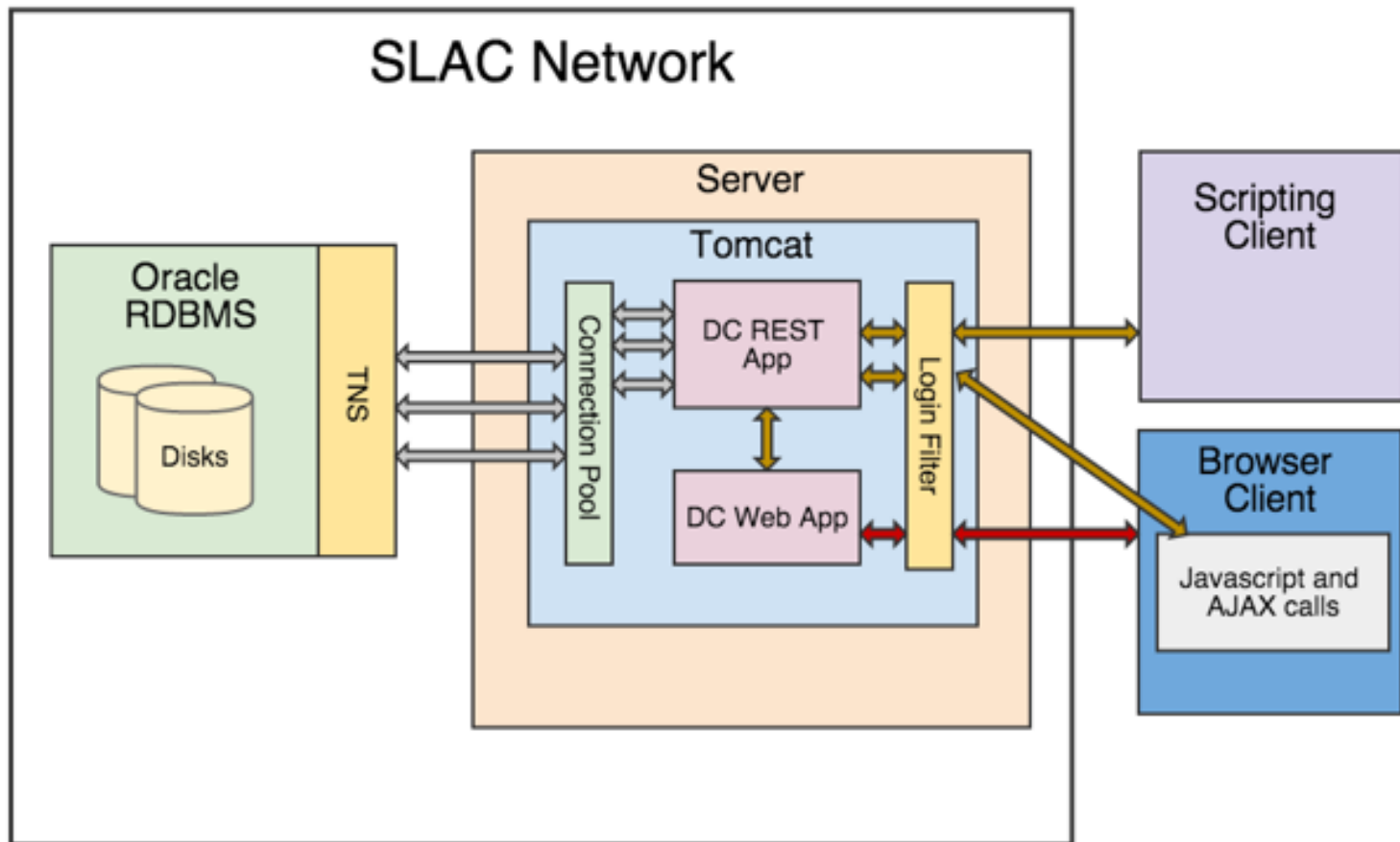## (Representational State Transfer)

Or...

Stateless Create/Read/Update/Delete
Implemented over HTTP
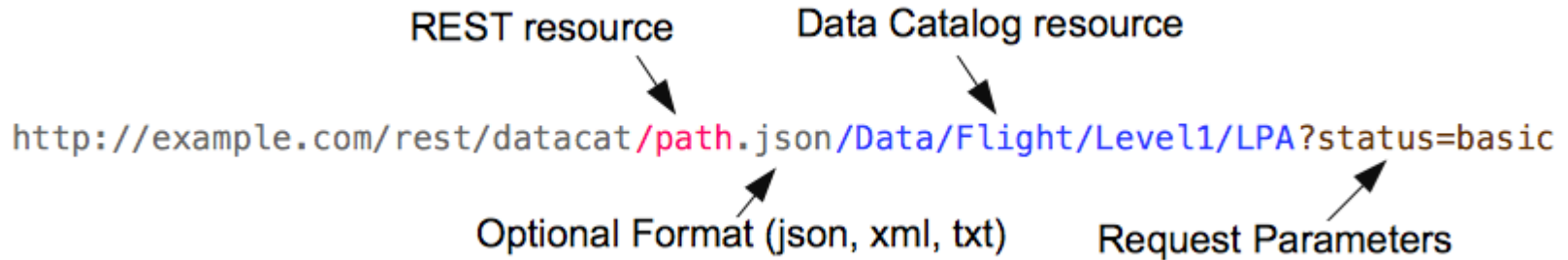## using Resources (not methods)

RESTful API

A scalable, supportable way to supporting [C/C++, Java, Javascript, Python, Perl, Ruby, etc...] is to use a RESTful API over HTTP.

- We will let the user determine the output they want
  - JSON, XML, text
  - Made easy thanks to the help of Jersey/JAX-RS
- HTTP/XML/JSON is well supported in nearly all languages
- We get a single point of API access = One API to develop
  - Easier to manage resource use
  - If needed, scaling horizontally is easy
- The Web Application is driven off the same REST API

RESTful API – What does this look like?

RESTful API – Resources

SLAC

REST resource   Data Catalog resource

`http://example.com/rest/datacat/path.json/Data/Flight/Level1/LPA?status=basic`

Optional Format (json, xml, txt)   Request Parameters

JSON
Output:

```
{
  "$type":"folder",
  "name":"LPA",
  "path":"/Data/Flight/Level1",
  "metadata":{
    "entry":[
      { "key":"nKeyData",
        "value":10 },
      { "key":"eventComponents",
        "value":"reco=RECON,digi=DIGI,merit=MERIT" },
      { "key":"FitsSkim",
        "value":"Level 1 LPA data" }
    ]
  },
  "stat":{
    "$type":"stat",
    "datasetCount":0,
    "groupCount":75,
    "folderCount":0,
  },
  "description":"Output from Level 1 processing of on-orbit data"
}
```

Brian Van Klaveren - A Modern Web Based Data Catalog - CHEP2013

RESTful API – /datasets (Dataset records)

Listing one dataset:

- `GET /datasets/Repro/P2/Overlay/r0239557414.root`

Searching for datasets:

- `GET /datasets/Repro/P2?filter=nRun=239557414+AND+quality+=+'GOOD'`

Register a dataset

- `POST /datasets/Data/MC/Co60?name=r0123.root&format=ROOT&type=MC \`
  `&site=SLAC&fspath=/afs/slac/g/mc/r0123.root`

Modify Dataset - add/modify new versions, locations, or dataset metadata

- `PUT /datasets/Data/MC/Co60/r0123.root`
  `[{"key":"nRun", "number":123},{"key":"source","string":"cobalt"}]`

DELETE - Again, deleting the DC record, <u>not the underlying file(s)</u>

- `DELETE /datasets/Data/MC/Co60/r0123.root`

# RESTful API – Additional resources

/children

- Returns the children (with option to omit datasets) of any container object, with optional paging
- Useful for building lightweight browsing interfaces

/groups, /folders

- Used for creating, modifying, or deleting groups and folders in the same way datasets is used for datasets.

Searching

Search expressions are composed terms and conditions. The parser creates a syntax tree and translates that to SQL or defers to a plugin.

```
name = "run_012345.fits"
```

Term   Condition
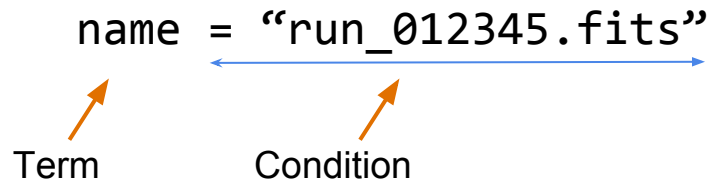
The parser is aware of three types of terms:
- A primitive's field name or an alias of that name (these are built-ins)
  - `name = "run_012345.fits"`
- A primitive's metadata key
  - `sIntent IN ('nomSciOps', 'nomSciOps_diagEna')`
- Or a plugin. In this case, "exp" is the plugin, and "quality" is the term
  - `exp.quality IN ('GOOD','GOLDEN')`

Searching - Conditions and examples

Use words or symbols. Conditions map to SQL:

- `<, >, <=, >=, =|==, !=|~=|<>`
- `LT, GT, LTEQ|LE, GTEQ|GE, EQ|EQUAL, NEQ|NE|NOT EQUAL`
- `[NOT] IN`
  - `nRun IN (1234, 5678, 2468)`
- `[NOT] NULL`
  - `runQuality IS NOT NULL`
- `[NOT] EXISTS`
  - `runQuality EXISTS`
- `AND | "&&"`
  - `(runQuality NEQ 'GOOD' AND nRun EQ 1234)`
- `OR | "||"`
  - `(createDate LT d'2013-10-14T17:57Z' OR nRun GT 9876)`

Searching - Plugins

A searching plugin will know how to relate to a Data Catalog object to extra, experiment-specific data (which is usually another table).

For example, when given the search expression:

`exp.quality IN ('GOOD', 'GOLDEN') AND exp.runType EQ 'data'`

The following will happen:
- The search parser understands to get the "exo" plugin
- The parser hands the plugin the current SQL statement, and the expression `quality IN ('GOOD', 'GOLDEN')`
  - Plugin performs any necessary setup to the SQL statement
  - Plugin returns a SQL expression to parser, if necessary
- Repeat the process for `runType EQ 'data'`

Security and Authentication

Security is handled two different ways

1. Session/Secure Cookies and Single Sign-on
    - Ideal when requests originate from web app/browsers
    - Possible to use via a script, but requires login every 8 hours

2. Hash Message Authentication Code (HMAC)
    - Relies on secret private key to generate a cryptographic hash of the HTTP request, ideal for batch jobs
    - Tools supported natively in many languages (Java, Python, etc...)
    - Heavily used in industry

HTML5

Modernizing our web application

- Move towards ajax/HTML5 support and frameworks
  - jQuery for general Javascript
  - Bootstrap for CSS
- Move to lazy loading for things that take a long time
  - Getting the # of files in 30 groups, each with 20k+ datasets, takes a while. Getting the byte size, event count, etc… longer.
  - So, let's load what we can in ~5 seconds, let the web app finish the rest asynchronously

HTML5 – New Layout with Bootstrap

HTML5 – Lazy and On-Demand loading

Lazy/Async Loading of Status

HTML5 – Responsive Design (thanks to Bootstrap)

Bootstrap gives us the additional feature of a fluid layout.

The web app can transform it's presentation depending on device or screen size.

Layout is stacked on narrower screen/viewport

Collapsible List

Conclusion

- The Data Catalog we've made for Fermi-LAT is very successful and used heavily across several experiments
- Generalizing the core and adding a plugin architecture allows us to support the needs of different experiments going forward
- RESTful interfaces are a good way to support many languages without repeating yourself, and a good way to support off-site use of our Data Catalog
- RESTful interfaces also make it easy to create responsive and adaptable human interfaces for data browsing and retrieval
- HTML5 technologies coupled with RESTful interfaces can enable us to create responsive web pages that scale

A bit more...

We are looking for other experiments who may be interested in using this!

- Current version has been in production for 5 years
- The complete rewrite is still 3-6 months out.
- The REST api has is used by EXO in production.
- Optimized for use with Oracle, but can work in MySQL, PostgreSQL
- More work is planned for the Download Manager
  - Optimized for xrootd at the moment

Informal Performance/benchmark notes:

- /path resource handled 600 concurrent connections at 1500 req/sec
- 4x2.6GHZ 16GB VM
- A SQL composing framework allows us to generate dynamic, optimized SQL at query time.