Managing A Rapidly Evolving Analytics Pipeline

Feng Peng- @feng

Data Pipeline Engineer, Analytics Infrastructure @Twitter

Strata Conference & Hadoop World, Oct 2013

Outline

- Analytics Pipeline Overview
- Challenges
- Our solution: Data Access Layer
- Examples

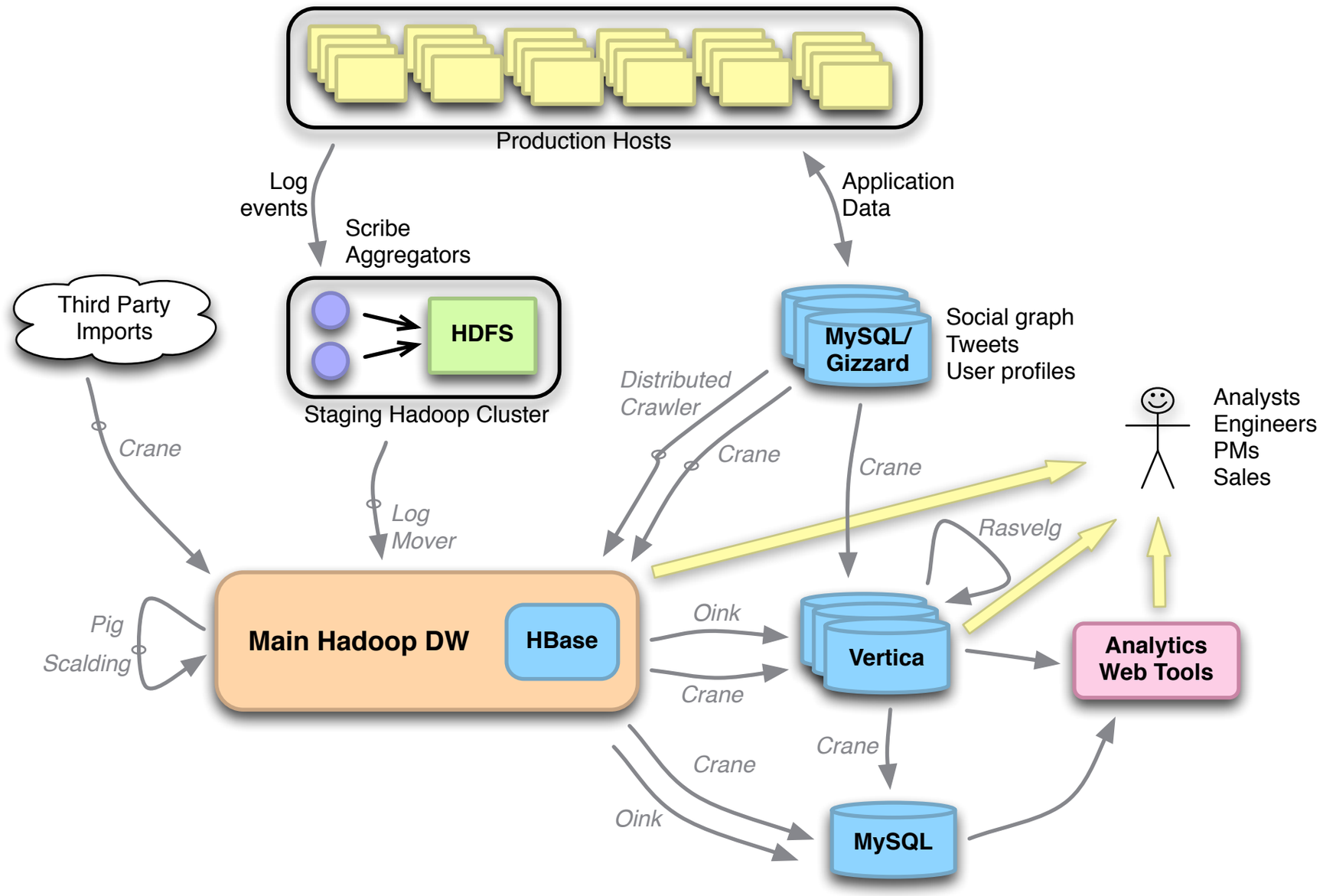
Big Data at Twitter

- Data
  - hundreds of data sources
  - tens of PBs data processed per day
  - thrift / protobuf / TSV / JSON
  - HDFS / HBase / Vertica / Mysql

Big Data at Twitter

- Applications
  - hundreds of engineers / data scientists / PMs
  - tens of thousands of production jobs per day
  - Pig / Scalding / Summingbird / Java ...

Twitter Analytics Pipeline



Beyond Hadoop

- Making use of big data is more than storing it on HDFS and writing M/R jobs
  - correct and complete data
  - consistent semantics across applications
  - verifiable results

ETL Pipeline

- Traditional ETL pipeline is only one part of the full analytics pipeline
  - controlled by a central team
  - well defined schema management
- Analytics pipeline
  - multi-DC, heterogeneous data systems
  - multi-team participation
  - multi-platform applications

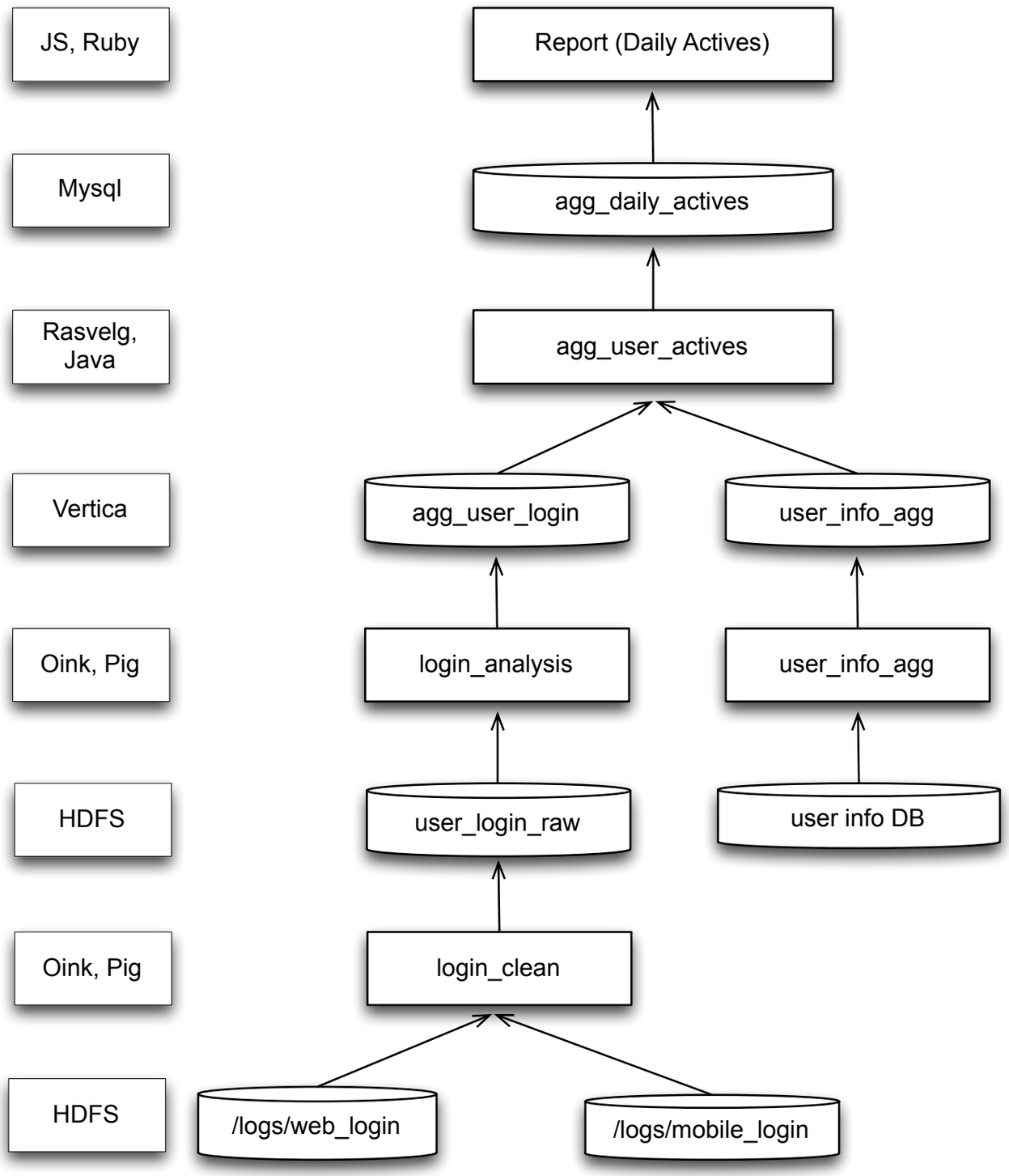
Growing Pains

- Data / application catalog
- Data evolution
- Data provenance
- Data discovery

Simple Questions Become Hard

- How to read the data stored on HDFS?
- What are the fields?
- What does each field mean?
- Who generates this data?

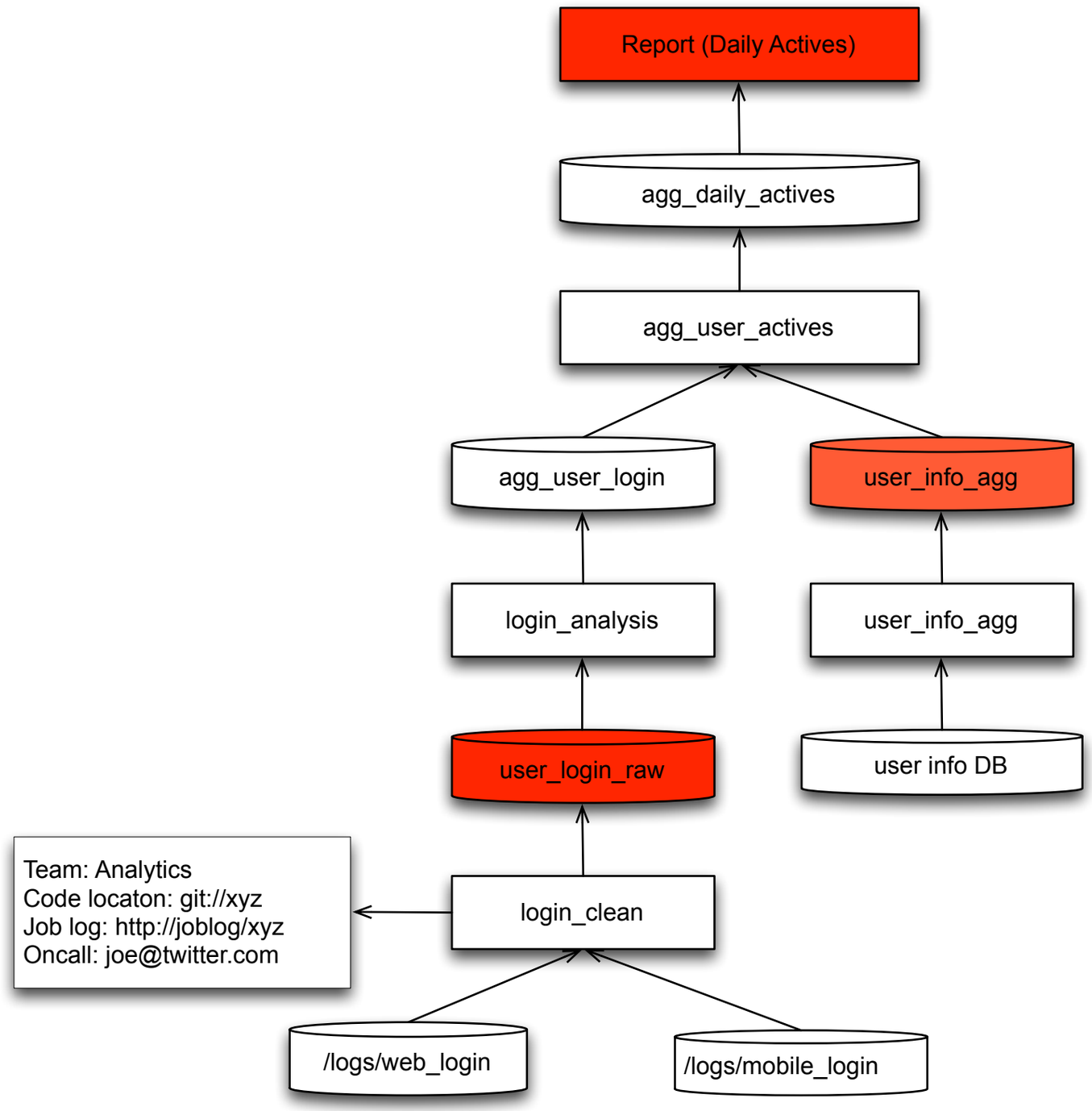
Case: a Delayed Report



More On-call Issues

- Root cause detection
- Alert prioritization
- Job / data ownership
- Ripple effects

Dependency Graph



Case: Tale of Two Teams

- Team A needs to change the semantics of a field.
- Team A searches the repo and notifies the users.
- Changes are synchronized and deployed.
- Team B finds their metric drops 10% (only two weeks later) because they use another repo.
- Totally fictional.

Deploying Data Changes

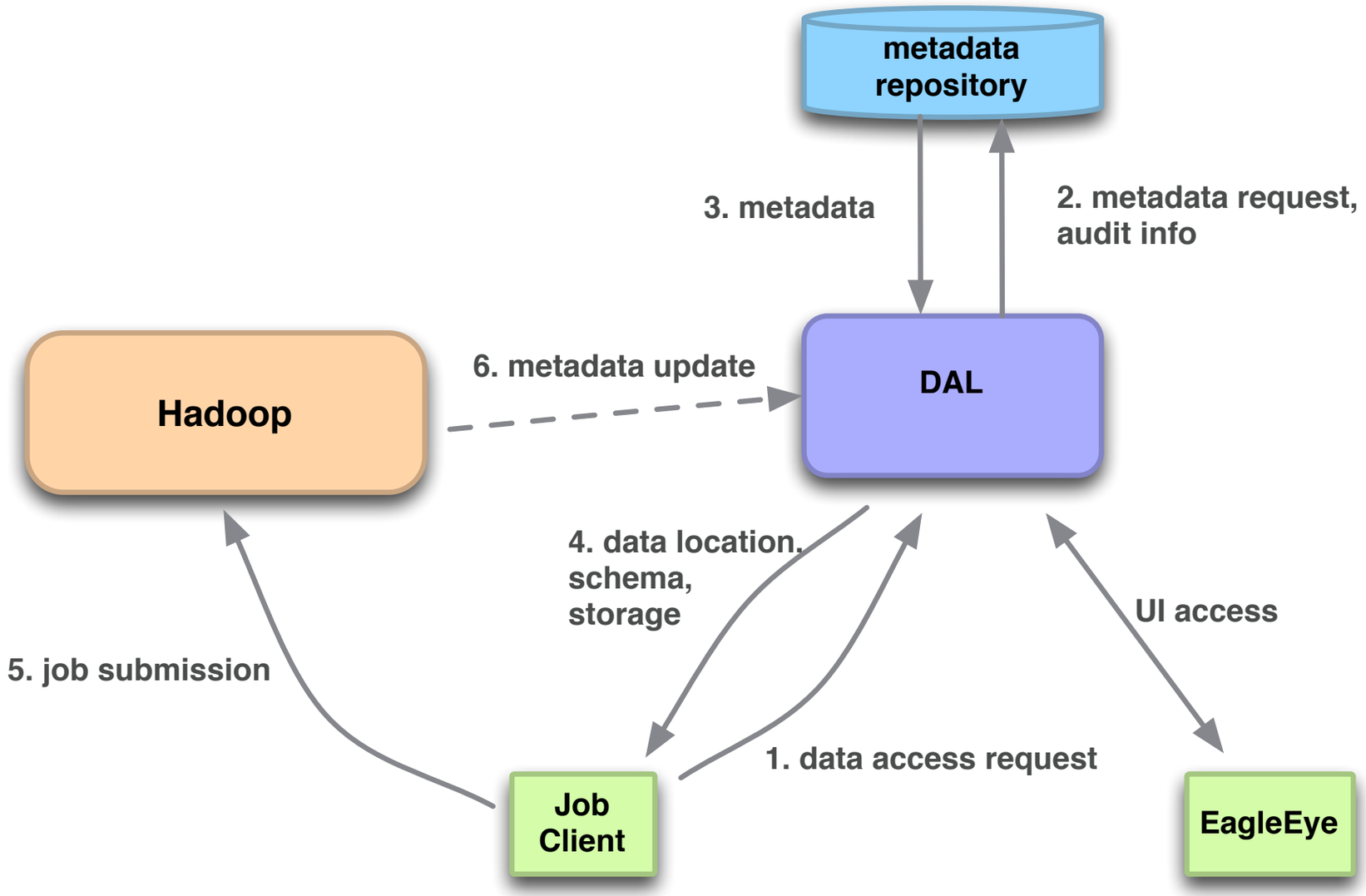
- Time consuming and error prone
- We do not want to do the following:

```
loadRange(start, end)
        R1 = load (start, 2012/12/31)
             from location A with field X
R2 = convert field X in R1
             to new value Y
R3 = load (2013/01/01, end)
        from location B with field Y
R  = union R2, R3
```

Data Access Layer (DAL)

- A universal data access layer with metadata service to provide:
  - schema management
  - storage abstraction
  - data provenance
  - usage auditing

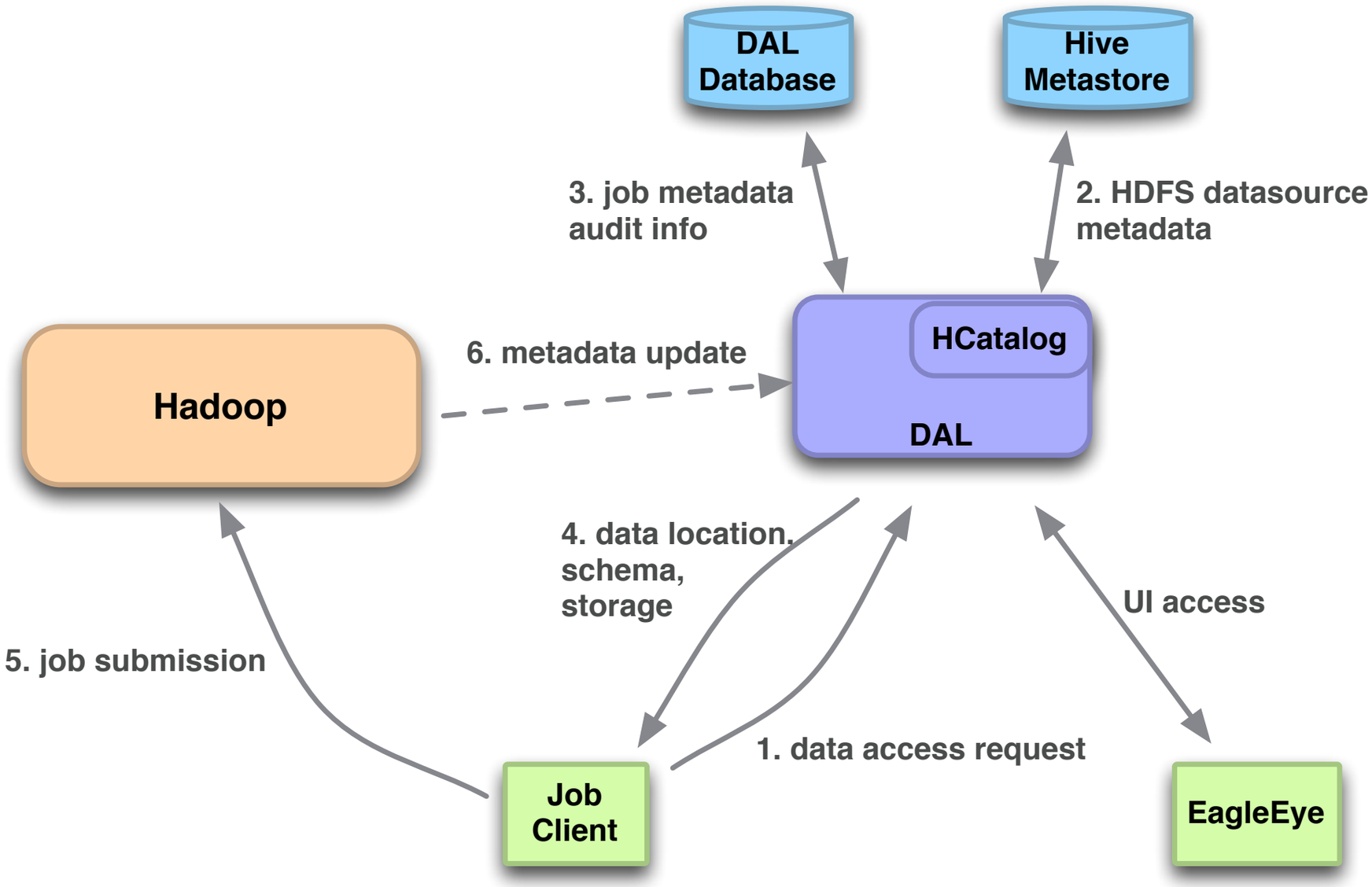
DAL Architecture



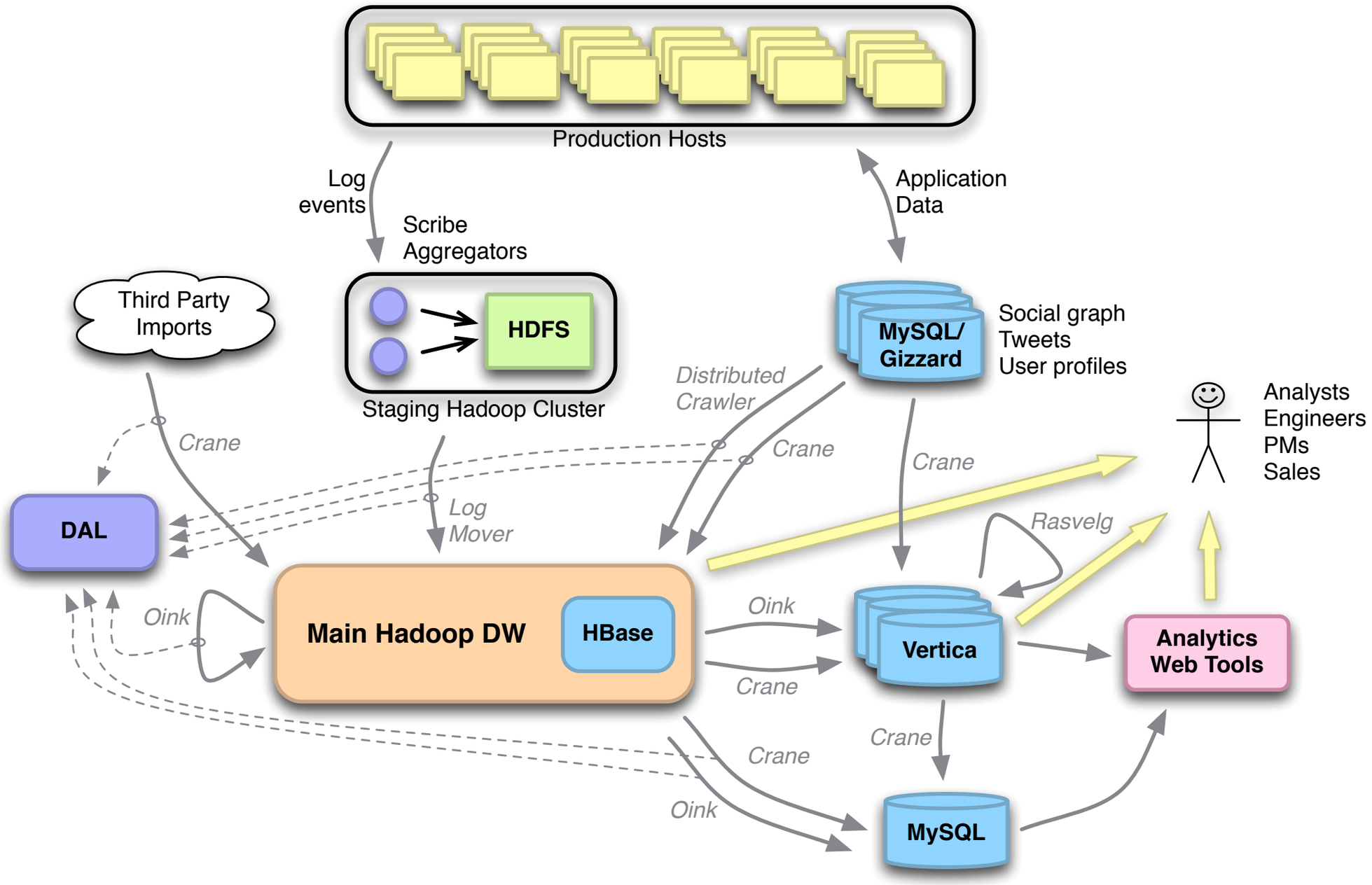
Simple idea:

```
fopen ("input.txt","r") => dalOpen("input.txt","r")
fopen ("output.txt","w") => dalOpen("output.txt","w")
```

HCatalog Integration



Analytics Pipeline with DAL



Main Components

- Data and application metadata repository
- Metadata and dependency management API
- Framework-specific adaptors
- GUI / CLI exploratory tools

DAL Database

- First class entities
  - applications
  - jobs
  - data sources
  - data partitions
- Entity properties, dependencies

Thrift API

- CRUD operations for first class entities
- Auditing functions

```
void addDependency(Application app,
                   DataSource dataSource,
                   DependencyType dependencyType)
```

- Property functions

```
void setAppProperty(Application app,
                    string propertyName,
                    string propertyValue)
```

A Pig Example

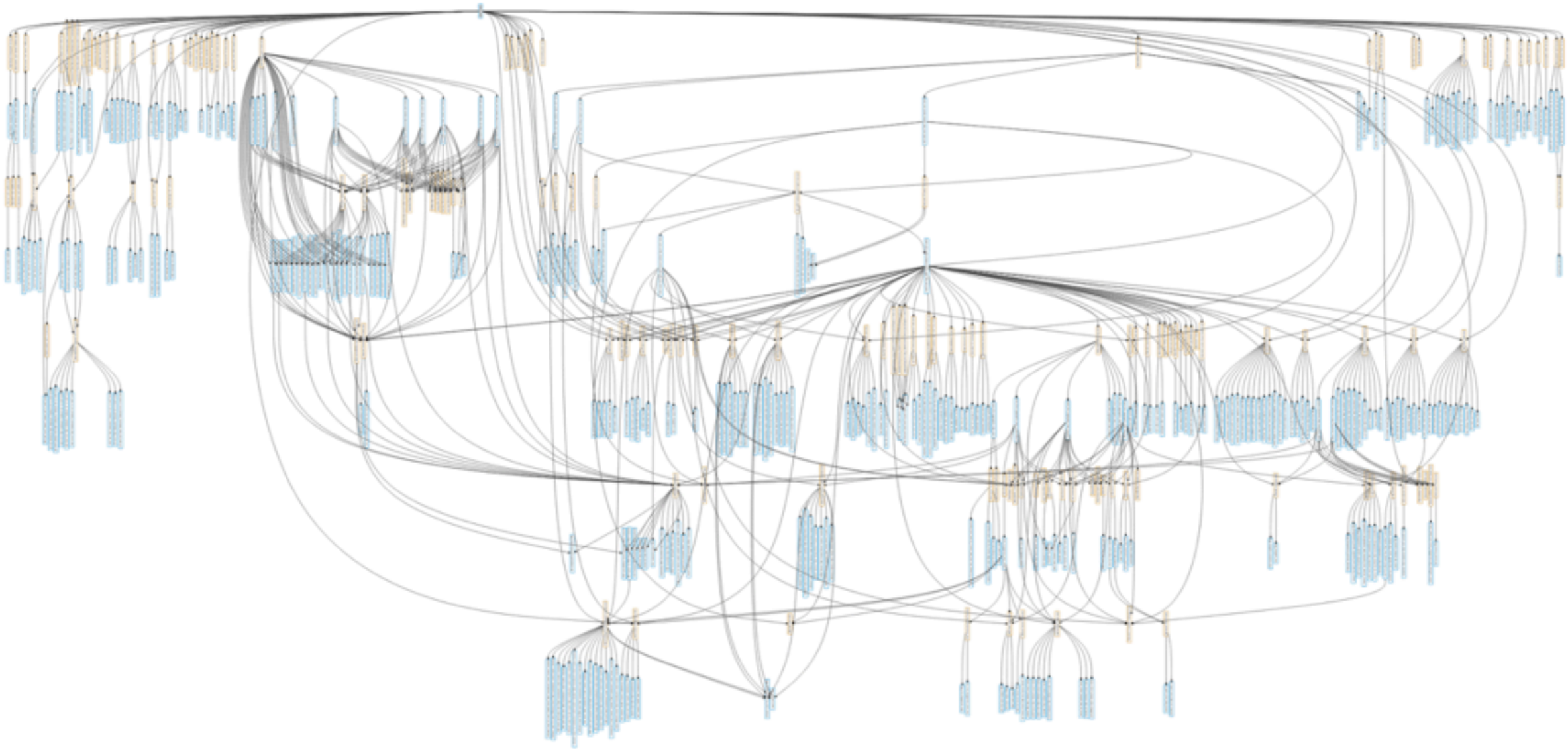
```
set application_name ‘user_analysis‘;
set user_name ‘feng‘;

users_load = load ‘dal://cluster.logs.users‘
                using dal_loader();
users = filter user_load by
          date >= ‘20130501‘ and date < ‘20130601‘
active = get_active_users(users);
store active into ‘dal://cluster.feng.result‘
        using dal_storer(date = ‘20130601‘)
```

Inside the dal_storer

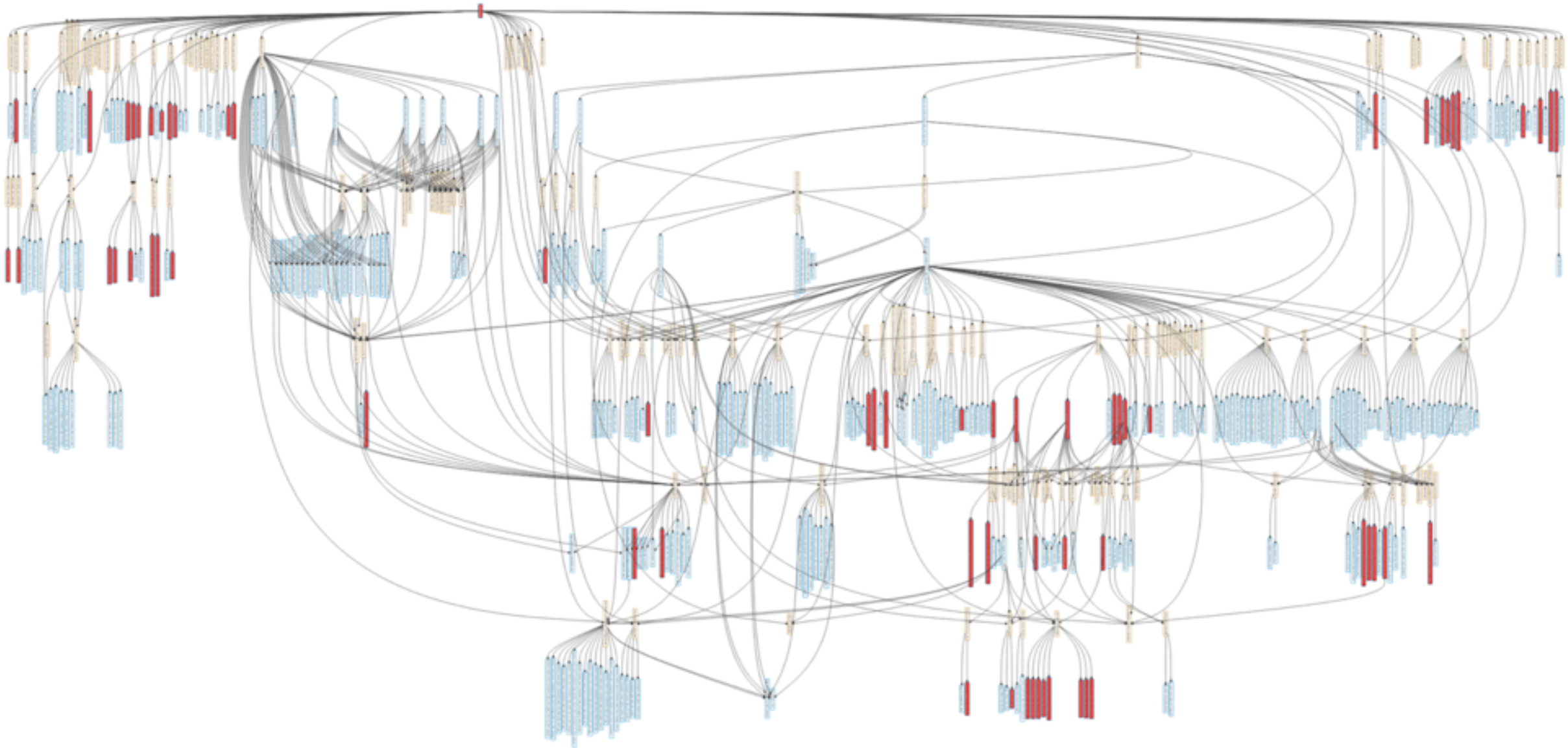
```
-- store active into 'dal://cluster.feng.result'
        using dal_storer(date = '20130601')
app = dal.createApp("user_analysis", "feng");
ds = dal.createDataSource("cluster.feng.result");
dal.addDependency(app, ds, WRITE);
```

```
store active into 'feng.result'
      using HCatStorer(date = '20130601')
```

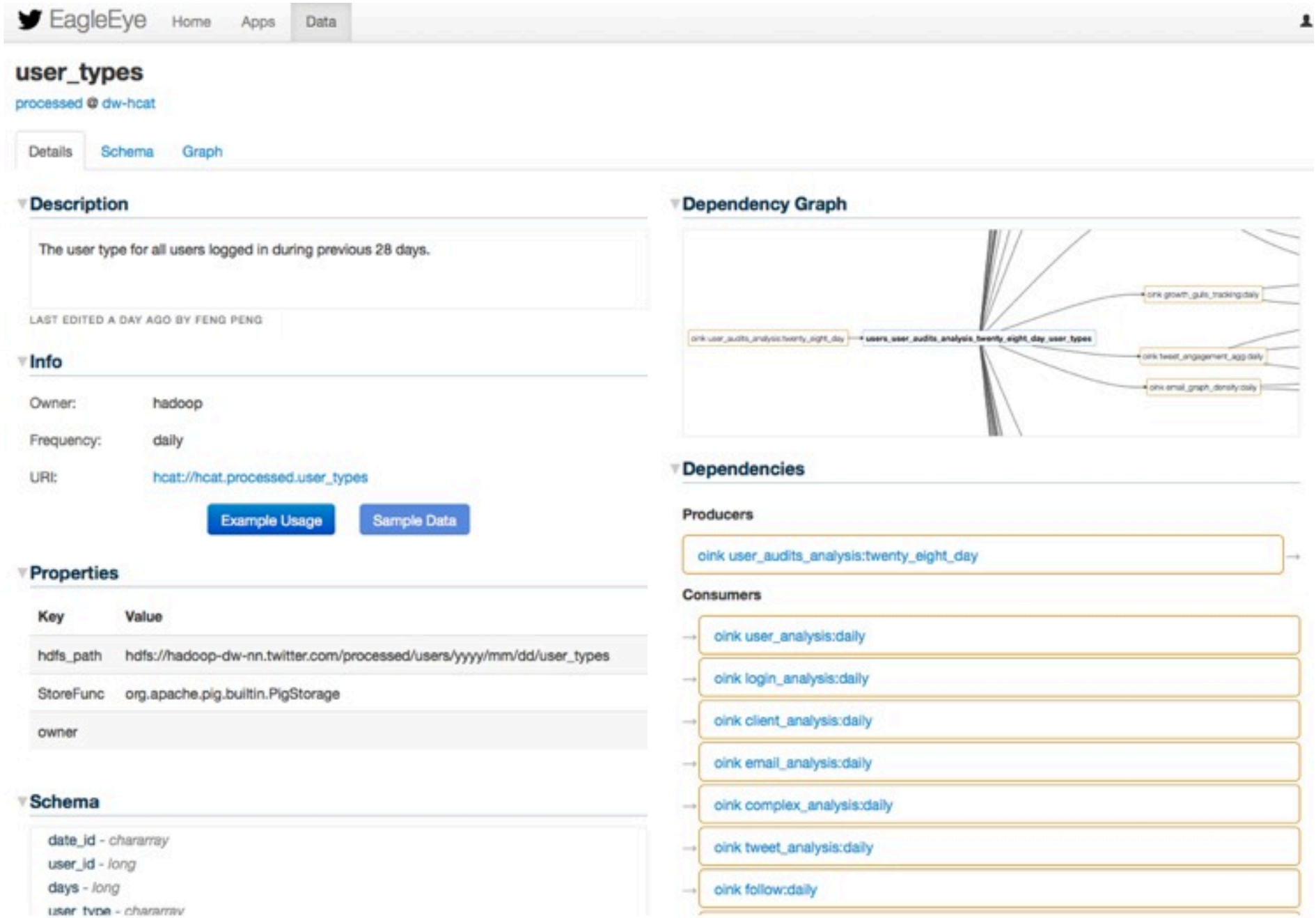
Dependency Graph



Dependency Graph with Latency



Data Exploration



Data Evolution

- Schema change
- Semantics change
- Field deprecation
- Storage change
- New data type replacing old

Example: Field Deprecation

- An optional field converter in the serde layer

```
 // when initializing
String converterName = tbl.getProperty("dal.objectconverter");
  if (converterName != null) {
    Class<?> converterClass = job.getClassByName(converterName);
    converter = (ObjectConverter) converterClass.newInstance();
  }

// when deserializing
return (converter == null)?result:converter.convert(result);
```

Challenges

- Migrate existing applications
- Support different frameworks
- Nuances caused by interactions between different frameworks
- Multi-cluster management
- Tooling

Acknowledgements

Arash Aghevli
@aaghevli

Joseph Boyd
@sluicing

Travis Crawford
@tc

Bill Graham
@billgraham

Michael Lin
@mlin

Akihiro Matsukawa
@amatsukawa

Feng Peng
@feng

Dmitriy Ryaboy
@squarecog

Questions?

Feng Peng - @feng