Galaxy

Code and storage architecture

www.getgalaxy.org
Galaxy application architecture
Job Manager → Job Handlers

<<thread>>
Splitting by process

galaxy.util.pastescript.serve

Application Stack

galaxy.webapps.galaxy.GalaxyWebApplication

---
galaxy.util.pastescript.serve

Application Stack

Job Manager

Job Handlers
UI Architecture

Views
- Web UI
- CLI
- JS Lib
- Python Lib (BioBlend)

Controller
Model
API
Code organization and standards

http://wiki.galaxyproject.org/Develop/BestPractices
(let’s look at the code)
These are best practices.

“foolish consistency is the hobgoblin of little minds”

Standards

Galaxy mostly follows PEP-8, readability is the ultimate goal

Avoid “from module import *”

Comment lines should be under 79 characters, code lines can be up to 200 characters if it improves readability

Docstrings need to be reStructured Text (RST) and Sphinx markup compatible, documentation is automatically generated

Whitespace: whatever is most readable, both for blank lines and space around operators

What about Javascript?
Galaxy data storage
Metadata stored in relational database
(PostgreSQL Please!)

Dataset files stored on filesystem
in objectstore
Data Abstraction

Galaxy

Object Store

Disk
Disk
S3
iRODS
Data Abstraction

```python
>>> fh = open( dataset.file_path, 'w' )
>>> fh.write('foo')
>>> fh.close()
>>> fh = open( dataset.file_path, 'r' )
>>> fh.read()

>>> update_from_file( dataset, file_name='foo.txt' )
>>> get_data( dataset )
>>> get_data( dataset, start=42, count=4096 )
```
Data Abstraction

Distributed Object Store

Galaxy

Distribution by weight
Zero weight

FS
FS
FS
FS
Data Abstraction

Benefits

• Grow beyond original capacity
• Avoid migrating data offline
• Tier storage
• Let your users bring their own storage
• Use resources w/o a shared filesystem (with iRODS)
• Remove IO bottlenecks
Job scheduling and running
Job Control

job_conf.xml

Diagram:

- Job Handler
  - Condor
  - Torque
  - DRMAA
  - SSH
    - SSH
      - Torque (CLI)
      - SGE (CLI)
  - Globus SSH
Extension: adding a new job runner
Running as actual user

User and group specific storage (?)
Galaxy workflow system
Workflow representation: data flow graph

Scheduling like any other job

New: fix and rerun from failure point (workflow automatically paused)

Future: background scheduling, decision points during scheduling
Galaxy API
Galaxy API

Technologies

• Representational State Transfer (REST)
• Sessionless operations via HTTP
• JavaScript Object Notation (JSON)
• Uses a key (rather than user/password)
Galaxy API

Core Interfaces

- Library permissions
- Forms
- Server configuration (view)
- Sample tracking requests and samples
- Manage users, roles, and quotas
- Execute tools and workflows
Galaxy API

Core Interfaces

- Histories and History Datasets
- Libraries and Library Datasets
- Tools
- Workflows
API Interactions with Galaxy

- $GALAXY_HOME/scripts/api
- Python lib: blend
- JavaScript lib
- Galaxy's REST API
- Galaxy
Bare Bones Galaxy API

> create (POST)

> display (GET)

> update (PUT)

> delete (DELETE)
Making the Calls

Wrapper methods exist (in /scripts/api/) to make the API calls easier:

```
./{action}.py <api key> http://<ip>/api/{module}/[unit] [args]
```

**action:** create | display | update | delete

**api_key:** obtained from the UI

**module:** datasets | forms | histories | libraries | permissions | quotas | requests | roles | samples | tools | users | visualizations | workflows

**unit:** dataset_id / history_id / library_id / ...

**args:** name / key-value pair / ...
Galaxy Code Documentation

Galaxy is an open, web-based platform for accessible, reproducible, and transparent computational biomedical research.

- **Accessible**: Users without programming experience can easily specify parameters and run tools and workflows.
- **Reproducible**: Galaxy captures information so that any user can repeat and understand a complete computational analysis.
- **Transparent**: Users share and publish analyses via the web and create Pages, interactive, web-based documents that describe a complete analysis.

Two copies of the Galaxy code documentation are published by the Galaxy Project:

- **Galaxy-Dist**: This describes the code in the most recent official release of Galaxy.
- **Galaxy-Central**: Describes the current code in the development branch of Galaxy. This is the latest checkin, bleeding edge version of the code. The documentation should never be more than an hour behind the code.

Both copies are hosted at ReadTheDocs, a publicly supported web site for hosting project documentation.

If you have your own copy of the Galaxy source code, you can also generate your own version of this documentation.

http://galaxy-central.readthedocs.org
Galaxy API Documentation

Background

In addition to being accessible through a web interface, Galaxy can now also be accessed programmatically, through shell scripts and other programs. The web interface is appropriate for things like exploratory analysis, visualization, construction of workflows, and rerunning workflows on new datasets.

The web interface is less suitable for things like

- Connecting a Galaxy instance directly to your sequencer and running workflows whenever data is ready
- Running a workflow against multiple datasets (which can be done with the web interface, but is tedious)
- When the analysis involves complex control, such as looping and branching.

The Galaxy API addresses these and other situations by exposing Galaxy internals through an additional interface, known as an Application Programming Interface, or API.

Quickstart
This gives detailed information about the specific member in question, in this case the History. To view history contents, do the following:

```
% ./display.py my_key http://localhost:4096/api/histories/8c49be44cfe29bc/content
Collection Members

#1: /api/histories/8c49be44cfe29bc/contents/6f91353f3eb0fa4a
  name: Pasted Entry
  type: file
  id: 6f91353f3eb0fa4a
```

What we have here is another Collection of items containing all of the datasets in this particular history. Finally, to view details of a particular dataset in this collection, execute the following:

```
% ./display.py my_key http://localhost:4096/api/histories/8c49be44cfe29bc/content
Member Information

  misc_blurb: 1 line
  name: Pasted Entry
  data_type: txt
  deleted: False
  file_name: /Users/yoplait/work/galaxy-stock/database/files/000/dataset_82.dat
  state: ok
download_url: /datasets/6f91353f3eb0fa4a/display?to_ext=txt
visible: True
genome_build: ?
model_class: HistoryDatasetAssociation
file_size: 17
metadata_data_lines: 1
id: 6f91353f3eb0fa4a
misc_info: uploaded txt file
metadata_dbkey: ?
```
Extending Galaxy
Tools & Datatypes

Hopefully everyone understands this by now?

There is another session, “Introduction to Tool and Data Source Configuration”
Visualizations

(See talk from Carl Eberhard)
Visualization and visual analytics
Architecture

Web browser

Galaxy
HTML UI

Galaxy
Tools

Datasets
Technical Highlights

Rough pluggable support for generic JavaScript visualizations + data providers

Data providers = fast, random access to data in Python, JS

API: run tools, run them on data subsets

Backbone + HTML5 objects for Web-based genomic visualizations
  * e.g. data managers, linear and circular views

JS binding to Galaxy API (blendJS?)
  * visualizations, tools, datasets
  * custom Galaxy UIs
// -- Viz set up. --

var genome = new Genome(JSON.parse('${ h.to_json_string( genome ) }'));
visualization = new GenomeVisualization(JSON.parse('${ h.to_json_string( viz_config ) }'),
viz_view = new CircsterView({
    width: 600,
    height: 600,
    // Gap is difficult to set because it very dependent on chromosome size and organization.
    total_gap: 2 * Math.PI * 0.2,
    genome: genome,
    model: visualization,
    radius_start: 100,
    dataset_arc_height: 15
});

// -- Render viz. --

viz_view.render();
$('#vis').append(viz_view.$el);
Eggs and dependency packaging
New controllers and webapps
Q&A