Galaxy

Internals, organization, control flow

www.getgalaxy.org

James, Dan, Greg

@jxtx / #usegalaxy
The Plan

1. What’s in the galaxy-central repository?
2. Galaxy web application architecture
3. Control flow in the Galaxy web application
4. Tools in the age of the toolshed
5. Galaxy Workflows
6. Galaxy data organization
0. The right ways to be involved with Galaxy
IRC: irc.freenode.net #galaxyproject

Trello: https://trello.com/b/75c1kASa/galaxy-development
1. Getting around the Galaxy repository
(lets look at the code)
2. Galaxy application architecture
Job Manager -> Job Handlers -> Job Runner

<<thread>>
The old way

- Renderer +
  progressive JS

- HTML on the wire,
  typically from mako

- controllers

The new way

- Backbone.js
  MVC on browser

- JSON on the wire

- controllers.api

- Universe App…
The old way

User stuff (prefs, etc)

Tool forms

Reports

Tool shed

*Many of these have an API but it is not yet used by the UI

The new way

Visualizations

History

Tool menu

Most grids

In between

Workflows

Data Libraries
Example of the new way: Tool menu generation
Example of the new way: Tool form generation
So many languages!

- Javascript
  - Mostly on the browser side, all new UI components

- JSON
  - API, database, etc

- Python
  - All of the core of Galaxy

- Other languages (e.g. C)
  - Only through Python eggs

- Handlebars
  - Browser side templating

- Cheetah
  - Only tool configuration files

- Mako
  - Most web controllers
3: Control flow: running jobs
4. Tools in the age of the toolshed
The old way

1. Each tool specified by a tool.xml somewhere on the local filesystem (but typically under tools)

2. Tools to be loaded specified in tool_conf.xml, loaded by Galaxy at startup — no representation in database beyond tool ids

No way to access old tool configurations after updates
In the ToolShed

stored as mercurial repo on disk in ToolShed
several types: unrestricted, suite, tool dependency
unrestricted can have multiple installable revisions

lib.galaxy.webapps.tool_shed / lib.tool_shed
In the ToolShed

- ToolShed Repository
  - stored as mercurial repo on disk in ToolShed
  - several types: unrestricted, suite, tool dependency
  - unrestricted can have multiple installable revisions

Each installable revision can have

- repository_dependencies.xml
- tool_dependencies.xml
- tool.xml

- Tool
  - Workflows
  - Datatypes
  - Data Managers
  - etc

- Installation Recipe
  - an installed package/binary
Installed in Galaxy

```
app.install_model

ToolShed Repository
  source: toolshed, owner, repo name, changeset revision
  metadata: json representation of repo contents
  one per installed installable revision

Repository Dependency
  backref
  via
  RepositoryRepositoryDependencyAssociation

ToolDependency
  dependency name
  dependency version
  dependency type: package, environment setting

ToolVersion
  tool_id
  ToolVersion Association
  tool_id
  parent ToolVersion allows tool lineage
```
5. Galaxy workflows
Workflow modules have:

Config time state — in the workflow editor used to generate the form associated with a given step and update it.

Runtime state — similar but used for parameters set at workflow runtime.

As well as conversion from JSON <-> Workflow Module instance <-> workflow_step encoded in database.
Workflow scheduling:

Currently workflows are scheduled like any other job.

All intermediate datasets and connections are created and each step is sent as a job to the JobManager.

Pausing: when intermediate steps fail the workflow is paused. Although, this actually applies to any dependent jobs.
6. Galaxy data organization
Where does data in Galaxy go?

1. “Metadata” is stored in a SQL database (preferable Postgres): Users, workflows, histories, dataset metadata… everything a user creates interacting with Galaxy except the raw contents of datasets

2. Dataset contents is stored in file_path, typically database/files

3. Data used by tools that is not user specific is stored in
https://wiki.galaxyproject.org/Admin/Internals/DataModel
Galaxy data model is not database entity driven

Entities are defined in galaxy.model as objects

SQLAlchemy is used for object relation mapping

Mappings are defined in galaxy.model.mapping in two parts — a table definition and a mapping between objects and tables including relationships

Migrations allow the schema to be migrated forward automatically

It *rarely* makes sense to access the Galaxy database directly
Where does data in Galaxy go?

1. “Metadata” is stored in a SQL database (preferable Postgres): Users, workflows, histories, dataset metadata… everything a user creates interacting with Galaxy except the raw contents of datasets.

2. Dataset contents is stored in file_path, typically database/files objectstore.

3. Data used by tools that is not user specific is stored in
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()
```

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

Distributed Object Store

Galaxy

Distributed Object Store

FS
FS
FS
FS

Distribution by weight
Zero weight
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
Data tables and location files
Data tables provide an abstraction which tools use to access indexes of data which can be accessed on the local filesystem
Tool config reference data table by name with abstract columns

```
<param name="maftype" type="select" label="Choose alignments">
  <options from_data_table='indexed_maf_files'>
    <filter type="data_meta" ref="input1" key="dbkey" column="dbkey">
      <validator type="no_options" message="No alignments are available">
        </validator>
    </filter>
  </options>
</param>
```
Data tables configuration maps abstract data table to a concrete file

```xml
<!-- Locations of MAF files that have been indexed with bx-python -->
<table name="indexed_maf_files">
  <columns>name, value, dbkey, species</columns>
  <file path="tool-data/maf_index.loc" />
</table>
```
Which can provide any information, but typically locations of data at a given site.
Data Managers
Special class of Galaxy tool which allows for the download and/or creation of data that is stored within Data Tables and their location files.

These tools handle e.g. the creation of indexes and the addition of entries/lines to the data table / .loc file via the Galaxy admin interface.

Data Managers can be defined locally or installed through the Tool Shed.

Available in: Admin GUI, Workflows, API
Special class of Galaxy tool

This tool writes a JSON description of new data table entries as content of tool output file. For example:

```json
{
  "data_tables": {
    "all_fasta": [
      {
        "path": "sacCer2.fa",
        "dbkey": "sacCer2",
        "name": "S. cerevisiae June 2008 (SGD/sacCer2) (sacCer2)",
        "value": "sacCer2"
      }
    ]
  }
}
```

This creates a new entry in the Tool Data Table:

<table>
<thead>
<tr>
<th>&lt;unique_build_id&gt;</th>
<th>&lt;dbkey&gt;</th>
<th>&lt;display_name&gt;</th>
<th>&lt;file_path&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>sacCer2</td>
<td>sacCer2</td>
<td>S. cerevisiae June 2008 (SGD/sacCer2) (sacCer2)</td>
<td>/Users/dan/galaxy-central/tool-data/sacCer2/seq/sacCer2.fa</td>
</tr>
</tbody>
</table>

Where the sacCer2.fa file was placed by the tool in the output file’s extra_files_path.
data_manager entry inside <data_managers> tag in data_mananger_conf.xml

```xml
<data_manager tool_file="data_manager/bwa_index_builder.xml" id="bwa_index_builder" version="0.0.1">
  <data_table name="bwa_indexes">
    <output>
      <column name="value" />
      <column name="dbkey" />
      <column name="name" />
      <column name="path" output_ref="out_file">
        <move type="directory" relativize_symlinks="True">
          <target base="${GALAXY_DATA_MANAGER_DATA_PATH}/${dbkey}/bwa_index/${value}"
        </move>
        <value_translation>${GALAXY_DATA_MANAGER_DATA_PATH}/${dbkey}/bwa_index/${value}/${path}</value_translation>
        <value_translation type="function">abspath</value_translation>
      </column>
    </output>
  </data_table>
</data_manager>
```

informs Galaxy about which data tables to expect for new entries and special handling of provided JSON values and files