Welcome!

This material is the result of a collaborative work. Thanks the Galaxy Training Network and all the contributors!
Galaxy Code Architecture
Questions

- How is the Galaxy code structured?
- What do the various other projects related to Galaxy do?
- What happens when I start Galaxy?
Objectives

- Explore various aspects of the Galaxy codebase.
- Understand the various top-level files and modules in Galaxy.
- Understand how dependencies work in Galaxy's frontend and backend.
Galaxy Architecture
We're here to answer your questions about Galaxy architecture!
Getting involved in Galaxy
Gitter: galaxyproject/Lobby
IRC: irc.freenode.net#galaxyproject
GitHub: github.com/galaxyproject
Twitter: #usegalaxy, @galaxyproject
Contributing

All Galaxy development happens on GitHub

The /galaxyproject projects
galaxyproject/galaxy

The main Galaxy application.

Web interface, database model, job running, etc...

Also includes other web applications including the ToolShed and Reports
galaxyproject/cloudman

Galaxy CloudMan - a web application which manages a Galaxy cluster in the cloud.

galaxyproject/cloudlaunch

CloudLaunch web application to make it easy to launch images on a cloud, drives https://launch.usegalaxy.org
Galaxy tools maintained by the *IUC* ("Intergalactic Utilities Commission").

A variety of tools, generally of high quality including many of the core tools for Galaxy main.

Demonstrates *current tool development best practices* - development on github and then deployed to test/main ToolSheds

Many older tools appearing on usegalaxy.org.
Tools Aside - More Repositories

Other repositories with high quality tools:

- Björn Grüning's repo
- Peter Cock’s repos:
  - blast repo
  - pico repo
  - mira repo
- ENCODE tools
- Biopython repo
- Galaxy Proteomics repo
- Colibread Galaxy Tools
- Greg von Kuster's repo
- TGAC repo
- AAFC-MBB Canada repo
- Mark Einon's repo
galaxyproject/starforge

Build Galaxy Tool dependencies for the ToolShed in Docker containers

Build Galaxy framework dependencies as Python wheels
galaxyproject/planemo

Command line utilities to assist in the development of Galaxy tools. Linting, testing, deploying to ToolSheds...

*The best practice approach for Galaxy tool development!*

galaxyproject/planemo-machine

Builds Galaxy environments for Galaxy tool development including Docker container, virtual machines, Google compute images
galaxyproject/\{ansible-*, \*-playbook\}

Ansible components to automate almost every aspect of Galaxy installation and maintenance.

Ansible is an advanced configuration management system

These playbooks are used to maintain Galaxy main, cloud and Docker images, virtual machines, ...
galaxyproject/pulsar

Distributed job execution engine for Galaxy.

Stages data, scripts, configuration.

Can run jobs on Windows machines.

Can act as its own queuing system or access an existing cluster DRM.
galaxyproject/bioblend

Official Python client for the Galaxy, ToolShed, and CloudMan APIs.

Best documented path to scripting the Galaxy API.
- galaxyproject/blend4php
- jmchilton/blend4j
- chapmanb/clj-blend

Galaxy API bindings for other languages.
High quality Docker containers for stable Galaxy environments.

Releases corresponding to each new version of Galaxy.

Many flavors available.
Principles
Whereas the architecture of the frontend (Web UI) aims for consistency and is highly opinionated, the backend (Python server) is guided by flexibility and is meant to be driven by plugins whenever possible.
The target audience is a *bench scientist* - no knowledge of programming, paths, or command lines should be assumed.

- Consistent colors, fonts, themes, etc...
- Reusable components for presenting common widgets - from the generic (forms and grids) to the specific (tools and histories).
- Tied to specific technologies:
  - JavaScript driven
  - Backbone for MVC
  - webpack & RequireJS for modules
A Plugin Driven Backend

Galaxy's backend is in many ways driven by *pluggable interfaces* and can be adapted to many different technologies.

- SQLAlchemy allows using sqlite, postgres, or MySQL for a database.
- Many different cluster backends or job managers are supported.
- Different frontend proxies (e.g. nginx) are supported as well as web application containers (e.g. uWSGI).
- Different storage strategies and technologies are supported (e.g. S3).
- Tool definitions, job metrics, stat middleware, tool dependency resolution, workflow modules, datatype definitions are all plugin driven.
Galaxy has long been guided by the principle that cloning it and calling the `run.sh` should "just work" and should work quickly.

So by default Galaxy does not require:

- Compilation - it fetches *binary wheels* for your platform.
- A job manager - Galaxy can act as one.
- An external database server - Galaxy can use an sqlite database.
- A web proxy or external Python web server.

A Plugin Driven Backend but...
Web Frameworks
Browser

HTTP

Server

Page Request

Static Content (HTML+JS+CSS)

MVC with backbone.js

API Request (JSON)

Build JSON response in Galaxy "API" controllers

API Response (JSON)

HTML rendered from client-side templates (in Backbone views).

Browser

Server
Backbone MVC
Browser

Server

HTTP

Page Request

Build HTML fragments using Mako Python library in Galaxy "web" controllers

Static Content (HTML+JS+CSS)

Render HTML fragments with JavaScript
Galaxy WSGI

galaxy.util.
pastescr ipt.serve

WSGI Web Server

Middleware Stack

galaxy.webapps.galaxy.
GalaxyWebApplication

galaxy.app.
UniverseApplication

Model

Toolbox

Job Creation

Datatypes Reg.

...
• Python interface for web servers defined by PEP 333 -
  https://www.python.org/dev/peps/pep-0333/.
• Galaxy moving from Paster to uwsgi to host the application.
  ○ http://pythonpaste.org/
  ○ https://uwsgi-docs.readthedocs.io/
A WSGI function:

```python
def app(environ, start_response):
    • Middleware act as filters, modify the environ and then pass through to the next webapp
    • Galaxy uses several middleware components defined in the wrap_in_middleware function of galaxy.webapps.galaxy.buildapp.
```
Galaxy’s WSGI Middleware

Middleware configured in
galaxy.webapps.galaxy.buildapp#wrap_in_middleware.

- paste.httpexceptions#make_middleware
- galaxy.web.framework.middleware.remoteuser#RemoteUser (if configured)
- paste.recursive#RecursiveMiddleware
- galaxy.web.framework.middleware.sentry#Sentry (if configured)
- Various debugging middleware (linting, interactive exceptions, etc...)
- galaxy.web.framework.middleware.statsd#StatsdMiddleware (if configured)
- galaxy.web.framework.middleware.xforwardedhost#XForwardedHostMiddleware
- galaxy.web.framework.middleware.request_id#RequestIDMiddleware
Setup on webapp in galaxy.web.webapps.galaxy.buildapp.py.

```python
webapp.add_route(
    '/datasets/:dataset_id/display/{filename:.+?}',
    controller='dataset', action='display',
    dataset_id=None, filename=None
)
```

URL /datasets/278043/display matches this route, so handle_request will

- lookup the controller named “dataset”
- look for a method named “display” that is exposed
- call it, passing dataset_id and filename as keyword arg

Simplified handle_request from lib/galaxy/web/framework/base.py.

```python
def handle_request(self, environ, start_response):
    path_info = environ.get('PATH_INFO', '')
    map = self.mapper.match(path_info, environ)
    if path_info.startswith('/api'):
        controllers = self.api_controllers
    else:
        controllers = self.controllers
    trans = self.transaction_factory(environ)

    controller_name = map.pop('controller', None)
    controller = controllers.get(controller_name, None)

    # Resolve action method on controller
    action = map.pop('action', 'index')
    method = getattr(controller, action, None)

    kwargs = trans.request.params.mixed()
    # Read controller arguments from mapper match
    kwargs.update(map)

    body = method(trans, **kwargs)
    # Body may be a file, string, etc... respond with it.
```
API Controllers

- lib/galaxy/webapps/galaxy/controllers/api/
- Exposed method take `trans` and request parameters and return a JSON response.
- Ideally these are *thin*
  - Focused on "web things" - adapting parameters and responses and move "business logic" to components not bound to web functionality.
Legacy Controllers

- `lib/galaxy/webapps/galaxy/controllers/`
- Return arbitrary content - JSON, HTML, etc...
- Render HTML components using `mako` templates (see `templates/`)
- The usage of these should decrease over time.
Application Components
Galaxy Models

- Database interactions powered by SQLAlchemy - https://www.sqlalchemy.org/.
- Galaxy doesn't think in terms "rows" but "objects".
- Classes for Galaxy model objects in `lib/galaxy/model/__init__.py`.
- Classes mapped to tables in `lib/galaxy/model/mapping.py`
  - Describes table definitions and relationships.
Galaxy Model Migrations

- A migration describes a linear list of database "diff"s to end up with the current Galaxy model.
- Allow the schema to be migrated forward automatically.
- Each file in lib/galaxy/model/migrate/versions/
  - 0124_job_state_history.py
  - 0125_workflow_step_tracking.py
  - 0126_password_reset.py
Database Diagram

https://galaxyproject.org/admin/internals/data-model/
**HistoryDatasetAssociation**
- hid: integer
- history_id: integer
- dataset_id: integer
- state: string
- name: string
- info: string

**Dataset**
- object_store_id: string
- external_filename: string
- _extra_files_path: string
- file_size: integer
- total_size: integer
Metadata

- Typed key-value pairs attached to HDA.
- Keys and types defined at the datatype level.
- Can be used by tools to dynamically control the tool form.
History

**HistoryDatasetAssociation**
- history_content_type = 'dataset'
- hid

**HistoryDatasetCollectionAssociation**
- history_content_type = 'dataset_collection'
- hid

**DatasetCollection**
- collection_type

**DatasetCollectionElement**
- element_index
- element_identifier
- element_index
- element_identifier
Stored Workflow → Workflow → Workflow Step → Workflow Module

(Tool Module, Input Dataset, etc)
Job Components

- Job is placed into the database and picked up by the job handler.
- Job handler (JobHandler) watches the job and transitions job's state - common startup and finishing.
- Job mapper (JobRunnerMapper) decides the "destination" for a job.
- Job runner (e.g. DrammaJobRunner) actual runs the job and provides an interface for checking status.
Object Store

```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)
```
ObjectStore

exists(obj)
file_ready(obj)
create(obj)
size(obj)
delete(obj)
get_data(obj)
get_filename(obj)
update_from_file(obj)
get_store_usage_percent()

DiskObjectStore
NestedObjectStore
S3ObjectStore

IRODSObjectStore
DistributedObjectStore
HierarchicalObjectStore
Visualization Plugins

Adding new visualizations to a Galaxy instance

- Configuration file (XML)
- Base template (Mako or JavaScript)
- Additional static data if needed (CSS, JS, ...)

Learn more about it with our visualization tutorial.
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE visualization SYSTEM "../../visualization.dtd">
<visualization name="Charts">
  <data_sources>
    <data_source>
      <model_class>HistoryDatasetAssociation</model_class>
      <test type="isinstance" test_attr="datatype" result_type="datatype">tabular.Tabular</test>
      <test type="isinstance" test_attr="datatype" result_type="datatype">tabular.CSV</test>
      <to_param param_attr="id">dataset_id</to_param>
    </data_source>
  </data_sources>
  <params>
    <param type="dataset" var_name_in_template="hda" required="true">dataset_id</param>
  </params>
  <entry_point entry_point_type="mako">charts.mako</entry_point>
</visualization>
Visualization Examples

All in config/plugins/visualizations:

- csg - Chemical structure viewer
- graphviz - Visualize graph data using cytoscape.js
- charts - A more elaborate builds on more Galaxy abstractions.
- trackster - Genome browser, deeply tied to Galaxy internals.
Data Providers

Provide efficient access to data for viz & API

Framework provides direct link to read the raw dataset or use data providers to adapt it

In config, assert that visualization requires a given type of data providers

Data providers process data before sending to browser - slice, filter, reformat, ...

Interactive Environments

Similar to visualizations: config and template
Within the base template, launch a Docker container running a web accessible process
Build a UI that accesses that process through a proxy

Learn more about it with our IE tutorial.
Interactive Environments - Examples

All in config/plugins/interactive_environments:

- jupyter
- rstudio
- phinch
- bam_iobio
Managers

High-level business logic that tie all of these components together.

Controllers should ideally be thin wrappers around actions defined in managers.

Whenever model require more than just the database, the operation should be defined in a manager instead of in the model.
Client Architecture
Client Directories

- Source stylesheets and JavaScript in `client/galaxy/{style|scripts}`
- "Packed" scripts served by Galaxy stored in `static/{style|scripts}`
  - webpack builds these "compiled" artifacts

Upshot - modify files in `client` and rebuild with `make client` before deployment.
Building the Client - Makefile Targets

client: grunt style ## Rebuild all client-side artifacts

grunt: npm-deps ## Calls out to Grunt to build client
    cd client && node_modules/grunt-cli/bin/grunt

style: npm-deps ## Calls the style task of Grunt
    cd client && node_modules/grunt-cli/bin/grunt style

npm-deps: ## Install NodeJS dependencies.
    cd client && npm install
Build tool for node/JavaScript, tasks in client/Gruntfile.js. Default task is

grunt.registerTask( 'default', [ 'check-modules', 'uglify', 'webpack' ] );

- check-modules Verifies node dependencies are correct and exact.
- uglify Compresses JavaScript modules in client and move to static and creates source maps.
  - JavaScript loads much faster but difficult to debug by default
  - Source maps re-enable proper stack traces.
- webpack Bundles modules together into a single JavaScript file - quickly loadable.
JavaScript Modules - The Problem

From https://requirejs.org/docs/why.html:

- Web sites are turning into Web apps
- Code complexity grows as the site gets bigger
- Assembly gets harder
- Developer wants discrete JS files/modules
- Deployment wants optimized code in just one or a few HTTP calls
JavaScript Modules - The Solution

From [https://requirejs.org/docs/why.html](https://requirejs.org/docs/why.html):

- Some sort of `#include/import/require`
- Ability to load nested dependencies
- Ease of use for developer but then backed by an optimization tool that helps deployment

RequireJS an implementation of AMD.
This is the workflow tool form.

```javascript
/*
 * This is the workflow tool form.
 */
define(['utils/utils', 'mvc/tool/tool-form-base'],
    function (Utils, ToolFormBase) {

        // create form view
        var View = ToolFormBase.extend({
            ...
        });

        return {
            View: View
        };
});
```
modules with dependencies

static assets
webpack in Galaxy

- Turns Galaxy modules into an "app".
- Builds two bundles currently - a common set of libraries and an analysis "app".
- https://github.com/galaxyproject/galaxy/issues/1041
- https://github.com/galaxyproject/galaxy/pull/1144
Webhooks is a system in Galaxy which can be used to write small JS and/or Python functions to change predefined locations in the Galaxy client.

In short: A plugin infrastructure for the Galaxy UI

You can learn more about webhooks using our webhook training.
Webhook masthead example

At the header menu: Enabling the overlay search, link to communities ...

You can learn more about webhooks using our webhook training.
Webhook tool/workflow example

1 job has been successfully added to the queue - resulting in the following datasets:

34: Filter on data 24

You can check the status of queued jobs and view the resulting data by refreshing the History pane. When the job has been run the status will change from 'running' to 'finished' if completed successfully or 'error' if problems were encountered.

Shown after tool or workflow execution. Comics, citations, support ...

You can learn more about webhooks using our webhook training.
Webhook history-menu example

Adds an entry to the history menu - no functionality as of now

You can learn more about webhooks using our webhook training.
Galaxy uses the less CSS preprocessor - https://lesscss.org/
Rebuild style with `make style`
Less files in `client/galaxy/style/less`
Build happens with grunt recipe in `client/grunt-tasks/style.js`
Dependencies
Dependencies - Python

script/common_startup.sh sets up a virtualenv with required dependencies in
$GALAXY_ROOT/.venv (or $GALAXY_VIRTUAL_ENV if set).

- Check for existing virtual environment, if it doesn't exist check for
  virtualenv.
- If virtualenv exists, use it. Otherwise download it as a script and setup a
  virtual environment using it.
- . "$GALAXY_VIRTUAL_ENV/bin/activate"
- Upgrade to latest pip to allow use of binary wheels.
- pip install -r requirements.txt --index-url
  https://wheels.galaxyproject.org/simple
- Install dozens of dependencies.
These come bundled with Galaxy, so do not need to be fetched at runtime.

- Dependencies are defined galaxy/client/bower.json.
- Bower (https://bower.io/) is used to re-fetch these.
- cd client; grunt install-libs
Galaxy Startup Process
Cloning Galaxy

$ git clone https://github.com/galaxyproject/galaxy.git galaxy
  Cloning into 'galaxy'...
remote: Counting objects: 173809, done.
remote: Total 173809 (delta 0), reused 0 (delta 0), pack-reused 173809
Receiving objects: 100% (173809/173809), 55.18 MiB | 11.08 MiB/s, done.
Resolving deltas: 100% (137885/137885), done.
Checking connectivity... done.
$ cd galaxy
$ git checkout -b master origin/master
  Branch master set up to track remote branch master from origin.
  Switched to a new branch 'master'
$ sh run.sh
CopyingConfigs

$ sh run.sh

Initializing config/migrated_tools_conf.xml from migrated_tools_conf.xml.sample
Initializing config/shed_tool_conf.xml from shed_tool_conf.xml.sample
Initializing config/shed_tool_data_table_conf.xml from shed_tool_data_table_conf.xml.sample
Initializing config/shed_data_manager_conf.xml from shed_data_manager_conf.xml.sample
Initializing tool-data/shared/ucsc/builds.txt from builds.txt.sample
Initializing tool-data/shared/ucsc/manual_builds.txt from manual_builds.txt.sample
Initializing tool-data/shared/ucsc/ucsc_build_sites.txt from ucsc_build_sites.txt.sample
Initializing tool-data/shared/igv/igv_build_sites.txt from igv_build_sites.txt.sample
Initializing tool-data/shared/rviewer/rviewer_build_sites.txt from rviewer_build_sites.txt.sample
Initializing static/welcome.html from welcome.html.sample
Setting up .venv and pip

Using real prefix '/usr'
New python executable in .venv/bin/python
Installing setuptools, pip, wheel...done.
Activating virtualenv at .venv
Collecting pip>=8.1
  Using cached pip-8.1.2-py2.py3-none-any.whl
Installing collected packages: pip
  Found existing installation: pip 7.1.2
  Uninstalling pip-7.1.2:
    Successfully uninstalled pip-7.1.2
Successfully installed pip-8.1.2
Collecting bx-python==0.7.3 (from -r requirements.txt (line 2))
  Downloading https://wheels.galaxyproject.org/packages/bx_python-0.7.3-cp27-cp27mu-manylinux1_x86_64.whl (2.1MB)
Collecting MarkupSafe==0.23 (from -r requirements.txt (line 3))
  Downloading https://wheels.galaxyproject.org/packages/MarkupSafe-0.23-cp27-cp27mu-manylinux1_x86_64.whl
Collecting PyYAML==3.11 (from -r requirements.txt (line 4))
  Downloading https://wheels.galaxyproject.org/packages/PyYAML-3.11-cp27-cp27mu-manylinux1_x86_64.whl (367kB)
Collecting SQLAlchemy==1.0.8 (from -r requirements.txt (line 5))
  Downloading https://wheels.galaxyproject.org/packages/SQLAlchemy-1.0.8-cp27-cp27mu-manylinux1_x86_64.whl (1.0MB)
Collecting mercurial==3.7.3 (from -r requirements.txt (line 6))
  Downloading https://wheels.galaxyproject.org/packages/mercurial-3.7.3-cp27-cp27mu-manylinux1_x86_64.whl (1.5MB)
...
Initial Debugging as App Starts

Activating virtualenv at .venv
INFO:galaxy.config:Logging at '10' level to 'stdout'
galaxy.queue_worker INFO 2016-06-23 19:11:51,925 Initializing main Galaxy Queue Worker on sqlalchemy+sqlite:///./database/control.sqlite?isolation_level=IMMEDIATE
tool_shed.tool_shed_registry DEBUG 2016-06-23 19:11:51,951 Loading references to tool sheds from ./config/tool_sheds_conf.xml.sample
tool_shed.tool_shed_registry DEBUG 2016-06-23 19:11:51,951 Loaded reference to tool shed: Galaxy Main Tool Shed
galaxy.app DEBUG 2016-06-23 19:11:51,956 Using "galaxy.ini" config file:
/home/john/workspace/galaxy-clean/config/galaxy.ini.sample
Database Migrations

migrate.versioning.repository DEBUG 2016-06-23 19:11:51,993 Loading repository lib/galaxy/model/migrate...
migrate.versioning.script.base DEBUG 2016-06-23 19:11:51,994 Loading script lib/galaxy/model/migrate/versions/0001_initial_tables.py...
migrate.versioning.script.base DEBUG 2016-06-23 19:11:51,994 Script lib/galaxy/model/migrate/versions/0001_initial_tables.py loaded successfully
migrate.versioning.script.base DEBUG 2016-06-23 19:11:52,009 Loading script lib/galaxy/model/migrate/versions/0002_metadata_file_table.py...

galaxy.model.migrate.check INFO 2016-06-23 19:13:32,812 Migrating 128 -> 129...
galaxy.model.migrate.check INFO 2016-06-23 19:13:33,436 Migration script to allow invalidation of job external output metadata temp files
galaxy.model.migrate.check INFO 2016-06-23 19:13:33,437 Migration script to change the value column of user_preference from varchar to text.
galaxy.model.migrate.check INFO 2016-06-23 19:13:34,325 Migration script to support subworkflows and workflow request input parameters

galaxy.model.migrate.check INFO 2016-06-23 19:13:35,633 Migration script to support subworkflows and workflow request input parameters
Everything after here happens every time
migrate.versioning.repository DEBUG 2016-06-23 19:13:35,635 Loading repository lib/tool_shed/galaxy_install/migrate...

migrate.versioning.script.base DEBUG 2016-06-23 19:13:35,636 Loading script lib/tool_shed/galaxy_install/migrate/versions/0001_tools.py...


migrate.versioning.script.base DEBUG 2016-06-23 19:13:35,636 Loading script lib/tool_shed/galaxy_install/migrate/versions/0002_tools.py...


migrate.versioning.script.base DEBUG 2016-06-23 19:13:35,636 Loading script lib/tool_shed/galaxy_install/migrate/versions/0003_tools.py...


migrate.versioning.script.base DEBUG 2016-06-23 19:13:35,636 Loading script lib/tool_shed/galaxy_install/migrate/versions/0004_tools.py...


migrate.versioning.script.base DEBUG 2016-06-23 19:13:35,636 Loading script lib/tool_shed/galaxy_install/migrate/versions/0005_tools.py...


tool_shed.galaxy_install.migrate.check DEBUG 2016-06-23 19:13:35,668 The main Galaxy tool shed is not currently available, so skipped tool migration...

Loading datatypes from ./config/datatypes_conf.xml.sample

galaxy.datatypes.registry DEBUG 2016-06-23 19:13:35,751 Retrieved datatype module galaxy.datatypes.data:GenericAsn1 from the datatype registry.
...
galaxy.datatypes.registry DEBUG 2016-06-23 19:13:35,833 Loaded build site 'ucsc': tool-data/shared/ucsc/ucsc_build_sites.txt with display sites: main,test,archaea,ucla


galaxy.datatypes.registry DEBUG 2016-06-23 19:13:35,834 Loaded build site 'ensembl': tool-data/shared/ensembl/ensembl_build_sites.txt


galaxy.tools.data DEBUG 2016-06-23 19:13:35,871 Loaded tool data table 'all_fasta'


... 


... 

galaxy.tools.data DEBUG 2016-06-23 19:13:36,210 Loaded tool data table 'vcf_iobio'

galaxy.tools.data INFO 2016-06-23 19:13:36,211 Could not find tool data tool-data/biom_simple_display.loc, reading sample

galaxy.tools.data DEBUG 2016-06-23 19:13:36,211 Loaded tool data table 'biom_simple_display'
Job Configuration, Citation Cache

galaxy.jobs DEBUG 2016-06-23 19:13:36,233 Loading job configuration from /home/john/workspace/galaxy-clean/config/galaxy.ini.sample
galaxy.jobs DEBUG 2016-06-23 19:13:36,233 Done loading job configuration
galaxy.tools.toolbox.base INFO 2016-06-23 19:13:36,279 Parsing the tool configuration ./config/tool_conf.xml.sample


galaxy.tools.toolbox.base DEBUG 2016-06-23 19:13:36,294 Loaded tool id: ucsc_table_direct1, version: 1.0.0 into tool panel.

galaxy.tools.toolbox.base DEBUG 2016-06-23 19:13:36,296 Loaded tool id: ucsc_table_direct_test1, version: 1.0.0 into tool panel.

galaxy.tools.toolbox.base DEBUG 2016-06-23 19:13:36,298 Loaded tool id: ucsc_table_direct_archaea1, version: 1.0.0 into tool panel.

...
Tool Dependency Resolution and Indexing

galaxy.tools.deps WARNING 2016-06-23 19:13:36,498 Path './database/dependencies' does not exist, ignoring
galaxy.tools.deps WARNING 2016-06-23 19:13:36,498 Path './database/dependencies' is not directory, ignoring
galaxy.tools.deps DEBUG 2016-06-23 19:13:36,503 Unable to find config file './dependency_resolvers_conf.xml'
galaxy.tools.search DEBUG 2016-06-23 19:13:37,709 Toolbox index finished. It took: 0:00:01.229406
Display Applications

galaxy.datatypes.registry  DEBUG  2016-06-23  19:13:37,797 Loaded display application 'ensembl_bam' for datatype 'bam', inherit=False.

...
Datatype Converters

galaxy.datatypes.registry DEBUG 2016-06-23 19:13:38,010 Loaded converter: CONVERTER_Bam_Bai_0
galaxy.datatypes.registry DEBUG 2016-06-23 19:13:38,011 Loaded converter: CONVERTER_bam_to_bigwig_0
galaxy.datatypes.registry DEBUG 2016-06-23 19:13:38,012 Loaded converter: CONVERTER_bed_to_gff_0
galaxy.datatypes.registry DEBUG 2016-06-23 19:13:38,012 Loaded converter: CONVERTER_bed_to_bgzip_0
galaxy.datatypes.registry DEBUG 2016-06-23 19:13:38,013 Loaded converter: CONVERTER_bed_to_tabix_0
...
Special Tools

Visualization Plugins

galaxy.visualization.plugins.config_parser INFO 2016-06-23 19:13:38,110 Visualizations plugin disabled: Circster. Skipping...
galaxy.visualization.plugins.config_parser INFO 2016-06-23 19:13:38,111 template syntax is deprecated: use entry_point instead
galaxy.visualization.plugins.config_parser INFO 2016-06-23 19:13:38,111 template syntax is deprecated: use entry_point instead
galaxy.visualization.plugins.config_parser INFO 2016-06-23 19:13:38,113 Visualizations plugin disabled: Sweepster. Skipping...
Tours

galaxy.tours INFO 2016-06-23 19:13:38,125 Loaded tour 'core.scratchbook'
galaxy.tours INFO 2016-06-23 19:13:38,158 Loaded tour 'core.galaxy_ui'
galaxy.tours INFO 2016-06-23 19:13:38,183 Loaded tour 'core.history'
Job Handler and Runners

galaxy.jobs.manager DEBUG 2016-06-23 19:13:38,196 Starting job handler
galaxy.jobs INFO 2016-06-23 19:13:38,196 Handler 'main' will load all configured runner plugins
galaxy.jobs.runners.state_handler_factory DEBUG 2016-06-23 19:13:38,198 Loaded 'failure' state handler from
galaxy.jobs.runners DEBUG 2016-06-23 19:13:38,198 Starting 5 LocalRunner workers
galaxy.jobs DEBUG 2016-06-23 19:13:38,200 Loaded job runner 'galaxy.jobs.runners.local:LocalJobRunner' as
galaxy.jobs DEBUG 2016-06-23 19:13:38,200 Legacy destination with id 'local:///', url 'local:///' converted
galaxy.jobs.handler INFO 2016-06-23 19:13:38,200 job handler stop queue started
galaxy.jobs.handler INFO 2016-06-23 19:13:38,222 job handler queue started
Ignore this...
Workflow Scheduler

galaxy.workflow.scheduling_manager DEBUG 2016-06-23 19:13:38,254 Starting workflow schedulers
Controllers
Middleware

galaxy.webapps.galaxy.buildapp DEBUG 2016-06-23 19:13:39,036 Enabling 'httpexceptions' middleware
galaxy.webapps.galaxy.buildapp DEBUG 2016-06-23 19:13:39,037 Enabling 'recursive' middleware
galaxy.webapps.galaxy.buildapp DEBUG 2016-06-23 19:13:39,042 Enabling 'error' middleware
galaxy.webapps.galaxy.buildapp DEBUG 2016-06-23 19:13:39,043 Enabling 'trans logger' middleware
galaxy.webapps.galaxy.buildapp DEBUG 2016-06-23 19:13:39,044 Enabling 'x-forwarded-host' middleware
galaxy.webapps.galaxy.buildapp DEBUG 2016-06-23 19:13:39,044 Enabling 'Request ID' middleware
galaxy.queue_worker INFO 2016-06-23 19:13:39,049 Binding and starting galaxy control worker for main
Starting server in PID 21102.
serving on http://127.0.0.1:8080
Production Galaxy - usegalaxy.org
<table>
<thead>
<tr>
<th>Default</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLite</td>
<td>PostgreSQL</td>
</tr>
<tr>
<td>Paste#http</td>
<td>uWSGI / nginx</td>
</tr>
<tr>
<td>Single process</td>
<td>Multiple processes</td>
</tr>
<tr>
<td>Single host</td>
<td>Multiple hosts</td>
</tr>
<tr>
<td>Local jobs</td>
<td>Jobs across many clusters</td>
</tr>
</tbody>
</table>

https://usegalaxy.org/production *
• Database server can scale way beyond default sqlite
• https://www.postgresql.org/
• github.com/galaxyproject/usegalaxy-playbook -> roles/galaxyprojectdotorg.postgresql
nginx (or Apache)

- Optimized servers for static content
- [https://www.nginx.com/resources/wiki/](https://www.nginx.com/resources/wiki/)
Threads in Python are limited by the GIL.

Running multiple processes of Galaxy and separate processes for web handling and job processing works around this.

This used to be an important detail - but uWSGI makes things a lot easier.
Cluster Support
Q&A
Key points

- Galaxy runs out of the box and fetches all needed dependencies.
- Running Galaxy in production requires work configuration.
- The architecture is designed to be pluggable and extendable.
- Learn more about different Galaxy aspects in our development section.
🎉 Congratulations on successfully completing this tutorial!

Found a typo? Something is wrong in this tutorial?
Edit it on GitHub