Galaxy in the wild

Galaxy Architecture and Supporting Production Level Genomics

Nate Coraor
The Galaxy Team
Penn State University
Ask questions (please!)
Who is this for?
- People who want to hack on Galaxy
- People who want to run a Galaxy server

Who isn’t this for?
- People who just want to use Galaxy (sorry)
Code architecture
Running a server
Galaxy mostly follows PEP-8 (but not entirely)

- Comment lines should be under 79 characters, code lines can be up to 200 characters if it improves readability
- Whitespace: whatever is most readable, both for blank lines and space around operators
Modularity and Reusability

- Datatypes: subclasses of more general datatypes
- Job runners: plugins, and subclasses of existing plugins
- API: standardized layout, framework
- Javascript: modeled data representation
- Code architecture
- Running a server
usegalaxy.org/production
Development/Production

- Galaxy is easy to start using:
  ```
  % sh run.sh
  ```

- Galaxy is not as easy to set up for a big lab, University, research org, etc.
Look at universe_wsgi.ini
Database

- SQLite (struck out)
- PostgreSQL
Galaxy process model
Galaxy multiprocess model
Galaxy multiprocess model

Bonus: upload and download via proxy
Cluster requires common filesystem
Cluster with LWR

Galaxy

translate paths
copy inputs in
copy outputs out

Cluster
(PBS, SGE, or local)

LWR

John Chilton
Job Execution Features

- All cluster DRM options available
  - walltime
  - number of cores, amount of memory
  - queue selection
- Dynamically choose the above based on
  - User
  - Tool
  - Tool parameters
  - Tool inputs
- Limit users’ concurrent jobs
Controlling Data

- Quotas
- Use Data Libraries for common data
  - Can upload from filesystem!
- Data removal
  - Allow users to “purge” unwanted data
  - Run dataset cleanup scripts to recover space
- Transparent compression if you can get it (zfs, btrfs)
Uploading data

- nginx upload module
- S(FTP)S upload
Reference Data

Figure S1. Schematic overview of reference data, location files, data tables, and tools. A Galaxy Data Manager handles downloading, creating and installing each of the required facets necessary to provide built-in reference data through the use of a web-based GUI.

Reference Genome
(*.fasta)

BWA Index Files on disk

File listing paths, genome build, descriptions, etc. of available BWA indexes
(bwa_index.loc)

Galaxy BWA Tool
(bwa_wrapper.xml)

Tool Data Table
(tool_data_table_conf.xml)

Dynamically use built-in data in any number of Galaxy Tools

abstraction layer

Galaxy's built-in reference data registry

bwa index [-a bwtsw|div|is] [-c] <in.fasta>
Reference Data

- Data tables and location files are a pain
- Use data managers
  - Versioned tools from the tool shed
  - Download data, build indexes

https://wiki.galaxyproject.org/Admin/Tools/DataManagers
Tricks and Technologies

- Run web processes from local disk to ensure network filesystem performance does not impact UI
  - Run handlers from shared filesystem
  - Process management via supervisord
  - Use Nagios to check individual handlers
  - Use sentry to aggregate tracebacks
  - Use config management
Config Management

- Formerly: CFEngine
- Now: Ansible
  - Dependencies: sshd, Python
  - No infrastructure required
  - Descriptions in YAML
  - Modules in Python
Ansible

- name: Update Galaxy to correct changeset
  hg: dest={{ galaxy_dir }} repo=http://bitbuck...

- name: Upgrade Galaxy database
  command: {{ galaxy_dir }}/manage_db.sh upgrade

- name: Install nginx
  apt: pkg=nginx-full
  when: ansible_os_family == "Debian"

- name: Copy nginx configs
  template: src=nginx.conf.j2 dest=/etc/nginx/...