Implementation of a standalone Galaxy instance for the creation of re-usable bioinformatics processing pipelines

Daniel J. Shea

August 16th, 2014

1 Introduction

Using yeast (Schizosaccharomyces Pombe and Saccharomyces Cervisiae) as our model organism, our research is conducted into understanding the biological processes underlying the formation, function, and epigenetic inheritance of silent chromatin domains. Three major areas of research are related to RNAi-mediated assembly of heterochromatin, SIR-mediated assembly of silent chromatin, and the regulation of recombination within the ribosomal DNA repeats. In order to improve the efficiency of data analysis via re-usable bioinformatics pipelines and to encourage collaboration and software re-use within the lab, we have chosen to implement a local instance of the Galaxy Project's open source web-based platform.

1.1 Heterochromatin's role in the silencing of genes

The physical organization of DNA in eukaryotes plays a role in the levels of genetic expression by restricting the physical access of transcriptional machinery to various regions of an organism's genome. The organization of DNA is structured around histones, octamers composed of highly conserved proteins, H2A, H2B, H3, and H4. These proteins form heterodimers H2A/H2B and H3/H4, respectively, that then further form tetramers. In turn, these tetramers then combine to form an octamer that is referred to as a histone. DNA winds around these histones forming a nucleosome, with small segments of DNA, referred to as linker DNA, between them. Structured around this basic organization of DNA are regions of tightly or loosely condensed chromatin, referred to as heterochromatin and euchromatin respectively. Heterochromatin plays a functional role in gene silencing through the physical restriction of access to DNA by transcriptional machinery involved in transcribing DNA into messenger RNA (mRNA).

1.2 Benefits derived from the implementation of Galaxy within the lab

Galaxy is an open, web-based platform for the analysis of data intensive biological research. It provides an interactive and readily accessible web interface to many of the commonly used bioinformatics tools available today. Computational pipelines may be constructed from within the web interface and then shared between users. Workflows are accessible via the workflow editor, which allows for the visualization and editing of processing pipelines. These workflows may then be abstracted for application to multiple data sets, and the workflows themselves can be shared between Galaxy users through the Galaxy web interface. This allows researchers within our environment to compose analytical workflows that may then be re-used or re-purposed by other collaborators in the lab with minimal effort.

2 Computational server architectural overview

Our computational server, chromosome.med.harvard.edu, serves as the platform on which we run our Galaxy instance. It should be noted however, that a local galaxy instance can be run on hardware with relatively modest specifications. Depending upon the data sets and types of analyses performed, you may be constrained by the available amount RAM on your server when dealing with particularly large data sets or application of computationally intensive analysis such as attempting to perform de novo assembly. The table below details the hardware configuration for chromosome.

2.1 Hardware configuration

chromosome.med.harvard.edu	
CPU:	(2) Intel(R) Xeon(R) CPU X5660 @ 2.80GHz, 6 Physical cores (12 Hyper-threaded)
Memory:	(18) 4GB DDR3-800 DIMMs for a total of 72GB
Disk:	(8) 1.817 TB SATA Drives configured as a RAID-6 array, providing 10.908TB of usable space
OS:	Ubuntu 12.04.4 LTS

3 Galaxy architectural overview

Galaxy provides default web server and local database storage for standalone instances. Depending on the use case, it is advised to reconfigure the instance to make use of more robust data storage and/or web server solutions. For our particular instance, we are currently utilizing a postgresql database as our back-end storage. Given the relatively small number of users within our lab utilization of nginx or apache as a web server, however, should the internal webserver prove to pose a bottleneck at a later date, the existing configuration may be migrated to nginx to take advantage of caching functionality to serve the static portions of the Galaxy web site. By default, sqlite internal database files are utilized for the storage of data and computational workflows. This proved promblematic as our anticipated data sets were determined to quickly tax a small flat file database system such as sqlite. Therefore, we opted to create a postgresql database wherein to store all of our data. Details on how to accomplish this are detailed below in the database subsection.

User authentication and role based authentication are handled by the Galaxy server itself. Larger installations with access to a centralized user identification management system such as LDAP or kerberos may be able to take advantage of identity caching through configuration of login caching and forwarding via apache or nginx. Attempts to tie our local instance to existing LDAP authentication mechanisms proved to break the existing administrative functionality of the server. These issues were raised with the development team of the Galaxy Project and it is currently being assessed as a possible feature request by Galaxy development team.

Below, we outline the necessary steps undertaken to install and configure our Galaxy standalone instance using the native python based web server and a postgresql back-end database for the storage of our data.

3.1 Galaxy and postgresql installation procedures

To install Galaxy we can retrieve the latest stable version from the mercurial repository. In order to do this, we must have the mercurial package installed. *(**Note:** Since we are using Ubuntu server as our linux distribution, we will demonstrate how to do this using the Ubuntu package management tools. Please refer to your operating system's package management documentation if you are using a different distribution of linux.)*

```
sudo apt-get install mercurial
```

Once mercurial is installed we may then install the latest stable Galaxy from the mercurial repository.

```
hg clone https://bitbucket.org/galaxy/galaxy-dist/
cd galaxy-dist
hg update stable
```

This is enough to configure a simple standalone Galaxy instance utilizing the internal sqlite database and python webserver. You can test the installation by attempting to start Galaxy by running the following command from within the `galaxy-dist` directory.

```
sh run.sh
```

The initial run of Galaxy will pull necessary python .egg files and start a local instance of the Galaxy server. We can now shut down the test instance as we will next install postgresql and configure Galaxy to rely on the external database as our back-end data store for all of our research data. Shutdown of Galaxy may be performed by issuing a `Ctrl-c` in the terminal where `sh run.sh` was instantiated.

To configure postgresql we will need to install the postgresql package.

```
sudo apt-get install postgresql
```

Further details regarding the configuration of postgresql may be obtained in your operating system's system administration documentation, or by accessing postgresql documentation available online. Once the database software is installed, we must create a database owned by the user id that will run the Galaxy server. In our case the user id is `galaxy`. *(**Note:** The name of the database user created in postgresql should match whatever username is chosen.)* Since the Galaxy platform controls its own schema within the database, we need to ensure that our galaxy user has the appropriate permissions and ownership of the the database for table creation. This is controlled through the postgresql configuration and instructions on how to assign roles and permissions to a postgresql user may be referenced in the postgresql documentation.

3.2 Galaxy standalone instance configuration

Once a database has been created and we have created both the relevant galaxy user in the operating system and in the database authentication systems, we can begin to configure Galaxy for use with an external database. Galaxy stores its relevant configuration information in a file located at the top-level directory `galaxy-dist` within a file named `universe_wsgi.ini`.

First we should ensure that the Galaxy server will be accessible from external interfaces if we are planning on setting up a web accessible standalone instance. This can be done by making the following change to the host entry within `universe_wsgi.ini`:

```
# The address on which to listen.  By default, only listen to
# localhost (Galaxy will not be accessible over the network).
# Use '0.0.0.0' to listen on all available network interfaces.
# host = 127.0.0.1}
host = 0.0.0.0
```

The next line we will need to edit is the `database_connection` line. This will point Galaxy to our newly created postgresql database.

```
# Connect to a local postgres database where database galaxy has been configured for
# access by the galaxy user. - djs 2014-05-06
# The SQLAlchemy PostgreSQL dialect has been renamed from 'postgres' to 'postgresql'.
# The new URL format is postgresql[+driver]://<user>:<pass>@<host>/<dbname>
database_connection = postgresql://galaxy:PASSWORD_GOES_HERE@localhost/galaxy
```

We will also enable the optional server side cursors which will improve the performance on large queries. This option is only available if we are using postgresql and was a consideration when determining which database to use as our storage model.

```
# If large database query results are causing memory or response time issues in
# the Galaxy process, leave the result on the server instead.  This option is
# only available for PostgreSQL and is highly recommended.
#database_engine_option_server_side_cursors = False
database_engine_option_server_side_cursors = True
```

The next three entries into our configuration will direct Galaxy where to manage tools for the tool shed repositories and where dependencies are to be stored on the server.

```
# Tool config files, defines what tools are available in Galaxy.
# Tools can be locally developed or installed from Galaxy tool sheds.
#tool_config_file = tool_conf.xml,shed_tool_conf.xml
tool_config_file = tool_conf.xml,shed_tool_conf.xml
```

```
# Default path to the directory containing the tools defined in tool_conf.xml.
# Other tool config files must include the tool_path as an attribute in the <toolbox>
    tag.
#tool_path = tools
tool_path = tools
```

```
# Path to the directory in which managed tool dependencies are placed.  To use
# the dependency system, see the documentation at:
# http://wiki.g2.bx.psu.edu/Admin/Config/Tool%20Dependencies
tool_dependency_dir = tool_dependencies
```

Now we will need to configure Galaxy to take advantage of Data Libraries, a method for sharing data within the lab. Directories in the configuration file listed here are relative paths, meaning, they are assumed to exist underneath our local galaxy-dist directory. It should be noted that if you define a directory within the configuration file, you must still create the directory yourself and ensure it has appropriate permissions for access by Galaxy. in our case the owner of the Galaxy processes is the `galaxy` user, so we created the relevant directories and ensured that the ownership was set appropriately.

Below is the entry in the configuration file telling Galaxy where files will be placed in order for the Galaxy administrator to upload them into Galaxy.

```
# -- Data Libraries

# These library upload options are described in much more detail in the wiki:
# http://wiki.g2.bx.psu.edu/Admin/Data%20Libraries/Uploading%20Library%20Files

# Add an option to the library upload form which allows administrators to
# upload a directory of files.
library_import_dir = MoazedLabData/Processed
```

And here is the relevant system directory, showing the appropriate permissions for that directory.

```
root@chromosome:/usr/local/galaxy-dist# ls -ld MoazedLabData/
drwxr-xr-x 4 galaxy galaxy 89 Jun 12 14:22 MoazedLabData/
```

Next we will enable the functionality that allows us to paste system specific paths via the Data Library management tool. Since our galaxy user is restricted to the Galaxy installation itself, this does not pose a security risk. However, it is strongly advised that you carefully consider the implications of enabling this feature if you choose to run galaxy with a normal account. It is advised that you create a galaxy account for running galaxy and restrict the galaxy user to only allow access to relevant system files. Please refer to your operating system's system administration documentation if you are unsure how to do this, as it is outside the scope of this article.

```
# Add an option to the admin library upload tool allowing admins to paste
# filesystem paths to files and directories in a box, and these paths will be
# added to a library.  Set to True to enable.  Please note the security
# implication that this will give Galaxy Admins access to anything your Galaxy
# user has access to.
allow_library_path_paste = True
```

Before we open up our Galaxy instance to users, it is very important that we change the default `id_secret` in the configuration file. Failing to do so poses a security risk. This value should be some random string, in this example I have generated a randomized string of data for use as the secret using the apg utility. If you do not have the apg utility installed on your system, in Ubuntu, it can be added by issuing the following command:

```
sudo apt-get install apg
```

Now we will want to use apg to generate our `id_secret`. This can be done by issuing the following command:

```
apg -a 1 -m 32 -x 32
```

`-a` tells `apg` to use random characters instead of attempting to provide a pronounceable password. `-m` and `-x` are flags indicating the minimum and maximum lengths of the generated passwords. Here we have chosen 32 for both values so that all passwords generated will be 32 characters in length.

Below is a sample output from the above referenced command.

```
]d_@{MF+cgItXmc+s[{J64eAKQKB{/6U
~D!N$M(\oF%*q1I!WSrfEydZ6MaK.-.`
g~V?"A\=,T.,9hi:]4e]<$8~y`B\+uJJ
lA=ivi3>GF09OB9qNbeGaDhp#eaO(q%j
t=Y!l_Ip^NsT!46U+>tJ##^ug($H{en<
9K%hA=P]F^x[5M@n2tp$S(ir-akB-%yQ
```

Each line represents a randomly generated 32 character password. Select one and place that string in the `id_secret` entry within the `universe_wsgi.ini` configuration file as shown below. This key is used by Galaxy to generate internal values involved with cookies and URLs. It must be changed or your Galaxy installation will not be secure from malicious users who could generate fake cookies using the default entry to gain access to a user's Galaxy account and thereby their data!

```
# -- Users and Security

# Galaxy encodes various internal values when these values will be output in
# some format (for example, in a URL or cookie).  You should set a key to be
# used by the algorithm that encodes and decodes these values.  It can be any
# string.  If left unchanged, anyone could construct a cookie that would grant
# them access to others' sessions.
#id_secret = USING THE DEFAULT IS NOT SECURE!
id_secret = g~V?"A\=,T.,9hi:]4e]<$8~y`B\+uJJ
```

The next setting we may need to change is to normalize emails. Galaxy user ids are constructed from the user's email address given at registration time. Normalization will force the email address to all lowercase and avoid any potential case issues when a user enters their email address to login to the system.

```
# If your proxy and/or authentication source does not normalize e-mail
# addresses or user names being passed to Galaxy - set the following option
# to True to force these to lower case.
normalize_remote_user_email = True
```

It is important if you are planning on administrating your galaxy instance to add your own email address to this entry and to create a user with a matching email address. This will enable you to access the administrative functions later when bringing up the Galay instance and is important to be able to create the shared Data Libraries.

```
# Administrative users - set this to a comma-separated list of valid Galaxy
# users (email addresses).  These users will have access to the Admin section
# of the server, and will have access to create users, groups, roles,
# libraries, and more.  For more information, see:
# http://wiki.g2.bx.psu.edu/Admin/Interface
#admin_users = None
admin_users = your.email@university.edu
```

Here we disable anonymous login, as only registered users will be given access to the laboratory instance of Galaxy.

```
# Force everyone to log in (disable anonymous access).
require_login = True
```

Currently, we have left this as the default, allowing anyone to register an account with our instance. Since our instance is behind a firewall an only accessible by individuals with access to the Private HMS network, we have decided to let our users create their own accounts and specify the email they would like to use. Depending upon your network configuration and whether or not your Galaxy instance is accessible over the internet, you may wish to change this value to `False`.

```
# Allow unregistered users to create new accounts (otherwise, they will have to
# be created by an admin).
#allow_user_creation = True
```

We have enabled administrative functionality to delete accounts. By default it is not enabled. Due to our rather lenient account creation policy we felt it necessary should individuals accidentally create a second account for themselves.

```
# Allow administrators to delete accounts.
#allow_user_deletion = False
allow_user_deletion = True
```

Once you have finished configuration of your Galaxy instance you can again bring it up and test that the configuration settings are working as intended. You can verify that the Galaxy server is able to access your postgresql database and a new schema should appear within the database now that Galaxy is managing its internal storage via the postgresql database. There is an important log associated with startup that should be mentioned at this point, as it will assist in troubleshooting any potential configuration issues and will allow you to determine when Galaxy has successfully started.

3.3 Galaxy log file rotation using logrotate

Located under the galaxy installation (in our case `/usr/local/galaxy-dist`) we will find a log file named `paster.log`. This file is where the Galaxy instance will log messages while it is running on the system.

On linux based systems, it might be beneficial to utilize `logrotate` to handle ensuring that `paster.log` is routinely rotated and compressed, to avoid continuously logging to a single file that will over time grow and consume system resources such as disk space. This can be done by implementing a logrotate configuration for galaxy by placing a file named `galaxy` in `/etc/logrotate.d` with the following configuration.

```
/usr/local/galaxy-dist/paster.log {
  weekly
  rotate 8
  copytruncate
  compress
  missingok
  notifempty
}
```

This tells the logrotate dæmon to rotate paster.log weekly, keeping the last 8 logs, copying the existing file and truncating the current log, and then compressing the copied logfile to reduce disk utilization storing the files. It is ok if the file is missing, and do not perform the above actions if the current log file is empty. For more information on how to configure the logrotate dæmon itself, please refer to your operating system administration documentation.

3.4 Automating start up and shut down of the Galaxy service

Since Galaxy will be treated as a system service on our server, we have opted to create a simple start up and shut down script on our Ubuntu server utilizing the Ubuntu upstart system. this ensures that when our system is brought down for maintenance, or re-started, that our Galaxy instance will be brought down cleanly during the shut down process and brought back up during the system's boot sequence. Below are the contents of our script for reference.

```
#!/bin/bash

### BEGIN INIT INFO
# Provides:          galaxy
# Required-Start:    $local_fs $remote_fs $syslog $named $network $time
# Required-Stop:     $local_fs $remote_fs $syslog $named $network
# Should-Start:
# Should-Stop:
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start/Stop the Galaxy service
### END INIT INFO

#--- config
SERVICE_NAME=galaxy
RUN_AS=galaxy
RUN_IN=/usr/local/galaxy-dist
#--- main actions

start() {
        echo "Starting $SERVICE_NAME... "
        cmd="cd $RUN_IN && sh run.sh --daemon"
        case "$(id -un)" in
                $RUN_AS)
                        eval "$cmd"
                        ;;
                root)
                        su - $RUN_AS -c "$cmd"
                        ;;
                *)
                        echo "*** ERROR *** must be $RUN_AS or root in order to control
     this service" >&2
                        exit 1
        esac
        echo "...done."
}

stop() {
        echo -n "Stopping $SERVICE_NAME... "

        cmd="cd $RUN_IN && sh run.sh --stop-daemon"

        case "$(id -un)" in
                $RUN_AS)
                        eval "$cmd"
                        ;;
                root)
                        su - $RUN_AS -c "$cmd"
                        ;;
                *)
                        echo "*** ERROR *** must be $RUN_AS or root in order to control
     this service" >&2
                        exit 1
        esac

        echo "done."
}

status() {
        echo -n "$SERVICE_NAME status: "

        while read pid; do
                if [ "$(readlink -m /proc/$pid/cwd)" = "$(readlink -m $RUN_IN)" ]; then
                        echo "started"
                        return 0
                fi
        done < <(ps ax -o 'pid cmd' | grep -P '^\s*\d+ python ./scripts/paster.py serve
    ' | awk '{print $1}')
        echo "stopped"
        return 3
}

notsupported() {
        echo "*** ERROR*** $SERVICE_NAME: operation [$1] not supported"
}

usage() {
        echo "Usage: $SERVICE_NAME start|stop|restart|status"
}


#---

case "$1" in
        start)
                start "$@"
                ;;
        stop)
                stop
                ;;
        restart|reload)
                stop
                start
                ;;
        status)
                set +e
                status
                exit $?
                ;;
        '')
                usage >&2
                exit 1
                ;;
        *)
                notsupported "$1" >&2
                usage >&2
                exit 1
                ;;
esac
```

4 Setting up a Data Library in Galaxy

Galaxy allows for the configuration of data libraries in order to share data files. The benefit of a data library is that access control may be permissioned on a user or group basis, if necessary. Furthermore, data stored in a data library is only stored once within the database. When users import data from a data library, a pointer to the original data is passed. This reduces disk utilization on the server and circumvents duplication of large data sets shared among multiple users.

We have implemented a Data Library within Galaxy as a centralized repository for all of our laboratory's publicly published data. This enables new researcher's to quickly make use of relevant available data, along with maintaining NCBI GEO and SRA accession identifiers should further relevant data not originating from the lab be required. This data is then readily accessible, and already in the proper data formats applicable, to existing bioinformatics workflows published within our Galaxy instance.

4.1 Example Data Library creation

5 Galaxy workflow creation

Workflows are created either by extracting a workflow from an existing analysis, or by building a workflow within the workflow editor. Workflows allow for the creation of a generalized pipeline for the processing of biological data using the many tools installed in the Galaxy tool shed. This approach utilizes a graphical flowchart detailing the inputs and outputs derived from each tool. Individual tool's outputs may be connected to the inputs of subsequent tools, forming a data pipeline. These workflows may then be saved and shared with other users of the Galaxy platform.

Workflows are modeled after data flow diagrams found in software engineering, whereby an input data set is transformed through a series of applied functions, resulting in output derived from the series of transformations applied to the data in the model. This allows for an abstract representation of the analysis to be performed on a type of data input which which result in data output of a specific format. Further complex analysis can then be derived by chaining relatively simple workflows to construct larger, more complex analytical pipelines. Presented below is an example, illustrating the workflow creation process, utilizing the workflow editor.

5.1 Example workflow

6 Conclusions