Production Galaxy Instances with CloudMan and CloudBioLinux Documentation

Release

John Chilton

December 29, 2012

CONTENTS

CHAPTER

ONE

INTRODUCTION

Typically, CloudMan is used to spin up a dedicated Galaxy virtual cluster for a single user or small group on Amazon EC2. This document outlines methods for deploying customized Galaxy instances that are more suitable for production environments with large user bases - these enhancements include enabling load balancing, external authentication, SSL, reporting, and taking advantage of external resources such as dedicated file and database servers and compute resources outside of the cloud.

Some of these enhancements can be used on EC2 with updated stock CloudMan images, however some require re-building a CloudMan image using CloudBioLinux (as is required when utilizing private cloud resources). *Appendix I: Getting Started with CloudBioLinux* therefore provides some guidance on building custom CloudMan images with CloudBioLinux. Readers without CloudBioLinux familiarity should review this appendix should be reviewed before continuing.

CHAPTER

TWO

SPLITTING GALAXY INTO MULTIPLE PROCESSES

CloudMan's default configuration launches just one Galaxy process. To really scale up the number of concurrent users Galaxy can handle it must be split into multiple processes as outlined and documented here:

```
configure_multiple_galaxy_processes: True
web_thread_count: 2
handler_thread_count: 2
galaxy_conf_dir: /mnt/galaxyTools/galaxy-central
```

The first option (`configure_multiple_galaxy_processes`) informs CloudMan to split Galaxy into multiple processes and `web_thread_count` and `handle_thread_count` specify how many web and handler threads to create (respectively).

The last option - `galaxy_conf_dir` - instructs CloudMan to setup a configuration for directory for Galaxy and is required for many of the options described in this document. See *Appendix III: Galaxy Configuration Directories* for more information.

When these options are enabled, CloudMan will rewrite the body of the `upstream galaxy_app {...}` in `nginx.conf` to load balance web traffic accross the number of web threads you specify.

It is a known complication with Galaxy that when using multiple Galaxy processes job admin functionality needs to be routed to Galaxy's job manager thread, this can be done by updating your `nginx.conf` file (see *Appendix II: Configuring nginx.conf*) to add the following `location /admin/jobs` subsection (as shown below):

```
location / {
    ...

    location /admin/jobs {
        proxy_pass  http://localhost:8079;
    }
}
```

CHAPTER

THREE

EXTERNAL AUTHENTICATION (LDAP)

Galaxy requires a proxy web server (in this case nginx) to enable external authentication. nginx can be configured to use LDAP authentication by modifing `nginx.conf` (see *Appendix II: Configuring nginx.conf*) as follows.

Modify `http` section of `nginx.conf` with LDAP connection information.

```
http {

    auth_ldap_url ldap://ldap.example.com/dc=example,dc=com?uid?sub?(objectClass=person);
    #auth_ldap_binddn cn=nginx,ou=service,dc=example,dc=com;
    #auth_ldap_binddn_passwd mYsUperPas55W0Rd
    #auth_ldap_group_attribute uniquemember; # default 'member'
    #auth_ldap_group_attribute_is_dn on; # default on

    ...

}
```

Modify the root location (`/`) of `nginx.conf` to require authentication and pass `REMOTE_USER` along to Galaxy and create a (`/api`) to make this authentication optional.

```
location / {
    auth_ldap_require valid_user;
    auth_ldap "LDAP Auth Source Description";
    proxy_set_header REMOTE_USER $remote_user;


    proxy_pass  http://galaxy_app;
    proxy_set_header   X-Forwarded-Host $host;
    proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
    proxy_set_header   X-URL-SCHEME https;

  ...
}

# For API access, set REMOTE_USER if available so Galaxy
# session based requests are let through, if REMOTE_USER is not
# available pass the request through and let Galaxy determine
# if a key is present and valid.
location  /api {
    proxy_set_header REMOTE_USER $remote_user;
    proxy_pass  http://galaxy_app;
    proxy_set_header   X-Forwarded-Host $host;
    proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
}
```

Additionally, nginx needs to be compiled with LDAP support. This should be done when building the cloud image with CloudBioLinux. To enable this compilation simply add the following option to your fabricrc file:

```
nginx_enable_module_ldap = true
```

Finally, Galaxy also must be configured to use the information that will be passed through by nginx.

```
galaxy_universe_use_remote_user: True
galaxy_universe_remote_user_maildomain: <domain_name (e.g. example.org)>
galaxy_universe_remote_user_logout_href: https://logout@<galaxy_url>/
galaxy_universe_require_login: True
```

Note, setting these properties requires also setting a `galaxy_conf_dir`.

More information about the nginx LDAP module can be found here. Interacting with other forms of external authentication will likely require compiling nginx with additional modules. Checkout out the functions `_get_nginx_modules` and `_get_nginx_module_ldap` in cloudbiolinux/master/cloudbio/galaxy/__init__.py for an outline of how to implement this.

If you are using external authentication in this fashion it is also likely a good idea to enable SSL.

CHAPTER

FOUR

ENABLE SSL

Your cloud account's security group will likely block port `443` by default. This must be opened.

If you are using Amazon EC2, when following the instructions on the CloudMan wiki site, be sure to add the `HTTPS` inbound rule in addition to the `HTTP` one mentioned.

Instructions for opening this port on private clouds will vary, the following command for instance will open it for OpenStack.:

```
nova secgroup-add-rule <security_group> tcp 443 443 0.0.0.0/0
```

CloudMan will need to setup the desired SSL *key* and *cert* before nginx starts up. The CloudMan can be configured to do this by passing them along as `conf_files` in the user data:

```
conf_files:
   - path: /usr/nginx/conf/key
     content: <base64 encoding of key>
   - path: /usr/nginx/conf/cert
     content: <base64 encoding of cert>
```

Finally, the `nginx.conf` (see *Appendix II: Configuring nginx.conf*) file for the instance will need to be updated. The server section will need to be adjusted to use port 443 and SSL with the supplied certificate and a new section should be created to redirect HTTP traffic to HTTPS (as shown below).

```
server {
   listen         80;
   server_name    mygalaxy.example.com;
   rewrite        ^ https://$server_name$request_uri? permanent;
}

server {
    listen 443 default_server ssl;
    ssl_certificate      /usr/nginx/conf/cert;
    ssl_certificate_key  /usr/nginx/conf/key;

    ....
}
```

CHAPTER

FIVE

REPORTS SERVER

The Galaxy reports webapp is a small webapp that runs in parallel to Galaxy and provides a wealth of valuable data on every job that Galaxy has run as well as disk usage accounting, etc.... It is an invaluable tool when hunting down problems reported by Galaxy users.

CloudMan can now enable the reports application by simply adding it to the list of services.:

```
services:
  - name: Galaxy
  - name: GalaxyReports
  - name: Postgres
```

By default no services need to be specified in the user data and `Galaxy` and `Postgres` are enabled. However, to add or remove any all desired services should be listed.

CHAPTER

SIX

EXTERNAL POSTGRES SERVER

When deploying to Amazon EC2, running a Postgres server right on the CloudMan master node makes a lot of sense. For private cloud deployments many institutions may already have well optimized, well maintained production Postgres servers however and utilizing these may be preferable. This section describes how to utilize such a server.

To disable CloudMan's default Postgres server, simply manually specify the list of services CloudMan should start and exclude Postgres. For instance:

```
services:
  - name: Galaxy
  - name: GalaxyReports
```

Galaxy must then simply be configured to use your external postgres server, this can be done by passing it in via the user-data variable `galaxy_universe_database_connection`.

```
galaxy_universe_database_connection: postgres://user:password@host:port/schema
```

Setting the database connection in this fashion also requires specifing a `galaxy_conf_dir` (see *Appendix III: Galaxy Configuration Directories*).

CHAPTER

SEVEN

EXTERNAL FILE SERVER

Two CloudMan user data options - `master_prestart_commands` and `workder_prestart_commands` - can be specified to run arbitrary shell commands before CloudMan starts up Galaxy on the master node or runs jobs on newly booted worker nodes.

The following example demonstrates how this used at MSI. The following commands mount Galaxy's data partition from an NFS export on the host `spider.msi.umn.edu` and a read-only partition from an NFS export on `buzzard.msi.umn.edu` (we use the second to store bio data such NGS indices, etc...).

```
master_prestart_commands:
  - "mkdir -p /mnt/galaxyData"
  - "mount -t nfs4 -o sec=sys spider.msi.umn.edu:/export/galaxyp /mnt/galaxyData/"
  - "mkdir -p /project/db"
  - "mount -t nfs4 -o ro buzzard.msi.umn.edu:/zprod2/misc/db /project/db/"
worker_prestart_commands:
  - "mkdir -p /mnt/galaxyData"
  - "mount -t nfs4 -o sec=sys spider.msi.umn.edu:/export/galaxyp /mnt/galaxyData/"
  - "mkdir -p /project/db"
  - "mount -t nfs4 -o ro buzzard.msi.umn.edu:/zprod2/misc/db /project/db/"
```

CHAPTER

EIGHT

RUNNING JOBS ON EXTERNAL COMPUTE RESOURCES

The method outlined here utilizes the LWR job runner. The LWR job runner is a Galaxy job runner and corresponding server-side application that can run jobs a remote server *without* requiring Galaxy filesystems be mounted on the remote host (as is required by DRMAA or PBS submissions).

The LWR works by transferring all input files to the remote host, rewritting paths in the Galaxy command-line as well as ‘‘configfile‘‘s, running the job remotely, and then transferring the outputs back to the Galaxy host upon completion.

This is being used at the Minnesota Supercomputing Institute to ship select jobs originating from an ephermeral Galaxy host in our OpenStack cloud to a permant Windows host outside the cloud. This is a useful tool for purchased node-locked and/or Windows only software.

In order to support this use case, CloudMan has been augmented to allow specifing tool runner URLs via user data. The following userdata option is used to tell CloudMan to configure Galaxy to run `proteinpilot` jobs on the remote Windows host `cobalt.msi.umn.edu` using the LWR job runner.:

```
galaxy_tool_runner_proteinpilot: "lwr://https://secretkey123@cobalt.msi.umn.edu:8913"
```

The secret key seen here is used to authorize Galaxy to submit jobs to the remote LWR host, and https is used to secure transport. Please consult the LWR documentation and source for details. Specifing job runners this way requires setting the `galaxy_conf_dir` option as well (eee *Appendix III: Galaxy Configuration Directories*).

Backend implementations for LWR targetting DRMAA and PBS are being developed. Progress can be tracked by following the LWR on Bitbucket.

8.1 An Aside

It MAY well be possible to configure Galaxy's standard job runner to submit Galaxy jobs directly from say a cloud host to a traditional, *if* all of the filesystems are mounted similarly between the Cloud and compute hosts and the remote compute server has a user that can run jobs with pid 1001 (the CloudBioLinux generated pid for Galaxy).

If this does work, one could imagine running jobs of type `tool_x` via the PBS host on `compute.example.com` by passing along the following user data to CloudMan at deploy time:

```
galaxy_universe_start_job_runners = drmaa, pbs  # Make sure drmaa is still enabled for Cloud-targette
galaxy_tool_runner_tool_x = pbs://compute.example.com/
```

At this point this is all untested speculation, but hopefully additional testing will be done and this documentation updated. (If you have tried this and have advice let me know)

CHAPTER

NINE

APPENDIX I: GETTING STARTED WITH CLOUDBIOLINUX

There are a few key configuration files used to configure CloudBioLinux + CloudMan. These include:

`fabricrc.txt` Fabric properties used to configure CloudBioLinux. For examples see the CloudBioLinux default or the version used for my CloudMan OpenStack bootstrap scripts

`userData.yaml` A YAML configuration file used by CloudMan to configure your instance(s) at startup.

`nginx.conf` Base configuration file for nginx server. See *Appendix II: Configuring nginx.conf* for description of how to replace the default contents of this file either at image build time or at instance boot time.

9.1 Checking Out CloudBioLinux

`git` should be used obtain `CloudBioLinux`.

```
git clone git://github.com/chapmanb/cloudbiolinux.git
```

If additional CloudBioLinux development is planned (installation of custom packages, services, etc...), it may be preferable to fork CloudBioLinux first and then checkout CloudBioLinux from the forked repository.

9.2 Installing CloudBioLinux

Prerequistes for installing CloudBioLinux include the following:

- A customized fabricrc file as described above (referred to as `/path/to/fabricrc.txt`).
- A running Ubuntu instance on your cloud platform. This document assumes key-pairs and security groups so that the instance (`test1.example.org`) is accessible via the private key `/path/to/private_key`.
- Required filesytem mounts (likely at least */mnt/galaxyData* and */mnt/galaxyTools*).

The specifics of these steps are beyond the scope of this document and will vary from setup to setup.

9.2.1 CloudMan Flavor

Once you have booted up an Ubuntu instance in your cloud environment, `CloudBioLinux` can be installed using fabric via the following command:

```
fab -u ubuntu -c /path/to/fabricrc.txt -i /path/to/private_key -H test1.example.org install_biolinux
```

This version of the install command is the simplest, and it will install some base packages and configure CloudMan. This is good for initial testing.

Ultimately, though you will likely want to install needed bioinformatics applications also - the following two subsections describe approaches to accomplish this.

9.2.2 Flavorless

```
fab -u ubuntu -c /path/to/fabricrc.txt -i /path/to/private_key -H test1.example.org install_biolinux
```

This second version of the `install_biolinux` command (no flavor specified) will install all of CloudBioLinux, which will include many packages you may not need for Galaxy, such as R libraries and desktop applications.

These packages will be installed into the instance's `/usr` directory, either using fabric commands defined in `cloudbiolinux/cloudbio/custom` or native pacakges.

If installing all of CloudBioLinux is not desired, one can comment out lines in `cloudbiolinux/config/main.yaml` and `cloudbiolinux/config/custom.yaml` and other files in that directory to prune the list of applications and libraries that get installed.

9.2.3 CloudMan + Tools Flavor

As mentioned above, the flavorless install will install all packages into `/usr`. This allows applications other than Galaxy to utilize these programs, but this has down sides.

With every program on Galaxy's path by default, it may be difficult to isolate certain types of problems. More importantly however, Galaxy cannot then easily target multiple versions of the same program with different tool wrappers.

`CloudBioLinux` can install specific Galaxy tools into a Galaxy tool dependency directories along with required `default` symbolic links and `env.sh` files. To install CloudBioLinux this way, the following command can be used:

```
fab -u ubuntu -c /path/to/fabricrc.txt -i /path/to/private_key -H test1.example.org install_biolinux
```

This approach also requires updating the target fabricrc file to instruct CloudBioLinux to install Galaxy dependencies.

```
galaxy_install_dependencies = true
```

The list of tools and versions that is installed can be found in `cloudbiolinux/contrib/flavor/cloudman/tools.yaml` `<https://github.com/chapman b/cloudbiolinux/blob/master/contrib/flavor/cloudman/tools.yaml>`. One can modify that file directly or specify an entirely new file by setting the `galaxy_tools_conf` property in fabricrc.

Warning: Installing Galaxy tools in this manor is not as well supported as the full CloudBioLinux approach. This approach is my (John Chilton) pet project, and my day job is proteomics not genomics so many of the tools are suitable for Galaxy circa mid-2012, not an updated Galaxy with Cufflinks 2, Tophat 2, etc.... I am happy to accept pull requests to integrate newer versions of these tools however.

9.3 Save the Image

The CloudBioLinux install will setup a script that get executed at instance startup to launch CloudMan. So the next step is to save this image and boot it up with the required CloudMan userdata. Details of how to do this will vary between Cloud platforms.

CHAPTER

TEN

APPENDIX II: CONFIGURING NGINX.CONF

`/usr/nginx/conf/nginx.conf` is the main configuration used by the nginx web server that proxies traffic to Galaxy. A custom ‘‘nginx.conf‘‘_ file can be configured either while building the base image via CloudBioLinux or while booting up the instance via CloudMan.

10.1 ... via CloudMan

CloudMan will replace `nginx.conf` with the base 64 encoded contents specified by the user data option `nginx_conf_contents`.

```
nginx_conf_contents: <base64 encoding of contents or URL>
```

One can use the *nix utility *base64* to generate the base 64 encoding for a file.

10.2 ... via CloudBioLinux

‘CloudBioLinux contains default template for nginx.conf, but this can easily ‘be substituted out for another file by setting the nginx_conf_path property ‘to point to this new file in the target fabricrc file.

```
nginx_conf_path = /local/path/to/nginx.conf
```

Alternatively the file can just be directly modified cloudbiolinux/installed_files/nginx.conf.template in the local clone of CloudBioLinux. Be careful if directly editing this file however, this is treated as a template so all *$* symbols must be escaped with a \ to avoid variable interpolation.

CHAPTER

ELEVEN

APPENDIX III: GALAXY CONFIGURATION DIRECTORIES

Creating a cloud instance the user data option `galaxy_conf_dir` will instruct CloudMan to launch Galaxy with the environment variable `GALAXY_UNIVERSE_CONFIG_DIR` set to the value specified by `galaxy_conf_dir`.

Since pull request 44, when Galaxy starts up with this set, it merges the files in the specified directory into a single `universe_wsgi.ini` file. This allows configuration by directory (like the `/etc/sudoers.d` or `/etc/apache/conf.d` directories in many modern Linux distributions).

Setting `galaxy_conf_dir` eases CloudMan's ability to configure Galaxy and is required for many other options described in this document - such as `configure_multiple_galaxy_processes`, `galaxy_config_*` and `galaxy_tool_runner_*`.