Pulsar Documentation

Release 0.15.3

The Galaxy Project

Aug 23, 2023

CONTENTS

Contents:

This project is a Python server application that allows a Galaxy server to run jobs on remote systems (including Windows) without requiring a shared mounted file systems. Unlike traditional Galaxy job runners - input files, scripts, and config files may be transferred to the remote system, the job is executed, and the results are transferred back to the Galaxy server - eliminating the need for a shared file system.

Full documentation for the project can be found on Read The Docs.

CHAPTER

ONE

CONFIGURING GALAXY

Galaxy job runners are configured in Galaxy's `job_conf.xml` file. Some small examples of how to configure this can be found here, but be sure to check out `job_conf.xml.sample_advanced` in your Galaxy code base or on Github for complete information.

CHAPTER

TWO

QUICKSTART

Full details on different ways to install Pulsar can be found in the install section of the documentaiton, but if your machine has the proper Python dependencies available it can be quickly download and a test job run with:

```
$ mkdir pulsar
$ cd pulsar
$ python3 -m venv venv
$ . venv/bin/activate    # venv\Scripts\activate.bat on Windows
$ pip install 'pulsar-app[web]'
$ pulsar-config
$ pulsar --daemon        # just `pulsar` on Windows
$ pulsar-check           # runs a test job
```

Please note that as of the 0.14.0 release, Pulsar no longer supports any version of Python 2. The minimum supported Python version is 3.5.

The configuration documentation has many details on securing your Pulsar server and enabling advanced features such as cluster integration and message queue communication.

CHAPTER

THREE

DEVELOPMENT AND TESTING

The recommended approach to setting up a development environment for Pulsar on Linux or macOS is rougly as follows:

```
$ git clone https://github.com/galaxyproject/pulsar
$ cd pulsar
$ python3 -m venv .venv
$ . .venv/bin/activate    # .venv\Scripts\activate on Windows
$ pip install -e '.[web]'
$ pip install -r dev-requirements.txt
```

The `-e` flag to `pip` installs Pulsar in "editable" mode, meaning that changes you make to the source code will be reflected when running the pulsar commands installed in the virtual environment.

This project is distributed with unit and integration tests (many of which will not run under Windows), the following command will install the needed python components to run these tests. The following command will then run these tests:

```
$ make tests
```

The following command will then produce a coverage report corresponding to this test and place it in the `coverage_html_report` subdirectory of this project.:

```
$ coverage html
```

Check out the Contributing documentation for many more details on developing and contributing to Pulsar.

Please note that this project is released with a Contributor Code of Conduct. By participating in this project you agree to abide by its terms.

CHAPTER

FOUR

SUPPORT

This documentation is an incomplete work in progress. There are more ways to configure and use Pulsar than are documented, and a growing number of Pulsar experts who would be more than happy to answer your questions and help with any problems you may run in to while setting up a Pulsar deployment. Please do not hesitate to reach out on the Galaxy Admins Gitter Channel

CHAPTER

FIVE

INSTALLING PULSAR

Tip: This documentation covers installing Pulsar by hand. The tutorial Running Jobs on Remote Resources with Pulsar in the Galaxy Training Network contains a step-by-step guide for installing Pulsar using Ansible.

There are two primary ways to deploy Pulsar. The newer and preferred method is to install Pulsar from PyPI using the standard pip and venv Python tools.

The older method also requires these tools to install Pulsar's dependencies but Pulsar itself is served directly from a clone of the Pulsar source tree - this mirrors how Galaxy is most typically deployed. This may be beneficial during Pulsar development and is required for certain experimental features such as Mesos support.

Both methods presented here require a Python 3.5 (or later) runtime for either Windows, Linux, or macOS. **Python 2 is no longer supported as of the 0.14.0 release of Pulsar**.

These instructions also require venv. Open a console on your machine and type `python3 -m venv` - if the module is missing you will need to install it. It is part of any full Python installation, but some Linux distributions (such as Debian and its derivatives) package it separately. On Debian systems, you can

5.1 From PyPI

Start by creating a directory for the Pulsar configuration files and setting up a venv to install Pulsar into using the following three commands.:

```
$ mkdir pulsar
$ cd pulsar
$ python3 -m venv venv
```

Next, activate this newly created venv. From a Linux or macOS terminal, this can be done with the command `. venv/bin/activate` and in Windows you can type `venv\Scripts\activate`.

Next install Pulsar using `pip`.:

```
$ pip install 'pulsar-app[web]'
```

If you are planning to use Pulsar in the optional message queue mode rather than traditional web mode (see the *configuration docs*), you can drop `[web]` from the install command (this extra qualifier causes web serving packages to be installed).

Next, create the required configuration files for use with Pulsar in this directory by running the following command.:

```
$ pulsar-config
```

The `pulsar-config` script can bootstrap various Pulsar deployment options, run `pulsar-config --help` for full details. For instance, Pulsar can be configured to monitor a message queue and skip the web server configuration - enable this by passing `--mq` to `pulsar-config`. Another useful option is `--supervisor` which will generate a Supervisord configuration for this directory and install Supervisord.

`pulsar-config` installs a few files into this directory. `app.yml` contains Pulsar configuration options and `server.ini` contains web server related information (it will not exist if configured `--mq`).

5.1.1 Launching Pulsar

The Pulsar server can be started by running:

```
$ pulsar [--daemon]
```

Under Linux and macOS the `--daemon` argument can be supplied to run Pulsar as a daemon, and stopped with `pulsar --stop-daemon`. If `--daemon` is not supplied, Pulsar will just run in the foreground (the only option for Windows).

The Pulsar deployment can be tested by running the following command, which will submit an example job and wait for its completion.:

```
$ pulsar-check
```

If Pulsar is not running on the default port `8913`, `pulsar-check` should be called with an explicit URL using the argument `--url=http://localhost:8913`. Likewise if a private token has been configured it can be supplied using `--private_token=<token>`.

5.2 From Source

Alternatively, Pulsar can be obtained from GitHub using the following command and ran directly from the source tree (like Galaxy is traditionally deployed):

```
$ git clone https://github.com/galaxyproject/pulsar
```

The following steps assume your current working directory is the newly created `pulsar` directory.:

```
$ cd pulsar
```

Create a new Python virtual environment called `.venv` in the `pulsar` root directory:

```
$ python3 -m venv .venv
```

Activate environment (varies by OS). From a Linux or macOS terminal:

```
$ . .venv/bin/activate
```

Or from a Windows terminal:

```
$ .venv\Scripts\activate.bat
```

Finally, install Pulsar's required dependencies into the virtual environment:

```
$ pip install -r requirements.txt
```

If using the standard webserver, it can be installed with:

```
$ pip install Paste PasteScript
```

5.2.1 Launching Pulsar

Before launching Pulsar, it may make sense to copy over the sample configuration files. `server.ini` is used to describe web server related properties and `app.yml` is used for Pulsar application-related configuration files:

```
$ cp server.ini.sample server.ini
$ cp app.yml.sample app.yml
```

Pulsar should now be launchable via the `run.sh` script under Linux or macOS or using the `run.bat` script under Windows. So under Linux or macOS, Pulsar can be launched with:

```
$ ./run.sh [--daemon]
```

This daemon can be stopped using `./run.sh --stop-daemon`. When run as a daemon, Pulsar will log to the file `paster.log`. If `--daemon` is not supplied, Pulsar will just run in the foreground.

Under Windows, Pulsar can be started using:

```
$ run.bat
```

and will run as long as that process is alive and log to standard output. A test job can be submitted using the command:

```
$ python run_client_tests.py
```

If Pulsar's `server.ini` has been modified and it is not running on the default port `8913`, `run_client_tests.py` should be called with an explicit URL using the argument `--url=http://localhost:8913`. Likewise if a private token has been configured it can be supplied using `--private_token=<token>`.

5.3 Pulsar Webservers

Pulsar's default webserver (if web dependencies are installed) is Paste. However, uWSGI or circus will be used instead, if found.

A precompiled version of uWSGI can be installed with:

```
$ pip install pyuwsgi
```

Or compiled from source with:

```
$ pip install uWSGI
```

5.4 Pulsar Dependencies

Several Python packages must be installed to run the Pulsar server. The core set of required dependencies were installed during the Pulsar installation in the previous section. Additional dependencies are required for features such submitting to a cluster (`drmaa`), communicating via message queue (`kombu`), etc.... Most of the time these can just be installed with `pip install <dependency_name>`.

CHAPTER

SIX

CONFIGURING PULSAR

If either installation procedure has been followed, your Pulsar directory should contain two files of interest: `app.yml` to configure the Pulsar application and `server.ini` to configure the web server (unless you are running Pulsar without a web server).

Default values are specified for all configuration options that will work if Pulsar is running on the same host as Galaxy (e.g. for testing and development). Otherwise, the `host` setting of `server.ini` will need to be modified to listen for external requests.

`app.yml` settings can be overridden by setting environment variables, just as with Galaxy, by prefixing the config setting name with `PULSAR_CONFIG_OVERRIDE_`. For example:

```
$ export PULSAR_CONFIG_OVERRIDE_PRIVATE_TOKEN=changed
$ pulsar
```

Defaults can also be set via environment variables by prefixing them with `PULSAR_CONFIG_`. For example, `PULSAR_CONFIG_PRIVATE_TOKEN`.

6.1 Security

Out of the box, **Pulsar essentially allows anyone with network access to the Pulsar server to execute arbitrary code and read and write any files the web server can access.** Hence, in most settings steps should be taken to secure the Pulsar server.

6.1.1 Private Token

If running Pulsar with a web server, you *must* specify a private token (a shared secret between Pulsar and the Galaxy server) to prevent unauthorized access. This is done by simply setting `private_token` in `app.yml` to some long random string.

Once a private token is configured, Galaxy job destinations should include a `private_token` parameter to authenticate these jobs.

6.1.2 Pulsar Web Server

The default Pulsar web server, Paste can be configured to use SSL and to require the client (i.e. Galaxy) to pass along a private token authorizing use.

Tip: SSL support is built in to uWSGI, an alternate webserver that can be installed (see *Installing Pulsar*).

pyOpenSSL is required to configure a Pulsar web server to server content via HTTPS/SSL. This dependency can be difficult to install and seems to be getting more difficult. Under Linux you will want to ensure the needed dependencies to compile pyOpenSSL are available - for instance in a fresh Ubuntu image you will likely need:

```
$ sudo apt-get install libffi-dev python3-dev libssl-dev
```

Then pyOpenSSL can be installed with the following command (be sure to source your virtualenv if setup above):

```
$ pip install pyOpenSSL
```

Once installed, you will need to set the option `ssl_pem` in `server.ini`. This parameter should reference an OpenSSL certificate file for use by the Paste server. This parameter can be set to * to automatically generate such a certificate. An unsigned certificate for testing purposes can be manually generated by the following method:

```
$ openssl genrsa 1024 > host.key
$ chmod 400 host.key
$ openssl req -new -x509 -nodes -sha1 -days 365  \
          -key host.key > host.cert
$ cat host.cert host.key > host.pem
$ chmod 400 host.pem
```

More information can be found in the paste httpserver documentation.

6.1.3 Message Queue

If Pulsar is processing requests via a message queue instead of a web server the underlying security mechanisms of the message queue should be used to secure communication - deploying Pulsar with SSL and a `private_token` described above are not applicable.

This can be done via two (not mutually exclusive) methods: client SSL certificates, or password authentication. In either case, you should configure your AMQP server with SSL.

If using client certificates, you will likely need to set the appropriate (for your PKI) combination of `amqp_connect_ssl_ca_certs`, `amqp_connect_ssl_keyfile`, `amqp_connect_ssl_certfile`, and `amqp_connect_ssl_cert_reqs`, in Pulsar's `app.yml` file. See `app.yml.sample` for more details.

If using password authentication, this information can be set in the `message_queue_url` setting in `app.yml`, e.g., with SSL:

```
message_queue_url: amqps://user:password@mqserver.example.org:5671//
```

You can consult the Kombu documentation for even more information.

6.1.4 User Authentication/Authorization

You can configure Pulsar to authenticate user during request processing and check if this user is allowed to run a job.

Various authentication/authorization plugins can be configured in *app.yml* to do that and plugin parameters depend on auth type. For example, the following configuration uses *oidc* plugin for authentication and *userlist* for authorization:

```
user_auth:
  authentication:
    - type: oidc
      oidc_jwks_url: https://login.microsoftonline.com/xxx/discovery/v2.0/keys
      oidc_provider: azure
      oidc_username_in_token: preferred_username
      oidc_username_template: *.
  authorization:
    - type: userlist
      userlist_allowed_users:
        - xxx
```

see plugins folder for available plugins and their parameters.

6.2 Customizing the Pulsar Environment (*nix only)

For many deployments, Pulsar's environment will need to be tweaked. For instance to define a `DRMAA_LIBRARY_PATH` environment variable for the `drmaa` Python module or to define the location to a find a location of Galaxy (via `GALAXY_HOME`) if certain Galaxy tools require it or if Galaxy metadata is being set by the Pulsar.

The file `local_env.sh` (created automatically by `pulsar-config`) will be source by `pulsar` before launching the application and by child process created by Pulsar that require this configuration.

6.3 Job Managers (Queues)

By default the Pulsar will maintain its own queue of jobs. While ideal for simple deployments such as those targeting a single Windows instance, if Pulsar is going to be used on more sophisticated clusters, it can be configured to maintain multiple such queues with different properties or to delegate to external job queues (via DRMAA, qsub/qstat CLI commands, or Condor).

For more information on configured external job managers, see *Job Managers*.

6.4 Galaxy Tools

Some Galaxy tool wrappers require a copy of the Galaxy codebase itself to run. Such tools will not run under Windows, but on *nix hosts the Pulsar can be configured to add the required Galaxy code a jobs `PYTHON_PATH` by setting `GALAXY_HOME` environment variable in the Pulsar's `local_env.sh` file (described above).

Most Galaxy tools require external command-line tools, known as *Galaxy Tool Dependencies*, to execute correctly. In Galaxy, these are provided by its Dependency Resolution system. Pulsar uses this same system, which can be configured via the `dependency_resolution` option in `app.yml`. See the example in app.yml.sample for additional information. In its default configuration, Pulsar will automatically install Conda but not automatically install missing tool dependencies. Administrators sending large numbers of tools to Pulsar most likely want to enable the `auto_install` option

on the `conda` dependency resolver or the `conda_auto_install` global option so that it is not necessary to manually install dependencies for tools sent to Pulsar. Both options are documented in the app.yml.sample file.

6.5 Message Queue (AMQP)

Galaxy and Pulsar can be configured to communicate via a message queue instead of a Pulsar web server. In this mode, Pulsar and Galaxy will send and receive job control and status messages via an external message queue server using the AMQP protocol. This is sometimes referred to as running Pulsar "webless".

Information on configuring RabbitMQ, one such compatible message queue, can be found in galaxy_with_rabbitmq_conf.

In addition, when using a message queue, Pulsar will download files from and upload files to Galaxy instead of the inverse. Message queue mode may be very advantageous if Pulsar needs to be deployed behind a firewall or if the Galaxy server is already set up (via proxy web server) for large file transfers.

A template configuration for using Galaxy with a message queue can be created by `pulsar-config`:

```
$ pulsar-config --mq
```

You will also need to ensure that the ` `kombu` Python dependency is installed (`pip install kombu`). Once this is available, simply set the `message_queue_url` property in `app.yml` to the correct URL of your configured AMQP endpoint.

AMQP does not guarantee message receipt. It is possible to have Pulsar (and Galaxy) require acknowledgement of receipt and resend messages that have not been acknowledged, using the `amqp_ack*` options documented in app.yml.sample, but beware that enabling this option can give rise to the Two Generals Problem, especially when Galaxy or the Pulsar server are down (and thus not draining the message queue).

In the event that the connection to the AMQP server is lost during message publish, the Pulsar server can retry the connection, governed by the `amqp_publish*` options documented in app.yml.sample.

6.6 Caching (Experimental)

Pulsar and its client can be configured to cache job input files. For some workflows this can result in a significant decrease in data transfer and greater throughput. On the Pulsar server side - the property `file_cache_dir` in `app.yml` must be set. See Galaxy's job_conf.xml example file for information on configuring the client.

More discussion on this can be found in this galaxy-dev mailing list thread and future plans and progress can be tracked on this Trello card.

CHAPTER

SEVEN

JOB MANAGERS

By default the Pulsar will maintain its own queue of jobs. Under Linux however, Pulsar can be configured to maintain multiple such queues with different properties or to delegate to external job queues (via DRMAA, qsub/qstat CLI commands, or Condor).

To configure job managers, uncomment the `managers` section of `app.yml` and modify it as needed. For instance, the default job manager corresponds to a configuration of

```
managers:
  _default_:
    type: queued_python
    num_concurrent_jobs: 1
```

The `type` of `queued_python` is indicating that the jobs are queued but that the queue is managed locally by Pulsar. Other possible values for `type` include `queued_drmaa`, `queued_condor`, `queued_cli`, `queued_external_drmaa` (examples of each follow).

7.1 Named Managers

The `managers` section can contain any number of named managers. For example:

```
managers:
  _default_:
    type: queued_python
    num_concurrent_jobs: 1

  example:
    type: queued_python
    num_concurrent_jobs: "*"
```

In this instance, Pulsar creates a second named queued (`example`) that will run as many concurrent jobs as the server has cores. The Galaxy Pulsar url should have `/managers/example` appended to it to use a named manager such as this.

7.2 DRMAA

The `queued_python` manager type is easy to configure but has serious limitations - for instance jobs running when Pulsar is restarted will be lost. For these reasons it is best to configure a real external job manager when possible.

Likely the cleanest way to interface with an external queueing system is going to be DRMAA. This method will likely work with Slurm, PBS Torque, LSF, etc.... In this case, one should likely setup a `local_env.sh` file and update it to set `DRMAA_LIBRARY_PATH` to point to the correct `libdrmaa.so` file. Also, the Python `drmaa` module must be installed (e.g. via `pip install drmaa`).:

```
managers:
  _default_:
    type: queued_drmaa
    native_specification: "-P bignodes -R y -pe threads 8"
```

Here the optional `native_specification` is going to depend on the underlying job manager.

In addition to the default dependencies described in the installation documentation, a DRMAA library will need to be installed and the python dependency `drmaa` will need to be installed as well to use the `queued_drmaa` manager. This can be done by activating Pulsar's virtual environment and running:

```
pip install drmaa
```

If you are using DRMAA, be sure to define `DRMAA_LIBRARY_PATH` in Pulsar's `local_env.sh` file.

7.3 Condor

Condor can also be used as a backend.

```
managers:
  _default_:
    type: queued_condor
    # Optional attributes...
    submit_universe: vanilla
    submit_request_memory: 32
    submit_requirements: 'OpSys == "LINUX" && Arch =="INTEL"'
    submit_rank: "Memory >= 64"
```

This would set universe, request_memory, requirements, and rank in the condor submission file to the specified values. For more information on condor submission files see the HTCondor quickstart for more information.

7.4 CLI

Pulsar can manage jobs via command-line execution of `qsub`, `qdel`, `stat` on the local machine.

```
managers:
  _default_:
    type: queued_cli
    job_plugin: Torque
```

`job_plugin` can also be `slurm` (to use `srun`, etc...) or `slurm_torque` (to use the Slurm varaint of `qsub`, etc...).

Pulsar can also login into a remote host before executing these commands if the job manager is not accessible from the Pulsar host.

```
managers:
  _default_:
    type: queued_cli
    job_plugin: Torque
    shell_plugin: SecureShell
    shell_hostname: queuemanager
    shell_username: queueuser
```

This will login to `queuemanager` as user `queueuser` to submit jobs. Be sure keyless SSH between Pulsar and the remote host is configured in this case.

7.5 Run-As-Real User DRMAA

All of the proceeding will run jobs as the same operating system user that Pulsar is running as. The `queued_external_drmaa` manager type will actually run DRMAA jobs via the user requested by the client (e.g. the Galaxy user).

```
managers:
  _default_:
    type: queued_external_drmaa
    production: true
    # Following are optional - should leave as defaults in most cases.
    #chown_working_directory_script: scripts/chown_working_directory.bash
    #drmaa_kill_script: scripts/drmaa_kill.bash
    #drmaa_launch_script: scripts/drmaa_launch.bash
```

For more information on running jobs as the real user, check out this discussion from the Galaxy mailing list.

7.6 More Options

Any manager can override the `staging_directory` used by setting this property in its configuration section.

The `min_polling_interval: 0.5` option can be set on any manager to control how frequently Pulsar will poll the resource manager for job updates.

For staging actions initiated by Pulsar (e.g. when driving Pulsar by message queue) - the following parameters can be set to control retrying these actions (if they) fail. (XXX_max_retries=-1 => no retry, XXX_max_retries=0 => retry forever - this may be a bit counter-intuitive but is consistent with Kombu.

```
preprocess_action_max_retries: -1
preprocess_action_interval_start: 2
preprocess_action_interval_step: 2
preprocess_action_interval_max: 30
postprocess_action_max_retries: -1
postprocess_action_interval_start: 2
postprocess_action_interval_step: 2
postprocess_action_interval_max: 30
```

CHAPTER

EIGHT

CONTAINERS

8.1 Galaxy and Shared File Systems

Galaxy can be configured to run Pulsar with traditional job managers and just submit jobs that launch containers. Simply setting `docker_enabled` on the job environment in Galaxy's job_conf.yml file will accomplish this.

There are limitations to using DRM systems that submit job scripts that launch containers though. Modern container scheduling environments (AWS Batch or Kubernetes or instance) are capable of scheduling containers directly. This is conceptually cleaner, persumably scales better, and side steps all sorts of issues for the deployer and developer such as configuring Docker and managing the interaction between the DRM and the container host server (i.e. the Docker server).

There are a couple approaches to scheduling containers directly in Galaxy - such as the Galaxy Kubernetes runner and the Galaxy AWS Batch runner. These approaches require Galaxy be deployed alongside the compute infrasture (i.e. on Amazon with the same EFS volume or inside of Kubernetes with the same mounts).

These two scenarios and some of their limitations are described below.

Fig. 1: Deployment diagram for Galaxy's AWS Batch job runner.

Fig. 2: Deployment diagram for Galaxy's Kubernetes job runner.

The most glaring disadvantage of not using Pulsar in the above scenarios is that Galaxy must be deployed in the same container with the same mounts as the job execution environment. This prevents leveraging external cloud compute, multi-cloud compute, and makes it unsuitable for common Galaxy use cases such as large public instances, Galaxy's leveraging institution non-cloud storage, etc… Even within the same cloud - a large shared file system can be an expensive prospect and Pulsar may allow making use of buckets and such more tractable. Finally, Pulsar offers more options in terms of how to collect metadata which can have big implications in terms of metadata.

8.2 Co-execution

Galaxy job inputs and outputs are very flexible and staging up job inputs, configs, and scripts, and staging down results doesn't map cleanly to cloud APIs and cannot be fully reasoned about until job runtime. For this reason, code that needs to know how stage Galaxy jobs up and down needs to run in the cloud when disk isn't shared and Galaxy cannot do this directly. Galaxy jobs however are typically executed in Biocontainers that are minimal containers just for the tool being executed and not appropriate for executing Galaxy code.

For this reason, the Pulsar runners that schedule containers will run a container beside (or before and after) that is responsible for staging the job up and down, communicating with Galaxy, etc..

Perhaps the most typical potential scenario is using the Kubernetes Job API along with a message queue for communication with Galaxy and a Biocontainer. A diagram for this deployment would look something like:

The modern Galaxy landscape is much more container driven, but the setup can be simplified to use Galaxy dependency resolution from within the "pulsar" container. This allows the tool and the staging code to live side-by-side and results in requesting only one container for the execution from the target container. The default Pulsar staging container has a conda environment configured out of the box and has some initial tooling to be connected to a CVM-FS available conda directory.

This one-container approach (staging+conda) is available with or without MQ and on either Kubernetes or against a GA4GH TES server. The TES version of this with RabbitMQ to mitigate communication looks like:

Notice when executing jobs on Kubernetes, the containers of the pod run concurrrently. The Pulsar container will compute a command-line and write it out, the tool container will wait for it on boot and execute it when available, while the Pulsar container waits for a return code from the tool container to proceed to staging out the job. In the GA4GH TES case, 3 containers are used instead of 2, but they run sequentially one at a time.

Typically, a MQ is needed to communicate between Pulsar and Galaxy even though the status of the job could potentially be inferred from the container scheduling environment. This is because Pulsar needs to transfer information about job state, etc. after the job is complete.

More experimentally this shouldn't be needed if extended metadata is being collected because then the whole job state that needs to be ingested by Galaxy should be populated as part of the job. In this case it may be possible to get away without a MQ.

8.3 Deployment Scenarios

8.3.1 Kubernetes

8.3.2 GA4GH TES

8.3.3 AWS Batch

Work in progress.

Fig. 3: Kuberentes job execution with a biocontainer for the tool and RabbitMQ for communicating with Galaxy.

Fig. 4: Kuberentes job execution with Conda dependencies for the tool and RabbitMQ for communicating with Galaxy.

Fig. 5: Kuberentes job execution with a biocontainer for the tool and no message queue.

Fig. 6: Kuberentes job execution with Conda dependencies for the tool and no message queue.

Fig. 7: GA4GH TES job execution with a biocontainer for the tool and RabbitMQ for communicating with Galaxy.

Fig. 8: GA4GH TES job execution with Conda dependencies for the tool and RabbitMQ for communicating with Galaxy.

Fig. 9: GA4GH TES job execution with a biocontainer for the tool and no message queue.

Fig. 10: GA4GH TES job execution with Conda dependencies for the tool and no message queue.

CHAPTER

NINE

GALAXY CONFIGURATION

9.1 Examples

The most complete and updated documentation for configuring Galaxy job destinations is Galaxy's `job_conf.xml.sample_advanced` file (check it out on GitHub). These examples just provide a different Pulsar-centric perspective on some of the documentation in that file.

9.1.1 Simple Windows Pulsar Web Server

The following Galaxy `job_conf.xml` assumes you have deployed a simple Pulsar web server to the Windows host `windowshost.examle.com` on the default port (`8913`) with a `private_token` (defined in `app.yml`) of `123456789changeme`. Most Galaxy jobs will just route use Galaxy's local job runner but `msconvert` and `proteinpilot` will be sent to the Pulsar server on `windowshost.examle.com`. Sophisticated tool dependency resolution is not available for Windows-based Pulsar servers so ensure the underlying application are on the Pulsar's path.

```
<?xml version="1.0"?>
<job_conf>
    <plugins>
        <plugin id="local" type="runner" load="galaxy.jobs.runners.local:LocalJobRunner"/
↪>
        <plugin id="pulsar" type="runner" load="galaxy.jobs.runners.
↪pulsar:PulsarLegacyJobRunner"/>
    </plugins>
    <destinations default="local">
        <destination id="local" runner="local"/>
        <destination id="win_pulsar" runner="pulsar">
            <param id="url">https://windowshost.examle.com:8913/</param>
            <param id="private_token">123456789changeme</param>
        </destination>
    </destinations>
    <tools>
        <tool id="msconvert" destination="win_pulsar" />
        <tool id="proteinpilot" destination="win_pulsar" />
    </tools>
</job_conf>
```

9.1.2 Targeting a Linux Cluster (Pulsar Web Server)

The following Galaxy `job_conf.xml` assumes you have a very typical Galaxy setup - there is a local, smaller cluster that mounts all of Galaxy's data (so no need for the Pulsar) and a bigger shared resource that cannot mount Galaxy's files requiring the use of the Pulsar. This variant routes some larger assembly jobs to the remote cluster - namely the `trinity` and `abyss` tools.

```
<?xml version="1.0"?>
<job_conf>
    <plugins>
        <plugin id="drmaa" type="runner" load="galaxy.jobs.runners.drmaa:DRMAAJobRunner"/
↪>
        <plugin id="pulsar" type="runner" load="galaxy.jobs.runners.
↪pulsar:PulsarRESTJobRunner"/>
    </plugins>
    <destinations default="local_cluster">
        <destination id="local_cluster" runner="drmaa">
            <param id="native_specification">-P littlenodes -R y -pe threads 4</param>
        </destination>
        <destination id="remote_cluster" runner="pulsar">
            <param id="url">http://remotelogin:8913/</param>
            <param id="submit_native_specification">-P bignodes -R y -pe threads 16</
↪param>
            <!-- Look for trinity package at remote location - define tool_dependency_dir
            in the Pulsar app.yml file.
            -->
            <param id="dependency_resolution">remote</param>
        </destination>
    </destinations>
    <tools>
        <tool id="trinity" destination="remote_cluster" />
        <tool id="abyss" destination="remote_cluster" />
    </tools>
</job_conf>
```

For this configuration, on the Pulsar side be sure to also set a `DRMAA_LIBRARY_PATH` in `local_env.sh`, install the Python `drmaa` module, and configure a DRMAA job manager for Pulsar in `app.yml` as described in *Job Managers*.

9.1.3 Targeting a Linux Cluster (Pulsar over Message Queue)

For Pulsar instances sitting behind a firewall, a web server may be impossible. If the same Pulsar configuration discussed above is additionally configured with a `message_queue_url` of `amqp://rabbituser:rabb8pa8sw0d@mqserver:5672//` in `app.yml`, the following Galaxy configuration will cause this message queue to be used for communication. This is also likely better for large file transfers since typically your production Galaxy server will be sitting behind a high-performance proxy while Pulsar will not.

```
<?xml version="1.0"?>
<job_conf>
    <plugins>
        <plugin id="drmaa" type="runner" load="galaxy.jobs.runners.drmaa:DRMAAJobRunner"/
↪>
        <plugin id="pulsar_default" type="runner" load="galaxy.jobs.runners.
↪pulsar:PulsarMQJobRunner">
```

(continues on next page)

(continued from previous page)

```
            <!-- Must tell Pulsar where to send files. -->
            <param id="galaxy_url">https://galaxy.example.org</param>
            <!-- Message Queue Connection (should match message_queue_url in Pulsar's app.
↪yml)
            -->
            <param id="amqp_url">amqp://rabbituser:rabb8pa8sw0d@mqserver.example.
↪org:5672//</param>
            </plugin>
        <plugin id="pulsar_hugenodes" type="runner" load="galaxy.jobs.runners.
↪pulsar:PulsarMQJobRunner">
            <param id="galaxy_url">https://galaxy.example.org</param>
            <param id="amqp_url">amqp://rabbituser:rabb8pa8sw0d@mqserver.example.
↪org:5672//</param>
            <!-- Set the 'manager' param to reference a named Pulsar job manager -->
            <param id="manager">hugenodes</param>
        </plugin>
    </plugins>
    <destinations default="local_cluster">
        <destination id="local_cluster" runner="drmaa">
            <param id="native_specification">-P littlenodes -R y -pe threads 4</param>
        </destination>
        <destination id="bignodes_cluster" runner="pulsar_default">
            <!-- Tell Galaxy where files are being stored on remote system, so
                 the web server can simply ask for this information.
            -->
            <param id="jobs_directory">/path/to/remote/pulsar/files/staging/</param>
            <!-- Remaining parameters same as previous example -->
            <param id="submit_native_specification">-P bignodes -R y -pe threads 16</
↪param>
        </destination>
        <destination id="hugenodes_cluster" runner="pulsar_hugenodes">
            <param id="jobs_directory">/path/to/remote/pulsar/files/staging/</param>
            <param id="submit_native_specification">-P hugenodes -R y -pe threads 128</
↪param>
        <destination>
    </destinations>
    <tools>
        <tool id="trinity" destination="bignodes_cluster" />
        <tool id="abyss" destination="hugenodes_cluster" />
    </tools>
</job_conf>
```

The `manager` param to the `PulsarMQJobRunner` plugin allows for using the same AMQP server and vhost (in this example, the default `/` vhost) between multiple Pulsar servers, or submitting jobs to multiple managers (see: *Job Managers*) on the same Pulsar server.

In this example, the `_default_` job manager will be used for `trinity` jobs, and the `hugenodes` job manager will be used for `abyss` jobs.

Note: If you only need to define different `submit_native_specification` params on the same cluster for these tools/destinations, it is not necessary to use a separate manager - multiple destinations can reference the same plugin. This example is for documentation purposes.

All of the amqp_* options documented in app.yml.sample can be specified as params to the PulsarMQJobRunner plugin. These configure Galaxy's connection to the AMQP server (rather than Pulsar's connection, which is configured in Pulsar's app.yml). Additionally, specifying the persistence_directory param controls where AMQP acknowledgement receipts will be stored on the Galaxy side.

For those interested in this deployment option and new to Message Queues, there is more documentation in galaxy_with_rabbitmq_conf.

Additionally, Pulsar ships with an RSync and SCP transfer action rather than making use of the HTTP transport method:

```
<?xml version="1.0"?>
<job_conf>
    <plugins>
        <plugin id="pulsar_mq" type="runner" load="galaxy.jobs.runners.
↪pulsar:PulsarMQJobRunner">
            <!-- Must tell Pulsar where to send files. -->
            <param id="galaxy_url">https://galaxyserver</param>
            <!-- Message Queue Connection (should match message_queue_url in
                 Pulsar's app.yml). pyamqp may be necessary over amqp if SSL is used
            -->
            <param id="amqp_url">pyamqp://rabbituser:rabb8pa8sw0d@mqserver:5671//?ssl=1</
↪param>
        </plugin>
    </plugins>
    <destinations default="pulsar_mq">
        <destination id="remote_cluster" runner="pulsar_mq">
            <!-- This string is replaced by Pulsar, removing the requirement
                 of coordinating Pulsar installation directory between cluster
                 admin and galaxy admin
            -->
            <param id="jobs_directory">__PULSAR_JOBS_DIRECTORY__</param>
            <!-- Provide connection information, should look like:

                    paths:
                        - path: /home/vagrant/  # Home directory for galaxy user
                          action: remote_rsync_transfer # _rsync_ and _scp_ are available
                          ssh_user: vagrant
                          ssh_host: galaxy-vm.host.edu
                          ssh_port: 22

            -->
             <param id="file_action_config">file_actions.yaml</param>
             <!-- Provide an SSH key for access to the local $GALAXY_ROOT,
            should be accessible with the username/hostname provided in
            file_actions.yaml
             -->
             <param id="ssh_key">-----BEGIN RSA PRIVATE KEY-----
            .............
            </param>
            <!-- Allow the remote end to know who is running the job, may need
                 to append @domain.edu after it. Only used if the
                 "DRMAA (via external users) manager" is used
             -->
            <param id="submit_user">$__user_name__</param>
```

(continues on next page)

(continued from previous page)

```
        </destination>
    </destinations>
    <tools>
        <tool id="trinity" destination="remote_cluster" />
        <tool id="abyss" destination="remote_cluster" />
    </tools>
</job_conf>
```

9.1.4 Targeting Apache Mesos (Prototype)

See commit message for initial work on this and this post on galaxy-dev.

9.1.5 Generating Galaxy Metadata in Pulsar Jobs

This option is often referred to as *remote metadata*.

Typically Galaxy will process Pulsar job outputs and generate metadata on the Galaxy server. One can force this to happen inside Pulsar jobs (wherever the Pulsar job runs). This is similar to the way that non-Pulsar Galaxy jobs work: job output metadata is generated at the end of a standard Galaxy job, not by the Galaxy server.

This option comes with a downside that you should be aware of, explained in Issue #234. Unless you are seeing high load on your Galaxy server while finishing Pulsar jobs, it is safest to use the default (remote metadata disabled).

In order to enable the remote metadata option:

1. Set `GALAXY_VIRTUAL_ENV` to the path to Galaxy's virtualenv (or one containing Galaxy's dependencies) when starting Pulsar. This can be done in the `local_env.sh` file. Instructions on setting up a Galaxy virtualenv can be found in the Galaxy Docs.
2. Instruct Pulsar with the path to a copy of Galaxy at the same version as your Galaxy server. This can either be done by setting `GALAXY_HOME` in `local_env.sh`, or by setting `galaxy_home` in `app.yml`.
3. In the Galaxy `job_conf.xml` *destination(s)* you want to enable remote metadata on, set the following params:

   ```
   <param id="remote_metadata">true</param>
   <param id="remote_property_galaxy_home">/path/to/galaxy</param>
   ```

   and one of either:

   ```
   <param id="use_metadata_binary">true</param>
   ```

   or:

   ```
   <param id="use_remote_datatypes">false</param>
   ```

9.2 Data Staging

Most of the parameters settable in Galaxy's job configuration file `job_conf.xml` are straight forward - but specifying how Galaxy and the Pulsar stage various files may benefit from more explanation.

`default_file_action` defined in Galaxy's *job_conf.xml* describes how inputs, outputs, indexed reference data, etc... are staged. The default `transfer` has Galaxy initiate HTTP transfers. This makes little sense in the context of message queues so this should be set to `remote_transfer`, which causes Pulsar to initiate the file transfers. Additional options are available including `none`, `copy`, and `remote_copy`.

In addition to this default - paths may be overridden based on various patterns to allow optimization of file transfers in production infrastructures where various systems mount different file stores and file stores with different paths on different systems.

To do this, the defined Pulsar destination in Galaxy's `job_conf.xml` may specify a parameter named `file_action_config`. This needs to be a config file path (if relative, relative to Galaxy's root) like `config/pulsar_actions.yaml` (can be YAML or JSON - but older Galaxy's only supported JSON). The following captures available options:

```
paths:
  # Use transfer (or remote_transfer) if only Galaxy mounts a directory.
  - path: /galaxy/files/store/1
    action: transfer

  # Use copy (or remote_copy) if remote Pulsar server also mounts the directory
  # but the actual compute servers do not.
  - path: /galaxy/files/store/2
    action: copy

  # If Galaxy, the Pulsar, and the compute nodes all mount the same directory
  # staging can be disabled altogether for given paths.
  - path: /galaxy/files/store/3
    action: none

  # Following block demonstrates specifying paths by globs as well as rewriting
  # unstructured data in .loc files.
  - path: /mnt/indices/**/bwa/**/*.fa
    match_type: glob
    path_types: unstructured  # Set to *any* to apply to defaults & unstructured paths.
    action: transfer
    depth: 1  # Stage whole directory with job and not just file.

  # Following block demonstrates rewriting paths without staging. Useful for
  # instance if Galaxy's data indices are mounted on both servers but with
  # different paths.
  - path: /galaxy/data
    path_types: unstructured
    action: rewrite
    source_directory: /galaxy/data
    destination_directory: /work/galaxy/data

  # The following demonstrates use of the Rsync transport layer
  - path: /galaxy/files/
    action: remote_rsync_transfer
```

(continues on next page)

(continued from previous page)

```
    # Additionally the action remote_scp_transfer is available which behaves in
    # an identical manner
    ssh_user: galaxy
    ssh_host: f.q.d.n
    ssh_port: 22

# See action_mapper.py for explaination of mapper types:
# - input: Galaxy input datasets and extra files.
# - config: Galaxy config and param files.
# - tool: Files from tool's tool_dir (for now just wrapper if available).
# - workdir: Input work dir files - e.g.task-split input file.
# - metadata: Input metadata files.
# - output: Galaxy output datasets in their final home.
# - output_workdir:  Galaxy from_work_dir output paths and other files (e.g. galaxy.json)
# - output_metadata: Meta job and data files (e.g. Galaxy metadata generation files and
#                    metric instrumentation files)
# - unstructured: Other fixed tool parameter paths (likely coming from tool data, but not
#                 nessecarily). Not sure this is the best name...
```

CHAPTER

TEN

SCRIPTS

This section describes some of the various scripts that are distributed with Pulsar.

10.1 `pulsar` (*nix)

Installing Pulsar will install the `pulsar` script. It is a lightweight wrapper abstracting out a few different ways to run Pulsar. Pulsar can easily be run inside a variety wsgi servers or stand-alone without a web server using `pulsar-main` - the `pulsar` script shouldn't be considered a best practice - it merely provides a minimal level of convenience that may be useful in some deployment scenarios.

Very simply, `pulsar` will source `local_env.sh` if it is present (to configure things like `DRMAA_LIBRARY_PATH`) and then determine which external application to use to run Pulsar (either a WSGI server or `pulsar- main`) and delegate to that method.

`pulsar` can be passed the `--mode` argument to explicitly describe which application should be used to run Pulsar. If `--mode` unspecified, `pulsar` will check the `PATH` and launch look for (in order) `uwsgi`, `circusd`, `chaussette`, and finally `paster` to determine which mode to use.

10.1.1 `paster` mode

Paste is installed with Pulsar and so is the fallback mode if none of the other web servers is available.

In this mode, Pulsar can be launched using the command:

```
pulsar
```

This will run the server in your terminal (not as a daemon) and the server will run as long as this command is running. To run Pulsar as a daemon, use the command:

```
pulsar --daemon
```

This will run Pulsar in daemon mode (i.e. run in the background). In daemon mode, paster creates a pid file in the current directory called `paster.pid` and a log file `paster.log`. The daemon can be stopped using the command:

```
pulsar --stop-daemon
```

10.1.2 `webless` mode

This mode can be used to launch Pulsar without a web server. This only makes sense if a `message_queue_url` is defined in `app.yml` and the client (e.g Galaxy) configures all staging to be triggered remotely (this is the default for the Galaxy job runner `galaxy.jobs.runners.pulsar:PulsarMQJobRunner`).

See the documentation for the `pulsar-main` for the arguments that may be supplied to `pulsar` in this mode.

10.1.3 Other Modes

`pulsar-config` will configure sections in `server.ini` that allow Pulsar to be launched using uWSGI, Cirucs, and Chaussette. `pulsar` will launch these servers when `--mode` is specified as `uwsgi`, `circus`, `chaussette` respectively.

See the documentation for the respective application for a full description of the arguments that can be used to configure that web server. Presumably each of these servers is more performant and better maintained than Paste but Paste is cross-platform and makes it trivial to configure SSL and so it remains the default for Pulsar for now.

10.2 `pulsar` (Windows)

`pulsar` is a lightweight wrapper around `paster serve` (see docs). It will check the current directory for a `server.ini` file and launch the described Pulsar server using Paste.

10.3 `pulsar-main`

Usage:

```
pulsar-main [-h] [-c CONFIG_DIR] [--ini_path INI_PATH]
            [--app_conf_path APP_CONF_PATH] [--app APP] [-d]
            [--daemon-log-file DAEMON_LOG_FILE] [--pid-file PID_FILE]
```

Help

Stand-alone entry point for running Pulsar without a web server.

In its simplest form, this method will check the current directory for an app.yml and run the corresponding configuration as a standalone applciation. This makes sense when `app.yml` contains a `message_queue_url` option so Pulsar is configured to listen to a message queue and doesn't require a web server.

The following commands can be used to bootstrap such a setup.:

```
mkdir pulsar-mq-config
cd pulsar-mq-config
pulsar-config --mq
pulsar-main
```

This script can be used in a standalone fashion, but it is generally better to run the `pulsar` script with `--mode webless` - which will in turn delegate to this script.

Options:

```
-h, --help            show this help message and exit
-c CONFIG_DIR, --config_dir CONFIG_DIR
                      Default directory to search for relevant Pulsar
                      configuration files (e.g. app.yml, server.ini).
--ini_path INI_PATH   Specify an explicit path to Pulsar's server.ini
                      configuration file.
--app_conf_path APP_CONF_PATH
                      Specify an explicit path to Pulsar's app.yml
                      configuration file.
--app APP
-d, --daemonize       Daemonzie process (requires daemonize library).
--daemon-log-file DAEMON_LOG_FILE
                      Log file for daemon, if --daemonize supplied.
--pid-file PID_FILE   Pid file for daemon, if --daemonize supplied (default
                      is pulsar.pid).
```

10.4 pulsar-config (Windows)

Usage:

```
pulsar-config [-h] [--directory DIRECTORY] [--mq] [--no_logging]
              [--host HOST] [--private_token PRIVATE_TOKEN]
              [--port PORT] [--install] [--force]
```

Help

Initialize a directory with a minimal pulsar config.

Options:

```
-h, --help            show this help message and exit
--directory DIRECTORY
                      Directory containing the configuration files for
                      Pulsar.
--mq                  Write configuration files for message queue server
                      deployment instead of more traditional RESTful web
                      based pulsar.
--no_logging          Do not write Pulsar's default logging configuration to
                      server.ini and if uwsgi is configured do not configure
                      its logging either.
--host HOST           Host to bind Pulsar to - defaults to localhost.
                      Specify 0.0.0.0 to listen on all interfaces.
--private_token PRIVATE_TOKEN
                      Private token used to authorize clients. If Pulsar is
                      not protected via firewall, this should be specified
                      and SSL should be enabled. See https://pulsar.readthed
                      ocs.org/en/latest/configure.html for more information
                      on security.
--port PORT           Port to bind Pulsar to (ignored if --mq is specified).
--install             Install optional dependencies required by specified
                      configuration (e.g. drmaa, supervisor, uwsgi, etc...).
--force               Overwrite existing files if they already exist.
```

10.5 pulsar-config (*nix)

Usage:

```
pulsar-config [-h] [--directory DIRECTORY] [--mq] [--no_logging]
              [--supervisor] [--wsgi_server {paster,uwsgi}]
              [--libdrmaa_path LIBDRMAA_PATH] [--host HOST]
              [--private_token PRIVATE_TOKEN] [--port PORT] [--install]
              [--force]
```

Help

Initialize a directory with a minimal pulsar config.

Options:

```
-h, --help            show this help message and exit
--directory DIRECTORY
                      Directory containing the configuration files for
                      Pulsar.
--mq                  Write configuration files for message queue server
                      deployment instead of more traditional RESTful web
                      based pulsar.
--no_logging          Do not write Pulsar's default logging configuration to
                      server.ini and if uwsgi is configured do not configure
                      its logging either.
--supervisor          Write a supervisord configuration file for managing
                      pulsar out as well.
--wsgi_server {paster,uwsgi}
                      Web server stack used to host Pulsar wsgi application.
--libdrmaa_path LIBDRMAA_PATH
                      Configure Pulsar to submit jobs to a cluster via DRMAA
                      by supplying the path to a libdrmaa .so file using
                      this argument.
--host HOST           Host to bind Pulsar to - defaults to localhost.
                      Specify 0.0.0.0 to listen on all interfaces.
--private_token PRIVATE_TOKEN
                      Private token used to authorize clients. If Pulsar is
                      not protected via firewall, this should be specified
                      and SSL should be enabled. See https://pulsar.readthed
                      ocs.org/en/latest/configure.html for more information
                      on security.
--port PORT           Port to bind Pulsar to (ignored if --mq is specified).
--install             Install optional dependencies required by specified
                      configuration (e.g. drmaa, supervisor, uwsgi, etc...).
--force               Overwrite existing files if they already exist.
```

10.6 pulsar-check

Usage:

```
Script used to run an example job against a running Pulsar server.
```

Help

Exercises various features both the Pulsar client and server.

Options:

```
-h, --help            show this help message and exit
--url=URL             URL of the Pulsar web server to target.
--private_token=PRIVATE_TOKEN
                      Private token used to authorize client, if the Pulsar
                      server specified a private_token in app.yml this must
                      match that value.
--transport=TRANSPORT
                      Specify as 'curl' to use pycurl client for staging.
--cache               Specify to test Pulsar caching during staging.
--test_errors         Specify to exercise exception handling during staging.
--suppress_output
--disable_cleanup     Specify to disable cleanup after the job, this is
                      useful to checking the files generated during the job
                      and stored on the Pulsar server.
```

CHAPTER

ELEVEN

PULSAR PROJECT CODE OF CONDUCT

This code of conduct outlines our expectations for participants within the Pulsar community, as well as steps to reporting unacceptable behavior. We are committed to providing a welcoming and inspiring community for all and expect our code of conduct to be honored. Anyone who violates this code of conduct may be banned from the community.

Our open source community strives to:

- **Be friendly and patient.**
- **Be welcoming**: We strive to be a community that welcomes and supports people of all backgrounds and identities. This includes, but is not limited to members of any race, ethnicity, culture, national origin, colour, immigration status, social and economic class, educational level, sex, sexual orientation, gender identity and expression, age, size, family status, political belief, religion, and mental and physical ability.
- **Be considerate**: Your work will be used by other people, and you in turn will depend on the work of others. Any decision you take will affect users and colleagues, and you should take those consequences into account when making decisions. Remember that we're a world-wide community, so you might not be communicating in someone else's primary language.
- **Be respectful**: Not all of us will agree all the time, but disagreement is no excuse for poor behavior and poor manners. We might all experience some frustration now and then, but we cannot allow that frustration to turn into a personal attack. It's important to remember that a community where people feel uncomfortable or threatened is not a productive one.
- **Be careful in the words that we choose**: We are a community of professionals, and we conduct ourselves professionally. Be kind to others. Do not insult or put down other participants. Harassment and other exclusionary behavior aren't acceptable. This includes, but is not limited to: Violent threats or language directed against another person, Discriminatory jokes and language, Posting sexually explicit or violent material, Posting (or threatening to post) other people's personally identifying information ("doxing"), Personal insults, especially those using racist or sexist terms, Unwelcome sexual attention, Advocating for, or encouraging, any of the above behavior, Repeated harassment of others. In general, if someone asks you to stop, then stop.
- **Try to understand why we disagree**: Disagreements, both social and technical, happen all the time. It is important that we resolve disagreements and differing views constructively. Remember that we're different. Diversity contributes to the strength of our community, which is composed of people from a wide range of backgrounds. Different people have different perspectives on issues. Being unable to understand why someone holds a viewpoint doesn't mean that they're wrong. Don't forget that it is human to err and blaming each other doesn't get us anywhere. Instead, focus on helping to resolve issues and learning from mistakes.

11.1 Diversity Statement

We encourage everyone to participate and are committed to building a community for all. Although we will fail at times, we seek to treat everyone both as fairly and equally as possible. Whenever a participant has made a mistake, we expect them to take responsibility for it. If someone has been harmed or offended, it is our responsibility to listen carefully and respectfully, and do our best to right the wrong.

Although this list cannot be exhaustive, we explicitly honor diversity in age, gender, gender identity or expression, culture, ethnicity, language, national origin, political beliefs, profession, race, religion, sexual orientation, socioeconomic status, and technical ability. We will not tolerate discrimination based on any of the protected characteristics above, including participants with disabilities.

11.2 Reporting Issues

If you experience or witness unacceptable behavior, or have any other concerns, please report it to any combination of the following people that makes you feel the most comfortable:

- Dave Clements (clementsgalaxy@gmail.com). Dave is the Galaxy Project community outreach manager and has experience handling Code of Conduct related issues.
- Dr. Mike Schatz (mschatz@cs.jhu.edu). Mike is Dave Clements' direct manager and issues related to Dave in some way should be reported to Mike.
- Helena Rasche (helena.rasche@gmail.com). Helena is a well-known, trusted community member, is LGBT+, and has completely separate funding and institutional affiliation from Dave and Mike.

All reports will be handled with discretion. In your report please include:

- Your contact information.
- Names (real, nicknames, or pseudonyms) of any individuals involved. If there are additional witnesses, please include them as well. Your account of what occurred, and if you believe the incident is ongoing. If there is a publicly available record (e.g. a mailing list archive or a public IRC logger), please include a link.
- Any additional information that may be helpful.

After filing a report, a representative will contact you personally, review the incident, follow up with any additional questions, and make a decision as to how to respond. If the person who is harassing you is part of the response team, they will recuse themselves from handling your incident. If the complaint originates from a member of the response team, it will be handled by a different member of the response team. We will respect confidentiality requests for the purpose of protecting victims of abuse.

11.3 Attribution & Acknowledgements

This code of conduct is based on the Open Code of Conduct from the TODOGroup.

CHAPTER

TWELVE

CONTRIBUTING

Please note that this project is released with a *Contributor Code of Conduct <https://pulsar.readthedocs.org/en/latest/conduct.html>*. By participating in this project you agree to abide by its terms.

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

12.1 Types of Contributions

12.1.1 Report Bugs

Report bugs at https://github.com/galaxyproject/pulsar/issues.

If you are reporting a bug, please include:

- Your operating system name and version, versions of other relevant software such as Galaxy or Docker.
- Links to relevant tools.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

12.1.2 Fix Bugs

Look through the GitHub issues for bugs. Most things there are up for grabs but the tag "Help Wanted" may be particulary good places to start.

12.1.3 Implement Features

Look through the GitHub issues for features (tagged with "enhancement"). Again, most things there are up for grabs but the tag "Help Wanted" may be particulary good places to start.

12.1.4 Write Documentation

Pulsar is cronically under documented, whether as part of the official Pulsar docs, in docstrings, or even on the web in blog posts, articles, and such.

12.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/galaxyproject/pulsar/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- This will hopefully become a community-driven project and contributions are welcome :)

12.2 Get Started!

Ready to contribute? Here's how to set up *pulsar* for local development.

1. Fork the *pulsar* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pulsar.git
```

3. Install your local copy into a virtualenv. Assuming you have `virtualenv` installed, this is how you set up your fork for local development:

```
$ cd pulsar/
$ virtualenv .venv
$ . .venv/bin/activate
$ pip install -r requirements.txt
$ pip install -r dev-requirements.txt
```

   If you have something like Slurm or Grid Engine configured on your local machine - you should also install `drmaa` with `pip install drmaa`.

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes lint:

```
$ make lint
```

and ensure the tests look good. The easiest way to test is with Docker if it is available (given the need to test commands with DRMAA, condor, sudo, etc…).:

```
$ docker run -v `pwd`:/pulsar -t jmchilton/pulsar_testing
```

This will mount your copy of *pulsar* in a Docker container preconfigured with all optional dependencies needed to run a wide range of integration tests. If Docker is to much of an ordeal many of Pulsar's tests can be executed by simply running `nosetests` from within an `virtualenv` configured as explained above.:

```
$ make tests
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

12.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. If the pull request adds functionality, the docs should ideally be updated. Put your new functionality into a function with a docstring. (Until the @jmchilton learns to do this consistently this is only a suggestion though.)
2. The pull request should work for Python 3.6 and later.

CHAPTER

THIRTEEN

PROJECT GOVERNANCE

This document informally outlines the organizational structure governing the Pulsar code base hosted at https://github.com/galaxyproject/pulsar. This governance extends to code-related activities of this repository such as releases and packaging and related projects. This governance does not include any other Galaxy-related projects belonging to the galaxyproject organization on GitHub.

13.1 Benevolent Dictator for Now (BDFN)

John Chilton (@jmchilton) is the benevolent dictator for now (BDFN) and is solely responsible for setting project policy. The BDFN is responsible for maintaining the trust of the developer community and so should be consistent and transparent in decision making processes and request comment and build consensus whenever possible.

The BDFN position only exists because the developers of the project believe it is currently too small to support full and open governance at this time. In order to keep things evolving quickly, it is better to keep procedures and process to a minimum and centralize important decisions with a trusted developer. The BDFN is explicitly meant to be replaced with a more formal and democratice process if the project grows to a sufficient size or importance.

The *committers* group is the group of trusted developers and advocates who manage the Pulsar code base. They assume many roles required to achieve the project's goals, especially those that require a high level of trust.

The BDFN will add committers as they see fit, usually after a few successful pull requests. Committers may commit directly or merge pull requests at their discretion, but everyone (including the BDFN) should open pull requests for larger changes.

In order to encourage a shared sense of ownership and openness, any committer may decide at any time to request a open governance model for the project be established and the BDFN must replace this informal policy with a more formal one and work with the project committers to establish a consensus on these procedures.

13.2 Committers

- John Chilton (@jmchilton)
- Nate Coraor (@natefoo)
- Helena Rasche (@hexylena)
- Marius van den Beek (@mvdbeek)

CHAPTER

FOURTEEN

DEVELOPING

This section contains documentation for maintainers of Pulsar.

14.1 Release Checklist

This release checklist is based on the Pocoo Release Management Workflow.

This assumes `~/.pypirc` file exists with the following fields (variations) are fine.

```
[distutils]
index-servers =
    pypi
    test

[pypi]
username:<username>
password:<password>

[test]
repository:https://test.pypi.org/legacy/
username:<username>
password:<password>
```

- Review `git status` for missing files.
- Verify the latest Travis CI builds pass.
- `make open-docs` and review changelog.
- Ensure the target release is set correctly in `pulsar/__init__.py` (`version` will be a `devN` variant of target release).
- `make clean && make lint && make tests`
- `make release`
  - Review Test PyPI site for errors.
  - Test intall `pip install -i https://testpypi.python.org/pypi pulsar-app`.

  This process will push packages to test PyPI, allow review, publish to production PyPI, tag the git repository, and push the tag upstream. If changes are needed, this can be broken down into steps such as:

  - `make release-local`
  - `make push-release`

CHAPTER

FIFTEEN

HISTORY

15.1 0.15.3 (2023-07-20)

- Fix Pulsar consumer state after `ConnectionResetError`. Pull Request 331
- User auth plugins (thanks to @SergeyYakubov). Pull Request 321

15.2 0.15.2 (2023-05-02)

- Fix Pulsar and Pulsar client reconnection to AMQP server. Pull Request 324
- Reduce verbosity of timeout exception catching. Pull Request 325

15.3 0.15.1 (2023-04-13)

- No changes, working around pypi isssue.

15.4 0.15.0 (2023-04-13)

- Updated Galaxy+Pulsar container. Pull Request 306
- Rework container execution - generalize Kubernetes execution to allow it to work without a message queue and to allow TES execution based on pydantic-tes (https://github.com/jmchilton/pydantic-tes). Pull Request 302
- Add documentation and diagrams for container execution scenarios. Pull Request 302
- Rework integration tests to use pytest more aggressively.
- Fixes to CI to run more tests that weren't being executed because Tox was not sending environment variables through to pytest.
- Add option `amqp_key_prefix` to direct task queue naming while retaining simple default manager names and such in container scheduling deployments. Pull Request 315
- Various typing and CI fixes. Pull Request 312, Pull Request 319
- Fixes for extra_file handling. Pull Request 318
- Separate tool_stdio and job_stdio handling. Pull Request 318
- Re-import MEMORY_STATEMENT.sh from Galaxy. Pull Request 297

- Add support for logging to sentry. Pull Request 322

15.5 0.14.16 (2022-10-04)

- Fix small regression related to building URLs for client action mapping.

15.6 0.14.15 (2022-10-03)

- Fix small regressions bugs in 0.14.14 - updating runner util code was bigger swap over than it seemed.

15.7 0.14.14 (2022-10-30)

- Bring in updated Galaxy runner util code. Pull Request 303
- Fix recovering "lost" jobs where the job directory does not exist at startup/recovery time (thanks to @natefoo). Pull Request 301
- Use urlencode to encode path (thanks to @mvdbeek). Pull Request 299
- Support the k8s_job_ttl_secs_after_finished option as in the Galaxy Kubernetes runner (thanks to @natefoo). Pull Request 287

15.8 0.14.13 (2021-12-06)

- Don't pass all environment variables to jobs launched by *Manager* (thanks to @nsoranzo). Pull Request 295
- Drop legacy job conf for Galaxy framework tests, test against `metadata_strategy: extended` (thanks to @mvdbeek). Pull Request 294

15.9 0.14.12 (2021-11-10)

- Fixes to bring HOME and temp directory handling closer to Galaxy native runners.
- Enable globbed from_work_dir outputs for remote metadata.

15.10 0.14.11 (2021-07-19)

- Fix and test for returncode handling in certain cases. Pull Request 274
- Modernize tox. Pull Request 271

15.11 0.14.10 (2021-07-17)

- Don't error out if annotated galaxy.json is absent. Pull Request 270

15.12 0.14.9 (2021-07-16)

- Implement dynamic file sources abstraction for parsing files to transfer from `galaxy.json` files. Pull Request 269
- Use tool classes to only test remote Galaxy tools. Pull Request 266
- Run Galaxy framework tests against dev and master branches of Galaxy (thanks to @mvdbeek). Pull Request 264

15.13 0.14.8 (2021-07-14)

- Fix Galaxy composite input references. Pull Request 262
- Run galaxy's tool framework tests against this repo's pulsar (thanks to @mvdbeek). Pull Request 259

15.14 0.14.7 (2021-07-13)

- Accept description of tool files to transfer from Galaxy. Pull Request 261
- Support globs in from_work_dir outputs (thanks to @natefoo). Pull Request 257
- Fix loading the Galaxy dependency resolvers config, plus additional config directory fixes (thanks to @natefoo). Pull Request 256

15.15 0.14.6 (2021-05-24)

- Fix for newer Galaxy tool profiles having isolated home directories.

15.16 0.14.5 (2021-04-15)

- Potential fix for setting file actions via job destination parameters.

15.17 0.14.4 (2021-04-14)

- Re-attempt release process - published wrong branch with 0.14.3.

15.18 0.14.3 (2021-04-13)

- Allow transferring fewer files from Pulsar when using extended metadata with Galaxy.

15.19 0.14.2 (2021-02-15)

- Fix the use of requests, limits, and walltime with coexecution pods. Pull Request 246

15.20 0.14.1 (2021-02-02)

- Fix the use of named managers with coexecution pods. Pull Request 242

15.21 0.14.0 (2020-09-17)

- fix the PyYAML "load() deprecation" warning (thanks to @gmauro). Pull Request 232
- Set the DRMAA workingDirectory to the job's working directory Pull Request 230
- Fix a unicode issue and polish a bit of variables (thanks to @gmauro). Pull Request 229
- Respond to MQ messages requesting status updates. Pull Request 228
- Fix REST connections broken with Py3 using standard transport Issue 227 Pull Request 231
- Drop Python 2.7 support in standard transport, drop Python 2.7 tests and fix Python 3.7 wheel install test, general test debugging enhancements. Pull Request 231
- drop python 2.6 and add 3.7 and update the testing infrastructure to a more recent Ubuntu setup (thanks to @bgruening). Pull Request 226
- Use is_alive in favour of isAlive for Python 3.9 compatibility (thanks to @tirkarthi). Issue 224 Pull Request 225
- Request and register ports for Galaxy ITs when using Kubernetes. Pull Request 223
- Implement killing k8s jobs. Pull Request 221
- Respond to MQ messages requesting status updates. Pull Request 228
- Drop python 2.6 and add 3.7 and update the testing infrastructure to a more recent Ubuntu setup (thanks to @bgruening). Pull Request 226
- Add a more descriptive message in case of error parsing an external id (thanks to @gmauro). Pull Request 213
- Use requests (thanks to @mvdbeek). Pull Request 216
- Use is_alive in favour of isAlive for Python 3.9 compatibility (thanks to @tirkarthi). Pull Request 225
- Debug connection string for AMQP. Pull Request 217
- Various small Kubernetes fixes and enhancements. Pull Request 218, Pull Request 219

- Improvements and fixes to container handling. Pull Request 202
- Fix a typo in exception logging thanks to @erasche. Pull Request 203
- Cleanup config file handling a bit by removing branch for very old Pulsar servers likely no longer supported. Pull Request 201

15.22 0.13.1 (2020-09-16)

- Pinned all listed requirements. This is the final version of Pulsar to support Python 2.

15.23 0.13.0 (2019-06-25)

- Various improvements and simplifications to Kubernetes job execution.

15.24 0.12.1 (2019-06-03)

- Retry botched release that didn't include all relevant commits.

15.25 0.12.0 (2019-06-03)

- Revise Python Galaxy dependencies to use newer style Galaxy decomposition. galaxy-lib can no longer be installed in Pulsar's environment, so you will likely need to rebuild your Pulsar virtualenv for this release. Pull Request 187
- Add a `Dockerfile` for Pulsar with CVMFS (thanks to @nuwang and @*afgane*). Pull Request 166
- Various small improvements to Kubernetes pod execution environment. Pull Request 190
- Improve readme linting. Pull Request 186
- Update example docs for Condor (thanks to @bgruening). Pull Request 189

15.26 0.11.0 (2019-05-16)

- Implement staging Galaxy metadata input files in the client. 39de377
- Fix 'amqp_ack_republish_time' in sample (thanks to @dannon). Pull Request 185
- Updated amqp_url in job_conf_sample_mq_rsync.xml (thanks to @AndreasSko). Pull Request 184
- Use wildcard char for pulsar version (thanks to @VJalili). Pull Request 181
- Refactor toward more structured inputs. f477bc4
- Refactor toward passing objectstore identifying information around. Pull Request 180
- Rework imports for new Galaxy library structure. da086c9
- Revert empty input testing, it really probably should cause a failure to transfer a non-existent file. 8bd5511
- Better client mapper documentation. b6278b4

15.27 0.10.0 (2019-05-06)

- Implement support for Kubernetes two container pod jobs - staging and tool execution as separate containers in the same job's pod. Pull Request 176, Pull Request 178

15.28 0.9.1 (2019-05-01)

- Fix duplicate inputs being a problem when staging Galaxy files. Pull Request 175
- Fix deprecated `assertEquals()` (thanks to @nsoranzo). Pull Request 173
- Fix a method missing problem. Pull Request 174
- Sync "recent" galaxy runner util changes. Pull Request 177

15.29 0.9.0 (2019-04-12)

- Add configuration parameter to limit stream size read from disk. Pull Request 157
- Pass full job status for failed and lost jobs. Pull Request 159
- Improve message handling if problems occur during job setup/staging. Pull Request 160
- Rework preprocessing job state to improve restartability and reduce job loss. **This change should be applied while no jobs are running.** Pull Request 164
- Add support for overriding config through environment variables (thanks to @nuwang). Pull Request 165
- Minor docs updates (thanks to @afgane). Pull Request 170
- Python 3 fixes in Pulsar client (thanks to @mvdbeek). Pull Request 172

15.30 0.8.3 (2018-02-08)

- Create universal wheels to enable Python 3 support when installing from PyPI (thanks to @nsoranzo). Pull Request 156

15.31 0.8.1 (2018-02-08)

- Update link for logo image. Pull Request 145
- Minor error and log message typos (thanks to @blankenberg). Pull Request 146, Pull Request 153
- Fixes/improvements for catching quoted tool files. Pull Request 148
- Fix config sample parsing so run.sh works out of the box. Pull Request 149

15.32 0.8.0 (2017-09-21)

- Support new features in Galaxy job running/scripting so that Pulsar respects `$GALAXY_VIRTUAL_ENV` and `$PRESERVE_GALAXY_ENVIRONMENT`. Fix remote metadata in cases where the tool environment changes the `python` on `$PATH`. Pull Request 137
- Precreate Galaxy tool outputs on the remote before executing (fixes a bug related to missing output files on stage out). Pull Request 141
- Support the remote_transfer file action without setting the `jobs_directory` destination param Pull Request 136
- Fix invalid character in job managers documentation (thanks to @mapa17). Pull Request 130
- Fix `conda_auto_*` option resolution and include a sample `dependency_resolvers_conf.xml` (thanks to @mapa17). Pull Request 132
- Fix tox/Travis tests. Pull Request 138, Pull Request 139, Pull Request 140
- Fix a bug with AMQP acknowledgement. Pull Request 143

15.33 0.7.4 (2017-02-07)

- Fix Conda resolution and add a test case. 11ce744
- Style fixes for updated flake8 libraries. 93ab8a1, 3573341
- Remove unused script. 929bffa
- Fixup README. 629fdea

15.34 0.7.3 (2016-10-31)

- Fix "AttributeError" when submitting a job as a real user. Pull Request 124, Issue 123

15.35 0.7.2 (2016-08-31)

- Fix bug causing loops on in response to preprocessing error conditions.

15.36 0.7.1 (2016-08-29)

- Do a release to circumvent a tool version logic error in Galaxy (released Galaxy versions think 0.7.0 < 0.7.0.dev3).

15.37 0.7.0 (2016-08-26)

- Update Makefile to allow release pulsar as an application and a library for Galaxy at the same time.
- Small update to test scripts for TravisCI changes.
- Improvements for embedded Galaxy runner. (TODO: fill this out)
- Remove support for Python 2.6. 60bf962
- Update docs to describe project goverance and reuse Galaxy's Code of Conduct. 7e23d43, dc47140
- Updated cluster slots detection for SLURM from Galaxy. cadfc5a
- Various changes to allow usage within Galaxy as a library. ce9d4f9
- Various changes to allow embedded Pulsar managers within Galaxy. ce9d4f9, d262323, 8f7c04a
- Introduce a separate working and metadata directory as required for Galaxy 16.04 that requires this separation. 6f4328e
- Improve logging and comments. 38953f3, a985107, ad33cb9
- Add Tox target for Python 2.7 unit testing. d7c524e
- Add `Makefile` command for setup.py develop. fd82d00

15.38 0.6.1 (2015-12-23)

- Tweak release process that left 0.6.0 with an incorrect PyPI description page.

15.39 0.6.0 (2015-12-23)

- Pulsar now depends on the new `galaxy-lib` Python package instead of manually synchronizing Python files across Pulsar and Galaxy.
- Numerous build and testing improvements.
- Fixed a documentation bug in the code (thanks to @erasche). e8814ae
- Remove galaxy.eggs stuff from Pulsar client (thanks to @natefoo). 00197f2
- Add new logo to README (thanks to @martenson). abbba40
- Implement an optional awknowledgement system on top of the message queue system (thanks to @natefoo). Pull Request 82 431088c
- Documentation fixes thanks to @remimarenco. Pull Request 78, Pull Request 80
- Fix project script bug introduced this cycle (thanks to @nsoranzo). 140a069
- Fix config.py on Windows (thanks to @ssorgatem). Pull Request 84
- Add a job manager for XSEDE jobs (thanks to @natefoo). 1017bc5
- Fix pip dependency installation (thanks to @afgane) Pull Request 73

15.40 0.5.0 (2015-05-08)

- Allow cURL downloader to resume transfers during staging in (thanks to @natefoo). 0c61bd9
- Fix to cURL downloaders status code handling (thanks to @natefoo). 86f95ce
- Fix non-wheel installs from PyPI. Issue 72
- Fix mesos imports for newer versions of mesos (thanks to @kellrott). fe3e919
- More, better logging. 2b3942d, fa2b6dc

15.41 0.4.0 (2015-04-20)

- Python 3 support. Pull Request 62
- Fix bug encountered when running `pulsar-main` and `pulsar-config` commands as scripts. 9d43ae0
- Add `pulsar-run` script for issues commands against a Pulsar server (experimental). 3cc7f74

15.42 0.3.0 (2015-04-12)

- Changed the name of project to Pulsar, moved to Github.
- New RESTful web services interface.
- SCP and Rsync file staging options added by E. Rasche. Pull Request
- Allow YAML based configuration.
- Support for more traditional `pip/setup.py`-style installs.
- Dozens of smaller bugfixes and documentation updates.

15.43 0.2.0

- Last version named the LWR - found on BitBucket.
- Still supported in Galaxy as of 15.03 the release.
- Introduced support for submitting to various queueing systems, operation as a Mesos framework, Docker support, and various other advanced deployment options.
- Message queue support.
- Framework for configurable file actions introduced.

15.44 0.1.0

- Simple support for running jobs managed by the Python LWR web process.
- https://bitbucket.org/jmchilton/lwr/branch/0.1

15.45 0.0.1

- See the original announcement and initial commit.

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search