turbo-hipster Documentation

Release 0.1

Joshua Hesketh

October 07, 2015

Contents

Contents:

CHAPTER 1

Turbo-hipster

Turbo-hipster works with the existing OpenStack code review system to implement testing-related plugins. Historically, whenever code has been written for Nova it has been tested against trivial datasets rather than real data. This can mean that when users run the updated code on their databases they can run into issues that were not found during testing. A variety of real-world databases have been collected, anonymized, and added to the database migration plugin used by turbo-hipster. Turbo-hipster is integrated into the existing code review system, and automatically runs tests against these larger test datasets. Turbo-hipster is specifically designed to flag issues where changes to the database schema may not work due to outliers in real datasets, and to identify situations where a migration may take an unreasonable amount of time against a large database.

Note: Database anonymity is important, and can be very time consuming. The databases used by turbo-hipster to test against are real-world databases that have been anonymized with a database anonymization tool called Fuzzy Happiness. Fuzzy Happiness takes markup in the sqlalchemy models file and uses that to decide what values to anonymize, and how to do so. This feature is still in development, and until it is complete turbo-hipster will not report back to Zuul automatically.

Additionally, turbo-hipster has been designed to be extensible, so it is possible to write other plugins to expand its capabilities.

1.1 Turbo-hipster and Zuul

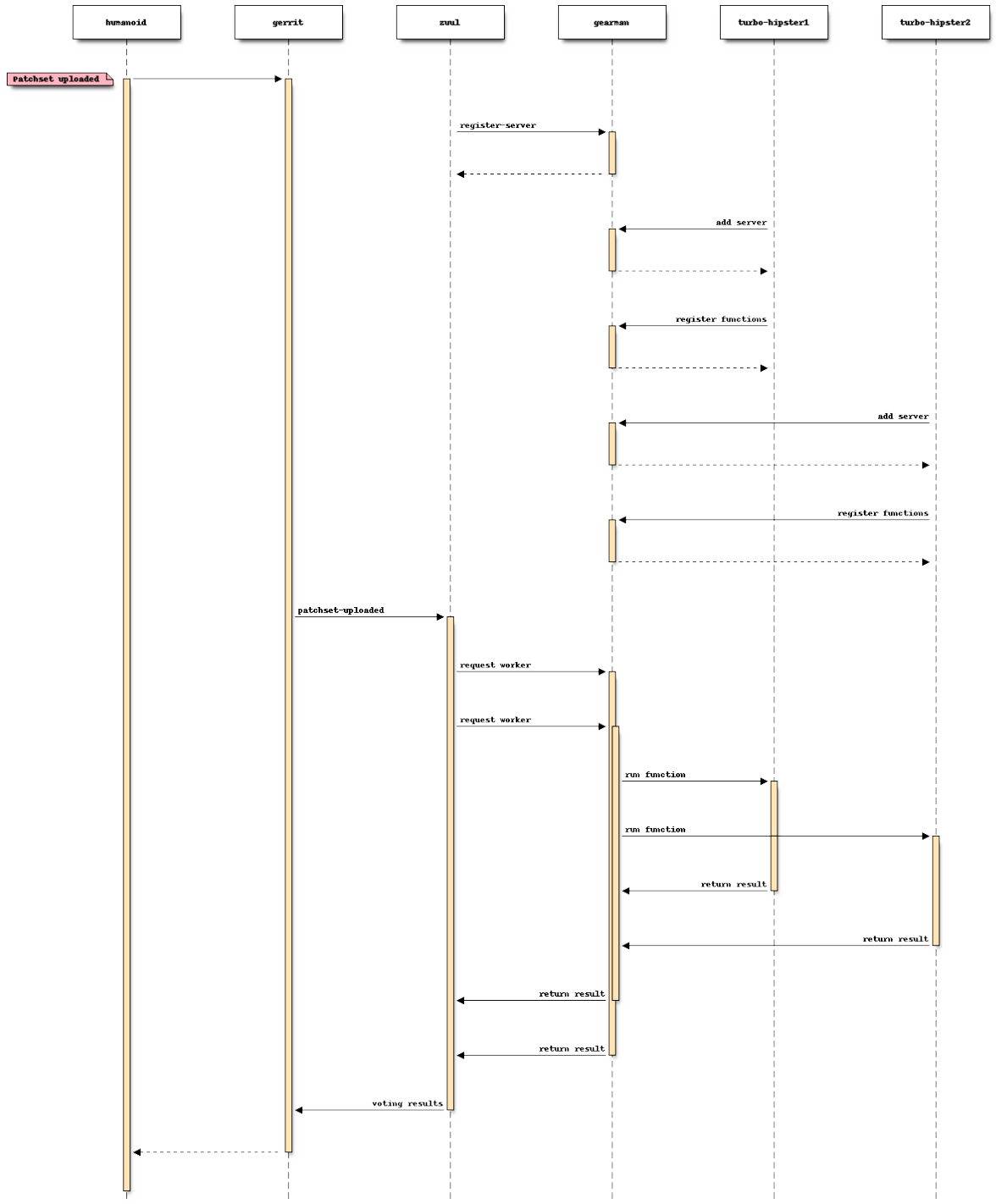
Turbo-hipster is a Gearman worker. Zuul provides arguments that turbo- hipster uses to check out the patch, perform the database testing, and then report back with success or failure. Zuul allows you to specify which jobs should be run against which projects. You can create a rule in Zuul for it to select jobs that require testing against a database. Turbo-hipster will then register as being able to complete that type of job. Gearman handles the connection between Zuul and turbo-hipster, recognizing when a job matches the rule, and passing it to turbo-hipster for testing. When turbo- hipster receives the patchset for the job, it creates a virtual environment to test it. The result of the test is sent back to Gearman as a json string, which contains links to compiled logfiles.

The simplified workflow for turbo-hipster:

1. Registers as a worker against Zuul's Gearman server
2. Receives jobs from Zuul as they arrive
3. Checks out the patchset
4. Sets up a new virtual environment for testing
5. Loads a representative subset of the available datasets
6. Runs the migration against each dataset, and checks the result

7. Reports the results to Zuul, using the Gearman protocol

1.2 Typical workflow diagram



CHAPTER 2

Installation

Turbo-hipster is installed directly into your Python `site-packages` directory, and is then run as a service. It is managed using a configuration file, which is in yaml format.

2.1 Installing turbo-hipster

1. Turbo-hipster can be installed directly to your Python `site-packages` directory:

    $ sudo python setup.py install

2. Copy the configuration file to a convenient location. By default, turbo-hipster will look in `/etc/turbo-hipster/config.yaml`:

    $ cp -R etc/turbo-hipster /etc/

3. The turbo-hipster configuration file is in yaml format. Open the `config.yaml` configuration file in your preferred editor and modify it for your environment:

```
**zuul_server**
  A dictionary containing details about how to communicate
  with zuul
     **git_url**
         The publicly accessible protocol and URI from where
         to clone projects and zuul_ references from. For
         example::
             http://review.openstack.org/p/
         or::
             git://review.example.org
     **gearman_host**
         The host of gearman_. zuul talks to its workers via
         the gearman protocol and while it comes with a built-
         in gearman server you can use a separate one.
     **gearman_port**
         The port that gearman listens on.
**debug_log**
  A path to the debug log. Turbo-hipster will attempt to create
  the file but must have write permissions.
**jobs_working_dir**
  Each job will likely need to write out log and debug
  information. This defines where turbo-hipster will do that.
**git_working_dir**
  turbo-hipster needs to take a copy of the git tree of a
  project to work from. This is the path it'll clone into and
```

```
  work from (if needed).
**pip_download_cache**
  Some of turbo-hipsters task plugins download requirements
  for projects. This is the cache directory used by pip.
**jobs**
  A list of registered jobs.
    **name**
      The name of the job to register. This is the function name
      for zuul's job. eg build:some_job.
    **plugin** (optional)
      The plugin to use. Defaults to shell_task.
  Any other variables the plugin may require for the job.
**plugins** (depreciated)
  This is depreciated in favour of jobs (above).
  A list of enabled plugins and their settings in a dictionary.
  The only required parameters are *name*, which should be the
  same as the folder containing the plugin module, and
  *function*, which is the function registered with zuul.
  Any other parameters are specified by the plugin themselves
  as required.
**publish_logs**
  Log results from plugins can be published using multiple
  methods. Currently only a local copy is fully implemented.
     **type**
         The type of protocol to copy the log to. eg 'local'
     **path**
         A type specific parameter defining the local location
         destination.
     **prepend_url**
         What to prepend to the path when sending the result
         URL back to zuul. This can be useful as you may want
         to use a script to authenticate against a swift
         account or to use *laughing_spice* to format the logs
         etc.
**conf_d**
  A path of a directory containing pieces of json confiuration.
  This is helpful when you want different plugins to add extra
  or even modify the default configuration.
```

4. Create a turbo-hipster user:

   $ useradd turbo-hipster

5. Create the directories listed in the configuration file, and give the `turbo-hipster` user write access:

   $ mkdir -p /var/log/turbo-hipster/ $ chown turbo-hipster:turbo-hipster /var/log/turbo-hipster/

   $ mkdir -p /var/lib/turbo-hipster/jobs $ chown turbo-hipster:turbo-hipster /var/lib/turbo-hipster/jobs

   $ mkdir -p /var/lib/turbo-hipster/git $ chown turbo-hipster:turbo-hipster /var/lib/turbo-hipster/git

   $ mkdir -p /var/cache/pip $ chown turbo-hipster:turbo-hipster /var/cache/pip

6. Open the MySQL log rotation configuration file in your preferred text editor, and edit it to ensure it is writable by `other`:

   $ vim /etc/logrotate.d/mysql-server # edit create 640 to 644.

Note: The turbo-hipster source code is also available for download from the turbo-hipster github page

$ git clone https://github.com/rcbau/turbo-hipster

Note: Debug logging must be configured for turbo-hipster, as it uses the Python logging framework to capture log messages from the task plugin code. To configure debug logging, set the `debug_log` configuration setting in the `config.yaml` configuration file.

CHAPTER 3

Running

3.1 Starting turbo-hipster

Turbo-hipster can be run from the command line:

```
$ turbo-hipster
```

This option allows you to pass parameters to turbo-hipster. Use the –help parameter to see a full list.

Short	Long	Description
-c	–config	Print the path to the configuration file and exit
-b	–background	Run as a daemon in the background
-p	–pidfile	Specify the PID file to lock while running as a daemon

Alternatively, you can start turbo-hipster as a service.

1. Copy the turbo-hipster init.d script to /etc/init.d/:

   $ sudo cp etc/init.d/turbo-hipster /etc/init.d/

2. Reload the script with the default configuration:

   $ sudo update-rc.d turbo-hipster defaults

3. Start the service:

   $ sudo service turbo-hipster start

CHAPTER 4

Plugins

Plugins can be used to extend turbo-hipster's capabilities.

Note: Currently, the only available plugin for turbo-hipster is the database migration plugin, `real_db_upgrade`, which tests code against a variety of real-world databases.

4.1 Installing plugins

Turbo-hipster plugins are responsible for handling the jobs that are passed to it. They must successfully build reports and publish them according to their configuration. They must also be able to communicate test results back to Zuul using Gearman.

Plugins must take a standard format in order to be able to work correctly with turbo-hipster. They must contain a `task.py` file with a `Runner` class.

Once you have created a turbo-hipster plugin, you need to configure it in the `config.yaml` configuration file.

4.2 Plugin: Database migration with `real_db_upgrade`

The database migration plugin, `real_db_upgrade`, is used to test datasets against real-world, anonymized, databases.

4.3 Migrating a database

In order to use turbo-hipster with the `real_db_upgrade` plugin, you need to set up the databases to test against, and point to the plugin in turbo-hipster's configuration file.

1. Create a directory for the datasets:

   $ mkdir -p /var/lib/turbo-hipster/datasets

2. Copy the json dataset to the directory you created:

   $ cp /my/dataset.json /var/lib/turbo-hipster/datasets/

3. Open the `/etc/turbo-hipster/config.yaml` file in your preferred editor, locate the plugins section, and add this line:

```
**plugins**
 real_db_upgrade
```

CHAPTER 5

Testing with turbo-hipster

When turbo-hipster completes a test, it sends the result of the test back to Gearman. These results contain a link to a compiled log file for the test.

If the test fails, or takes too long to complete, turbo-hipster will add a review to your patchset that looks like this:

```
gate-real-db-upgrade_nova_mysql_devstack_131007 SUCCESS in 4m 12s
gate-real-db-upgrade_nova_mysql_devstack_150 SUCCESS in 4m 17s
gate-real-db-upgrade_nova_mysql_user_001  SUCCESS in 14m 52s
gate-real-db-upgrade_nova_percona_devstack_131007 SUCCESS in 4m 20s
gate-real-db-upgrade_nova_percona_devstack_150 SUCCESS in 4m 11s
gate-real-db-upgrade_nova_percona_user_001 SUCCESS in 14m 22s
gate-real-db-upgrade_nova_mysql_user_002  FAILURE - Did not find the end of a migration after a start in 4m 16s (non-voting)
gate-real-db-upgrade_nova_percona_user_002 FAILURE - Did not find the end of a migration after a start in 4m 47s (non-voting)
```

5.1 Reading test reports

An example of a standard log file: http://thw01.rcbops.com/results/54/54202/5/check/gate-real-db-upgrade_nova_mysql_devstack_150/ddd6d53/20130910_devstack_applied_to_150.log

An example of the same logfile, using the javascript logviewer: http://thw01.rcbops.com/logviewer/?q=/results/54/54202/5/check/gate-real-db-upgrade_nova_mysql_devstack_150/ddd6d53/20130910_devstack_applied_to_150.log

5.2 Test failure codes

This section gives a list of failure codes, including some steps you can take for troubleshooting errors:

FAILURE - Did not find the end of a migration after a start

If you look at the log you should find that a migration began but never finished. Hopefully there'll be a traceroute for you to follow through to get some hints about why it failed.

WARNING - Migration %s took too long

In this case your migration took a long time to run against one of the test datasets. You should reconsider what operations your migration is performing and see if there are any optimizations you can make, or if it is really necessary. If there is no way to speed up your migration you can email us at rcbau@rcbops.com for an exception.

FAILURE - Final schema version does not match expectation

Somewhere along the line the migrations stopped and did not reach the expected version. Our datasets start at previous releases and have to upgrade all the way through to the most current release. If you see this, inspect the log for traceroutes or other hints about the failure.

FAILURE - Could not setup seed database. FAILURE - Could not find seed database.

These errors are internal errors. If you see either of these, contact us at rcbau@rcbops.com to let us know so we can fix and rerun the tests for you.

FAILURE - Could not import required module.

This error probably shouldn't happen as Jenkins should catch it in the unit tests before Turbo-Hipster launches. If you see this, please contact us at rcbau@rcbops.com and let us know.

If you receive an error that you think is a false positive, leave a comment on the review with the sole contents of "recheck migrations".

If you have any questions/problems please contact us at rcbau@rcbops.com.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search