Code Review for System Administrators

ELIZABETH KRUMBACH JOSEPH

Elizabeth Krumbach Joseph is an Automation and Tools Engineer at HP working on the OpenStack Infrastructure team. She is also a member of the Ubuntu Community Council, one of the two governing bodies of the Ubuntu Project, and on the Board of Directors for Partimus.org, a non-profit in the San Francisco Bay area providing Linux-based computers to schools in need.
lyz@princessleia.com

Over the past year, OpenStack [1] has seen fast-paced adoption by major vendors and a staggering growth in its developer community from companies around the world. As a result, the relatively new cloud platform has become quite the popular topic throughout the open source community. The infrastructure used to manage the growth of the project over this time has had to handle this load, while still managing to review and test the code going into the project. In this article, I'll look at how code review practices for OpenStack can be applied to system administration.

The infrastructure [2][3] for the OpenStack project is fully open source and managed by a geographically distributed team of systems administrators from multiple companies who are responsible for the installation and maintenance of the tools used by the OpenStack project. This infrastructure includes the full code review and continuous integration system, plus wiki, pastebin, etherpad, chat bots, and other tools used by project members on a day to day basis. A major boon to the ability for all of us to collaborate effectively is by not only imposing review upon code going into OpenStack itself, but also to all the configuration files and code that we deploy in production for the infrastructure.

The code review and continuous integration system used in OpenStack is built with the needs of OpenStack in mind. OpenStack is essentially a big project made up of many smaller projects that operate largely independently within the OpenStack umbrella. Each project has different team leads, core contributors, and reviewers. As such, the system needed extensive integration testing as part of what is tested before code is committed to the git repositories. We're using Gerrit [4] for the front end code review mechanism and Jenkins [5] as our continuous integration server that launches tests. To glue these together, we use a test worker distributor, Gearman [6], to hand things off to our Jenkins servers. We use a couple other tools to manage the queuing of testing and merging of code (Zuul [7]), and to manage the pool of machines used to do tests (Nodepool [8]), both of which are open source and were developed by the infrastructure team.

Systems administration does not typically need the same kinds of tests that fully integrated code within OpenStack does. Instead, sysadmins on my team use this same system, but we have set up a series of checks to run against all changes to our config files and other code that the team checks in, including:

- flake8 (running pep8 and pyflakes) for our Python scripts
- puppet parser validate & puppet-lint against any changes in Puppet
- XML syntax checking on some XML files where the structure is known

These checks are run as soon as we submit a code or configuration change into the code review system, giving the system administrator feedback within a few minutes about whether their code has passed these basic tests. We regularly assess this list, improve it

References

[1] OpenStack: http://www.openstack.org/

[2] OpenStack infrastructure: http://ci.openstack.org/

[3] OpenStack infrastructure code repository: https://git.openstack.org/cgit/openstack-infra

[4] Gerrit: https://code.google.com/p/gerrit/

[5] Jenkins: http://jenkins-ci.org/

[6] Gearman: http://gearman.org/

[7] Zuul: http://ci.openstack.org/zuul/

[8] Nodepool: http://ci.openstack.org/nodepool.html

and add more when we expand the types of code or configuration files we are submitting so that we get as much benefit as possible from automated tests.

Next the changes are reviewed by contributors to the infrastructure team. As an open source project team, we allow anyone to do basic code reviews, but restrict the higher levels of approval to core project members who have a history of being trustworthy and providing a high level of code review expertise.

This human element of the system is perhaps the most valuable part of having a code review process for system administrators. The process provides an opportunity to have multiple eyes on even simple changes before submitting something that could possibly impact hundreds of active developers from dozens of different companies. A single system administrator is no longer responsible for applying a change. Instead, a team collaboratively reviews and approves it. This collaboration can make for a lower stress, higher reward work environment.

As a distributed team, our review process gives us a great platform for checking in a “Work in Progress” change that we can actively collaborate on by commenting on the changes inside the code review system. Also, team members have an opportunity to see the solution that one of us came up with, and make suggestions for tackling the issue in an entirely different way now that we’ve seen how the proposed suggestion may work. By having the proposed changes in front of us, we’re also able to test code independently of the submitter before giving our approval, which often catches edge cases that are found in our varied personal test environments.

Because all of our changes go through code review, including those from core contributors, there is little technical difference between a submission provided by a new contributor or someone who has been with the project for a long time. Every review is handled independently, and we have the same social requirements for accepting a change (two core reviewers should give their approval). This also means that no one has the ability to commit directly to the code repository. Changes by core members, like those from anyone else, must pass syntax checking and get reviews.

Once the code lands in our git repository, the Puppet master picks it up and it is deployed automatically to the appropriate servers. If the change is to code being run on a server, we use a Puppet mechanism for handling code repositories that regularly checks for updates in the repository and restarts services on changes as needed.

All of these processes have trained members of the team to be collaborative by default, an important thing for a distributed team or one that tends to have different systems administrators focusing on different projects. This has helped tremendously when the limitations for managing systems completely through code review and code repositories are considered.

The first obvious limitation is bootstrapping this process. To get the process going, you need basic servers set up, and when adding new servers to the infrastructure from our pool of OpenStack-based virtual machines, there are still portions we have not been able to automate completely. Also, there is the handling of passwords, SSL keys, and other sensitive data that cannot be made available generally to every member of the project. These limitations are both handled by having a “root” team that has access to creating new servers and to adding, viewing, and manipulating the sensitive data.

We’ve also had to handle the inevitable problem that comes up where you simply must log into a server for some reason. Perhaps a MySQL database needs a manual edit, or the Puppet agent running on the server has crashed. We may also need to debug something, so shell access to browse logs and run diagnostic tools is essential. To handle this, root team members have access to all the servers, and then access is granted on a server by server basis to team members with expertise in working on specific applications in the infrastructure. We also run public monitoring of our servers via Cacti and have a Puppet dashboard so basic statistics about system resources and the application of changes via Puppet can be tracked and reviewed by any contributor.

Complicated upgrades or migrations also are difficult to manage through a code review system. In these cases, shell access often is required, but our collaborative culture makes it so that we’re always working together on these projects. Typically, resolving these situations starts off in an online team meeting in which we flesh out the migration plan in a collaborative editing tool (such as etherpad), then we schedule the maintenance window when multiple team members will be available. Once we get to the maintenance time, we work together on Internet Relay Chat (IRC) to run through the list of tasks defined in the etherpad and work together if anything goes awry.

There are also the inevitable emergencies that crop up from time to time. In these cases, the root admins do have the ability to log in and shut down the Puppet agent and make changes manually to unblock us. The team has been disciplined so that these incidents are rare, as we’d much rather fix it via a commit, and followed up with as soon as the emergency has passed and more core members are available to assess and permanently solve the problem.

Fortunately these limitations are a small percentage of what we encounter, and our primary contact with our systems on a day to day basis is through the code review system. In addition to public Cacti and Puppet dashboard that any contributor can access, we also are diligent about maintaining our team documentation for how we run and make changes to our various services and make sure all our configurations and scripts are browseable by contributors in our public git repository. This makes it relatively easy for new contributors to join our team and get up to speed with our full infrastructure, or simply make a single change to address a pain point in our infrastructure, from adding a new test to the continuous integration system to adding a favicon to the project status page. Documentation also allows our existing system administrators to focus on their core skills and slowly get up to speed with other portions of the infrastructure by reading documentation, doing reviews, and watching other team members work.