Proposal for Modernization of KDE Infrastructure

Jan Kundrát

January 27, 2015

Executive Summary

This paper proposes a few ideas for modernizing the infrastructure that is provided to KDE developers. At the core is an idea for adopting Gerrit, an advanced code-review and Git hosting platform. Gerrit has been under evaluation by a couple of KDE projects since September 2014. We have received very positive feedback from people who participate in Gerrit evaluation, and now we have a stable, production setup connected to our Continuous Integration. This means that developers know whether a patch which they are reviewing introduces regressions, and whether their project will still build on an ancient machine with Qt 4.6 and GCC 4.4.

This proposal goes far beyond just saying "use Gerrit". We outline how to integrate it with other tools, and how to make sure that the end result is a featureful infrastructure which is easy to maintain for years to come.

Another part of this proposal is opening up of the process of infrastructure maintenance. By treating service configuration as code, we are able to free up sysadmin resources and leverage help from community. Because we are embracing code review and automated Continuous Integration for sysadmin tasks as well, the help from community can be accepted without excessive risks.

The paper discusses high-level motivation, provides an overview of the proposed features, and wraps up with a detailed discussion explaining how to implement our suggestions.

1 Pain Points of the Current Setup

We are living in a world where GitHub has influenced fellow developers' expectations. People are used to sending pull requests around. Day-to-day changes are implemented through self-service portals, with no human intervention. If a task can be automated, it better be automated *well*. There is little point in wasting valuable human on tasks which can be handled by machines.

The current set of tools which KDE offers is a bit antiquated. Reviewboard is not integrated with the Continuous Integration (CI) infrastructure or bug tracking. Uploading patches and testing changes from Reviewboard requires developers to take many manual steps. These are prone to errors and constitute extra work for everybody. Users have no idea that a patch exists unless they actively follow multiple systems.

Current project hosting is based on Redmine/ChilliProject which has reached its breaking point as well. The anongit setup is held together by duct tape and sysadmins' much appreciated work. Events which propagate instantly on competing, proprietary code hosting platform might take an hour on our infrastructure. Some of the software which we use is no longer maintained. This constitutes a problem for our developers. People expect an excellent service, but our current tools cannot deliver that.

Readers are encouraged to read sysadmins' excellent summary of these problems to gain further insight into the list of current issues.

2 Why Gerrit?

The proposal centers around using Gerrit for primary Git repository hosting and code review. Gerrit is a mature, yet actively developed tool. Originating at Google, it has since been adopted by a number of large open source projects, including the Qt Project, LibreOffice, Android, Eclipse, OpenStack, WikiMedia, Tizen, CyanogenMod and many others. Gerrit is regularly spoken about at conferences and developer meetups. It receives substantial commercial backing and is the code-review platform of choice for corporations and small businesses alike, including Google, eBay, SAP, Sony Ericsson, KitWare and others. It is also available as a hosted service from an independent vendors on a commercial basis. It has a friendly upstream and a responsive community that provides advice when needed. Patches are accepted with little bikeshedding.

Gerrit itself focuses on doing one thing, and doing it well. As such, it provides an integrated platform for code review which is tied with a Git server. This puts Gerrit into a unique position where pushing changes for a review is just a matter of `git push`, and checking what others have created a matter of `git pull`. Integration with third party tools is implemented via a rich set of stable APIs and via plugins.

The proposal is based on an existing infrastructure of the OpenStack project. With tens of thousands of commits per month, hundreds of repositories and thousands of committers, a solution which scales for OpenStack is guaranteed to handle KDE's scale as well. We can also leverage the work of others and to benefit from problems which other large-scale projects have solved already.

Gerrit is currently used for code review and CI at the Qt Project. Because many KDE developers are also Qt contributors, familiarity with Gerrit helps people move between these two high-profile and related projects without much effort.

3 Technical Solution

The rest of the text deals with technical aspects of providing service for KDE developers. One of the focus points is discussion on how to integrate various services together. In several cases, there are multiple approaches which will all lead to a success. We have tried to provide a balanced opinion on how to proceed based on the past experience, real-world feedback and field testing of our Gerrit deployment, as well as on experience of other large-scale FLOSS projects. Nonetheless, chances are that the proposed ideas are not the best option out there. In that case, we are looking forward to hearing about what can be improved.

3.1 Identity Management

KDE already has an identity service based on LDAP. It manages a list of users along with their roles within KDE. A custom web application used for self-service account management is deployed on KDE infrastructure. These services will be preserved without modifications. Should a need to change them arise in future, it will be easy to e.g. deploy another web frontend without affecting the rest of our infrastructure.

The choice of LDAP is an excellent technical decision. LDAP is an industry-standard for identity management, and it is rare to find a mature application which needs user accounts, yet cannot integrate with LDAP these days.

In future, this setup might well be combined with a native Single-Sign-On (SSO) service. That way, users not only have just a single password to use, but they also will not have to enter that password to multiple services or web pages all over again.

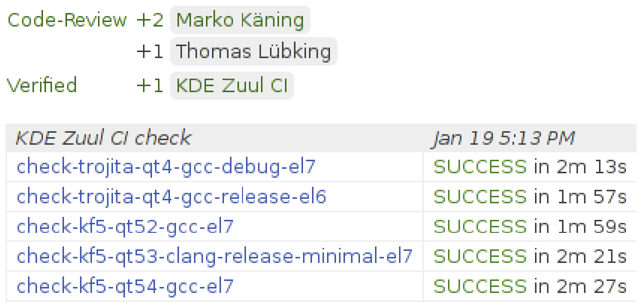
3.2 Git Repository Hosting

Gerrit will act as a primary repository host. This will be completely transparent to the users. Developers who do not want to change their workflow will witness no user-visible changes. All existing clones will work, and developers will be able to direct push and bypass code review if they so choose.

There will be no need for mandatory code review on a KDE-wide basis. Projects will be free to opt-in for mandatory reviews and/or CI coverage with a potential for manual override. With an exception of infrastructure projects which directly correspond to code running at our servers or root access, all KDE developers will have the same level of access.

3.2.1 Code Review

One of Gerrit's strongest points is its support for fine-grained code reviews. KDE developers can approve changes, while general public is free to offer their comments. A difference between a bystander's review ("this looks good, but I don't really know the details") and a maintainer's review ("yes, this is it") is clearly indicated.



Code-Review	+2	Marko Käning
	+1	Thomas Lübking
Verified	+1	KDE Zuul CI

KDE Zuul CI check	Jan 19 5:13 PM
check-trojita-qt4-gcc-debug-el7	SUCCESS in 2m 13s
check-trojita-qt4-gcc-release-el6	SUCCESS in 1m 57s
check-kf5-qt52-gcc-el7	SUCCESS in 1m 59s
check-kf5-qt53-clang-release-minimal-el7	SUCCESS in 2m 21s
check-kf5-qt54-gcc-el7	SUCCESS in 2m 27s

The reviews can tackle each change on various levels; there is support for commenting on the change as a whole, on particular lines or ranges of characters, and adding file-level comments as well. Individual reviews can be replied to. An SSH command for conducting code reviews is available as well.

Support for adding image attachments is available in upstream's Git repository, and will be deployed when we upgrade to the next release.

Maintainers can push changes into the Git repository straight from the code review UI. This eliminates the current source of pain where a KDE developer gives his "Ship it!" on Reviewboard, and then nothing happens because the patch author has moved on, or doesn't have a Git account at all.

There are two different web interfaces, and all functionality is made available via REST APIs. A console, TUI application called Gertty[1] written in Python is available which is equal, feature-wise, with the web interface. Extending other applications, such as KDevelop or the Qt Creator, is possible – a good opportunity for this will be the Google's Summer of Code project. As a proof of concept, Eclipse is an example of an IDE which already integrates with Gerrit and Bugzilla.[2]

Third-party tools integrate with the code review process either via SSH, or through the REST APIs. The bundled web UIs are implemented as a client-side JavaScript application that talk to Gerrit via the REST APIs. There is no artificial feature gap between what can be done with official tools and what is available to other UIs. Alternative web UIs using various modern web frameworks are under development.

The recommended way of pushing changes for review is via a magic Git branch, `refs/for/master`. A development version of Gerrit contains features for interactively editing files and proposing changes over the web interface.[3] A complementary application for uploading patches via web is also available.[4]

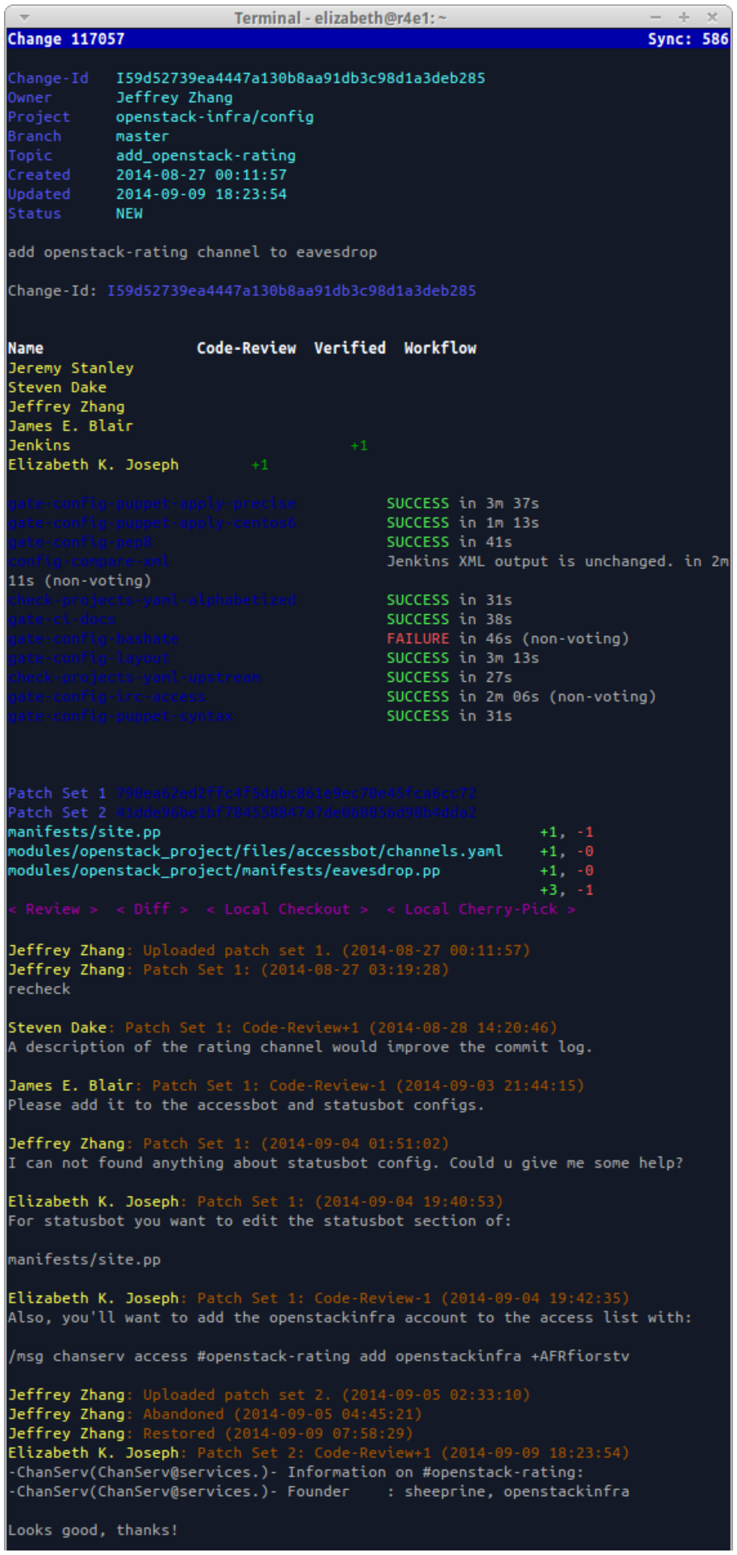
Each version of each proposed change is available for download as a Git ref, a unified diff, or a Zip/TGZ/... archive. A regular Git clone does *not* download these changes by default. Full power

[1] Additional screenshots and more information are at `http://princessleia.com/journal/?p=9785`.

[2] An impressive tutorial showing how to work with Bugzilla, Gerrit and Jenkins from within an IDE is available at `http://www.eclipse.org/community/eclipse_newsletter/2013/september/article4.php`.

[3] See `https://gerrit-review.googlesource.com/Documentation/user-inline-edit.html` for an overview and screenshots.

[4] See WikiMedia's `https://tools.wmflabs.org/gerrit-patch-uploader/`.



```
Change 117057                                                             Sync: 586

Change-Id   I59d52739ea4447a130b8aa91db3c98d1a3deb285
Owner       Jeffrey Zhang
Project     openstack-infra/config
Branch      master
Topic       add_openstack-rating
Created     2014-08-27 00:11:57
Updated     2014-09-09 18:23:54
Status      NEW

add openstack-rating channel to eavesdrop

Change-Id: I59d52739ea4447a130b8aa91db3c98d1a3deb285


Name                 Code-Review  Verified  Workflow
Jeremy Stanley
Steven Dake
Jeffrey Zhang
James E. Blair
Jenkins                                +1
Elizabeth K. Joseph        +1

gate-config-puppet-apply-precise          SUCCESS in 3m 37s
gate-config-puppet-apply-centos6          SUCCESS in 1m 13s
gate-config-pep8                          SUCCESS in 41s
config-compare-xml                        Jenkins XML output is unchanged. in 2m
11s (non-voting)
check-projects-yaml-alphabetized          SUCCESS in 31s
gate-ci-docs                              SUCCESS in 38s
gate-config-bashate                       FAILURE in 46s (non-voting)
gate-config-layout                        SUCCESS in 3m 13s
check-projects-yaml-upstream              SUCCESS in 27s
gate-config-irc-access                    SUCCESS in 2m 06s (non-voting)
gate-config-puppet-syntax                 SUCCESS in 31s



Patch Set 1 798ea62ed2ffc4f5dabc861e9ec70e45fca6cc72
Patch Set 2 41dde96be1bf784558847a7de060856d98b4dda2
manifests/site.pp                                          +1, -1
modules/openstack_project/files/accessbot/channels.yaml    +1, -0
modules/openstack_project/manifests/eavesdrop.pp           +1, -0
                                                           +3, -1
< Review >  < Diff >  < Local Checkout >  < Local Cherry-Pick >

Jeffrey Zhang: Uploaded patch set 1. (2014-08-27 00:11:57)
Jeffrey Zhang: Patch Set 1: (2014-08-27 03:19:28)
recheck

Steven Dake: Patch Set 1: Code-Review+1 (2014-08-28 14:20:46)
A description of the rating channel would improve the commit log.

James E. Blair: Patch Set 1: Code-Review-1 (2014-09-03 21:44:15)
Please add it to the accessbot and statusbot configs.

Jeffrey Zhang: Patch Set 1: (2014-09-04 01:51:02)
I can not found anything about statusbot config. Could u give me some help?

Elizabeth K. Joseph: Patch Set 1: (2014-09-04 19:40:53)
For statusbot you want to edit the statusbot section of:

manifests/site.pp

Elizabeth K. Joseph: Patch Set 1: Code-Review-1 (2014-09-04 19:42:35)
Also, you'll want to add the openstackinfra account to the access list with:

/msg chanserv access #openstack-rating add openstackinfra +AFRfiorstv

Jeffrey Zhang: Uploaded patch set 2. (2014-09-05 02:33:10)
Jeffrey Zhang: Abandoned (2014-09-05 04:45:21)
Jeffrey Zhang: Restored (2014-09-09 07:58:29)
Elizabeth K. Joseph: Patch Set 2: Code-Review+1 (2014-09-09 18:23:54)
-ChanServ(ChanServ@services.)- Information on #openstack-rating:
-ChanServ(ChanServ@services.)- Founder    : sheeprine, openstackinfra

Looks good, thanks!
```

Reviewing a change with Gertty, a third-party TUI.
(The color scheme is configurable.)

of Git is available to assess differences between versions of each change. There is no need to learn another VCS system, as all functionality is made available via pure Git.

Gerrit can be configured to automatically assign reviewers based on who has recently authored lines which are touched by a patch. This settings is controlled on a per-project basis, and there is an upper limit on how many people can be added that way to prevent excessive spam.

3.2.2 Conflicting Changes

When a project is big enough, sooner or later there will be patches laying around which are mutually incompatible. By default, Gerrit uses merge algorithm that solves conflicts on a file level. A list of changes which modify the same files is shown within the UI. Changes which cannot be submitted due to a merge failure are clearly marked as needing a rebase in all UIs. That way, a developer can make sure that a conflict is solved in a meaningful way and without introducing bugs.

3.2.3 Scratch Repositories

The functionality of scratch repositories will be preserved. A trivial script available via SSH and/or HTTPS will allow users to create and manage personal repositories under `scratch/username/`. This is made possible due to Gerrit's rich set of APIs:

```
if not check_ldap_group($user, "developers"):
  die "you are not a KDE developer"
if not regexp_match($proj, '[-a-zA-Z0-9_]'):
  die "invalid project name"
if operation == "delete":
  return exec "ssh bot@gerrit delete --yes-really-delete --force scratch/$user/$
      proj"
if operation != "create":
  die "invalid operation"
return exec "ssh bot@gerrit create-project --owner $user --parent sysadmin/gerrit-
    user-scratch-projects scratch/$user/$proj"
```

If a need arises, it is possible to offer this functionality via a plugin and to integrate it into Gerrit's web UIs and SSH and REST APIs as well.

3.2.4 Personal Clones

KDE developers who wish to share their work which is not quite ready for merging with the world are encouraged to upload it to Gerrit as a change *not for review*. That way, there is no need to create and clone another repository. All clones are equally visible on a single place. It is possible to distinguish between changes intended for review, and these not-intended-for-submission changes. Dashboards are provided for browsing changes in each category separately. Regular Git users will not be cloning these personal-work branches.

Another, complementary solution for clones are per-user branches. These are perfectly supported in Gerrit, and can be moved away to a hidden namespace if it is not desired to "pollute" everybody's clone by these refs.

3.2.5 Repository Browsing

Gerrit links with an arbitrary WWW Git frontend. The proof-of-concept deployment which has been running in KDE uses KDE's QuickGit for repository and commit browsing. A native deployment will likely use a dedicated VM with cGit. An alternative is to reuse KDE's existing GitPHP frontend to minimize the amount of required changes.

Pushes will be replicated in real time via Gerrit's native replication. There are no technical limitations on force pushes, and no problems will arise if projects decide to allow them. Support for mirroring to GitHub can be provided out-of-box if the community wishes so.[5]

[5]There are production-ready solutions for automatically managing GitHub repositories, including automated

3.2.6 Post-Commit Review

Finished changes can be commented on, and a prominent "Revert" button for proposing a revert for code review is available. The comments are sent to the right people, and the author of the troublesome patch is Cc-ed on the revert request.

A new feature of the (not yet released) Gerrit 2.11 is support for proposing changes right from the web interface. We do not have experience with this functionality yet, but we fully expect that it can be used as a basis for adding repository auditing later on.

3.2.7 Anongit – Replicating for Read-Only Git Mirroring

Gerrit has a built-in, native plugin for smart replication. The replication is congestion-aware, and can create and/or delete remote repositories on each replica. People are using this over the globe, with mirrors on different continents and over unreliable WAN links. The replication is push-based with an event queue with each replica being synced as fast as it can. If a replica falls behind, attempts are transparently retried and merges are combined together for efficiency.

3.2.8 User Management

Gerrit supports LDAP authentication, so it will simply talk to KDE's existing LDAP. Information about developer status is obtained from LDAP as well. If we were to introduce more fine-grained access control to some repositories, LDAP groups would be used.

SSH keys cannot be looked up from LDAP at this time. This is a similar situation to the current setup which requires a script to synchronize keys due to OpenSSH and Gitolite limitations.

One option is to let users manage their SSH keys right within Gerrit. This is possible because except for translators, no other uses of SSH keys are needed when SVN is deprecated. A better option is keeping the SSH keys in LDAP and introducing a simple script which deletes keys that are no longer desired and adds new ones into Gerrit. Because Gerrit has an API for managing user's SSH keys via a service account, the script is going to be reasonably simple and straightforward. The script can be triggered in response to LDAP's syncrepl events.

A truly hackish approach which would nonetheless get the job done is:

```
exec "ssh bot@gerrit set-account $user --remove-ssh-keys ALL"
exec "ssh bot@gerrit set-account $user --add-ssh-key - < authorized_keys"
```

A more elegant tool should inspect the list of SSH keys and only make changes which are actually necessary. This will be written by `jkt`.

3.2.9 E-mail Notifications

Gerrit can send e-mails about various aspects of changes, including events about new changes, added comments, reviews, and changes hitting the Git tree. Users can pick what projects are watched, and can set up fine-grained filters, including conditions for branches, file paths and authors.[6] This way, users can opt-in to receive notifications about projects, branches or files they care about. There is no support for e-mailing info about direct pushes at this time. This will be developed by `jkt` and proposed for upstream integration.

A backup alternative is reusing KDE's existing hooks and the commitfilter. However, moving all settings into a single place is of course a better solution.

3.2.10 IRC Notifications

In the current setup, this feature is implemented by sending e-mails from a post-commit hook to a machine mailbox. A bot is watching the recipient address, and acts on the incoming e-mails by forwarding them to an IRC bot, which in turn sends them to appropriate channels.

closing of pull requests with a friendly note asking users to contribute via project's official channel.

[6]Check `https://gerrit.vesnicky.cesnet.cz/r/Documentation/user-search.html` for a list of supported search terms and expressions.

We propose to use Gerrit's events and an existing GerritBot project. It is already providing notifications about Gerrit changes at `#kde-gerrit` and `#trojita`. Configuration of this bot and channel-to-project mapping will be handled in the same was as any other project metadata. Because the data will be stored in a Git repository, users will be able to propose changes to them and their validity will be checked by Continuous Integration and the regular code review process.

3.2.11 Bugzilla Integration

There is much more to bug tracker integration than just sending e-mails. Typically, a bug tracker and a code review system focus on different groups of users and have different UIs to reflect this difference. It is common to have a discussion starting at, say, Bugzilla where we try to reproduce a bug. When a patch is written, we have to check not only whether it fixes a bug, but we have to also make sure that the patch in question fits the project, and whether it fixes the problem *correctly*.

A patch under review should be *visible* from the bug tracker so that interested parties or power users know that some work is going on. This might be combined with automatically modifying a bug's status so that it's clear that it is not an unconfirmed report anymore, but rather something which is worked on. When a patch gets merged or when it is decided that it is not actually a good fit for the problem at hand, this should again be made obvious in the issue tracker's UI. The amount of notifications should be kept to a reasonable state. The users in particular typically have no intention in following developer's discussion about coding style.

In a similar manner, the patch review UI should not just say "this fixes bug 123456". The information about the associated bug(s) should be made in a visually pleasant way, and all required information such as a bug summary should be visible right in the tool's UI, without any extra mouse clicks.

KDE's current setup relies on e-mail hooks which fire when a patch gets merged into any branch of the target repository. This leaves much to be desired; Bugzilla and Reviewboard remain two isolated islands that are only connected through hyperlinks. Another problem of this approach is that only commits by KDE developers can actually close bug reports. When a KDE developer merges a commit done by a third-party, perhaps when merging a pull request, a bug won't be closed despite a proper `BUG:` keyword.

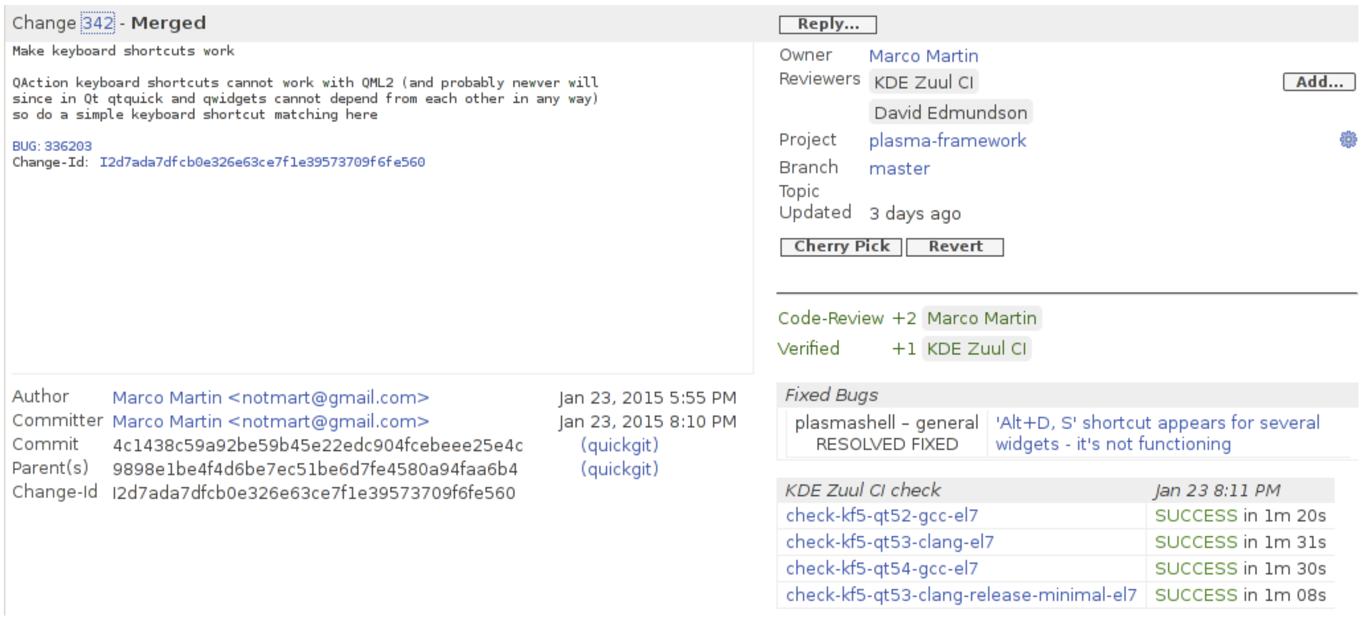
In KDE's Gerrit deployment, we added JavaScript code which relies on public APIs of both Gerrit and Bugzilla to provide interactive previews of the information stored in the other system. For developers, nothing is changed and they rely on the familiar `BUG` and `CCBUG` keywords. Gerrit pulls live data from Bugzilla, which is why a merged change in Gerrit correctly shows the bug as `RESOLVED FIXED`.

A similar integration was added to Bugzilla. The current version was designed to require minimal modifications to our instance. In particular, we did not want to add another custom field to the Bugzilla installation, or to inject JavaScript in there. The demo therefore relies on a client-side GreaseMonkey script for including the required JavaScript.[7] If the proposed system is approved and gets rolled out, no user-side changes will be required. The total maintenance burden of these extensions and the level of complexity is similar to using a modification of an existing Bugzilla theme.

We propose to use Gerrit's `its-bugzilla` plugin for making modifications to bugs when a change gets uploaded for the first time, and when it gets merged. Should KDE ever migrate to another platform for issue tracking, Gerrit ships with plugins for interacting with Jira, Phabricator and others. Adding support for more bug trackers is easy because a common core is cleanly abstracted away. The JavaScript integration bits which are Bugzilla-specific comprise a single AJAX call, and can be easily replaced in future.

The plugin is configurable on a per-project basis. To provide just a small example, it is possible to e.g. enforce mandatory presence of `BUG` keywords on release branches, or to specify that only patches merged to a stable branch will update Bugzilla records. We do not expect to need such

[7]Download from `http://jkt.flaska.net/gerrit-bugzilla-userscript.js`.



Change 342 - **Merged**

Reply...

```
Make keyboard shortcuts work

QAction keyboard shortcuts cannot work with QML2 (and probably newver will
since in Qt qtquick and qwidgets cannot depend from each other in any way)
so do a simple keyboard shortcut matching here

BUG: 336203
Change-Id: I2d7ada7dfcb0e326e63ce7f1e39573709f6fe560
```

Owner	Marco Martin
Reviewers	KDE Zuul CI, David Edmundson
Project	plasma-framework
Branch	master
Topic	
Updated	3 days ago

Add... Cherry Pick Revert

Code-Review	+2	Marco Martin
Verified	+1	KDE Zuul CI

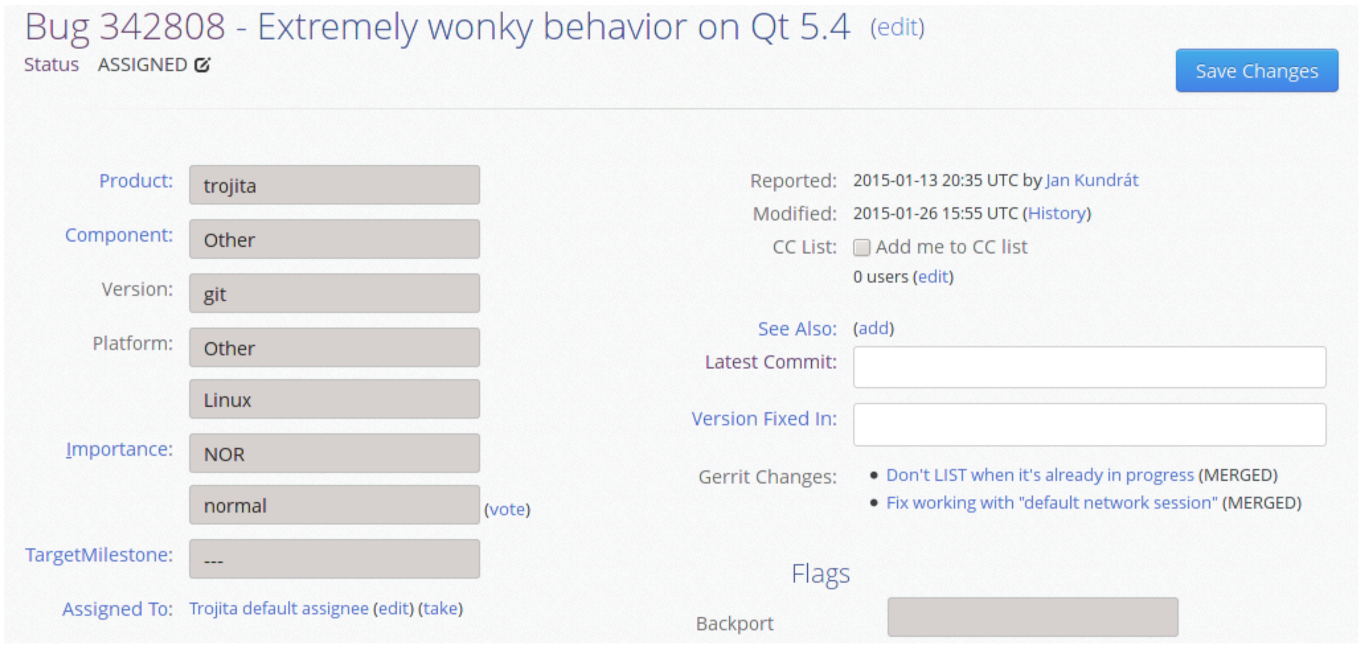
Author	Marco Martin <notmart@gmail.com>	Jan 23, 2015 5:55 PM
Committer	Marco Martin <notmart@gmail.com>	Jan 23, 2015 8:10 PM
Commit	4c1438c59a92be59b45e22edc904fcebeee25e4c	(quickgit)
Parent(s)	9898e1be4f4d6be7ec51be6d7fe4580a94faa6b4	(quickgit)
Change-Id	I2d7ada7dfcb0e326e63ce7f1e39573709f6fe560	

Fixed Bugs

plasmashell - general RESOLVED FIXED	'Alt+D, S' shortcut appears for several widgets - it's not functioning

KDE Zuul CI check	*Jan 23 8:11 PM*
check-kf5-qt52-gcc-el7	SUCCESS in 1m 20s
check-kf5-qt53-clang-el7	SUCCESS in 1m 31s
check-kf5-qt54-gcc-el7	SUCCESS in 1m 30s
check-kf5-qt53-clang-release-minimal-el7	SUCCESS in 1m 08s

Showing live Bugzilla data within Gerrit at `https://gerrit.vesnicky.cesnet.cz/r/342`. See `https://gerrit.vesnicky.cesnet.cz/r/1` for rendering of non-existing bugs.



Bug 342808 - Extremely wonky behavior on Qt 5.4 (edit)

Status ASSIGNED

Save Changes

Product:	trojita
Component:	Other
Version:	git
Platform:	Other
	Linux
Importance:	NOR
	normal (vote)
TargetMilestone:	---
Assigned To:	Trojita default assignee (edit) (take)

Reported:	2015-01-13 20:35 UTC by Jan Kundrát
Modified:	2015-01-26 15:55 UTC (History)
CC List:	☐ Add me to CC list
	0 users (edit)
See Also:	(add)
Latest Commit:	
Version Fixed In:	
Gerrit Changes:	• Don't LIST when it's already in progress (MERGED) • Fix working with "default network session" (MERGED)

Flags

Backport	

Bugzilla showing links to commits under review in Gerrit an advanced (and maybe annoying) configuration within KDE; the above is just to offer a glimpse of the level of functionality which is supported.

3.2.12 Project Management

There are many possible meanings of a "project management". In the existing KDE's setup, various roles are handled by Redmine/ChilliProject. It is *not* used for release planning, issue tracking, Gantt diagrams, file hosting, wiki pages nor any additional fancy purposes. Redmine's role is limited to act as a project browser, Git repository viewer, and a tool for storing additional metadata such as a proper place of the project in KDE's project groups (implemented as a custom patch).

Right now, KDE uses two essentially separate project organization concepts. At one layer, the list of projects is flat, with no hierarchy and no namespacing. All project identifiers (repository names) must be unique within the whole set. At another layer, the entire family of KDE projects is split into a tree. The top-level entries such as `frameworks/` or `extragear/` can be further divided into e.g. `extragear/graphics/` or `extragear/pim/`. Interestingly enough, the KDE Frameworks' concept of Tiers is not contained in this hierarchy and all KF5 projects are called just `frameworks/something`.

The philosophy of Gerrit is to store as many data as possible in Git repositories. A project's permission configuration is stored in a special file within the `refs/meta/config` Git branch. The same space is also used for various plugin-specific option storage.

We therefore propose to use the same place for storing arbitrary information that is needed by the rest of KDE's infrastructure. Benefits of such an approach include being able to treat modifications in the same way as patches to code – simply submit them for review, and let project maintainers approve, improve or reject these. A Continuous Integration setup can verify these changes for sanity, as well as be used for preparing a pre-generated `kde_projects.xml` file. All sorts of data, including long descriptions, stable branch assignments or localization options will be stored there. Metadata such as active status of the project will be generated based on Gerrit's native features for project enabling/disabling.

The missing bits, compared to the current solution, is mapping of people to project maintainers. These bits will be moved to LDAP where a group will be created for every project that *needs* special attention. From earlier discussions on `kde-core-devel` and `kde-scm-interest`, it appears that the intention is to keep all developers equal, and explicitly listing maintainers will serve no additional purpose. If this analysis is wrong, a list of maintainers can be added without any difficulty.

3.2.13 Audit Hooks

Audit hooks refer to a set of validators and helpers which are run during each push to KDE's Git. Their purpose is to serve as a rudimentary sanity check to ensure that obviously wrong stuff won't make it to our Git servers.

Gerrit *does* support running of hooks that are capable of rejecting a particular commit. We could therefore take the existing hook setup as-is and simply call it from within Gerrit. However, Gerrit *also* offers native support for majority of this functionality, and adopting native features appears to be a better solution in the long run.

Gerrit has, for example, native support for commit blacklisting. Even the API to control which commits are to be always rejected during a push is very similar to KDE's existing setup – a new command is `ssh user@gerrit ban-commit $project $commitSha1`. As a nice bonus, the command accepts a `--reason` option which can be used as a human-readable message to explain to users what to do if they accidentally push a blacklisted commit again.

Similar check applies to e-mail validation. An ACL verifies whether an e-mail matches one of user's registered address. These addresses are either read from LDAP, or validated by a mail probe to make sure that they actually exist and belong to the user in question. This validation can be configured on an LDAP group basis, so it is possible to allow KDE developers to push commits on behalf of third-party contributors while preventing regular users from faking their identity.

Commits can be rejected early if they include an "obviously wrong" file name, etc.

More serious checks are supposed to be integrated via the code review system. That way we can have sanity checks on e.g. Tab characters for projects which settled to use all spaces for indenting, or Krazy runs to provide an early EBN coverage.

Another improvement is building an audit trail via Git notes. KDE's Gerrit is already configured to make use of this feature. Here is how a typical record for a change looks like:

```
Notes (review):
    Verified+1: KDE Zuul CI
    Code-Review+1: Thomas Lübking <thomas.luebking@gmail.com>
    Code-Review+2: Marko Käning <mk-lists@email.de>
    Submitted-by: Jan Kundrát <jkt@kde.org>
    Submitted-at: Mon, 19 Jan 2015 19:07:43 +0100
    Reviewed-on: https://gerrit.vesnicky.cesnet.cz/r/339
    Project: trojita
    Branch: refs/heads/master
```

These data are available to regular users via another Git ref. This is a standard Git feature, and users who are interested in this can read these data through e.g. `git log --notes=review` invocation. Users who have no use for this do not fetch these data by default.

3.3 Continuous Integration

This section proposes an ambitious plan of replacing our Jenkins instance with an alternative. It is by no means a requirement to apply this part of proposal in order to work with Gerrit. Gerrit will work just fine with Jenkins as well. Please see the rest of this section for rationale on why we suggest this change. Also note that KDE's existing build scripts *and* Scarlett's work on Jenkins cleanup *and* Marko's OS X CI efforts will remain relevant even if this proposal is accepted.

KDE has been traditionally using Jenkins, an industry-standard for automated builds. Over the years, custom scripts were written for managing dependencies and for launching build jobs with multiple configurations. However, the way in which Jenkins is used actually does not make use of many of Jenkins' advanced features. Even a Git workspace preparation is handled by a custom shell script, and build artefacts are saved using a manually managed `rsync` depot. As of 2014-2015, the main role of Jenkins is to serve as a way of launching jobs on remote servers, and for providing an overview of whether a last build failed.

In Jenkins, build jobs cannot be easily defined in a declarative manner. Various solutions exist for managing the pain of uploading XML scripts over a web interface. As of early 2015, none are deployed by KDE yet. Other projects are struggling with Jenkins in a similar manner with various level of success. It is common to hear "today we would do it differently" when discussing mature Jenkins deployments.

A serious CI covers more than just launching a job and reporting back a result. If a change is proposed to Git, chances are that *another* change will land before a developer reviews the original patch, or that the contributor used a week old checkout as a basis of her work. It is therefore important to not check the change as-is, but rather test a future result of the change merged to the current state of the Git repository.[8]

It is also not enough to test a result of each `git push`. Developers all make mistakes, and it is pretty common, especially when rebasing locally, to introduce a regression by accident, only to be fixed by an immediate follow-up commit. When such an error happens, it makes it difficult to use `git bisect` when looking for (unrelated) regressions because a bisection process gets interrupted

[8]This is amplified by Git's emphasis on non-linear histories where the history tree can involve many merge operations.

by a presence of a spontaneous build breakage. A powerful CI therefore must also check each commit separately and make sure that it does not break the build or the unit tests.

In addition to that, some projects have dependencies on other projects. It is desirable, if the HW resources allow, to verify whether a change in a shared library breaks its consumers. While it is not realistic to do this for each commit to a Qt mirror or a core library such as `kdelibs` from KDE4 times, there are situations where the cost is not prohibitive as the space of dependencies is reasonably sized.

The OpenStack project has tackled these issues with Zuul, a build job scheduler. Zuul listens to events within Gerrit, merges Git trees, launches build jobs and reports back to Gerrit about their status. It can be configured to *enforce* a CI pass; that is, each commit must pass all tests before a change is merged into Git.

Zuul fully supports parallel execution of build jobs. If multiple independent changes are merged into Git before a CI run completes, Zuul will speculatively schedule build jobs under an assumption that an already-human-approved change is likely to succeed. If an unrelated change fails, the subsequent changes in the testing queue get resubmitted while skipping the commit which caused breakage. This process respects Git commit parents, so real commit dependencies are handled properly.

Interested readers are encouraged to check out a description of Zuul's operation in its excellent documentation at `http://ci.openstack.org/zuul/gating.html`. Zuul is not a trivial system, but the offered feature set and no need to work around Jenkins' limitations more than offset having to master yet another system.

3.3.1 KDE's Use of Zuul

An instance of Zuul has been deployed for use with KDE's Gerrit. It has been running in production since December 2014, verifying builds of a substantial chunk of KDE-Frameworks, up to and including `plasma-framework`. The KDE-specific parts of the build are reused from the Jenkins environment; in particular, the Python scripts and project interdependency data are an important piece of the functionality. There are no plans to deprecate these in our use of Zuul.

We currently offer builds with Qt 4, Qt 5.2, Qt 5.3 and Qt 5.4, using both release and debug builds of Qt, with GCC and Clang/LLVM compilers, on multiple versions of Linux. This will be extended with Mac OS X and Windows builds when time permits and depending on Scarlett's and Marko's results. The only client-side requirement of the current setup is Python 2.7.

The CI results are fed back to Gerrit as a vote against change approval. The setup was able to identify multiple regressions and breakages in proposed changes.

3.3.2 Test Failures

Some of the automated test are inherently unstable. A common situation which leads to non-deterministic results are test which talk to a network service. Another case of frequent test failures are hard-coded delays or timer values. Regardless of the actual cause, an unstable test suite is a fact of life which must be dealt with by a CI system.

Our Zuul instance is configured to allow developers to request a test re-run by simply saying `recheck` as a comment on a change.

The OpenStack's Gerrit and Zuul setup is accompanied by an automated build log crawler based on ElasticSearch. This tool analyzes the gathered logs and subjects them to statistical processing. If a common error pattern is found, developers are pointed to an appropriate bug number, or at least list of logs which recently triggered said condition. The log harvester also checks the `recheck` comments for references to bug numbers for extra context. In common cases, a developer whose change hit a known bug will be notified by an automated script explaining what to do, and with a pointer to a likely cause of the failure. This setup is available under Apache2 license, and can be deployed on KDE's infrastructure if time permits and people are interested.

3.3.3 Build Warnings

Projects which want to enforce warning-free builds deal with this requirement by appending `-Werror` to a reference build profile. The usual approach is focusing on no warnings on the latest compiler and latest version of Qt. This works well for e.g. Trojitá where we've been running like this for years.

3.3.4 Build Status Matrix and Status Reports

Some of KDE's projects do not subscribe to the concept of code review. At the same time, it is crucial for the release team to have an insight into the current status of the code. It would be suboptimal to release a version of a library with known build failures. While an ideal solution is mandatory CI verification prior to changes being merged, this is not realistic to request from all KDE projects. As such, build results have to be tracked, and a dashboard of their status needs to be provided.

A simple dashboard consisting of static JSON files gathered from builds which are triggered during each `ref-updated` build will be provided. The complexity of this setup is expected to be a small fraction of e.g. project dependency handling.

3.3.5 Third-Party Continuous Integration

Another feature not easily offered by Jenkins is support for a Continuous Integration feedback from interested third parties. In order to interact with Jenkins, a third-party service would have to gain a privileged level of access, something which might be hard to justify due to the associated security risks.

In contrast to that, *anyone* can vote in Gerrit, and people are free to consume events from the Gerrit event stream via e.g. `ssh bot@gerrit stream-events`. This has been demonstrated by the OpenStack project to scale quite well. Relevant examples of applications of this feature are enthusiastic users who are willing to e.g. test a subset of KDE projects on an exotic operating system or on a hardware whose capacity would not be able to accommodate the whole of KDE. To give a specific example, it probably makes little sense to test all of KDE on Android phones, but certain applications which already offer an Android UI would immensely benefit if they were subject to CI on that platform.

4 Additional Services

Because Gerrit focuses on code hosting and repository management, it does not provide features such as a bundled issue tracker, or a task management system. When evaluating competing infrastructure services, it is important to always compare full stacks of what it takes to provide a given level of service.

An often heard argument is that the fewer separate services are needed to provide a fixed feature set the better – especially for maintenance reasons. In this context, we should always estimate an actual price tag of each service's maintenance. Deploying and maintaining five services where each require 10 minutes of work each month is actually *better* then taking care of a single service which needs an hour of babysitting over the same time. The cost of the initial deployment has to be factored in, too.

Service bundling comes with a tax in scalability as well. Depending on the service architecture, it might be needlessly hard to split the entire infrastructure to multiple machines, or to load-balance these tools if a need arises as KDE grows.

The takeaway message is to compare actual costs rather than rough estimates, and to measure rather than guess. First profile, and then optimize.

4.1 Integrating Independent Services

The mere fact that our services are provided by different projects do not necessarily mean that they do not play well with each other. As an example, Gerrit's web UI was extended to offer a live preview of build status straight from Zuul. The change was originally authored by the OpenStack project, and is built in JavaScript to embed a continuously-updated JSON status report into Gerrit's main pages. As a result, once a change is uploaded to Gerrit, a live preview of the build status is visible on the review's web, and this review gets updated to include the actual build result in real time.

We implemented a similar thing between Gerrit and Bugzilla, again using a JavaScript approach. The integration goes beyond simple hotlinks common with other solutions. The web applications are complemented by modular JavaScript code that fetches data from the other service, integrating them back with a native look and feel. There is no need to maintain a possible fragile synchronization because data are always fetched from their corresponding origin as needed. The developer experience remains unchanged and relies on the well-known `BUG` and `CCBUG` keywords. See section 3.2.11 for screenshots and additional information.

A similar extension can be written when KDE adapts e.g. a Trello equivalent.

4.2 Single Sign On (SSO)

Chances are that we will always have to run more than one service. While having a single set of credentials to use over the whole KDE infrastructure is nice and *almost* possible today,[9] it is not a full answer to this problem. Users still have to enter the same passwords to various login boxes, and it is hard to be sure that each service to which one is logging in is a secure and official one provided by KDE.

The usual answer to this is some sort of an SSO (Single Sign On) approach. Each web page which requires a login would typically redirect its users to a single, central place to enter the user's credentials. The result of a successful authentication is a short-lived token which is automatically passed to the original service, and that service can use the identity vouch embedded in the token to authorize the user's action. Similar tools which hand out tokens exists for console access as well.

This proposal does not pick a favorite SSO solution. Adopting SSO is proposed for a later time, when resources permit.

4.3 Improvements to Task Tracking, Bug Reports and Issue Management

We are no experts on Kanban task planning or tools for agile development and scrum meetings. We will not discuss various existing tools which could be used for these purposes. These topics are a prime example of an area which is still under development, and no clear and *free* winners have emerged so far. Rather than tie in with a single solution which might not be a best one in the long term, we would like to propose to remain open to various alternatives at this point.

One promising example is StoryBoard,[10] an application designed to keep all aspects of user and developer stories and to tie them into project's development process. It will be worthwhile to keep an eye on its development and possible adoption.

[9] Bugzilla and Mailman are two notable exceptions. Each of them requires its own separate login credentials right now, and fixing this could be a non-trivial problem.

[10] See `https://wiki.openstack.org/wiki/StoryBoard`

5 Managing Infrastructure as Code

This is an optional part of this paper. We have no intention to push unwanted changes to people doing the work — we are just offering help. The proposed solution will reduce the time needed for infrastructure maintenance in the long run. We are sharing information about what has worked well for others who have been solving a similar problem. Other solutions to this problem are possible — we had to pick and *use* one approach.

While following this approach is by no means necessary for modernizing our infrastructure and improving the developer experience, all of the scripts and configuration recipes have been written and debugged and are now ready for use. As such, adopting this approach for areas covered by this proposal will not constitute additional work.

A question of whether to use a configuration management system for the rest of the KDE infrastructure is outside of scope of this paper. In particular, we are absolutely *not* pushing for adoption of our favorite tools. If KDE was already using some tool for configuration management, we would have simply followed suit.

This section talks about a proposal for switching away from manual service administration and maintenance, and calls for embracing some popular, widely deployed configuration management system. The idea is to start with an empty, minimal install of a Linux distribution, and write a *program* in a declarative language describing how the end result should look like. This approach puts emphasis on the expected final state, and delegates actual work to mature, battle-tested utilities.

Providing even a gentle introduction to this area would make this text much longer. Rather than doing a poor job of explaining the technical details in an extremely incomplete manner, let us emphasize the following bonuses which one gets by adopting such an approach to system management:

Repeatability The entire configuration is described in a set of files saved in Git. The tools guarantee that no steps which are needed will be skipped when a server needs to be reinstalled.

Audit It is easy to tell who changed what because the data driving all actions are stored in Git.

Transparency Everything but secret keys and passwords can be safely made available to the community, and community can help by offering specific suggestion as *patches*. This is a development process which we all know and can follow.

Scale It doesn't matter whether a configuration is applied to a single machine, or to hundreds of VMs in a cloud. The tools are built for scalability.

Modularity Configuration can be split into reusable modules, similar to how we use shared libraries for repetitive or low-level tasks. Need to backup a box? Write that code once, and parametrize the backup setup with e.g. a list of directories. Got to back up an SQL database? Add parameters for DB name and login credentials, and write that `pgdump`-launching cron job once.

The most common heard objection is "we only have a couple of servers". Most of the advantages listed above however apply to each-server-is-unique deployments as well.

The future is not all bright, of course, and there are also dangers inherent to adopting a *different* approach to what one has been doing for years. It is also easier for an "innocent mistake" to take down a substantial chunk of one's infrastructure. As usual, careful testing, a staging environment and peer review of code can mitigate most of these risks.

Making one-off changes with a configuration management system is usually a slower operation than `ssh`ing into a machine and editing files by hand. The small decrease of speed however pays off as soon as some new member joins the sysadmin team. A unified structure of configuration and common patterns help newcomers navigate the jungle of complex setups and service interactions, and assist people make correct changes at the right places.

The situation around infrastructure-as-code approach is similar to programmers' view about unit tests and source code versioning. There are some excellent programmers out there who do not write test cases. There are also excellent programmers who do not put their code into a SCM. However, there is only a very small set of programmers who have already tried a SCM such as Git, yet choose not to use one for their work anymore.

5.1 Choice of Tools

There are several high-profile contestants in this area, and we had to choose one for implementing our proposal. KDE does not currently use any such system, so there were no considerations for picking a particular tool to align with the rest of the infrastructure. Due to prior experience and knowledge and thanks to its adoption within the OpenStack Infrastructure team, by Digia's CI infrastructure and many other high-profile projects including CERN, we chose to use Puppet.

All services which are covered in this proposal, including Gerrit's installation and configuration, the entire Zuul CI deployment, all IRC notifiers etc., are covered by Puppet manifests. These installation and configuration scripts serve as basic documentation, and make it possible to redeploy the entire setup quite easily.

Our reasons for picking this Puppet approach were quite simple – we were looking for *some* tools to use for repeatable deployment, and Puppet manifests proved to fit the job pretty well. Had KDE used something else already, we would have followed suit. There is no intention to push something over to the KDE sysadmins; Puppet was just a tool which we used to let our job done. We expect to use Puppet for this deployment of the infrastructure, if approved, because it will make a KDE-wide adoption require less time.

6 Possible Alternatives

A prominent alternative proposed by KDE's sysadmins is deployment of Phabricator. Please see their original proposal and the `kde-core-devel` archives for a discussion about its merits, as well as for a list of other possible alternatives. We are deliberately trying not to evaluate Phabricator's performance or suitability for KDE in this document. We have no practical experience with it at this time. A detailed plan for its adoption and a roadmap for fixing the missing bits would have to be provided before it can be considered for migration.

7 Conclusion

This paper presents a vision for how a next generation KDE infrastructure can be implemented. It does not build on a green field, but rather reuses a proven architecture which has been field-tested by the OpenStack Infrastructure project.

We describe how a set of independent yet well-integrated tools can *together* provide an infrastructure which is easy to use and maintain. A proof-of-concept trial has been running within KDE since September 2014 with excellent feedback from all participants so far.

It is our belief that the setup described in this paper is capable of handling KDE's needs, and we recommend to evaluate this architecture as an alternative to a possible adoption of Phabricator.

8 Summary of Required Future Work

The following list summarizes what has to be done in order to enable all features described in this proposal. It is built on a real experience with the tools in question. This means that the list is very close to being exhaustive; we do not expect additional major items to appear. In particular, this is *not* just a list created after a high-level documentation reading.

Everything else which is proposed in this paper but not explicitly mentioned as a work item below has been already implemented, and is now ready for immediate deployment. A realistic estimate is that migration of all KDE Git projects can be completed well within a month, including testing and verification periods and a staged migration.

- Install `gerrit-patch-uploader` (see 3.2.1)
- Debug script for self-service repository creation (see 3.2.3)
- Write documentation about a `Workflow` label for "clones" (see 3.2.4)
- Write script for syncing SSH keys to Gerrit (see 3.2.8)
- Extend Gerrit with support for e-mail notifications about direct pushes (see 3.2.9)
- Configure IRC bot to notify about commits as well (see 3.2.10)
- Configure the `its-bugzilla` plugin (see 3.2.11)
- Install JavaScript add-on for querying Gerrit to `bugs.kde.org` (see 3.2.11)
- Documentation: improve KDE-specific tutorials and best-practices on how to work with Gerrit effectively
- Write script generating `kde_projects.xml` (see 3.2.12)
- Dashboard indicating state of the CI (see 3.3.4)
- Bootstrap KDE's Puppet and import the already-developed Puppet configuration in there (see 5)

About the Author

Jan Kundrát has been a KDE developer since 2007. When he is not busy programming Trojitá, automating large infrastructures or messing with build systems, he likes photography, hiking and other sorts of outdoor activities out in the woods and up in the mountains. Between 2006 and 2012, he was on a team dealing with a 350+ node cluster; the servers were processing scientific data originating at the CERN's LHC accelerator. At that job, Jan was driving a successful effort of switching from ad-hoc manual administration to a cfengine based automated setup. Since 2013, Jan has been running a public cloud service based on OpenStack.