Continuous Delivery of Debian packages

Michael Prokop
Terminology

- **Continuous Integration**
  - well known from software development
- **Continuous Deployment**
  - Q/A criteria says OK? Ship/deploy!
- **Continuous Delivery**
  - release whenever you decide it's useful to do so (= business decision!)
Why?
Costs of a Bugfix

Source: Barry Boehm's „EQUITY Keynote Address“
Independence

Source: http://decarabia.soup.io/post/241926962/Image
Reproducible

Source: https://wiki.debian.org/ReproducibleBuilds
Predictable

I'm just outside town, so I should be there in fifteen minutes.

Actually, it's looking more like six days.

No, wait, thirty seconds.

The author of the Windows file copy dialog visits some friends.

Source: https://xkcd.com/612/
Problems
Problems $company experienced 1/2

- Mess with golden images (to ship a custom software stack to customers)
- Long build times (e.g. single change → rebuild full image, upload to customer,...)
- Builds non-reproducible (unmanaged build infrastructure, devs can build + include their own packages,...)
Problems $company experienced 2/2

- Release process holding back ongoing development work (VCS freezes are preventing ongoing work)
- Getting more and more customers → not scaling (golden images → even worse)
- Tried Debian source package uploads to custom build service → many pitfalls + developers still needed to manually build/release packages (some of them not even using Debian/Ubuntu → tools like git-dch, debuild,... unavailable)
What do we want?
Deployment Pipeline

Source: http://continuousdelivery.com/2010/02/continuous-delivery/
Workflow

1. Development/Testing
   - git commit & git review

2. Debian builds (+PPA)
   - Jenkins verify (-1/+1)
   - Code Reviewers (-2/-1/0/+1+2)

3. Submit to {master,$branch}

4. Debian package, Puppet,...

5. Internal tooling

6. Available to Customers
   - $Release (incl. Q/A)
   - Final Debian build

7. Release dashboard

$Product
How did we get there?
Principles

• Rely on Debian packages + Debian repositories for everything (no exceptions)
• Only what's under version control matters (no option to build something manually on your own system)
• Automate infrastructure handling (Puppet/Ansible)
• Automated debian/changelog handling to simplify releasing of new package versions (devs don't need Debian/Ubuntu at all)
• Automated release branch handling (release 0.42 is available as such a branch)
• VMs for testing/development (via Vagrant → run `vagrant up $product-$version`, automated box builds at least once per day)
• PPAs for development (no VCS freezes, fast-forward + release branches only)
Improvements

• Usage of tmpfs/eatmydata, ccache,... for build speedups
• Dashboards for abstraction + let people focus on their tasks instead of tools
• Code review system (improves code quality but also sharing knowledge + introducing new people)
Jenkins-debian-glue
“The nice thing about standards is that there are so many of them to choose from.”

Source: https://xkcd.com/927/
Jenkins?

- Hudson: 2004
- Jenkins: 2011
- Weekly releases + LTS versions
- MIT license
- >1000 plugins available
- >120k registered installations (07/15)
- Disclaimer: written in Java, but absolutely not restricted to Java projects
Why jenkins-debian-glue?

• Formalize existing knowledge into a customizable framework
• Provide a common ground to base (further) work on
• Gather feedback from what other users (might) need
• Community building
• Don't create new tools and standards, instead rely on existing and working ones
• Should be easy to use also for non-Debian folks
What's behind j-d-g?

- Open Source Project (MIT license)
  - started in 2011
  - >25 contributors
  - written mainly in shell, easy to adjust + extend
- CI server (Jenkins)
- Build environment (cowbuilder/pbuilder)
- VCS (git + svn OOTB)
- Repository management (reprepro + freight)
- Q/A tools: piuparts, lintian, autopkgtest, pep8, perlcritic, shellcheck, checkbashism
Who's using j-d-g?

- Grml (incl. dpkg, FAI, initramfs-tools,...)
  - https://jenkins.grml.org/
- PostgreSQL
  - https://wiki.postgresl.org/wiki/Apt
- LLVM
  - http://llvm.org//apt/
- Kamailio
  - https://kamailio.sipwise.com
- Wikimedia
  - https://integration.wikimedia.org/ci/view/Ops-DebGlue/
- ... and many more
Setup? Automatic deployment

% wget https://raw.github.com/mika/jenkins-debian-glue/master/puppet/apply.sh
% sudo bash ./apply.sh $your_password

http://jenkins-debian-glue.org/getting_started/
### Jenkins

**jenkins-debian-glue** Continuous Integration labs

<table>
<thead>
<tr>
<th>Build Queue</th>
<th>Name</th>
<th>Last Success</th>
<th>Last Failure</th>
<th>Last Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>No builds in the queue.</td>
<td>jenkins-debian-glue-binaries</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Build Executor Status</td>
<td>jenkins-debian-glue-piuparts</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>#</td>
<td>jenkins-debian-glue-source</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Icon: S M L

Legend: RSS for all, RSS for failures, RSS for just latest builds

---

Help us localize this page


REST API

Jenkins ver. 1.480.2
generate Debian source package using VCS
  – (Upstream) Source (orig.tar.xz)
  – Debian changes (debian.tar.xz) [optional]
  – Control file (.dsc)
• Script `generate-\{git,svn\}-snapshot`
• Automates changelog generation (git-dch ftw, thanks Guido!)
• Important: needs to be run only once per project (exception: multi distribution usage in one repository)
${\text{project}}$-binaries

- Debian Binary Packages (*.deb)
- Script build-and-provide-package
  - Automates pbuilder/cowbuilder setup, usually nothing to do manually
- Important: build once per architecture/distribution (exception: “Architecture: all”)
${\text{project}}$-piuparts

- Install/deinstall/upgrade testing (optional)
- Useful since you might forget about it otherwise
Repository Handling

- Automatic handling of repositories without manual interaction
  - reprepro
  - freight
- By default part of ${project}-binaries job
- Separate usage via ${project}-repos job:
  - BUILD_ONLY vs PROVIDE_ONLY
Further Q/A testing available OOTB

- Lintian
- Autopkgtest
- perlcritics/checkbashism/shellcheck/pep8/...
- Results as TAP/jUnit/... reports in Jenkins available
Example of a Build Pipeline

- Automatic lintian integration in *-source + *-binaries
- Automatic autopkgtest-integration in *-binaries
- Optionally Code-Review + automatic merge to Master after Q/A
- Optionally further static code analysis, web tests, performance tests,...
Managing many Jenkins jobs without driving nuts?

- usage of jenkins-job-builder to create and manage Jenkins jobs
  - http://docs.openstack.org/infra/system-config/jjjb.html
  - https://github.com/sipwise/kamailio-deb-jenkins (example)
- YAML file(s) for configuration
  - No webinterface clicking!
  - Version control!
  - Code review for job changes!
Lessons learned
Lessons learned 1/3

• Developers needs vs operations/distribution needs (\$package or \$version not available)
  – Contribute back to Debian when reasonable
• Diverse people improve overall quality
  – Homogeneous systems, diverse people
• Code review requires good remote working culture
  – Open Source folks are used to remote working :(
Lessons learned 2/3

• Avoid external dependencies
  – Github, CPAN, PyPI, RubyGems, Puppetlabs, Percona, $local_debian_mirror... unreachable?
  → set up local mirrors
  • Speedup!
  • Staging options
• Configuration management (e.g. for setup of Jenkins slaves) is essential → infrastructure as code
• Consistent timezones (UTC) + time (NTP!)
Lessons learned 3/3

• Catch 22
  – build scripts broken but build infrastructure receives updates via build infrastructure/scripts? → recursion problem
  – upgrading from wheezy to jessie, deployment of configuration management depends on unit-tests which don't work on jessie yet
• Provide test infrastructure for setup, configuration,... changes without breaking production
• Rebuild of a system might look different from currently running one, even with cfgmgmt → use testing also for cfgmgmt (serverspec, mspectator, Tests::Server, Test Kitchen,...)
Tips 1/2

• Regular rebuilds of all packages → apply recent policies + package build infrastructure changes so packages are up2date
• mr/myrepos is very useful for dealing with large amounts of repositories (thanks joeyh!)
• Integrate CI/CD system into your monitoring environment
Tips 2/2

• Collect metrics independent from Jenkins & CO to be able to remove jobs/builds without losing metrics
• Use gertyt cmdline tool if you don't like gerrit web interface
• Set up "jenkins-verify" job to ensure Jenkins works as needed
Antipatterns 1/3

- Manual SSH → provide debugging options instead
- Flaky Tests (fast vs slow hardware, „sleep X”, ...) → people don't trust + care any longer
- Polling/pull/cronjobs instead of triggering → get immediate actions + effects
Antipatterns 2/3

- Manual setup of machines/configs → snowflake pattern (AKA they look alike but are still different)
- No standarized output in tools → makes parsing hard[er]
- Checklists → use automation instead
• Hardcoding (IP addresses, hostnames, port number, test system,...) instead of configurability
• Same thing gets built multiple times in the deployment pipeline → share artifact instead
• Lack of notifications for failing builds/tests/... → developer starts to wait + poll
Unresolved problems 1/2

- dependency management alla wanna-build to get package builds automatically in the right order (package `foo` Build-Depends on package `bar` → build `bar` before `foo`)
- Build-Depends vs Depends, but no „Test-Depends” (bundler, carton,...)
• „High frequency“ (CI/CD) Debian repositories causing apt to often fail while mirror is updated (“Hashsum mismatch”)
• piuparts: successful runs even though there have been issues, e.g. package that gets tested has dependency issues though removing the package itself is considered a valid solution
Recap - projects possibly worth a look

- Debian :)  
- Jenkins  
- Jenkins-debian-glue  
- Vagrant  
- Gerrit + Gertty  
- Jenkins-Job-Builder
Recap – tl;df

• Put everything under version control
• Automation (deployment, cfgmgmt, release process)
• Custom Dashboards
• Tests, tests, tests
• Rely on established workflows + tools
• PS: Once you're used to that working in non-CD environments feels bad
Jenkins-debian-glue BoF

• Date: 2015-08-21
• Time: 18:00-19:00
• Room: Helsinki
• Purpose: In this BoF session we provide an opportunity to meet developers + contributors of the jenkins-debian-glue project, discuss issues for improvements, upcoming new features and get your questions answered.
Questions || Wishes?

@mikagrm
mika (at) debian.org

http://michael-prokop.at/blog/
http://jenkins-debian-glue.org/

Thanks for feedback to Christian Hofstaedtler + Victor Seva