Continuous Integration at a Distribution Level

Topics

1. 14a Debian dev, $>$ 12a Ubuntu dev
2. U: For 4 years: CI for 30,000 source packages
3. still only distro that does test-based gating; share some experience, try to convince you

Motivation
- old distro model: break/freeze/fix
- new distro model: rolling release
- force developers to finish transitions
- make use of existing tests
- cover packaging and integration

1. first few years: 4 months of feature dev, FF, try to find and fix half of the regressions
2. daily morning exercise to unbreak boot, X.org, packages; not enough non-devs were using devel series
3. archive wide changes (lib transitions, deps for major Qt version) not finished - SEP, later
4. once popular enough, mission critical, commercial products: not good enough
5. devel series is stable and usable at all times, safe to use by non-devs, ratchet towards perfection
6. many upstreams have tests (during build) and moved to CI, but no uniform way to run them downstream, don't run at the right time

Continuous Integration at a Distribution Level

Idea

1. several iterations of standalone desktop/server test suites, QA team responsible, Jenkins
2. didn't work socially (blame game) and technically (no gating, noone pays attention)
3. conclusion: devs must be responsible for testing, used for gating, QA team only does infra and consulting
4. add tests to source packages that exercise the binary packages as-installed
5. trigger on uploads of pkg or rdeps, gate
6. autopkgtest: both test driver and name for this kind of test
7. submitted as Debian Enhancement Proposal #8

Idea

http://dep.debian.net/deps/dep8/

```
Title: autopkgtest - automatic as-installed package testing
Drivers: Ian Jackson <ijackson@chiark.greenend.org.uk>,
 Iustin Pop <iustin@debian.org>,
 Stefano Zacchiroli <zack@debian.org>
Abstract:
 Establish a standard interface to define and run "as-installed"
 tests of packages, i.e. the testing of packages in a context as
 close as possible to a Debian system where the packages subject
 to testing are properly installed.
```

Continuous Integration at a Distribution Level

Simple CLI test: gzip

1. one of the simplest and oldest tests: gzip
2. d/t/control: metadata: enumerate tests, deps, other properties/testbed reqs (later)
3. d/t/testname: executable, exit 0 iff pass

Simple CLI test: gzip

debian/tests/control:

```
Tests: simple-gzip
Depends: gzip
```

debian/tests/simple-gzip:

```
#!/bin/sh -e
echo "Bla" > bla.file
cp bla.file bla.file.orig
gzip bla.file
gunzip bla.file.gz
cmp bla.file bla.file.orig
```

Continuous Integration at a Distribution Level

Running the test

Running the test

```
autopkgtest gzip -- qemu ubuntu-xenial-amd64.img

autopkgtest ~/debian/gzip-1.6/ -- schroot sid

autopkgtest http://git.debian.org/gzip.git -- \
          lxd images:debian/jessie/i386
```

1. autopkgtest program: create temp testbeds, copy in test, run, copy results back out, logging, influencing
2. various ways of specifying the test: package name, directory, git tree, etc.
3. backends with different capabilities/isolation levels; tests in schroot (first backend), lxc, lxd, qemu, or arbitrary ssh, cloud, adb
4. production: QEMU for x86 and Power, lxd for ARM/zSeries

Continuous Integration at a Distribution Level

Simple lib test

Simple lib test

debian/tests/control:

```
Tests: build-login
Depends: build-essential, libsystemd-dev
```

1. slightly more useful/elaborate: libraries: compile/link/run simple program

Continuous Integration at a Distribution Level

debian/tests/build-login:

```
#!/bin/sh -e
cat <<EOF > loginmonitor.c
#include <systemd/sd-login.h>

int main(int argc, char **argv) {
    sd_login_monitor* mon = NULL;
    int res = sd_login_monitor_new(NULL, &mon);
    if (res < 0) {
        fprintf(stderr, "sd_login_monitor_new failed: %i\n", res);
        return 1;
    }
    assert(sd_login_monitor_get_fd(mon) > 0);
    return 0;
}
EOF

gcc -Wall -Werror -o loginmonitor loginmonitor.c \
    $(pkg-config --cflags --libs libsystemd)
echo "build: OK"
./loginmonitor
echo "run: OK"
```

1. first look: simple, second look: lots of things that can break, they do fail
2. -dev package is missing deps, forgets to install header files or pkgconfig
3. upstream pkgconfig is broken, toolchain/multi-arch lib lookup issues
4. same pattern in a lot of libraries these days

Continuous Integration at a Distribution Level

More complicated tests

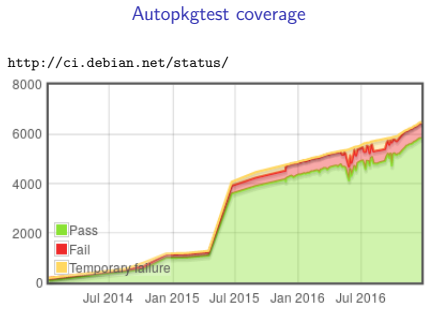
More complicated tests

```
Tests: networkd-test.py
Tests-Directory: test
Depends: systemd, policykit-1, dnsmasq-base
Restrictions: needs-root, isolation-container
```

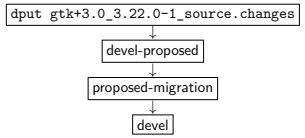
1. run test shipped by upstream: test/networkd-test.py in systemd
2. Dir: normally look for test executable in debian/tests
3. needs root, needs container or better due to veth and starting services
4. isolation-machine: NetworkManager (mac80211hwsim), kernel (stress-ng)
5. systemd: simulate suspend for logind, create scsi-debug LUKS partition, install/check start of NM, lightdm, crucial services, no failed services, boot smoke, upstream QEMU tests
6. don't want to go into too many details here, just give a broad overview of how devs use this

Continuous Integration at a Distribution Level

Autopkgtest coverage



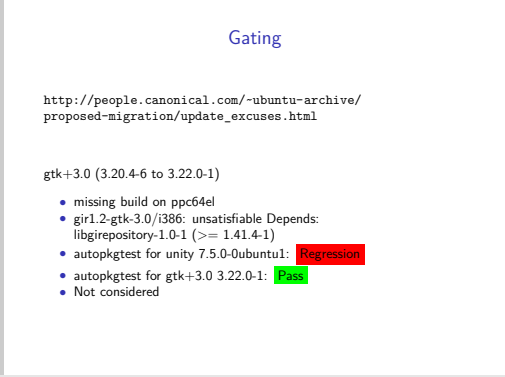
1. tests run in Debian too since 2014, but D does not gate yet
2. pushed Ubuntu tests to Debian, vast majority come directly from D now
3. great success: $>$ 6000 packages, covers much more through rdeps
4. big leap: generic tests for perl/ruby/dkms modules



1. dev prepares and uploads new GTK
2. put into proposed pocket: overlay archive, staging area; no human users
3. p-m checks builds, installability, tests
4. once all good: p-m lands verified package groups in devel, otherwise kept in proposed
5. packages in devel never regress in architecture support, installability, or tests
6. might need further uploads to adjust reverse dependencies to new ABI, removing broken packages, manual overrides possible

Continuous Integration at a Distribution Level

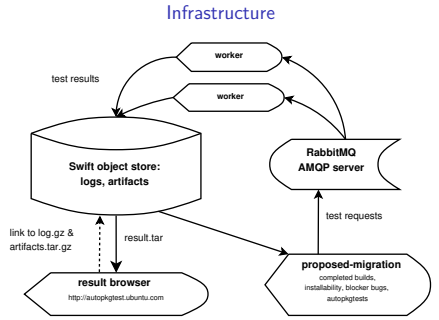
Gating



1. proposed packages appear on this report
2. simplified output (5 arches, 1 test), tests don't start if unbuilt/uninstallable
3. simple case, consider glibc, perl, python, apt; land them with confidence
4. not just devel, also stables

Continuous Integration at a Distribution Level

Infrastructure



1. started with Jenkins, but brittle, hard to maintain, losing requests, hard to set up locally, SPOF
2. standard cloud tech, small/loosely connected components
3. policy entity (proposed-migration or GitHub): request tests to AMQP
4. RabbitMQ; job distr system; robust, parallel, atomic, simple API
5. workers: grab requests from queues they can service, call autopkgtest, run test in temporary cloud instance, put logs/artifacts into swift; many dozen parallel ones, different arches
6. Swift: OpenStack data storage, all test results, logs, artifacts
7. requestor polls swift for results
8. web UI: present test results/logs/artifacts to developers; independent, uncritical, replaceable
9. Juju charms, simple to deploy locally into 3 containers for dev/testing, redeploy in minutes without any loss

Continuous Integration at a Distribution Level

Impact for Ubuntu

Impact for Ubuntu

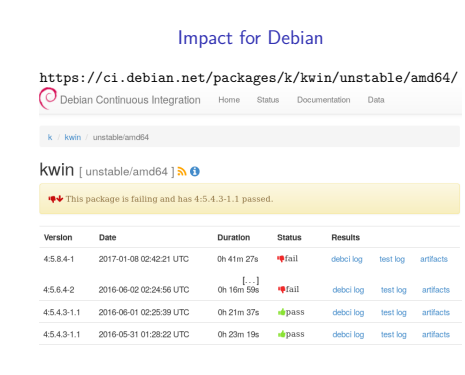
- *Don't you break my software!*
- cross-package changes land completely or not at all
- usable devel series
- release team: ☹→ ☺

- *Argh, something broke the tests!*
- maintain infrastructure and cloud
- broken tests/updates imported from Debian

1. effective carrot and stick for developers
2. carrot: better tests - harder for other people/packages to break your software
3. kernel > lxc, systemd, apparmor; X > Qt > KDE
4. cross-package changes: complete or not at all, pointless to whine against a machine; good devel series
5. release team does not have to clean up behind changes tossed over the fence
6. cost: keep tests passing; break for weird reasons (infra/cloud changes, external web sites, changes not covered by CI
7. test infra is not free: reliable CI service on necessarily unreliable hw, demanding tests; cloud/infra maintenance
8. broken tests or updates (ruby) imported from Debian; no manpower -> ignore
9. after a few months people got used to it, "if" not disputed, just tweak policy and infra

Continuous Integration at a Distribution Level

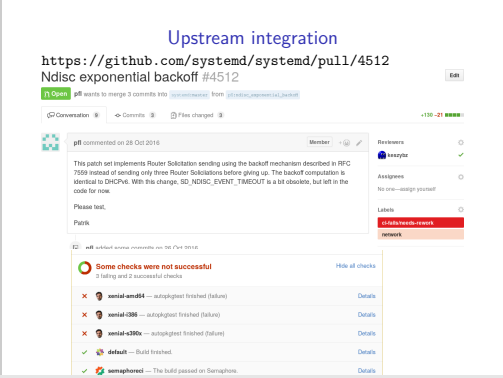
Impact for Debian



1. works great for native Ubuntu sw (installer, Unity, Juju, snapd, MaaS
2. finds bugs and keeps them from landing; report upstream, fix
3. gating in Ubuntu conceptually too late for upstream software
4. running/gating tests in Debian

Continuous Integration at a Distribution Level

Upstream integration



1. systemd: new upstream release, week or two to find and fix regressions
2. use the tests at the point where damage happens: upstream PRs
3. tweaked our systemd package to be able to build unmodified upstream source and adjust tests for Debian specific behaviour
4. GitHub PRs call out to Ubuntu infra
5. dev heaven: every commit and release in master builds, passes tests
6. daily builds are easy (PPA), new release == write nice changelog
7. not limited to systemd; takes a bit of effort to set up