OpenStack Packing for Ubuntu

How we do Cloud Archive things

Bug Fix backports and Archive Pockets

Bug fix and backport

1. Fix upstream
2. backport to upstream stable branches
    1. rocky
    2. stein
    3. etc.
3. start backporting SRU after landing in stable branches
    1. propose backport as SRU to Ubuntu release
    2. backport to UCA
    3. staging
    4. proposed
    5. updates

Archive pockets

- Ubuntu - Main archives
- *Staging* pocket - first place we backport a package to
    - should come from the LTS
    - Make sure the packages build
- Proposed pocket - has to be here at least 7 days (SRU bug has to be updated with vefirication tags)
- Updates pocket - Users are only expected to run Updates

Tools

rmadison : Package versions in ubuntu

cmadison : Package versions in UCA

> The following three don't pull git repos

pull-lp-source : Download the package sources

pull-uca-source : Download the package sources from the UCA

pull-debian-source swift : Download the package sources from Debian

git clone https://git.launchpad.net/~ubuntu-server-dev/ubuntu/+source/nova : Clone the source package source

To check what the difference between two builds is (useful when changing `debian/rules`), using python-magnumclient as an example:

If we go to the magnumclient LP page, we can see multiple builds. At the moment, for Groovy, we can see that there is a 2.11 and a 3.1 build for it. We can cliek through to the 2.11 or 3.1 buildlog, scroll to the bottom, and see what files are included in the package. By comparing these, we can confirm that our expectations are correct regarding that the changes are.

Making a package in a PPA

Hey Chris,

This is how to create a PPA, make a minor package change, build a package, and upload to a PPA.

1. Useful env vars:

```
export GPGKEY=0DCDF806
export EDITOR=/usr/bin/vim
export DEBFULLNAME="Corey Bryant"
export DEBEMAIL="corey.bryant@canonical.com"
export QUILT_PATCHES=debian/patches
```

2. Go to your launchpad page (e.g. https://launchpad.net/~chris.macnaughton) and click "Create a new PPA".
3. If working with bionic (queens by default), no changes are needed for your PPA dependencies. If working with xenial-queens (cloud archive) you'll need to "Edit PPA dependencies" and make your PPA depend on queens-staging. You may be able to get hints from https://launchpad.net/~corey.bryant/+archive/ubuntu/xenial-queens.
4. Clone package source: git clone https://git.launchpad.net/~ubuntu-server-dev/ubuntu/+source/keystone; cd keystone; git checkout stable/queens
5. ls debian # the debian directory is where all package customization occurs
   - poke around and see what's there
   - maybe cherry-pick an upstream patch to debian/patches
6. dch -i # make a changelog entry
7. Bonus:
   1. cherry pick a patch from the corresponding upstream branch that is not yet in the package
   2. or use quilt to create a new patch (quilt new; quilt edit ; quilt refresh; quilt pop -a)
   3. patch goes in debian/patches; patch name goes in debian/patches/series; 'git add' any new files
8. `debcommit -a`
9. `git log` # see your change?
10. `gbp buildpackage -S -sa`
11. `ppa_upload.sh --yes -d bionic --upload ppa:chris.macnaughton/bionic-rocky ../build-area/keystone*.dsc` # [1]
12. look for package building in ppa
13. install package from ppa - ie. `sudo add-apt-repostiory ppa:chris.macnaughton/bionic-rocky && sudo apt update && sudo apt install keystone-common`

[1] ppa_upload.sh

```
#!/bin/bash
#
# This script just makes it easier to upload the same package version to a PPA multiple times.
#
# ppa_upload.sh -d bionic --upload ppa:corey.bryant/bionic-queens ../build-area/keystone*.dsc

backportpackage -S ~ppa`date +%Y%m%d%H%M` $@
```

> Trouble shooting:

> * git build-package may fail with something like "dpkg-checkbuilddeps: error: Unmet build dependencies: apache2-dev dh-apache2 openstack-pkg-tools (>= 23~) python3-sphinx (>= 1.6.2) You will need to apt install these build dependency packages and try again. Note: look in debian/control at Build-Dependencies (that's where these are defined) and Depends are run-time dependencies

pbuilder stuff

```
pbuilder-dist focal create
pbuilder-dist focal update
pbuilder-dist focal build ../build-area/cinder*dsc

pbuilder-dist bionic login --save-after-login  # login to chroot, make changes, and save



# Pbuilder drop into chroot on failure
mkdir /var/cache/pbuilder/hooks; ln -s /usr/share/doc/pbuilder/examples/C10shell
/var/cache/pbuilder/hooks/C10shell; echo HOOKDIR=/var/cache/pbuilder/hooks | sudo tee -a
/etc/pbuilderrc



pbuilder-dist experimental create # debian experimental
```

When logged into the chroot:

```
fakeroot debian/rules clean
fakeroot debian/rules build
fakeroot debian/rules install
fakeroot debian/rules binary
```

New OpenStack Release

Note: If uscan fails you may need to first update debian/watch to get the release tarball from opendev.org (see aodh).

Example of steps taken for aodh to pick up rc1 release:

```
git clone lp:~ubuntu-server-dev/ubuntu/+source/aodh
cd aodh

git checkout pristine-tar
git checkout upstream
git checkout master

uscan --verbose --download-version 10.0.0~rc1 --rename
gbp import-orig --no-interactive --merge-mode=replace ../aodh_10.0.0~rc1.orig.tar.gz
```

If doing an update for a release that goes straight to the cloud archive, make sure that you use `~cloud0` and pick up the last d/changelog version from the uca. ie:

https://git.launchpad.net/~ubuntu-server-dev/ubuntu/+source/barbican/commit/?id=a4cce949f0988ca75690d95ff5632b38abf466ba

```
dch -i
sed -i "1s/1ubuntu1/0ubuntu1/" debian/changelog
# again see aodh as an example

debcommit -a  # this will commit the changes with a git log that is the same as your changelog in d/changelog

# check the git log to see what changes are in new release - ie.

git diff HEAD~2 HEAD~1
# update debian/control based on any new requirements in the new release
# if debian/control updated then also update debian/changelog with "  * d/control: Align (Build-)Depends with upstream."

# it'll take some time to understand what's actually needed here or not.
```

```
# doc requirements.txt and test-requirements.txt go in Build-Depends-(Indep) and
# run time requirements.txt generally go in Build-Depends-(Indep) and Depends.

# accuracy of build-depends are not quite as important as run-time depends.

# again

debcommit -a
```

Next, try to build the source package: `gbp buildpackage -S -sa`

if patches fail to apply it's possible they need to be rebased or perhaps they landed upstream recently and can be dropped from d/patches. useful commands: quilt push, quilt edit, quilt refresh, quilt pop. if making patch changes, again update d/changelog with details and use debcommit -a to commit.

Finally when ready to upload to the testing PPA I'm using: ~/src/scripts/ppa/ppa_upload.sh --yes -d focal --upload ppa:openstack-ubuntu-testing/ussuri ../build-area/aodh_10.0.0~rc1-0ubuntu1.dsc

But for you, for now at least, if you can push all branches (upstream, pristine-tar, master) and all tags (git push --all && git push --tags) to your own namespace on launchpad then I can merge those into the ubuntu-server-dev repos and push them to the ppa for you.

I think what we will do is keep the changelogs UNRELEASED until final release since we're just upload to the testing ppa for now. At that point we can release/tag and upload groovy (from master branches), and then cut stable/ussuri branches and adjust the package version slightly so that groovy supsercedes focal, then release/tag and upload focal to the focal unapproved queue where the SRU team will review and accept.

Package Specifics

ceph

Ceph uses sbuild to build the package

Dependencies

```
# sbuild dependencies
sudo apt install -yq sbuild debhelper ubuntu-dev-tools piuparts
# Ceph dependencies
sudo apt install -yq \
    default-jdk javahelper junit4 libatomic-ops-dev \
    libboost-dev libboost-program-options-dev libboost-system-dev \
    libboost-thread-dev libedit-dev libfcgi-dev libfuse-dev \
    libgoogle-perftools-dev libjerasure-dev libkeyutils-dev \
    libleveldb-dev libs3-dev libsnappy-dev libxml2-dev python-nose yasm
```

Setup sbuild

`mkdir -p /home/ubuntu/ubuntu/logs`

create /etc/schroot/sbuild/fstab

```
# fstab: static file system information for chroots.
# Note that the mount point will be prefixed by the chroot path
# (CHROOT_PATH)
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
/proc           /proc           none    rw,bind         0       0
/sys            /sys            none    rw,bind         0       0
/dev/pts        /dev/pts        none    rw,bind         0       0
tmpfs           /dev/shm        tmpfs   defaults        0       0
# Mount a large scratch space for the build, so we don't use up
# space on an LVM snapshot of the chroot itself.
/var/lib/sbuild/build  /build   none    rw,bind         0       0
```

Create $HOME/.sbuildrc:

```
# *** VERIFY AND UPDATE $mailto and $maintainer_name BELOW ***

# Mail address where logs are sent to (mandatory, no default!)
$mailto = 'chris.macnaughton@canonical.com';

# Name to use as override in .changes files for the Maintainer: field
$maintainer_name='Chris MacNaughton <chris.macnaughton@canonical.com>';

# Directory for chroot symlinks and sbuild logs.  Defaults to the
# current directory if unspecified.
$build_dir='/home/ubuntu/ubuntu/build';

# Directory for writing build logs to
$log_dir="/home/ubuntu/ubuntu/logs";
# don't remove this, Perl needs it:
1;
```

Setup some chroots!

```
for distro in focal bionic; do
mk-sbuild $distro
done
```

Build Ceph

`sbuild ceph -c chroot:focal-amd64 --arch=amd64 --dist=focal --nolog -A`

Horizon

Horizon vendors xstatic so needs weird things

```
# This example is using Train as an example, during a release of the 16.2.0
# stable point release. Note that the previous version of the package is
# 16.1.0, so we're pulling the xstatic tgz from there.
wget https://launchpad.net/~ubuntu-cloud-archive/+archive/ubuntu/train-staging/+so
urcefiles/horizon/3:16.1.0-0ubuntu1~cloud0/horizon_16.1.0.orig-xstatic.tar.gz
mv horizon_16.1.0.orig-xstatic.tar.gz horizon_16.2.0.orig-xstatic.tar.gz
```

Alternately, to pull in the xstatic bits for a new release, you can run `debian/rules refresh-xstatic` as documented in `debian/README` in the package.

Build the source package

```
cd horizon
# debuild can handle multiple sources when building packages, so that's
# what we're using to build th package, rather than `gbp`
debuild -S -sa -us -uc
```

Now we want to confirm that the package built correctly

> The following could specify the version more explicitly but I like the generic wildcard version over `horizon_16.2.0-0ubuntu1.dsc`

```
pbuilder-dist eoan build ../horizon*.dsc
```

Point Release

1. http://10.245.162.25/ and look at the corresponding upstream report
   - Train: http://10.245.162.25/train_upstream_versions.html
2. scroll through to find versions were behind on but only for the core packages that are on this wiki page
3. Process update the same way as a new OpenStack release
4. Open a bug similar to: 1818069
5. (After we upload it to the unapproved queue) subscribe the ubuntu-sru team to the bug
6. Add a card to the distro trello board

Updating a backport patch (Qemu)

pull-lp-source qemu focal (do that twice into 2 different directories) (one sec looking at patch)

git clone lp:~ubuntu-cloud-archive/ubuntu/+source/ca-patches

in one of the qemu repos do: patch -p1 < lp:~ubuntu-cloud-archive/ubuntu/+source/ca-patches/ussuri/qemu.patch

it won't apply cleanly as expected so fix up that (d/p/series I think is the issue)

then on both run: debuild -S -sa -us -uc

then debdiff qemu1.dsc qemu2.dsc > /tmp/qemu.patch

then diff -Naur /tmp/qemu.patch ubuntu-cloud-archive/ubuntu/+source/ca-patches/ussuri/qemu.patch # just fix up patch comments at this point

cp /tmp/qemu.patch ubuntu-cloud-archive/ubuntu/+source/ca-patches/ussuri/qemu.patch

git commit and git push

Do a merge / sync

1. grab the automated merge results grab-merge python-mistralclient && cd python-mistralclient
2. download ubuntu package source git clone lp:~ubuntu-server-dev/ubuntu/+source/python-mistralclient
3. untar debian dir from LATEST debian package tar xvf python-mistralclient_4.0.1-2.debian.tar.xz
4. copy merged debian dir to our package source rsync -avrz --delete ../python-mistralclient-4.0.1-2ubuntu1/debian/ ./python-mistralclient/debian/
5. read the REPORT file (note the Conflicts) vi REPORTS (in this case don't worry about any that aren't in debian dir)
6. resolve the Conflict files cd python-mistralclient (fix up conflicting files, you'll see conflicts surrounded by <<<<<<< ======= >>>>>>>)
7. update d/changelog with your name/email/stamp dch -i # you'll need to maintain the old version; not sure of a better way
8. compare latest debian and new ubuntu package diff -Naur ../debian/ debian/|less
9. note any differences* in "Remaining changes" vi debian/changelog
10. verify git diff changes make sense git diff
11. commit changes debcommit -a # only keep "Merge from Debian unstable. Remaining changes:" paragraph in commit message
12. push to launchpad git push lp:~ubuntu-server-dev/ubuntu/+source/python-mistralclient
13. create new upstream release if needed

Patch headers

https://packaging.ubuntu.com/html/patches-to-packages.html#patch-headers

We recommend that you tag every patch with DEP-3 headers by putting them at the top of patch file. Here are some headers that you can use:

Description: Description of what the patch does. It is formatted like Description field in debian/control: first line is short description, starting with lowercase letter, the next lines are long description, indented with a space.

Author: Who wrote the patch (i.e. “Jane Doe packager@example.com”). Origin: Where this patch comes from (i.e. “upstream”), when Author is not present.

Bug-Ubuntu: A link to Launchpad bug, a short form is preferred (like https://bugs.launchpad.net/bugs/XXXXXXX). If there are also bugs in upstream or Debian bugtrackers, add Bug or Bug-Debian headers.

Forwarded: Whether the patch was forwarded upstream. Either “yes”, “no” or “not-needed”.

Last-Update: Date of the last revision (in form “YYYY-MM-DD”).

Uploading to the cloud archive

```
debuild --no-lintian -S -sa -nc -uc -us -v2015.1.1-0ubuntu2 --changes-option=-
DDistribution=bionic  (-v specifies the previous package version; bionic is the base ubuntu
LTS release)
debsign ../$package.changes
dput ppa:ubuntu-cloud-archive/<openstack-codename>-staging package*source.changes
```

Prior to Ubuntu Groovy (OpenStack Victoria), the cloud archive has been backed by private PPAs.

Running Autopkgtests on a package

First, make sure that you have a container ready for the matching series:

```
autopkgtest-build-lxd ubuntu:focal
# the development release is only on the dailiy servers
autopkgtest-build-lxd ubuntu-daily:hirsute
```

Then we run the test with the built package:

```
autopkgtest --shell-fail -U --apt-pocket=proposed ../build-area/${PWD##*/}_*changes -- lxd
autopkgtest/ubuntu/groovy/amd64
```

Regression Testing

Horizon (openstack-dashboard)

IN addition to clicking raound to verify that the basic functionality is present, create a basic heat stack through the dashboard:

```
heat_template_version: 2013-05-23

description: Simple template to deploy a single compute instance

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      image: cirros
      flavor: m1.tempest
      key_name: nova-key
      networks:
      - network: private
```

Updating a batch of packages

Rouch outline for updating a batch of packages

```
# For released bits
for thing in $packages; do upstream-release $thing xena; done
# For snapshots
for thing in $packages; do ~/pkg-scripts/pkg-snapshot-version-git $thing master; done
cd pkg

# Check if packages match what's uploaded
for thing in $packages; do cat $thing/debian/changelog | grep -q $(rmadison $thing | grep
impish | awk -F '|' '{print $2}' | tail -n 1) || echo "$thing has differences"; done



for thing in $packages; do echo $thing; (cd $thing; ~/pkg-scripts/pkg-update-deps impish);
done

# Diff all of the $packages to see if updates are needed

# can skip releaseing if desired, but will need to re-do the builds after
for thing in $packages; do (cd $thing; dch -r; debcommit -ar); done

for thing in $packages; do echo $thing; (cd $thing; (gbp buildpackage -S -sa || touch
../$thing.gbp.fail) | tee ../$thing.gbp.log); done
# Check for *.fail

for thing in $packages; do echo $thing; (cd $thing; (pbuilder-dist impish build ../build-
area/${PWD##*/}_*dsc || touch ../$thing.pbuilder.fail) | tee ../$thing.pbuilder.log); done
# Check for *.fail

dput build-area/*.changes

for thing in $packages; do git -C $thing push --all; git -C $thing push --tags; done
```