redhat.

Restructuring ceph-container

March 2018

Why restructuring ceph-container?

- Some metrics (commit b5f0fd6e261ee1c3895a4075a8766e78bebb84ee)
  - 3 Ceph releases (Jewel, Kraken, Luminous)
  - 7 OS versions (Ubuntu {14.04 |16.04}, RHEL 7.3, OpenSuse {leap 42.2, 42.3}, Fedora 24, CentOS 7)
  - 10 flavors (luminous/ubuntu/16.04 is one flavor)
  - 586 files including 316 symlinks (53 %)
  - One file can be linked up to 5 times !

Why restructuring ceph-container?

- Issues triggered by the current structure:
  - Hard to understand: is it a specific file or a link? Not really easy to understand when patching a flavor
  - Almost impossible to estimate the impact of a change as the file is linked by other flavors
  - Impossible to make a specific change on a linked file without breaking the link and duplicate the code
    i. A single value, like the distro code name, is enough to request a file duplication
    ii. Some similar commands are duplicated up to 8 times (i.e wgetting confd from github)
    iii. This duplicated code is never updated in all flavors
  - When adding a new entrypoint file was error prone as you could forget to add the symlink
  - Build process is done via a user script that move files and only one flavor at a time
  - Using docker hub for building the devel & release containers is damn slow
    i. 1 flavor at a time, long queuing, hours of build

Why restructuring ceph-container?

- Analysing the current code
  - differences & commonalities
    i. Differences between flavors are small if we'd use variables
    ii. Some files are common to all flavors
    iii. Some files are very specific to some or a single flavor
  - The core of ceph-container project has :
    i. Little changes between Ceph releases
    ii. Little changes between OSes & releases
  - Packages names are almost similar between deb & rpm
    i. But the way to install them is different (dnf / yum / apt)

Rethinking the architecture, a matriochka approach

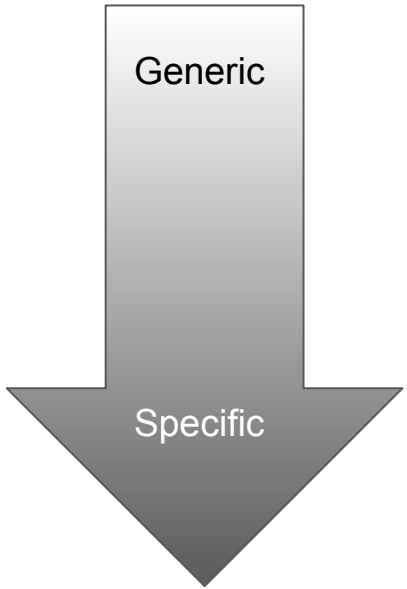
- Ideas to ease maintainer's life
  - Throwing symlinks to hell
  - Avoiding code duplication as much as possible
  - Avoid collateral damages when editing specific files
    i. making a change for Ubuntu should not break CentOS
  - Improving collaboration with external contributors to get more support from other downstreams projects

- Why not building containers by stacking files?
  - All containers receive the generic files
  - Specific flavors can add files or override the generic files if needed
  - Using a templating approach to turn a generic file into a specific one

redhat.

Matriochka approach - prioritizing files

- Defining a priority order to add files (extracted from CONTRIBUTING.md)
  - src/FILE
  - src/{daemon-base,daemon}/FILE
  - ceph-releases/ALL/FILE
  - ceph-releases/ALL/{daemon-base,daemon}/FILE
  - ceph-releases/ALL/<os distro>/FILE
  - ceph-releases/ALL/<os distro>/{daemon-base,daemon}/FILE
  - ceph-releases/ALL/<os distro>/<os release>/FILE
  - ceph-releases/ALL/<os distro>/<os release>/{daemon-base,daemon}/FILE
  - ceph-releases/<ceph release>/FILE
  - ceph-releases/<ceph release>/{daemon-base,daemon}/FILE
  - ceph-releases/<ceph release>/<os distro>/FILE
  - ceph-releases/<ceph release>/<os distro>/{daemon-base,daemon}/FILE
  - ceph-releases/<ceph release>/<os distro>/<os release>/FILE
  - ceph-releases/<ceph release>/<os distro>/<os release>/{daemon-base,daemon}/FILE



redhat.

Matriochka approach - an example

Example of a file existing in 5 versions (located in different sub-directories)

1. src/plop.sh
2. ceph-releases/ALL/ubuntu/plop.sh
3. ceph-releases/ALL/ubuntu/14.04/plop.sh
4. ceph-releases/jewel/plop.sh
5. ceph-releases/kraken/ubuntu/14.04/plop.sh

What file is chosen regarding the OS / Ceph matrix ?

	Ubuntu 16.04	Ubuntu 14.04	Centos 7
Luminous	2	3	1
Jewel	4	4	4
Kraken	2	5	1

Note : A tool called “find-src”located into the staging dir, reports the origin of every file

redhat.

Matriochka approach

- Important concepts
  - Current & future versions first !
    - src/ is today equals to luminous and future releases
    - A new feature is implemented in src/
    - Required exceptions should be added for older releases
    - When older releases are deprecated, exceptions disappear at the same time
  - Until its proven, everything is identical on all flavors
    - It’s important to think code for being used everywhere
      - Less code, better support
    - Exceptions can be handled with empty variables
      - __CEPH_MGR_PACKAGE__ is empty in Kraken as no package for it

Templating

- To maximize the commonalities between flavors, we use a template which :
  - Implement nested substitution
  - Based on files and a flat tree
  - Has defined values provided by :
    i. Makefile (ARCH, CEPH_VERSION, OS_NAME, OS_VERSION)
       1. Will be replaced in a string called like STAGE_REPLACE_WITH_ARCH
    ii. __SOMETHING__ files
  - As per the matriochka approach, it's easy to override a __VARIABLE__ by another which is located in a specific directory
    i. __RADOSGW_PACKAGE__ = “radosgw” for debs while equals “ceph-radosgw” for rpms

How to use it ?

- Staging the code
  - Staging assemble files for a given flavor and prepare the directory that will be built
  - Staged files have all their variables substituted, useful for debugging (staging/<flavor> directory)
  - “make” is enough to stage the default targets

- Building
  - Calls docker build on staged flavors
  - “make build.parallel” is recommended to improve the build speed
    i. It does create the staging + start the docker build

How to use it ?

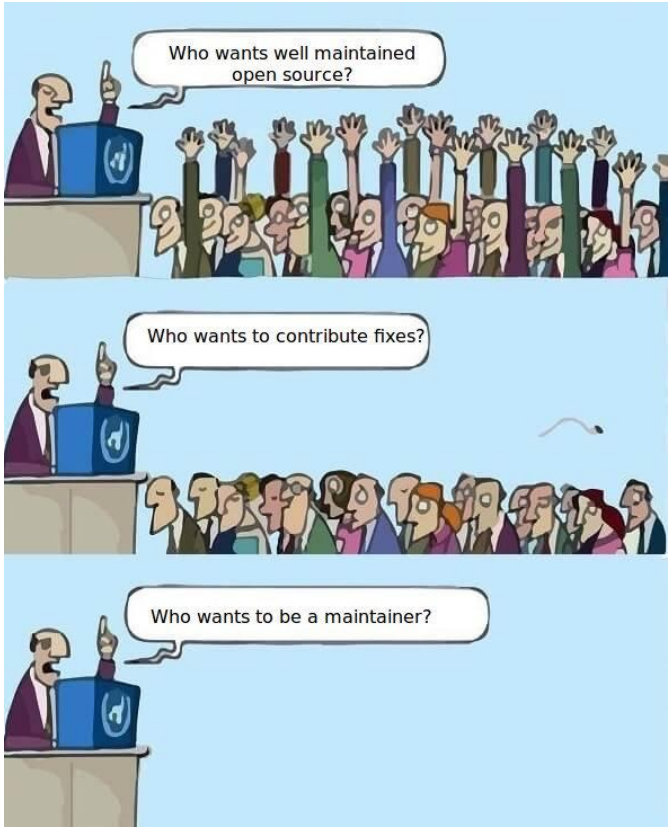
- Pushing to Docker Hub
  - “make push” will push the built images to Docker Hub according to your credentials
  - “make push.parallel” do the same task but parallelize the processing

- Some variables could be used during those commands
  - FLAVORS= to enforce a particular set of flavors to build
  - RELEASE= to enforce the tag name
    i. By default, RELEASE equals the branch name

Benefits

- Some metrics (commit d10b7e45a2950a36090ab91c54030e0c26cabbe8)
  - 66 commits later
  - 3 ceph releases (Luminous, Jewel, Kraken)
  - 4 OS versions (ubuntu {14.04 |16.04}, opensuse 42.3, centos 7)
  - 8 flavors
  - 143 files (¼ of the initial size) including 0 symlinks
  - Adding new distributions
    i. Centos needs 9 files and 1736 chars,
    ii. OpenSuse needs 13 files and 1833 chars
    iii. Adding a distro doesn't require any specific code to support all ceph-releases
    iv. Having new versions of the distro doesn't induce more code as we have built-in variables
  - Easier/native arch support (currently x86_64 only)

Benefits - Managing the community



redhat.

Benefits

- Community
  - Every distro has its own directories
    i. We define a __DOCKERFILE_MAINTAINER__ per distro to give responsibility to contributors
    ii. A PR pointing those distro directories induce
      1. No collateral damage for any other distro
      2. This PR has to be handled by the maintainer
  - Adding new major contributor
    i. Blaine Gardner from Suse has been a major contributor in this restructuration
    ii. He is very active and offers high quality PR and generates a good dynamic
    iii. He is maintaining the OpenSuse port and merge its stuff alone (quality commits are required)
  - Supporting multiple distributions
    i. Enlarges the user base
    ii. Provides more feedbacks & patches from other downstreams

Benefits

- Better traceability
  - Each container embedded a set of traceability items
    - Git repository url, branch name & commit id
    - Git status (aka was the local repo was clean at build time = no local edit since commit id)
    - A Release tag embedding the branch name or a release version (via a git tag)
- Better CI integration
  - We now have a Jenkins job that gets triggered after each PR is merged. This job will:
    - Build images on a slave
    - Push them on the Docker Hub
    - Tag latest
    - The whole process for the 7 default flavors takes only 14 minutes
- Images tags can be found at https://hub.docker.com/r/ceph/daemon/tags/

Extra benefits that comes with this change

- Make it easy to be consumed by other upstream projects doing containerized Ceph
- We decided to go back to using 2 images instead of one.
  - Daemon-base: contains Ceph packages
  - Daemon: contains ceph-container specific files/entrypoint
    - It depends on daemon-base at build time (Docker file, FROM daemon-base)
- For example: Rook uses our daemon-base image and puts its own entrypoint on top of it

- Smaller container image size
- The build mechanism shrink all the layers to a single one, this drastically reduces the size of the final container image.

Useful links

Read the contribution guide:
https://github.com/ceph/ceph-container/blob/master/CONTRIBUTING.md

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos