Release Process

Gabriele Comoretto
Configuration and Release Engineer

DMLT Face-to-Face
November 6-8 2018

Presentation Summary

- Releases End Users
- Release Process in Use
- Assumptions
- Release Process To Be
- Pragmatic Approach
- Patch Releases
- Semantic Versioning

Releases End Users

- It is all about what we need releases for...

- From DMTN-044:
  - Regular LSST developer
  - Close Collaborators
  - External Users (science collaborators)

- Are these users really requiring Official Releases?
  - Or are we talking about snapshots? Release Candidates? Weekly or daily releases?
  - Maybe what is needed is a stable reference point for development activities
  - This should be available in each commit on master, that has passed the CI system

Releases End Users - 2

- What about DM Operations data processing?
  - Prompt processing
  - DRP processing
  - Calibration Processing
  - … other DM SW Products?
- We need to be able to provide software releases for operations:
  - These release shall not contain
    - Source code,
    - Test data
    - Build Tools
  - we shall deploy only binaries
- We should be doing this during commissioning and OPS rehearsals...

Releases End Users - 3

- So we have 2 different use cases:
  - **We need a release process that can provide the release of the required software product (see Product Tree) for operational processing**
  - We still need to be able to provide a software distribution to the external community, that distribution shall includes everything:
    - Build tools and environment definition
    - DM Software
    - Test data
    - Source code
    - …
  - If we can't combine these use cases, we will need 2 different release processes

Release Process in Use

- Based on weekly releases automatically generated
  - The first release candidate is based on the announced weekly (just a copy)
- Validation of the release candidate with unit test (and demo package)
- Additional release candidates created in case of problems / additional changes
  - until the final release candidate is identified
- Creation of the final release
- Naming schema: M.m
  - No semantic versioning: in each release all packages SW will get the same version

Release Process in Use - 2

- It is a simple approach
  - But monolithic
- Slow: the build job takes several hours
  - It includes everything, (3rd party packages, check packages like python, build tools, test data, etc)
  - DM SW is fragmented (more than 120 pkgs)
  - It is complex
- In summary:
  - It fulfills the requirement to provide a distribution to the external users

Assumptions

- The source of true is the SW repository
- A SW Product Release is identified by
  - A Tag in the repository, that includes all source code, default configurations, dependencies information
  - A Release Note that includes all descriptive information
  - Any packaging is just a technicality on top of the SW Release
- A SW Product has only one git repository (not true for DM)

Release Process To Be

- Oriented to release only a SW Product defined in the product tree:
  - For example DRP software product: will includes only the SW packages needed to run the DRP
    - Reason:
      - we have to deploy in operations (NCSA, Base, Summit, others?) only the binaries that are required to implement the operational services
  - It assumes that Dependencies have been released already following the same process
- To handle in a separate procedure:
  - The environment management (conda?)
  - The build tools (including sconsUtils)
  - The packaging has to go on top of the release (eups? conda?)
  - The distribution (docker?)

Release Process To Be - 2

- Example 3rd party package Boost, release 1.68.0:
  - Source code is available at:
    - https://dl.bintray.com/boostorg/release/1.68.0/source/boost_1_16_0.tar.bz2
  - It do not contains any dependencies, that are:
    - Compiler (depends on the platform)
    - icu
    - bzip2
    - xz
    - zlib
  - Using a conda recipe, for example:
    - https://github.com/gcomoretto/boost-cpp-feedstock
  - a distribution package can be created:
    - https://anaconda.org/lsst-dm/boost-cpp
  - that can be used to build DM science pipelines

```
requirements:
  build:
    - {{ compiler('cxx') }}

  host:
    - icu                 # [unix]
    - bzip2               # [unix]
    - xz                  # [unix]
    - zlib

  run:
    - icu                 # [unix]
    - bzip2               # [unix]
    - xz                  # [unix]
    - zlib
```

- **The Release is not depending from the Conda PKG**

Release Process To Be - 3

- Release steps on a single SW Product:
  - Cut a branch
  - Create a release candidate (on a release branch) in the SW repository
  - Validate the release candidate
  - Create new release candidates as needed and validate them
  - Create the final release, a Tag in the repository, and document it with the Release Note.
- Distribution Packages (eups, conda) can be created on the release candidates, on each commit and on the Official Release!
- This is not feasible now for DM:
  - The actual build systems is requiring all the software to be build at once
  - The number of SW packages is too high
- What can be done now ?

Pragmatic Approach

Can be done now: moving all 3rd party packages into conda environment as a first step:

- DM-15495: successful build afw moving all 3rd party packages into conda environment
  - 24 (3rd party) packages moved into conda
  - Recipe created (or updated) for the ones that was not available in conda
    - Boost, minuit2, apr, apr-util pytest-session2file, ndarray,
    - Packages now available at: https://anaconda.org/lsst-dm
  - Remains 15 DM SW packages to build using lsstsw - up to afw - on mac and dev01 cluster

(thanks to Tim and Jim for the support provided)

```
(scipl15) gcomoret@lsst-dev01 ~/demo15/lsstsw (master) $rebuild -r tickets/DM-15495 afw
                afw:  ok (7.7 sec).
          daf_base:  ok (1.4 sec).
             utils:  ok (1.3 sec).
              base:  ok (1.1 sec).
         sconsUtils:  ok (1.3 sec).
     pex_exceptions:  ok (1.1 sec).
    daf_persistence:  ok (2.6 sec).
               log:  ok (1.1 sec).
        pex_policy:  ok (1.5 sec).
        pex_config:  ok (1.9 sec).
              geom:  ok (2.0 sec).
           sphgeom:  ok (1.3 sec).
           astshim:  ok (1.3 sec).
       starlink_ast:  ok (5.6 sec).
           afwdata:  ok (191.7 sec).
# BUILD ID: b3902
         sconsUtils: tickets.DM-15495-gd75b50e6d4 ok (30.8 sec).
       starlink_ast: lsst-dev-gd6cc4e835a ..........................................
...................ok (171.8 sec).
           astshim: tickets.DM-15495-g7001fd91b3 .ok (59.9 sec).
              base: tickets.DM-15495-g94c015e6c5 ok (21.4 sec).
     pex_exceptions: tickets.DM-15495-gd50fb9fc60 ok (26.6 sec).
           afwdata: 16.0-1-ge3096bf+17 ok (10.8 sec).
           sphgeom: tickets.DM-15495-g87815cb334 ok (30.4 sec).
             utils: tickets.DM-15495-g972e103ea4 ok (36.1 sec).
              geom: tickets.DM-15495-g9fba1c17a6 ...........ok (78.5 sec).
          daf_base: tickets.DM-15495-g39c1881d78 ok (40.7 sec).
               log: tickets.DM-15495-gb47864af19 ok (26.4 sec).
        pex_policy: tickets.DM-15495-gc7ffbb6c17 ok (44.8 sec).
    daf_persistence: tickets.DM-15495-g4a94016bcb .ok (59.9 sec).
        pex_config: tickets.DM-15495-gda6958d695 ok (38.6 sec).
               afw: tickets.DM-15495-g00feb39d5f ..................................
.....................................................................................
.....................................................ok (431.6 sec).
# BUILD b3902 completed.
(scipl15) gcomoret@lsst-dev01 ~/demo15/lsstsw (master) $
```

Pragmatic Approach - 2

- Can be done now: manage the conda environment
- In the near future (6 -12 month?):
  - Manage the build system
    - Build tools: lsstsw (and others) not to be part of the released SW Product
      - To be released separately (TBD)
    - Conda packages required (they are defined in lsstsw)
  - Need to reduce the complexity:
    - Removing from the release process what is not part of the release
    - Reducing the number of packages
      - Merging packages → SW Product
      - Using git-submodules → Meta Package → SW Product
      - Can we (try to) control the creation of new git packages?

Pragmatic Approach - 3

- In the long term:
  - Improve the build tools and dependency management
  - Build the SW Products as identified in the product tree:
    - Keep lsst_distrib for distribution to the science community (as it is now, in a first moment)
  - Release each SW Products separately
    - Just Tags in github
  - Provide separate distribution packages (eups / conda) for each SW Products
    - `conda install drp`
  - Create Docker images based on the new provided distribution packages
  - Create a distribution for the scientific community that includes:
    - All Science Pipelines SW products
    - The build tools and the environment definition

Summarizing

- The release is just a Tag (and release note)
  - All information is contained in the SW repository
- The packaging and distribution are different processes:
  - They go “on top” of the release
  - Can be done on a snapshot (like a weekly), for each sha1, or for an Official Release
- The development tools and environment need to follow their own release process
- The release process should be applied to only one SW product at once

Questions?

Patch Releases

- Patch releases are not done at the moment (at least not for release 15.0 and 16.0 of the science pipelines).
- Can be done using release branches
  - Back-porting ticket/branches → Development procedure to be defined
  - Potentially there may be lots of conflicts
    - I have back-ported DM-16235 (cherry-pick) on a release branch based on w.2018.42 (afw, jointcal) in a test repository.
- What we need is:
  - a controlled process, since back-porting and fixing conflicts may cost time
  - a development procedure for back-porting (the developer is the driver here)
  - a documentation procedure (use Jira Fix in Version Field, as proposed 6 months ago)
- During operations and commissioning we may need to do lots of patches
  - Potentially one per day at the beginning
  - We need a way to shortcut the process… but first we need to *dominate* the regular process.

Patch Releases - 2

- Patches on the actual release process:
  - All the packages will be released with the patch version
    - New tag in the repository and new eups packages
  - This requires time, but it may be quicker than doing a new release from master
- Patches on the proposed release process:
  - Only the impacted SW Products need to be released
  - Semantic Versioning can be applied
- Do we always need an official patch release?
  - In some cases a stable release branch can be used instead:
    - Snapshots on the release branch or a patch release candidate
  - This may be OK for:
    - Validation / Integration / Commissioning
    - Scientist that need a stable SW base to playing with
  - An official patch release can always be done

Semantic Versioning

- https://semver.org/spec/v2.0.0.html
- Three digits version identification: **MAJOR.minor.*patch***
  - new **MAJOR**: when incompatible API are introduced in the code
  - new **minor**: when backward compatible functionalities are introduced in the code
  - new **patch**: when backward compatible bug fixes are introduced in the code
- It may be really helpful for managing patch releases
  - I think we need to sort out other things first (decisional procedure, development procedure and documentation).
- But is it feasible?
  - In some cases we may need to have breaking changes in minor releases or even in patches (especially at the beginning of OPS and pre-OPS)
    - This is optimistic: in 20 years of my experience, almost always a patch contained breaking changes (it is not me to decide)
- We may need to relax the application of SemVer, and enforce it lather.

Questions?

Other processes

- Development
  - Management of the build tools
  - This will require a release process on the build tools SW packages and conda environment
- Dependencies
  - Add a requirements.txt in the existing SW packages
- 3rd party packages handle
  - Use as much as possible available anaconda packages
  - Fork existing conda recipes if available and need to be changed
  - Create lsst conda recipes otherwise
- Packaging DM SW
  - Use conda in parallel to EUPS (?)
  - I would nice to just type: `conta install lsst_distrib`(?)
- Distributing DM SW
  - To operations: docker
  - To science community:
    - lsst_distrib is not a SW product, it is a distribution that shall include all: SW Release, Build System, Test Data