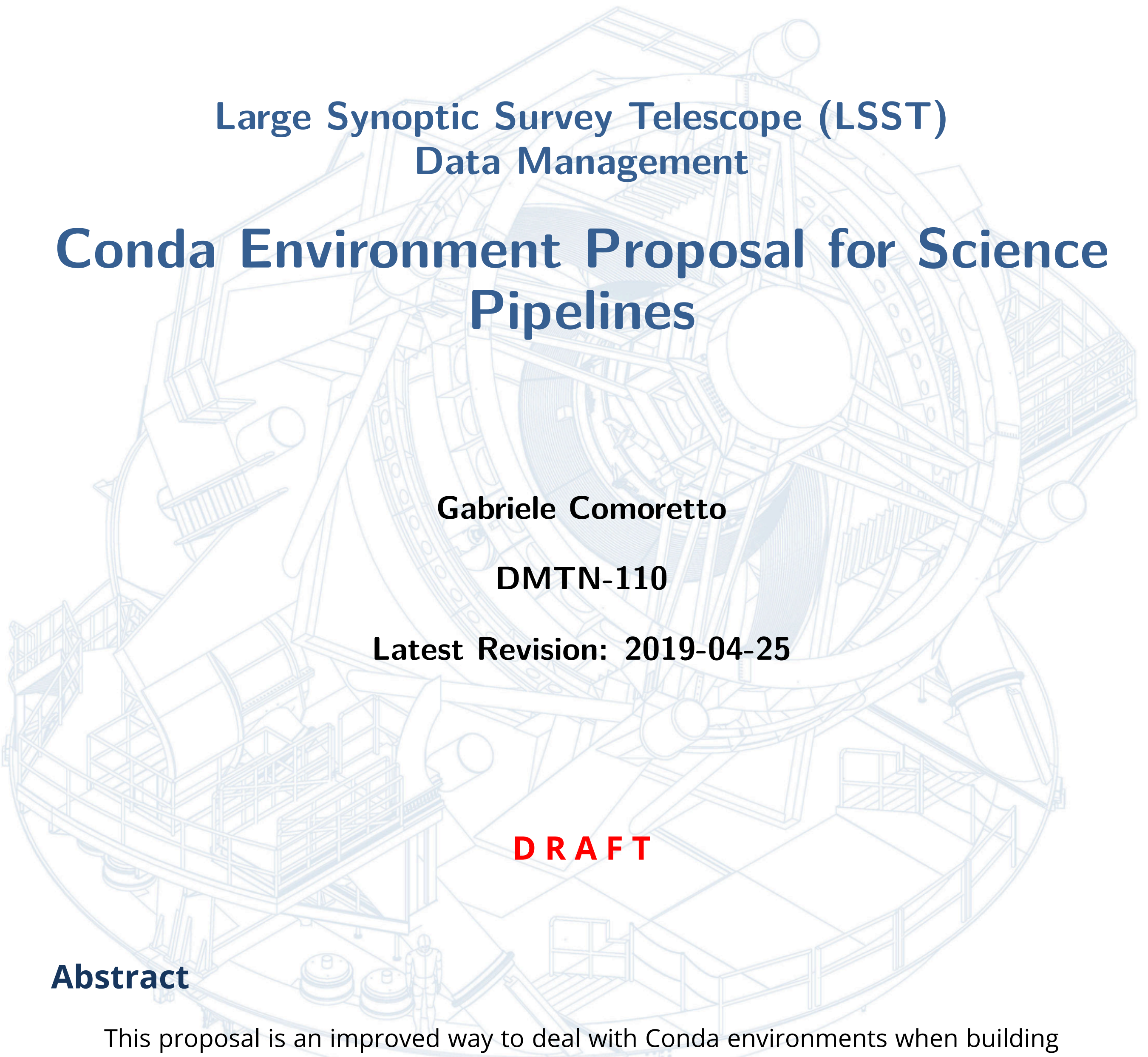
LARGE SYNOPTIC SURVEY TELESCOPE



Large Synoptic Survey Telescope (LSST)
Data Management

Conda Environment Proposal for Science Pipelines

Gabriele Comoretto

DMTN-110

Latest Revision: 2019-04-25

DRAFT

Abstract

This proposal is an improved way to deal with Conda environments when building the Science Pipelines software. The benefits of implementing it can be seen in both the development and release processes. The effort required for its implementation is limited since it reuses all tooling already in place.

Change Record

Version	Date	Description	Owner name
	2019-03-01	First draft	Gabriele Comoretto

Contents

Conda Environment Proposal for Science Pipelines

1 Introduction

The proposal made in this document is oriented to have a more functional approach in the environment management for Science Pipelines builds.

The benefits that can be obtained by implementing this proposal are mainly for developers, who will have more control over the environments they are using. This will permit a more dynamic evolution of the used environment, allowing more frequent upgrades to newer versions of packages in the environment.

The release process will benefit as the global build time for the Science Pipelines will be reduced. A reduced build time allows more frequent releases and faster development turnaround both locally and using continuous integration. This will be very important during commissioning and the initial operations phase, when multiple releases per day can be needed.

At the moment there are few EUPS packages *stubs* that are not always working as expected. Implementing this proposal will remove these packages, and the overall build process will be more robust.

Implementing this procedure will have very low impact on the DM development team. The immediate benefit will be a better management of the conda environment. Activities documented in section 5 can be scheduled afterward, depending on resource availability and avoiding impacts on project milestones.

1.1 Applicable Documents

1.2 Definitions

This document uses certain terms which are often overloaded. In this section we define these terms as intended in this document.

1.2.1 Environment

An environment is a set of libraries, executables and configurations, that are predefined for a specific context or function.

An environment is usually needed in the following contexts:

- software development: the environment includes all libraries and tools required to build and debug a software product
- software test and verification: the environment includes, on top of build and debug tools, additional tools to permit/facilitate the testing and validation activities.
- operations: the environment shall include only tools required for the execution of the operational products. It shall not contain any build or debug tool. It shall be optimized for the operational activities. [1]

These are the most common uses of an environment. Additional scenarios can be identified depending on the needs.

In some cases, for simplicity, the same environment is used in development, tests and operations, but this does not justify the inclusion of the environment definition in the Software Product itself.

1.2.2 Software Product

A Software Product is a component of the subsystem (DM) product tree.

The following activities are done on a SW Product:

- development activities
- test activities, usually unit tests and system test
- release activities
- packaging activities
- operational activities, in stand alone mode, in an operational pipeline or as part of a service. Some SW Products are libraries, therefore are not operationally, but used as dependencies by other Software Products.

[1] In case a problem is found in operation the first thing to do is to replicate it in a different environment, where also debug and further analysis is possible.

A Software Product shall correspond to a single repository. In other words, there shall be a correspondence *1:1* between Software Products and git packages. All the Software Prouct dependences are different Software Products and therefore released separately.

This is not the case of DM, where a Software Product is comprised of multiple git packages. The compromised solution in this case is to have a GitHub *metapackage* that identify the Software Product. All git packages composing the Software Product are dependencies in a GitHub *metapackage* and released at the same time. In this case, a git package shall be related only to one Software Product, and it is not by himself a Software Product. In other words, there is a correspondence *1:many* between Software Product and Git packages. There shall never be a *many:many* correspondence.

The following elements should not be part of a Software Product:

- build tools, otherwise we will have build tools deployed in operations.
- environment definition, since the environment depends on the final instantiation of the Software Product that are not visible in the development phase.

These elements shall follow a parallel development / release process.

Test data should also not be included in a Software Product. However including small datasets for enabling the unit tests is not a bad practice.

The content of this subsection is a *prerequisite* for any release procedure to be applied.

1.3 Software Release

A software release is a consciously identified tag of a Software Product repository documented with a software release note. The identifier is usually of the form `M.n.p`.

The tag in the Github repository and the software release note shall be sufficient to identify the release, and therefore for a developer to resolve the dependencies, build the binaries and execute the software.

The software release is a monolithic snapshot of the Software Product to be used as is, or in its corresponding binary package(s), by the downstream processes or users. In theory, it should be possible to identify the release with a checksum calculated automatically on the delivered package. That checksum can be used to identify the release installed in an environment, or included in a distribution.

1.4 Third Party Packages

Third party packages are those packages that are not developed in DM, and follow their own documentation and release process.

2 Current Implementation

2.1 Development Workspace Preparation

As of March 2019, the following steps are used to configure a build workspace for the Science Pipelines:

- clone the GitHub package `https://github.com/lsst/lsstsw`
- deploy the environment using `./bin/deploy`
- activate the environment by sourcing `bin/setup.sh`

This will make available locally all tooling required to build the science pipelines source code using the command `rebuild <-r ref> lsst_distrib`.

The environment deployed and activate using the above steps corresponds to a fixed revision of the *scipipe_conda_env* git package. This revision is hardcoded in the *lsstsw* build tool. It is also hardcoded in the continuous integration scripts versioned in the *jenkins-dm-jobs* GitHub repository.

2.2 Science Pipelines Distribution

Following the instructions provided in `pipelines.lsst.io`, it is possible to configure a build workspace using the *newinstall.sh* script. This script is available in the *lsst/lsst* git repository and it is tagged weekly together with the Science Pipelines weekly build.

The instantiation of the work space is described in the `pipelines.lsst.io` documentation page, and can be summarized as follows:

- download from GitHub the *newinstall.sh* script in an empty folder.
- execute the script: it will deploy the tools and the environment needed for the build. It will also create an additional shell script, *loadLSST.bash*, to be execute to activate the Conda environment. The script is generated different shell flavors also: csh, ksh and zsh.
- source the *loadLSST.bash* script in order to activate the environment

The workspace will be ready to build the Science Pipelines using `eups distrib install ....` builds. If *newinstall.sh* has been execute with the options *-ct*, binary packages will be downloaded from EUPS instead of building the source code, when building.

Note that the Conda environment reference is hardcoded in the *newinstall.sh* script.

The script may have changed from one week to an other. This imply that each EUPS weekly build may work only with its corresponding *newinstall.sh*.

2.3 Conda Environment Management

The repository *scipipe_conda_env* defines a list of Conda packages to be installed in the Conda environment. These packages are resolved by Conda from the default channels.

The *scipipe_conda_env* repository includes two different types of environment definition:

- **bleed**: the list of the Conda packages that are required by the Science Pipelines. No version is specified.
- **pinned**: the list of all Conda packages resolved in the environment. This includes the bleed packages and all required dependencies resolved automatically by Conda. The precise resolved version is given for each package listed in the file.

The *pinned* version of the environment definition is the one used to instantiate the environment for the Science Pipelines build as specified in sections 2.1 and 2.2.

The *bleed* version is used to update the *pinned* definition to the latest packages version that have been made available in the public Conda channels since the previous update. The *pinned* file is updated every 6 months approximately.

For the *linux* platform, an extra package is required, *nomkl*[2]. For this reason two different bleed files and two different pinned files are provided.

[2]See JiraDM-8146

3 Problems

3.1 Static Environment

As described in section 2, the environment deployed and activated using the actual version of the tooling is hardcoded in a few places.

This implies that, each time the environment is updated, the hardcoded value need to be updated, the tools redeployed, and restart from scratch any build or development activities.

The tools used should be able to deploy and activate a different environment, without the need to have it hardcoded.

3.2 Unknown Environment Status

In some cases, the science pipelines build is failing due to some packages misalignment. It is very common to see support requests on *slack* channels, that end up been a problem of Conda environment.

Since developers are not constrained to use the recommended environment, there should be an easy way to compare the active environment with a reference one, locally deployed or a reference in *scipipe_conda_env* git package.

3.3 High Time to Rebuild

The 17.0 release of the Science Pipelines stack comprises 128 packages.

All packages are rebuilt all the times a new *lsstsw* development workspace is configured. Each time a release or a release candidate are created, they are rebuilt also, with the effect to slow down considerably the release process.

Many of these packages are 3rd party libraries that are not available in the public Conda channels, or that require some changes in order to be used in the Science Pipelines stack.

These packages are not DM software, they are not formally part of the release, and they should be considered as part of the Conda environment. Moving them to the conda environment will decrease the build time and improve the release as is.

In order to move these packages to the Conda environment, there are two problems:

- create and maintain Conda packages suitable for DM usage
- dynamically manage the Science Pipelines Conda environment, in order to be able to include and use new versions of the required packages. This is the critical point.

See appendix A for a list of packages proposed to be moved to the conda environment.

The expected benefits are:

- more efficient environment management;
- reduced build time and therefore easier to release the Science Pipelines;
- have a more stable build system, since reducing the number of package to build every time will reduce the possibility of random failures.

3.4 Missing Usecases

As a result of the problems exposed above, following use cases should be taken in account:

- be able to deploy a new Conda environment without the need to update and have a new clone of *lsstsw*.
- be able to activate different Conda environments using a *lsstsw* command.
- be able to check the differences between Conda environments.
- record which Conda environment has been used for a specific build, *lsstsw* `rebuild`, and be able to activate it again.
- be able to activate the correct environment before running and EUPS build using `eups distrib install`.
- be able to run a CI build using a specific environment reference.
- be able to manage Conda packages for the third party libraries that requires DM rebuild or customization.

4 Proposed Implementation

Here follows the proposed changes required to satisfy the usecases from the previous section.

4.1 Define Environment References

The package *scipipe_conda_env* to be tagged in GitHub for each weekly and each time an EUPS build is created. This will guarantee that each EUPS weekly build will have a corresponding well known Conda environment, that can be used for rebuilt them. The Conda environmemt tag shall be in line with the EUPS tag. The SHA1 can be identified also, but should not be strictly needed.

All changes implemented in the *scipipe_conda_env* repository shall follow the usual development process. The tooling improvements proposed bellow, will permit the validation of those changes and include them in the development workflow.

4.2 Use Different Environments in lsstsw Git Package

A new command, or command line option, shall be added to lsstsw to deploy and activate a specific Conda environment reference from the *scipipe_conda_env* GitHub package.

The possible options should be:

- deploy/activate a specific *scipipe_conda_env* hash
- deploy/activate a specific *scipipe_conda_env* branch
- deploy/activate a specific *scipipe_conda_env* tag
- deploy/activate the tip of *scipipe_conda_env* master

This can be done with the following changes in lsstsw package.

4.2.1 Update deploy Command

The **deploy** lsstsw command should accept the environment reference **envref** as an additional command line option.

The default behavior should be the same as now. If no envref is provided, the usual fixed hash shall be deployed.

In case the envref is *master*, it shall deploy the latest master, if not already deployed. In this way, this option can be used to check if there is a newer master version of the conda environment and deploy it.

The environment shall have a unique name, not the fixed one that is currently used.

4.2.2 Update rebuild Command

The **rebuild** command should record the environment reference used for each build.

This will permit in the future to retrieve the information, in order to manually activate the same conda environment before running a build.

The **rebuild** command could also raise a warning, in case the active environment is different from the one used previously on the same package. However this is not a problem related to the environment management but a potential improvement in the build tooling.

4.2.3 New envconfig Command

A new command shall be added, that permits to choose an available environment to activate. The environment need to be already available locally, deployed for example with *deploy* command describe abode in refsec:deploy.

It shall accept one of following options:

- a *scipipe_conda_env* environment reference to activate
- an existing build number, in order to activate the environment that has been used for that build.
- list the available environments, ready to be activated
- compare the active environment with an other one, a *scipipe_conda_env* reference or a deployed environment

If no option is provided, the default reference shall be activated, in order to keep the current behavior.

4.3 Changes to lsst/lsst Git Package

The distribution script *newinstall.sh* should be updated in order to be able to deploy and activate the environment corresponding to the EUPS build that is going to be installed.

This imply that the generate scripts *loadLSST.** shall accept an environment parameter as input, and deploy the environment reference before activating it, if not already deployed.

4.4 Changes to Jenkins builds

The Jenkins jobs, for example *run-rebuild*, shall be updated in order to accept an environment reference as an optional parameter. In this way it will be possible to verify that the science pipelines builds with a new environment.

5 Following activities

The changes proposed in the previous section 4 can be implemented with minimal impact to the day to day activities.

The new functionalities introduced can already facilitate the management of the conda environment and development activities in general.

However, the resolution of the problems exposed in 3, requires the reduction of the number of packages included in the shared stack, moving third party libraries in the conda environment.

5.1 Conda Packaging 3rd Party Libraries

Some of the packages listed in appendix A are already available in Conda public channels or *PyPi* (see following 5.1.2). They are now included in the Science Pipelines stack as EUPS packages since when they were initially needed, they were not available in any Conda channels.

Some other packages require to be updated more often, therefore they have been included as EUPS packages, since the Conda environment is not sufficiently dynamic.

Packages that require customizations with substantial code changes, need to remain part of the stack, at least in a first phase. Packges that require minor patches can be handled by a new recipe.

A small team shall be defined in DM to maintain the environment and the needed Conda recipes.

5.1.1 Conda Packages Managed by DM

DM shall use the packages from the Conda public channels, if available.

DM shall maintain a Conda recipes for all those packages that are not available in Conda / PyPi, or that requires to be rebuild with a specific flag. This is the case of *boost*. These packages shall have a specific naming convention, so it will be clear they have been *re-packaged* by LSST DM.

A DM maintained Conda package shall be dismissed, not maintained anymore, as soon as the corresponding package available in the public Conda channel can be used in the Science Pipelines build.

5.1.2 PyPi Packages

The conda environment definition files in *scipipe_conda_env* can accept also pip package.

This implies that packages available in *PyPi* package repository and that can be installed in the environment. However, if available, the preference shall be given to the Conda package.

In case a pip package will bring into the environment too many PyPi dependencies, DM can decide to maintain a Conda recipe for it.

5.2 Change Control

Since maintaining conda recipes may require a considerable amount of effort, the DMCCB shall monitor when new packages are added to the environment or are used in the codebase.

Changes to the environment shall be proposed using RFC Jira issues.

Third party packages shall also be documented and included the DM product tree. Each package shall have an internal reference person.

5.3 More Packages to Conda

Considering the inclusion of the 3rd party libraries listed in appendix A as a first step, follows a list of activities that can improve the overall DM codebase management and facilitate the development activities.

- create conda packages for the main DM tools used for the build process, such as EUPS, sconsUtils, lsst_build.
- include in the Conda environment those third party libraries that are customized by DM with specific code changes, such as *starlink_ast*.
- define environments for different uses, for example development and software debug, operations, science data analysis, etc.
- create conda packages for all DM software.

Implementing the first bullet should not require much effort, but will simplify the tooling we are using for instantiate a new development workspace, such as *lsstsw/deploy* or *newinstall.sh*.

The second bulled is a long term activity, that will permit to apply semantic versioning to each git package, and therefore have proper dependency management.

A Packages to be Moved

The following (eups) packages are proposed to be moved from science pipelines to the conda environment:

- apr (*)
- apr_util (*)
- astrometry_net
- astropy (*)
- boost (*)
- cfitsio (*)
- coord
- doxygen (*)
- eigen (*)
- esutil
- fftw (*)
- flake8
- galsim
- gsl (*)
- healpy
- libyaml
- lmfit
- log4cxx (*)
- mathplotlib (*)
- minuit2 (*)
- mpi

- mpi4py
- mpich
- numpy (*)
- ndarray (*)
- pep8_naming (*)
- pybind11 (*)
- pycodestyle
- pyflakes
- pykg_config
- python_coverage
- python_execnet
- python_future
- python_psutil
- python_py
- pytest (*)
- pytest_cov (*)
- pytest_forked
- pytest_flake8 (*)
- pytest_session2file (*)
- pytest_xdist (*)
- python_mccabe
- pyyaml (*)
- requests
- scipy

- scons (*)
- sqlalchemy
- starlink_ast
- treecorr
- wcslib
- ws4py

In the DM-15495 exercise, we manage to build *afw* moving the 3rd party libraries marked with **(*)** to the conda environment definition.

Other packages, like for example *starlink_ast*, may require additional work in order to have them available as a conda package. This can be done in a second time.

A References

References

B Acronyms used in this document

Acronym	Description
CI	Continuous Integration
DM	Data Management
DMCCB	DM Change Control Board
DMTN	DM Technical Note
EUPS	Extended Unix Product System
LSST	Large Synoptic Survey Telescope
RFC	Request For Comment
SW	Software (also denoted S/W)