Astronomy and Computing 24 (2018) 45–51

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

Full length article

KERN

G. Molenaar *, O. Smirnov

SKA South Africa, 3rd Floor, The Park, Park Road, Pinelands, 7405, South Africa
Department of Physics & Electronics, Rhodes University, PO Box 94, Grahamstown, 6140, South Africa

ARTICLE INFO

Article history:
Received 22 June 2017
Accepted 26 March 2018
Available online 15 June 2018

Keywords:
Software
Packaging
Radio astronomy
Reproducible science
Containerisation

ABSTRACT

KERN is a bi-annually released set of radio astronomical software packages. It should contain most of the standard tools that a radio astronomer needs to work with radio telescope data. The goal of KERN is to save time and prevent frustration in setting up of scientific pipelines, and to assist in achieving scientific reproducibility.

1. Introduction

The installation of scientific software for use in astronomy can be notoriously challenging. The radio astronomy community has a limited number of dedicated software engineers and often lacks the human resources to dedicate to industrial-quality software development and maintenance. It is not uncommon that poorly written and badly maintained software packages of high complexity are used by scientists around the world, as these provide some set of algorithmic features not available elsewhere.

KERN has been created to facilitate scientists and system administrators. KERN is the name of the project to structure and automate the packaging of scientific software. The main deliverable is the KERN suite, a bi-annually released set of 3rd party open source scientific software packages.

The primary goals of KERN are as follows: to make it easier to install the scientific software, to supply a consistent working environment to a scientist and to improve interoperability and interchangeability.

Due to human resource limitations, we target KERN to one operating system and distribution. This is unfortunate, but recent development and adaptation of containerisation technology make it easier to deploy packaged software on most platforms. Limiting us to only one platform enables us to focus on performing the packaging only once, and to do this well. The choice of this one platform is then based on install base (desktop, server) and ease of use for user and developer (package creator).

The intended audience of this paper is threefold:

- the user, who is interested in using the software bundled with KERN,
- the developer, who wants his radio astronomy software available to a wider range of users and
- the system administrator who is setting up systems intended to be used for radio astronomical data reduction.

The name KERN means ‘core’ in Dutch and Afrikaans.

2. The target platform

A quick look around various astronomy institutes and universities shows that GNU/Linux and OS X are the most used personal computing platforms. On the server side it is without question GNU/Linux. Compared to OS X, GNU/Linux is an open source and a freely available platform, which is also a clear advantage. These facts combined result in the choice for Linux as the KERN platform.

However, further consideration was required before selection of the most suitable platform. There are various flavours of GNU/Linux, with different design philosophies and varying packaging formats. The most popular distributions can be split into two groups, RPM (Red Hat Package Manager) package and Debian package based distributions. There is no major advantage or disadvantage to either package format. Although there are diverse local trends, it is our experience that in the South African radio astronomy community, the majority of frequently utilised platforms are Debian based, specifically Ubuntu LTS. This distribution also appears to have popularity worldwide. Therefore, it was the most logical choice as KERNs target platform.

* Corresponding author at: Department of Physics & Electronics, Rhodes University, PO Box 94, Grahamstown, 6140, South Africa.
E-mail address: gijs@pythonic.nl (G. Molenaar).

https://doi.org/10.1016/j.ascom.2018.03.004
2213-1337/

3. Other packaging methods

In this section we discuss other packaging systems available to us, and motivate our choice not to use these.

3.1. Anaconda

A packaging effort named Anaconda is currently gaining popularity. Anaconda is a cross platform set of scientific software, with a focus on Python. It supports GNU/Linux, Windows and OS X. OS X is also often used as a desktop environment in radio astronomy. Supporting OS X would be advantageous for many end users.

We have performed experiments with packaging packages for Anaconda. Users have reported that Anaconda is easy to operate; however, at the time of writing, the packaging procedure is cumbersome for the developer. In effect, developers cannot generate the same high quality, seamlessly installable packages, as is achievable with native Linux packaging methods. Additionally, Anaconda lacks an equivalent to Debians Lintian, a packaging tool that dissects a Debian package and tries to find bugs and violations of the Debian policies.

Various software packages in KERN are not created with OS X support in mind thus requiring various modifications to the source code. Also, compilation procedures can vary greatly across platforms, doubling the packaging and maintenance effort if we would support OS X as a platform.

In addition, Linux distributions come with a large set of prepackaged software, which eliminates the need to package many dependencies. Using Anaconda would necessitate packaging up many dependencies ourselves.

The limitations of Anaconda led to the preference of Debian- over Anaconda packages.

3.2. Python and pip

The Python programming language has become the most widely used language in astronomy (Momcheva and Tollerud, 2015). Python comes with a package manager called pip. Pip assists in downloading and installing Python packages from the Python package index (PyPi). Another useful tool for setting up Python environments is called `virtualenv`. Virtualenv enables a user to set up one or more isolated Python environments without system administrator rights. The combination of these two tools enables the end user to set up various custom environments with specific versions of dependencies. For pure Python projects, pip and virtualenv are cross platform and independent of the host operating system package manager. However, pip is less suited for impure Python projects. Some Python libraries depend on non-Python run time libraries and/or non-Python development headers compile time, making them "impure". A recent improvement to the Python Packaging system is the introduction of wheels. Wheels are pre-compiled binary Python packages. These do not require compilation and will work if the packaged library does not have unusual requirements. An example of a independent binary wheel is Numpy. Numpy only depends on Python and a small set of system libraries. The Application Binary Interface (ABI) differs across host platforms and python versions, requiring a wheel for every platform and python combination. These are supplied on PyPi and the correct version is automatically selected by pip on installation.

Problems arise for example, with the python-casacore package, which has more unusual dependencies. Python-casacore depends on the casacore package and both packages need to have a matching ABI. If the version of casacore differs between compile time and run time, the library does not work. Pip does not have any control over reinforcing the shared library version. This would imply that we need to create, upload and maintain wheels for every casacore released. Moreover, with every casacore release we would also need to create a wheel for every OS and supported profile. The exponential growth of this cartesian product quickly becomes cumbersome for the package maintainers.

These limitations combined with pip's inability to handle non-Python libraries, make pip an ill suited candidate for our packaging effort. Nevertheless, this does not mean pip and KERN cannot be combined.

The approach we adopt is that we prepackage impure python libraries and bundle them with KERN. These Python packages are then precompiled against a set of specific library versions. This guarantees the ABIs always match up. Users can then augment the system python installation, or a Python virtual environment, with packages from the packaging index using pip.

Although KERN supports both Python 2 and Python 3, most Python packages in KERN only support Python version 2. For now, the only package that supports Python 3 is python-casacore. The KERN Python 3 package for casacore is named python3-casacore.

3.3. Collaboration with Debian

Ubuntu is directly based on Debian and thus is similar to Debian. Nonetheless, due to version differences in the bundled libraries in each distribution, KERN packages are unlikely to run on Debian. Fortunately the packaging procedure is identical for Debian and Ubuntu, which makes creating true Debian packages a matter of a recompilation of the source package.

In contrast, the build system, dependency management and library management for RPM is completely different. Porting our packages to RPM is non-trivial, and maintaining support for RPM based distributions would imply doubling the required effort.

We have established a collaboration with Debian developers, and some packages from KERN (e.g. `casacore` and `aoflagger`) have been incorporated into Debian directly. These packages have been uploaded to the Debian archive, and changes are synchronised between KERN and the Debian archive.

Not all KERN packages are suitable for uploading to Debian. Packages with a small user base or packages that are fragile and receive continuous fixes (as opposed to formal release) are not well suited to this distribution model, since it can take some time before a package ends up in a Debian release. For the more stable packages in KERN, we expect a continuation of this effort, with more packages ending up in Debian in the future.

4. Usage

To use the packages of KERN, one needs to add the KERN remote repository to the system. It is recommended to use the latest released version, which is KERN-2 at time of writing. KERN-2 is packaged for Ubuntu 16.04: using the packages on a different distribution or version will most likely fail. If running Ubuntu 16.04 is not an option on a particular system, we recommend using Docker, Singularity (see below) or a virtual machine.

The `add-apt-repository` command should be used to add the KERN repository to a system. Some packages in KERN depend on Ubuntu packages in the `multiverse` and `restricted` repositories (CUDA is an example of such a dependency). Once the local cache is updated using `apt-get update` the package cache can be searched using `apt-cache search` *`PACKAGE`* and packages can be installed using the `apt-get install` *`PACKAGE`* command.

In case of an unexpected fault, it is important to ensure that the latest versions of all packages are being used (by running `apt update` and `apt upgrade`), before reporting new issues.

Missing packages can be nominated for inclusion in KERN by requesting the packaging on the issue tracker.[1]

[1] https://github.com/kernsuite/packaging/issues.