Python Wheels in Galaxy

The wheels on the Galaxy go round and round

Galaxy ADMINS

February 18, 2016 GalaxyAdmins Meetup
Nate Coraor

https://docs.galaxyproject.org/en/release_16.01/admin/framework_dependencies.html

Overview

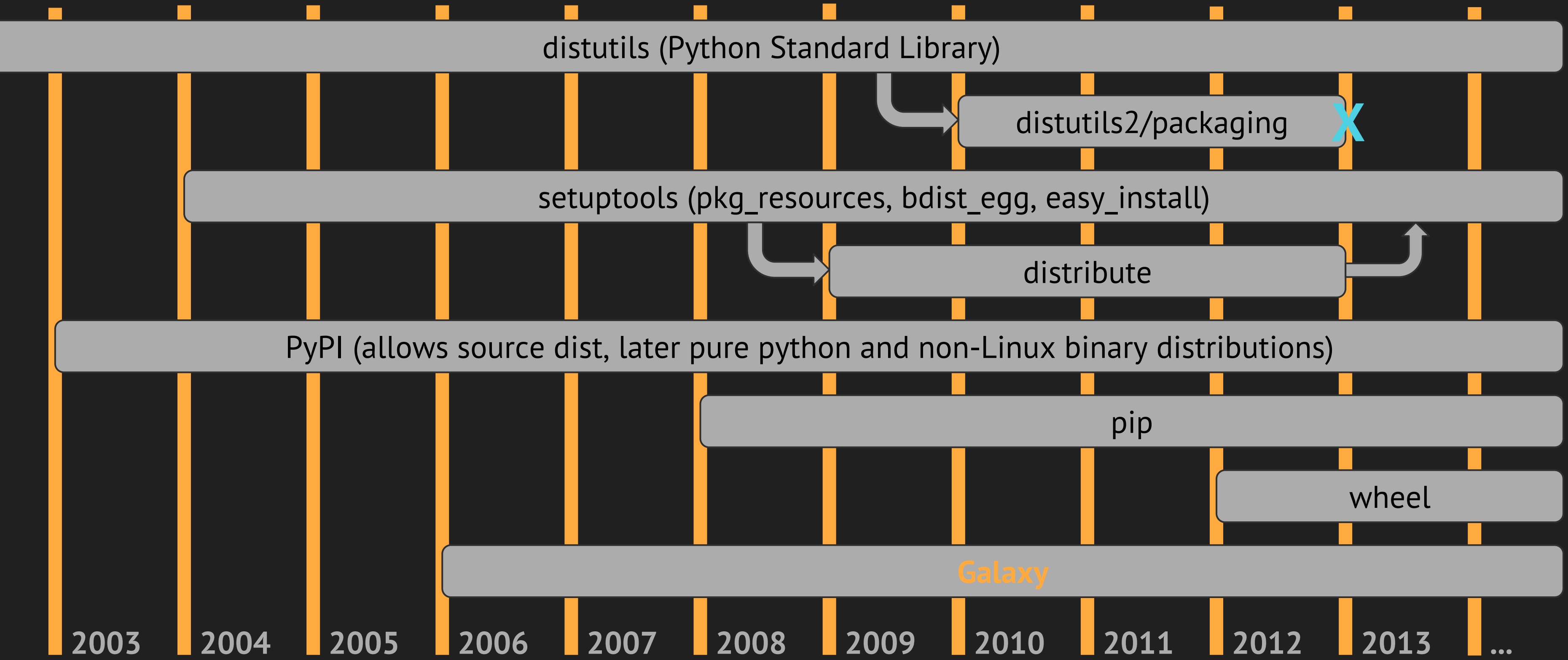
- A history lesson
- Wheels
- What has changed?
- Nitty gritty stuff
- Future plans

A history lesson: Galaxy dependency management

Most Python projects use `requirements.txt` and `pip` to manage dependencies, most of which are compiled at first run. Galaxy's approach has always been heavier:

- The Galaxy framework's dependency list is **long** and building the list is **slow**
- Many include C extensions and some are **not easy to compile**
- Some require **non-standard packages** to be installed on the system to build, but not to use
- For tool developers, we want Galaxy to be **quick, easy,** and **foolproof** to start
- Galaxy solution: prepackage and provide **binary distributions** (eggs, wheels)
- Wheels have not solved all the problems yet (but we'll get to that)

Python packaging history lesson



A history lesson: Eggs

Wheel, and egg before it, are “built” distribution formats

An (example) non-pure Python package installation process using setuptools:

```
% python setup.py install
running install
  running bdist_egg
    running egg_info
    running install_lib
    running build_py    # “build” pure python contents
    running build_ext   # build C extensions
                                    PyYAML-3.11-py2.7-linux-x86_64.
                                                               egg
```

A history lesson: Egg compatibility

Egg: setuptools' built distribution format (and intermediary format for installation)

```
  six-1.9.0-py2.7.egg
PyYAML-3.11-py2.7-linux-x86_64.egg
```

compatible Python version

compatible "platform" for non-pure Python packages

What is Python's ABI compatibility?

What is `linux_x86_64`'s ABI compatibility?

A history lesson: Python ABI compatibility

Prior to Python 3.4, the default Python build uses UCS-2 (UTF-16) encoding, but most vendor (Linux distribution) Pythons use UCS-4 (UTF-32) encoding. *Binary artifacts created on these two are incompatible!*

Also incompatible: Debug, PyMALLOC build-time options (but in practice, all CPythons are debug disabled, PyMALLOC enabled), the Python interpreter itself (PyPy, Jython, etc.)

Galaxy solution: hack UCS string into egg filename, ignore debug, PyMALLOC, and interpreter. See eggs.galaxyproject.org:

```
  PyYAML-3.10-py2.7-linux-x86_64-ucs2.egg
  PyYAML-3.10-py2.7-linux-x86_64-ucs4.egg
```

A history lesson: Linux ABI compatibility

What does `linux_x86_64` mean? Are wheels built on Debian compatible with Red Hat Enterprise Linux?

NOPE.

sorta

1. Code built on a certain version of GLIBC can not be run with an older GLIBC due to symbol versioning
2. Some libraries linked by a package are not part of a “standard” Linux install
3. Other libraries (e.g. `libz`) do not version symbols and rarely version filenames, but their ABIs may not be compatible

A history lesson: Linux ABI compatibility

Galaxy solution:

1. Build eggs on a system with a very old GLIBC (Debian etch/4.0)
2. Statically link non-standard libraries directly into the wheel
3. Doesn’t end up being a problem in practice

Downsides: Statically linked libraries will not receive security/bug updates like the vendor-provided version would, do not include vendor modifications, may not include all features

Wheels: What are wheels?

Wheels are a “built” distribution format with a defined specification[1] designed to solve most of the deficiencies in the (unspecified) egg format.

Very similar to eggs in practice (zip file, most metadata files the same), but wheel does not include Python bytecode (`*.pyc`) files and the wheel filename has more descriptive tags[2][3] which handle the Python ABI compatibility problem:

```
PyYAML-3.11-cp27-cp27mu-linux_x86_64.whl
```

But the generic `linux_x86_64` tag is still there. More on that later...

Wheels: How does Galaxy use wheels?

Beginning with the 16.01 Galaxy release, eggs are replaced with wheels. Wheels are fetched from wheels.galaxyproject.org.

This is done by installing a *modified version of* `pip`. The modifications are necessary to add specificity to the *platform tag* section of the wheel filename (e.g. `linux_x86_64`)

Three types of wheels are supported:

- Pure python
- Platform specific
- Platform generic

Wheels: Platform specific wheels

We detect the Linux distribution and version so we can produce wheels like:

```
psycopg2-2.6.1-cp26-cp26m-linux_x86_64_ubuntu_15_04.whl
```

- Pros:
  - `libpq` is dynamically linked - the (maintained) vendor version will be used
- Cons
  - The the target system will need to have `libpq` installed
  - One wheel per interpreter/UCS/arch/distro combination: https://wheels.galaxyproject.org/simple/psycopg2/

Wheels: Platform generic wheels

We build on Debian 6 and produce wheels “that should work on any modern Linux” like:

```
PyYAML-3.11-cp27-cp27mu-linux_x86_64.whl
```

- Pros:
  - Great for simple wheels with no non-standard external dependencies
  - Only one wheel per interpreter/UCS/arch combination (no distribution)
- Cons
  - Cannot link to “non-standard” libraries

But… see manylinux (more on this later)

What has changed?

- Some things change
  - The `eggs/` directory is no longer used
  - A virtualenv is created at `.venv/` and dependencies are installed there
  - Galaxy development versions use unpinned dependencies (soon)
- Some things stay the same
  - Galaxy still manages its dependencies upon startup
  - For Galaxy releases, pinned versions of dependencies are installed (for reproducibility)
- However
  - The standard tooling is used: `pip` and `wheel`
  - You can use standard `pip` to install dependencies from source without using our wheels

What has changed?

```
% sh run.sh
New python executable in .venv/bin/python
Installing setuptools, pip, wheel...done.
Activating virtualenv at .venv

Ignoring indexes: https://pypi.python.org/simple
Collecting pip==8.0.2+gx2
  Downloading https://wheels.galaxyproject.org/packages/pip-8.0.2+gx2-py2.py3-none-any.whl (1.2MB)
    100% |████████████████████████████████| 1.2MB 37.2MB/s
Installing collected packages: pip
  Found existing installation: pip 7.1.2
    Uninstalling pip-7.1.2:
      Successfully uninstalled pip-7.1.2
Successfully installed pip-8.0.2+gx2

Collecting bx-python==0.7.3 (from -r requirements.txt (line 2))
  Downloading https://wheels.galaxyproject.org/packages/bx_python-0.7.3-cp27-cp27mu-linux_x86_64.whl (1.7MB)
    100% |████████████████████████████████| 1.7MB 37.5MB/s
...
```

What has changed?

```
Installing collected packages: bx-python, ...

Collecting psycopg2==2.6.1 (from -r /dev/stdin (line 1))
  Downloading https://wheels.galaxyproject.org/packages/psycopg2-2.6.1-cp27-cp27mu-linux_x86_64_debian_stretch_sid.whl
(357kB)
    100% |████████████████████████████████| 360kB 54.5MB/s
Installing collected packages: psycopg2
Successfully installed psycopg2-2.6.1

Unsetting $PYTHONPATH
Activating virtualenv at .venv
galaxy.queue_worker INFO 2016-02-17 16:34:41,651 Initalizing main Galaxy Queue Worker on sqlalchemy+postgresql:///nate
tool_shed.tool_shed_registry DEBUG 2016-02-17 16:34:41,666 Loading references to tool sheds from .
/config/tool_sheds_conf.xml.sample
tool_shed.tool_shed_registry DEBUG 2016-02-17 16:34:41,666 Loaded reference to tool shed: Galaxy Main Tool Shed
galaxy.app DEBUG 2016-02-17 16:34:41,666 Using "galaxy.ini" config file: /home/nate/git/galaxy/config/galaxy.ini
...
```

Nitty gritty stuff: Virtualenv placement

For a managed installation you may not want to use `.venv/` (or you may be using uWSGI, `galaxy-main`, etc.).

1. Create your own virtualenv
2. Activate it
3. Use `scripts/common_startup.sh` (and `run.sh`, if applicable) with the `--no-create-venv` option

More complicated stuff (using with Conda, supervisor, uWSGI) in the documentation

Nitty gritty stuff: Platform binary compatibility

Some Linux distributions (like CentOS and Red Hat Enterprise Linux) have the same ABI and are binary compatible, meaning that wheels produced on one should work on the other. Enter `binary-compatibility.cfg`:

```
{
     "linux_x86_64_centos_6_7": {
          "install": ["linux_x86_64_rhel_6_7"]
     },
     "linux_x86_64_centos_6": {
          "install": ["linux_x86_64_rhel_6"]
     }
}
```

Future plans: better platform generic wheels

While all of this *Galaxy pip/wheel* work was going on, some scientific Python folks went about solving the same problems in a similar way, and the result is **manylinux**[4]

Wheels using the `manylinux_x86_64` platform tag are built on CentOS 5 and work much like Galaxy's "platform generic" wheels, except:

1. The manylinux PEP[5] includes a specific list of allowed external libraries
2. They do not reuse the generic, underspecified `linux_x86_64` platform tag
3. The `auditwheel` utility ensures that a wheel is manylinux1 compatible
4. `auditwheel` can bundle unallowed external libraries into the wheel

In the near future (16.04 release?), generic wheels will be replaced with manylinux

Future plans: Standardize platform specific wheels

Platform specific wheels still have a useful case: wheel deployers may prefer not to use a bundled version of an unallowed library, and instead prefer to rely on a system package version

For this reason, I plan to write a PEP for official specific platform tag support, and hope to get this support added into upstream `pip`

Future plans: pip install galaxy, packages

Galaxy installations are cumbersome to manage. Wheel support makes simplifying this easier. I am working on support to be able to `pip install galaxy` (aiming for the 16.04 release). See pull request #921[6], issue #1152[7]

Additionally, we should be able to create Galaxy packages in other formats (e.g. `.deb`, `.rpm`). See issue #1472[8]

References

1. PEP 0427 -- The Wheel Binary Package Format 1.0 https://www.python.org/dev/peps/pep-0427/
2. PEP 0425 -- Compatibility Tags for Built Distributions https://www.python.org/dev/peps/pep-0425/
3. PEP 3149 -- ABI version tagged .so files https://www.python.org/dev/peps/pep-3149/
4. The manylinux project https://github.com/pypa/manylinux
5. PEP 0513 -- A Platform Tag for Portable Linux Built Distributions https://www.python.org/dev/peps/pep-0513/
6. Galaxy Pull Request #921: [WIP] Make Galaxy packageable/installable https://github.com/galaxyproject/galaxy/pull/921
7. Galaxy issue #1152: Support `pip install galaxy` with dependencies https://github.com/galaxyproject/galaxy/issues/1152
8. Galaxy issue #1472: Distribute Galaxy packages https://github.com/galaxyproject/galaxy/issues/1472

Galaxy Documentation: Galaxy Framework Dependencies: https://docs.galaxyproject.org/en/release_16.01/admin/framework_dependencies.html