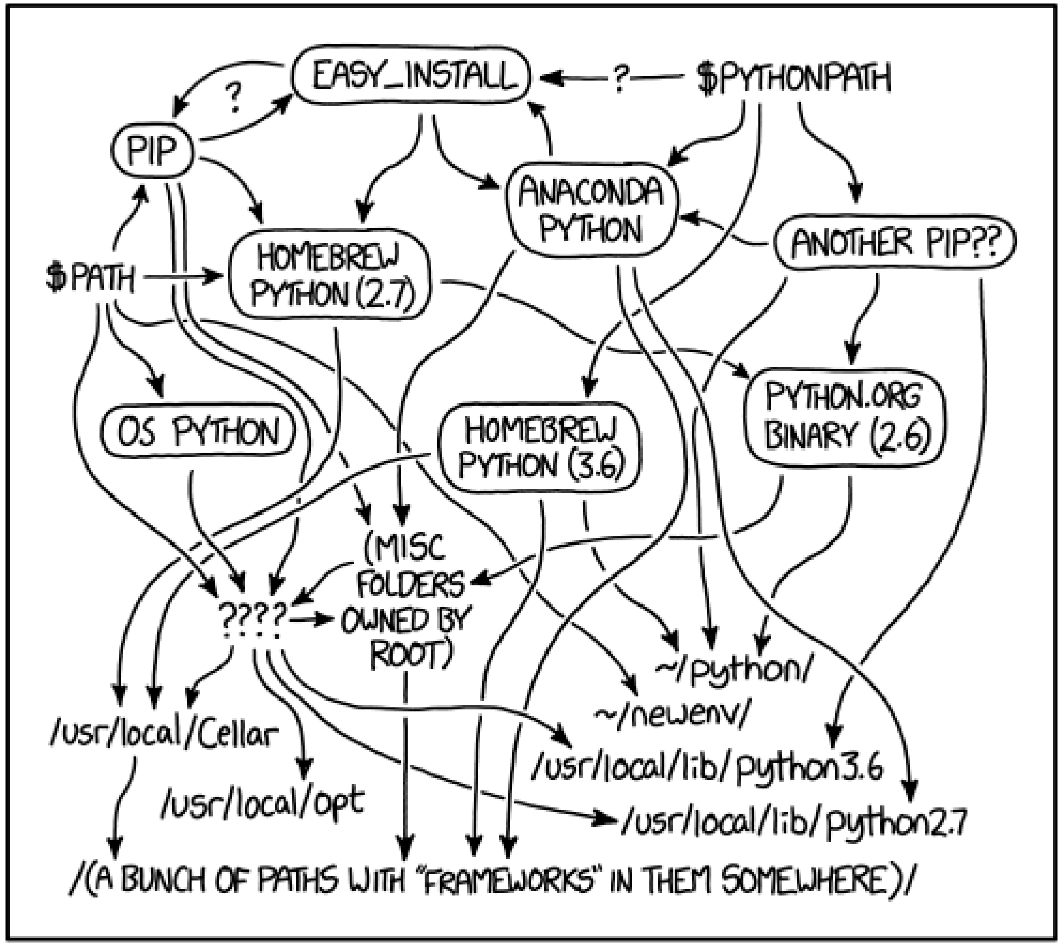
Changes to the Python Infrastructure

25 July 2018

OLCF User Conference Call

Presented by Matt Belhorn

Python environments can get messy...



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY ~~LAPTOP~~ HAS BEEN DECLARED A SUPERFUND SITE.

supercomputer

...more so in HPC

Credit: https://xkcd.com/1987/

New Approach to Providing Python and Python Extensions

- Anaconda Distributions
  - Includes commonly used packages out-of-the box
  - Extended, customized with conda environments
- Minimal native python environment modules
  - Can use wheels, but doesn’t rely on pre-compiled binaries at system level
  - OLCF will no longer provide modules for every extension
  - Extend the base with your own virtualenvs
- Complete DIY is always an option
  - All of this is in userspace anyway, tune your environment from the ground up
  - Choose from native python, anaconda/miniconda at the python version you need

Anaconda Basics

- Provided as modulefile on Titan, Eos, Rhea
  - `python_anaconda{M}/{M}.{m}.{u}-anaconda{M}-{REL}`
- Not yet provided on Summit, Summitdev
  - Coming soon, in meantime, see DIY in appendix
- `PYTHONUSERBASE` set to unique location
  - `${HOME}/.local/${HOST}/python/${MODULENAME}`
- Relies heavily on pre-compiled binaries
- Extended through *conda environments*
  - `conda` similar to `pipenv`: package manager, virtual environment all-in-one

{M}: Python Major Version
{m}: Python minor Version
{u}: Python micro Version
{REL}: Anaconda Release

Conda Initial Setup

```
cat $HOME/.condarc
envs_dirs:
  - /ccs/proj/<projid>/<user>/virtualenvs/<host>...
  - /ccs/home/<user>/.local/share/virtualenvs/<host>...
```

- Setup your conda config to put conda envs on NFS filesystem.
- If need on CN, choose `/ccs/proj/<projid>`; not `$HOME` or Lustre.
- Recommended to use env names that separate project and host.

Creating Conda Environments

```
conda create <pkgs>... -c <channel> -p <path>
source activate <conda_env>
conda install numpy pyyaml [<pkg>…]
source deactivate
```

- Pre-compiled packages (with external dependencies) pulled from *channels*
- Binaries typically highly optimized for generic architectures
- Pre-compiled binaries don’t always work on HPC resources
- Building packages from source possible, discussed later

SLES11 Crays vs Anaconda

- Pre-compiled binaries
  - Assumes/ships with OMPI, must re-build packages if conflicts with machine
- `glibc` on SLES11 (Titan, Eos) older than packagers expect
  - Little can be done about this. Rebuild package from source
  - Packages that ***require*** newer glibc (e.g. Tensorflow) must be provisioned other ways
- Anaconda libraries generally collide with OS libs.
  - Relative-RPATH’s: works fine on Cray FENs, fail on CNs due to ALPS.
  - Add anaconda libs to `LD_LIBRARY_PATH` only on CNs:
  - `aprun -e LD_LIBRARY_PATH="${CONDALIBS}:${LD_LIBRARY_PATH}"  ...`

Native Python (from environment modules)

- `module load python/{M}.{m}.{u}`
- Basic packages included in root `site-packages`
  - virtualenv, pip, setuptools, etc.
  - Will consider generic, unoptimized numpy/scipy/matplotlib, other pure-python extensions
- OLCF no-longer providing lots of extensions via environment modules
  - Some packages still be provided by environment modules. Eg, mpi4py
  - Bindings for specific external frameworks no longer provided this way (h5py, pynetcdf, etc)
  - Packages with specific external dependencies (scipy, numpy) also not provided (for now)
  - Build these for your own needs; in virtualenvs or arbitrary prefixes

Providing your own extensions

- Python packages can exist anywhere: add to `PYTHONPATH`
- But avoid `PYTHONPATH` pollution
  - packages for varying python versions, machine architectures, and external dependencies
  - Major problem providing packages via environment modules
  - Don’t modify the `PYTHONPATH` in your shell init files
- Consider using virtualenvs

Venv/Virtualenvs

- Best practice: provides isolated python environment
- python3: `python3 -m venv <path>`
- python2: `virtualenv <path>`
- Activate several ways
  - from command line: `. <path>/bin/activate`; `deactivate`
  - from shebang line: `#!/path/to/venv/bin/python3`
- Load all environment modules first, deactivate to before changing environment modules

Building Packages from Source

- Can be trickier in HPC environment
  - Still better for your application to have isolated, reliable dependencies
  - Much easier to manage at a personal level than for site-provided environment modules that work for everyone
- Let `pip` do it for you: `CC=cc MPICC=cc pip install --no-binary <pkg> <pkg>`
- Or use distutils/setuptools: `python setup.py install`
  - Check package docs. May need to get creative passing HPC environment parameters.
- See appendix for expanded examples

General Guidelines

- Follow PEP394 (https://www.python.org/dev/peps/pep-0394/)
  - Call `python2` or `python3` instead of ambiguous `python`
  - Same in scripts: `#!/usr/bin/env python2` or `#!/usr/bin/python3`
- Don't mix anaconda/conda envs and native python/virtualenvs
- Avoid mixing virtualenvs and environment modules
  - Environment module changes generally conflict with virtualenvs
- Use venv python in script shebang lines
  - eg: `#!/path/to/your/venv/bin/python3`
- Use care with `pip install --user ...` -
  - ensure `$PYTHONUSERBASE` is unique to python version and machine architecture.
  - `$HOME` *is shared on a variety of architectures.*

Current status

- Anaconda
  - Changes are already live on Titan, Eos, Rhea
  - Coming soon to Power systems
- Native python
  - Extension environment modules will be deprecated in coming weeks.
  - Future versions of Python will have minimal extensions, in root site-packages.
  - External environment module python extensions for older python versions will remain as-is until further notice, but won’t work with newer python interpereters.
  - Additions to python interpereter root site-packages are rolling work-in-progress, check site-packages before loading additional extensions from modulefiles.

Feedback Welcome

Don’t hesitate to contact us: help@olcf.ornl.gov

- Report problems you discover with the software we provide.
- Get help building your application in a virtualenv/conda env
- Suggest certain packages be included in root site-packages

Appendix

Resources

- Venv/Virtualenv
  - venv (py3): https://docs.python.org/3.6/library/venv.html
  - virtualenv (py2): https://virtualenv.pypa.io/en/stable/
- Anaconda Documentation
  - conda: https://conda.io/docs/user-guide/getting-started.html
  - Installing your own: https://conda.io/docs/user-guide/install/linux.html
- Check the package documentation
  - Installation procedure in package docs is often not as simple as described when applied to an HPC environment.

Source Installs with Pip

```
module swap PrgEnv-pgi PrgEnv-gnu
module load cray-hdf5-parallel/1.10.2.0  # sets HDF5_DIR envvar
source /path/to/venv/bin/activate
CC=cc HDF5_MPI=”ON” HDF5_VERSION=1.10.2 pip install -vv --no-binary=h5py h5py
```

```
module swap PrgEnv-pgi PrgEnv-gnu
module load cray-hdf5/1.10.2.0  # sets HDF5_DIR envvar
module swap craype-istanbul craype-mc8
source /path/to/venv/bin/activate
CC=cc HDF5_VERSION=1.10.2 pip install -vv --no-binary=h5py h5py
```

- Most python packages assume the use of GCC. Use the GCC PE or GCC raw compilers when possible for easiest builds.
- Use the `--no-binary` flag to build packages from source.
  - Comma separated list of package/dependency names
  - Can use :all: to build all dependencies from source.
  - Use verbose output `-vv` to identify build errors.
- Most packages accept environment variables to configure source builds. Check package documentation.
- If package will run on login/batch nodes and using Cray wrappers, target the login node architecture.
- Must have all external dependencies loaded at runtime (PrgEnv-gnu, cray-hdf5, etc. excluding compiler target)

Setuptools/Distutils Source Builds

```
module swap PrgEnv-pgi PrgEnv-gnu
module load cray-hdf5-parallel/1.10.2.0
. /path/to/venv/bin/activate
python setup.py configure --hdf5=$HDF5_DIR
python setup.py configure --hdf5-version=1.10.2
python setup.py configure --mpi
python setup.py install
```

- Pip source builds actually do this under the hood.
- Allows complex builds by
  - editing `setup.cfg` (or other, see package docs)
  - passing arguments to `setup.py configure`
  - Global distutils options can be set in your user-config (~/.pydistutils.cfg) or a temporary (preferred) site-config using `setup.py setopt`/`setup.py saveopt`. (https://setuptools.readthedocs.io/en/latest/setuptools.html#configuration-file-options)
  - See `setup.py --help-commands` for build steps

Conda source builds

- Try to use conda first w/ alternate channels
  - https://conda.io/docs/user-guide/tasks/manage-pkgs.html
- Can use pip or setuptools to install PyPI packages as normal with venv
  - This doesnt’ take advantage of isolated libraries provided by pre-built conda packages
- Use conda-build to make your own “portable” conda packages from recipes.
  - More complex, you’re bundling dependencies into a pre-built binary nominally for distribution from anaconda repositories (channels).
  - https://conda.io/docs/user-guide/tasks/build-packages/install-conda-build.html#install-conda-build
  - https://conda.io/docs/user-guide/tutorials/build-pkgs-skeleton.html
  - https://conda.io/docs/user-guide/tutorials/build-pkgs.html

DIY: Install your own Anaconda/Miniconda/Native CPython

```
module unload python xalt
module unload PrgEnv-pgi PrgEnv-gnu PrgEnv-intel PrgEnv-cray
module load PrgEnv-gnu dynamic-link

HOST="${HOSTNAME%%-*}"
PROJID="stf007"
PREFIX_BASE="/ccs/proj/$PROJID/$USER/${HOST}"
PREFIX="${PREFIX_BASE}/opt/anaconda3/5.2.0"

# Get installer from https://repo.continuum.io/archive/ for your target
# architecture: Linux-x86_64 or Linux-ppc64le
wget https://repo.continuum.io/archive/Anaconda2-5.2.0-Linux-x86_64.sh
md5sum Anaconda2-5.2.0-Linux-x86_64.sh  # Verify the hash
chmod a+x Anaconda2-5.2.0-Linux-x86_64.sh
./Anaconda2-5.2.0-Linux-x86_64.sh -b -p $PREFIX

# Add anaconda deploy to the PATH
if [ -z "$OPATH" ]; then
  OPATH="$PATH" # Save current PATH for easy restore
  export PATH="$PREFIX/bin:$PATH"
fi

which pip
which python

# Update pip to latest version
pip install -v --upgrade pip

# If on a Cray,
if [ -n "${CRAYPE_VERSION}" ]; then
  # Install mpi4py in anaconda root, built against cray-mpich
  CC=cc MPICC=cc pip install -v --no-binary :all: mpi4py
else
  pip install -v --no-binary :all: mpi4py
fi
```

- Be sure to add to your PATH at runtime.
- Consider also setting:
  - unset PYTHONSTARTUP
  - export PYTHONUSERBASE=/path/to/choice
- **Example script to install Anaconda on a Cray** *(change PREFIX_BASE before use)*: https://code.ornl.gov/m9b/nccs_python_reference/blob/master/install_anaconda_on_cray.sh
- **Example script to install both py2 and py3 natively**: https://code.ornl.gov/m9b/nccs_python_reference/blob/master/build_raw_python.sh