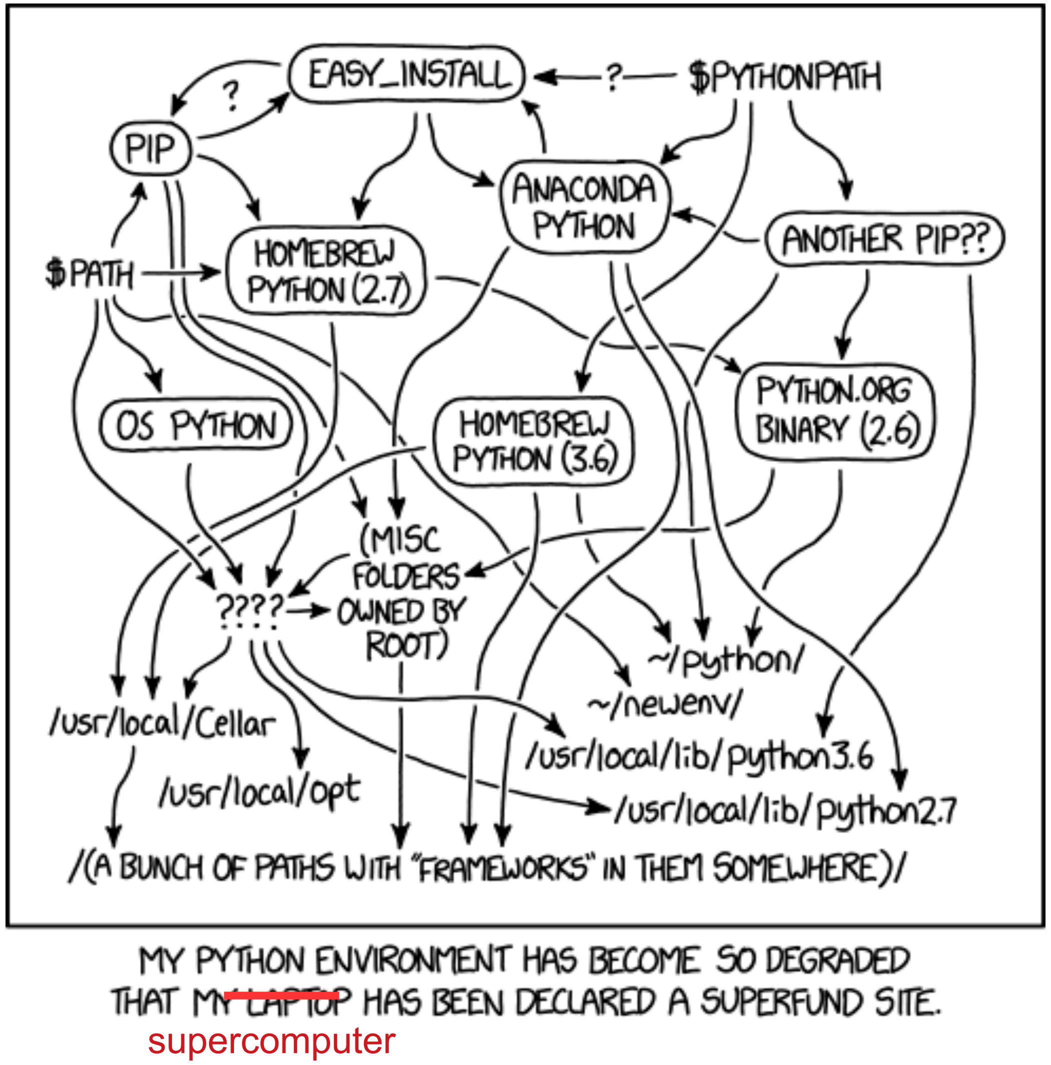
Python on Summit

Matt Belhorn
Oak Ridge Leadership Computing Facility
Summit Training Workshop
11 February 2019

Python environments can get messy...



...more so in HPC

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

Provided Python Environments and Extensions

- Anaconda Distributions
  - Includes commonly used packages out-of-the box
  - Extendable/customizable with `conda` environments
- Minimal native python environment modules
  - OLCF can't feasibly provide env-modules for every extension
  - Extend the standard library with `virtualenvs`
- DIY is always an option
  - More work, but also more stable and tuned to your needs.

Anaconda

- Provided via modulefile on Summit, Ascent
  - `python/{M}.{m}.{u}-anaconda{M}-{REL}`

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

- `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

Native Python (from environment modules)

- Provided via module files
  - `module load python/{M}.{m}.{u}`
  - Versions 3.7.0 and 2.7.15 from Jan 1
  - 3.5.2 and 2.7.12 also on some systems

Native Python (from environment modules)

- Basic packages included in root site-packages*
  - `virtualenv`, `pip`, `setuptools`, etc for setting up virtualenvs.
  - Only for python interpreters outside a compiler environment. Unload all compilers to get a python environment with these pre-installed to setup a virtualenv.
- OLCF no-longer providing lots of extensions via environment modules
  - Some packages still provided by environment modules. Eg, `mpi4py`
  - Will consider generic, unoptimized `numpy/scipy/matplotlib`, and pure-python extensions
  - Generally you will need to setup a virtualenv for additional extensions

Native Python (from environment modules)

- Bindings for specific external frameworks no longer provided this way (`h5py`, `pynetcdf`, etc)
  - Packages with specific external dependencies (`scipy`, `numpy`) may be present but not recommended for use
  - Build these for your own needs
- Extension env modules do not load their dependencies
  - Neither external libraries
  - Nor *extra (often required) python extensions*

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 problems exist providing packages via environment modules
  - Not recommended to modify the `PYTHONPATH` in your shell init files
- Best approach: use virtualenvs or conda envs

Creating Conda Environments

- Pre-compiled packages pulled from *channels*
  - Generally comes with pre-compiled external dependency libraries
  - Binaries typically optimized for generic architectures
  - Pre-compiled binaries don’t always work on HPC resources
  - Building packages from source possible

```
conda create <pkgs>... -c <channel> -p <path>
source activate <conda_env>
conda install numpy pyyaml [<pkg>…]
pip install --no-binary mpi4py install mpi4py
source deactivate
```

Venv/Virtualenvs

- 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 before changing environment modules

Building Packages from Source

- Can be tricky in HPC environment
- Easier to manage at a personal level than for site-provided environment modules that work for everyone
- Let `pip` do it for you:

```
[CC=gcc MPICC=mpicc] pip install \
                          -v --no-binary <pkg> <pkg>
```

- Or use distutils/setuptools: `python setup.py install`
  - Check package docs. May need to get creative passing HPC environment parameters.

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`
- Python environments generally don’t mix
  - conda envs
  - Virtualenvs
  - Native python

General Guidelines

- Avoid mixing virtualenvs and python extension env 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.

Thanks for listening

- Questions or comments regarding the Summit programming environment?

  Contact `help@olcf.ornl.gov`

  We’re happy to help with any issues and questions you have.

Backup

What about ML/DL?

- Tensorflow, PyTorch, Keras, etc. usually require extra dependencies.
- Some of these claim to be provided by Anaconda for ppc64le, but that's not always a truthful claim.
- We are working on other, non-anaconda solutions for these packages.
- In the meantime…

What about ML/DL?

```
module load python/3.7.0-anaconda3-5.3.0
conda create tensorflow-gpu \
             keras-gpu \
             ipython \
             -p ~/tf_conda_env
bsub -P stf007 -n1 -W 60 -Is $SHELL
source activate ~/tf_conda_env
jsrun … ~/tf_keras_test.py
```

```python
#!/usr/bin/env python3
import tensorflow as tf
import keras
mnist = keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = keras.models.Sequential([
  keras.layers.Flatten(),
  keras.layers.Dense(512, activation=tf.nn.relu),
  keras.layers.Dropout(0.2),
  keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Matplotlib Backends

- Matplotlib backends
  - In scripts:
    ```
    import matplotlib
    matplotlib.use(‘tkagg’)  # not case sensitive
    import matplotlib.pyplot as plt
    ```
  - Globally:
    ```
    cat ~.matplotlib/matplotlibrc
    backend : tkAgg
    ```

Changes to python code not being honored?

- Python compiles source to bytecode caches at runtime
  - Files/dirs such as `` `__pycache__` ``, `` `*.pyc` ``, `` `*.pyo` ``
- Old bytecode may be used if source changes undetected
- Solution: `` `export PYTHONDONTWRITEBYTECODE=1` ``
  - Useful when actively developing python code
  - Lesser performance, not recommended for production runs

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.

Conda Initial Setup

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

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

Source Installs with Pip

- Most python packages assume use of GCC.
- Use the `--no-binary` flag to build packages from source.
  - Comma separated list of packages or `:all:`
  - Use verbose output `-vv` to identify build errors.
- Check package documentation for configuration.
- External dependency env modules must be loaded at runtime

```
module load hdf5  # sets HDF5_DIR envvar
source /path/to/venv/bin/activate
CC=gcc HDF5_MPI=”ON” HDF5_VERSION=1.10.2 pip install -v --no-binary=h5py h5py
```

Setuptools and distutils Source Builds

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

Setuptools and distutils Source Builds

```
module load hdf5
. /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
```

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 doesn't use libraries provided by pre-built conda packages
- Use `conda-build` to make your own "portable" conda packages from recipes.
  - More complex; bundles dependencies into a pre-built collection for distribution, nominally from anaconda 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.html