Advanced Anaconda Training

Dan Udwary and Mario Melara
DSEG / UEG
Joint Genome Institute - June 7th, 2018
Agenda

- **Conda Basics and Concepts Recap**
  - A quick overview of the basics and some concepts to get everyone on the same page
- **Tips and Tricks**
  - Ways to effectively use Anaconda
- **Packaging**
  - A run through of how to create packages for some of your tools
- **Discussion**
  - JGI software policy
Conda Basics and Concepts Recap
Conda Basics Recap

- Python provided at NERSC on Denovo via modules
  - module avail python
    - python/2.7-anaconda (default)
    - python/3.5-anaconda
    - python/3.6-anaconda
- Can use Miniconda
  - Anaconda - conda, Python, 200 packages
  - Miniconda - conda, Python
- Installing packages
  - conda install numpy
  - conda install numpy=1.13 (other e.g. => == |)
- Creating environments
  - conda create --name my_env numpy
Conda Environments

Conda Base - module load python

/usr/common/usg/languages/python/2.7-anaconda

bin  lib  shared  include
Conda Environments

Conda Base - module load python

$ conda create --name my_env numpy
$ source activate my_env

PATH=~/conda/envs/my_env/bin:$PATH
Conda Channels

- Channels are URLs where packages are fetched
  - [https://repo.continuum.io/pkgs/](https://repo.continuum.io/pkgs/) (first looked here)
- Forge
- BioConda
- Channels can be set in .condarc
  - Useful for maintaining own private packages
- conda install -c bioconda scikit-learn
Tips and Tricks
Installing Packages

- Install all your packages at once
- **REMEMBER:** All your packages are installed under ~/.conda/envs/[env]
  - If you have a lot of environments, it will eat up space in $HOME
  - If you’re using the virtualenv for a massively parallel job, you’re going to have a bad time (and so will everyone else)
- Recommendation:
  - $HOME - testing
  - /project/projectdirs/ - permanent
  - Sandbox - permanent
  - $BSCRATCH - good for one-off large experiments
  - conda create --prefix=/where/you/want/env
Exporting/Cloning Environment

- Want to share environment or move it to another machine?
  - conda env export > my_environment.yml
  - conda env create -f environment.yml
- Want to create a clone of an environment?
  - conda create --name [new_env] --clone [other_env]
- Consider how to incorporate version control with your environment.
  - Anaconda doesn’t have vc built in at the moment.
Saving Environment Variables

- Conda allows you to save environment variables
  - Good if you need to store secret keys
  - To save these variables do the following:

```bash
cd /path/to/anaconda/envs/myEnv
mkdir -p ./etc/conda/activate.d
mkdir -p ./etc/conda/deactivate.d
touch ./etc/conda/activate.d/env_vars.sh
touch ./etc/conda/deactivate.d/env_vars.sh
```

```bash
#!/bin/sh
# in activate.d/env_vars
export MY_KEY="secret_key"
```

```bash
#!/bin/sh
# in deactivate.d/env_vars
unset MY_KEY
```
Clean Environment with `conda clean`

- Conda has cache files
  - Index cache
  - Lock files
  - Tarballs
  - Unused cache packages
  - Source cache

- Clean everything with `conda clean --all`
- If you’re left in a bad state -- try clearing caches
Miniconda and Anaconda

- Miniconda allows you full control of your environments and packages
- **Don’t use Miniconda and then load a Anaconda-Python module**
  - Issues can arise if you source activate an environment while trying to use Miniconda
- Unset PYTHONSTARTUP
  - Environment variable mainly relevant to system python. We don’t want users to use system Python
Conda and Pip

- Conda doesn’t track pip installed pkgs
- Causes duplication
  - Env degraded
- Dependency problems
- Try as best to install conda only pkgs
- Use pip as last resort
- Fix incoming down the road
  - As of this presentation (5/7/18)

```
$ pip install tensorflow
Requires:
  tensorboard,
  html5lib 0.999999999 (x7)
  bleach=1.5.0
```
### .condarc

- Customize how conda acts when you’re installing packages and creating envs.
  - conda config --describe
    - Shows all the options that you can tweak.
  - Useful configurations:

<table>
<thead>
<tr>
<th>create_default_packages:</th>
<th>envs_dirs:</th>
<th>pkgs_dirs:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- pip</td>
<td>- $BSCRATCH/.conda/envs</td>
<td>- $BSCRATCH/.conda/pkgs</td>
</tr>
<tr>
<td>- ipython</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- scipy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Don’t see a .condarc already in your $HOME?
  - conda config --describe > ~/.condarc
.condarc - adding values

- conda config --add|--append [key] [value]
  - For list entries
- conda config --set [key] [value]
- Examples:

  conda config --add channel [channel]
  conda config --append channel [channel]

  conda config --set always_yes true
If You Want to Learn More!

RTFM: https://conda.io/docs/
Check out videos - Anaconda Youtube
Read NERSC docs - Anaconda Python
Keep It Simple

MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.
Packaging Your Own Stuff

NERSC
Some caveats:

- Remember that Anaconda can pull down anything — it’s not just a Python package manager
  - So, you can provide libraries, executables, code in other languages
- There’s more than we can cover in an hour
  - Though, the basics are pretty simple and straightforward
  - Use working examples! Don’t reinvent the wheel!
  - What can NERSC do to help? Workshop? Hands-on?
- What should JGI’s software strategy be?
  - Public channel(s)? Local repo? Git repo to pull from?
  - Anaconda Cloud purchase required for private channels
  - That’s all going to be up to JGI, not NERSC.
Prerequisites and Requirements

$ conda install anaconda-client
$ conda install conda-build

• **Know where your pkgs and conda-bld directories are located**
  - Typically: $HOME/conda-bld (created first time you use “conda build”)
    - $HOME/.conda/pkgs (pkgs installed for virtual envs)
    - $PYTHON_DIR/pkgs (central installation)

• **Note: conda-build and client is already installed and available to you within NERSC python modules.**
  - You’ll probably only need to do this if you’re using Miniconda, or building/testing/experimenting on your computers
Four simple steps to building your package

1. **Have your source code in a version-controlled repo**
   - Bitbucket, Github, GitLab, ftp – anything with a url
   - You can use local files, but seriously don’t be that kind of person

2. **Prepare a recipe**
   - meta.yaml and a build.sh script (and/or bld.bat for Windows)

3. **“Conda build” your package**
   - You’ll build locally.
   - Might need to make decisions about what platforms to build for if compiling

4. **Conda upload to an Anaconda channel**
   - You don’t have to upload, and can “conda install” from a local directory
Step 1 - Put your source somewhere

Are you really not already using a version controlled repo?
A simple example meta.yaml:

```
package:
  name: pyinstrument
  version: "0.13.1"

source:
  git_rev: v0.13.1
  git_url: https://github.com/joerick/pyinstrument.git

requirements:
  build:
    - python
    - setuptools
  run:
    - python

test:
  imports:
    - pyinstrument

about:
  home: https://github.com/joerick/pyinstrument
  license: BSD
  license_file: LICENSE
```

Simple build.sh example:

```
$PYTHON setup.py install \
  --single-version-externally-managed \
  --record=record.txt
```

meta.yaml contains metadata, source locations, dependencies, tests

build.sh is the commands needed to build the software on your machine(s)
Example meta.yaml for bioconda’s Blast

package:
  name: blast
  version: 2.7.1
source:
  fn: ncbi-blast-2.7.1+-src.tar.gz
  patches:
    - boost_1.0.5.400.patch
  sha256: 10a78d3074130a6d4e983d2a0cbf03ef84b622b82bd9a59c6b9fbdd9e9d0298ca
build:
  noarch: '
  noarch_python: false
  number: 3'
  string: boost1.64_3
requirements:
  build:
    - boost 1.64.0 py36_4
    - boost-cpp 1.64.0_1
    - bzip2 1.0.6_1
    - ca-certificates 2018.1.18_0
    - certifi 2018.1.18 py36_0
    - gettext 0.19.8.1_0
    - gmp 6.1.2_0
    - gnutls 3.5.17_0
    - icu 58.2_0
    - intel-openmp 2018.0.0_8
    - libgfortran 3.0.1 h93005f0_2
    - libidn11 1.33_0
    - mkl 2018.0.2_1
    - mkl_fft 1.0.1 py36_1
    - mkl_random 1.0.1 py36_0
    - ncurses 5.9_10
    - nettle 3.3_0
    - numpy 1.14.2 py36hd90ae307_1
    - openssl 1.0.2n_0
    - pcres 8.41_1
  run:
    - boost 1.64*
    - bzip2 1.0.6*
    - gnumon
    - nettle 3.3.3.3*
    - pcres
    - perl
    - perl-list-moreutils
    - perl-archive-tar
test:
  commands:
    - blastn -help
    - blastp -help
    - makeblastdb -help
    - touch libstdcpp.so.6; LD_LIBRARY_PATH=. makeblastdb -help && rm libstdcpp.so.6
    - update_blastdb.pl --help
  files:
    - test.fa
    - testdatabase.fa
about:
  license: Public Domain
  summary: BLAST+ is a new suite of BLAST tools that utilizes the NCBI C++ Toolkit.
  extra:
    final: true
Example build.sh for Blast

Any files created by build.sh that are written to its directory will become part of the package.tar.bz2

```bash
$ more build.sh
#!/bin/bash

cd $SRC_DIR/c++/

export CFLAGS="-SFLAGS -02"
export CXXFLAGS="-SCXXFLAGS -02"
export CPPFLAGS="-SCPPFLAGS -ISPREFIX/include"
export LDFLAGS="-DLDFLAGS -L$PREFIX/lib"

if test x"uname" = x"linux"; then
    # only add things needed; not supported by OSX ld
    LDFLAGS="-L$PREFIX/lib -Wl,-as-needed"
fi

LIB_INSTALL_DIR=$PREFIX/lib/ncbi-blast+

./configure
    --without-openssl
    --without-gcrypt
    --with-nettle=$PREFIX
    --with-krb5

projects="algo/blast/ app/ objmgr/ objtools/align_format/ objtools/blast/"

rm -rf ReleaseMT

cd $SRC_DIR/c++

# The "datatool" binary needs the libs at build time, create
# link from final install path to lib build dir:
ln -s $SRC_DIR/c++/ReleaseMT/lib $LIB_INSTALL_DIR

rm $LIB_INSTALL_DIR

mkdir -p $PREFIX/bin $LIB_INSTALL_DIR

rm $SRC_DIR/c++/ReleaseMT/bin/*_unit_test

chown -R $USER $PREFIX/bin

chmod +x $PREFIX/bin/*

sed -i.bak '1 s/\$PREFIX\$USER/bin/env perl 1g' $PREFIX/bin/update_blastdb.pl
```
Step 3 – Build your package

$ conda build yourpackagedir
  • Conda-build will pick up the meta.yaml file and run the build script
  • Output will be a package.tar.bz2 file
    • typically to be found in $HOME/conda-bld/linux-64

  • Can directly install that package.tar.bz2 file
    $ conda install --use-local /path/to/package.tar.bz2

  • Many additional options for setting up your meta.yaml and build.sh:
    https://conda.io/docs/user-guide/tasks/build-packages/define-metadata.html
Step 4 – Upload to Anaconda

• Register an account on Anaconda Cloud
  • [http://anaconda.org](http://anaconda.org)
  • Free is fine for your own public channel
  • Paid for private channels and collaborative groups

$ anaconda login

$ anaconda upload /path/to/yourpackage.tar.bz2
Now your package is available to everyone

- For better or worse
- $ anaconda search packagename

<table>
<thead>
<tr>
<th>package</th>
<th>version</th>
<th>type</th>
<th>platform</th>
<th>python versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>conda-forge/pyinstrument</td>
<td>2.0.2</td>
<td>conda</td>
<td>linux-64, win-32, win-64, osx-64</td>
<td>py36_0, py34_0, py35_0, py27_0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A call stack profiler for Python. Inspired by Apple's Instruments.app</td>
</tr>
<tr>
<td>conda-forge/pyinstrument_cext</td>
<td>0.1.2</td>
<td>conda</td>
<td>linux-64, win-32, win-64, osx-64</td>
<td>py36_0, py35_0, py27_0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A CPython extension supporting pyinstrument</td>
</tr>
<tr>
<td>danudwaryjgi/pyinstrument</td>
<td>2.0.2</td>
<td>conda</td>
<td>linux-64</td>
<td>py27_0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A call stack profiler for Python. Inspired by Apple's Instruments.app</td>
</tr>
<tr>
<td>davidliwei/pyinstrument</td>
<td>0.13.1</td>
<td>conda</td>
<td>osx-64</td>
<td>py34_0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A call stack profiler for Python. Inspired by Apple's Instruments.app</td>
</tr>
</tbody>
</table>
Some helpful resources on packaging

- Anaconda’s documentation on “Working With Packages”
  https://docs.anaconda.com/anaconda-cloud/user-guide/tasks/work-with-packages

- A tutorial on building conda packages directly from PyPI packages
  https://conda.io/docs/user-guide/tutorials/build-pkgs-skeleton.html

- A more extensive hands-on training from AnacondaCon 2018. Covers some history of python package management, provides suggestions on third-party packaging tools, and has several exercises.