OAK RIDGE National Laboratory

Python on Frontier

Michael Sandoval

HPC Engineer - User Assistance Group
Oak Ridge Leadership Computing Facility (OLCF)
Oak Ridge National Laboratory (ORNL)

February 16, 2023

Overview

- What to Expect on Frontier
- Virtual Environments
  - What are they and how do they work?
  - Options on Frontier
    - Inherent feature: `venv`
    - Anaconda distribution: `conda`
- Using `venv`
- Installing and Using Miniconda
- General Best Practices

Moving to Frontier: What to Expect

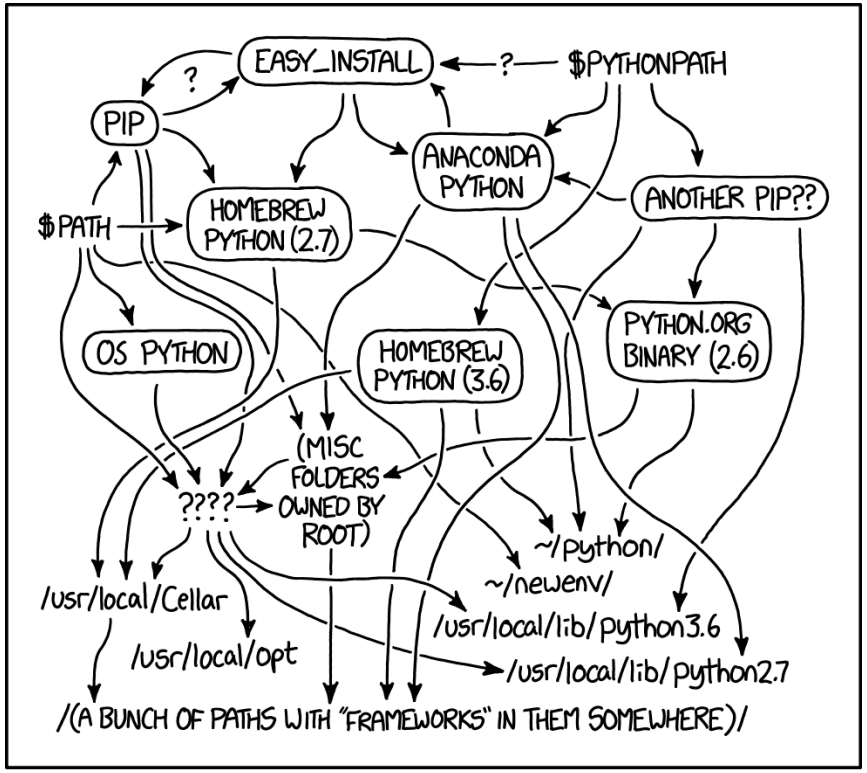
The big takeaway….

- No more Power architecture, x86 is back!
  - Easier to install from pre-compiled binaries
  - Source installs are "easier"
  - Plays nice with conda/mamba and pip
- Should play nicer with Slurm (see: Andes)
- GPU workflow is now the biggest hurdle with the switch to AMD
  - Won't be talking about this today
- Currently, no plans for Anaconda module, but please let us know at help@olcf.ornl.gov if this would be better for your workflow.

Virtual Environments

- What are they?
  - Isolated directory trees that help you manage various packages or different versions of Python
- Why are they beneficial?
  - Dependencies of one package might clash with dependencies of another package
  - Allows you to install new packages without modifying a "base" installation
  - Unique environments can be used on a per-project basis
- We will only be discussing Python3 approaches

Managing Python gets complicated sometimes….



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

Image credit: https://xkcd.com/1987/

Various Virtual Environment Options

- Native option to Python: `venv` environments
  - Included with every installation
  - Extends managing your ***current*** installation
- Anaconda distributions: `conda` environments
  - Highly customizable environments with a large repository of supported packages
  - Not only for Python
- There's also things like pyenv, pipenv, Poetry, [insert fancy new niche manager here]…but we won't be covering those

What about on Frontier?

Two main options:

1. Use the `cray-python` module
   - Supports `venv` syntax
   - Comes with pre-installed packages like numpy, scipy, mpi4py tuned for Cray machines

2. Install your own Miniconda
   - Supports `conda` syntax
   - Similar workflow to what is used on Andes, Summit

Comparing the options on Frontier

- `cray-python` module
  - Pros:
    - Works out of the box
    - Don't need to install any additional things
    - Pre-installed libraries tuned for Cray machines
  - Cons:
    - Extremely minimal
    - Highly dependent on pip
    - Restricted to version of module
    - Can't switch between different Python versions that easily using `venv`

- Personal Miniconda
  - Pros:
    - Let's you manage multiple Python versions, not just environments
    - "Easy" to install dependencies based on your current environment
    - Highly customizable
    - Similar workflow across other OLCF systems
  - Cons:
    - Not included on Frontier as a module, must install yourself
    - Can clash with loaded modules if not careful
    - Highly dependent on pre-compiled binaries*

* can still use pip

The `cray-python` Module

- See what's available:

```
[msandov1@login2.crusher ~]$ module -t avail cray-python
/opt/cray/pe/lmod/modulefiles/core:
cray-python/3.9.4.2
cray-python/3.9.7.1
cray-python/3.9.12.1*
cray-python/3.9.13.1*
```

- Version of module corresponds to version of Python:

```
[msandov1@login2.crusher ~]$ module load cray-python/3.9.4.2
[msandov1@login2.crusher ~]$ python3 -V
Python 3.9.4

[msandov1@login2.crusher ~]$ module swap cray-python/3.9.4.2 cray-python/3.9.13.1
[msandov1@login2.crusher ~]$ python3 -V
Python 3.9.13
```


* also on Frontier

The `cray-python` Module: `venv`

- Create virtual environments by doing:

```
python3 -m venv /path/to/my_env
```

- This creates a set of directories at the specified location, which will contain everything unique to that virtual environment
- How to activate and deactivate the environment:
  - From the command line: `source /path/to/my_env/bin/activate`
  - From the command line: `deactivate`
  - Using a shebang line : `#!/path/to/my_env/bin/python3`
- After activating, you can then install new packages using pip:

```
python3 -m pip install ....
pip install ....
```

The `cray-python` Module: Installing with pip

- In general:

```
pip install <pkg>
```

- From source installs:

```
pip install --no-binary=<pkg> <pkg>
```

- Incorporating environment variables (e.g., gcc):

```
CC=gcc pip install <pkg>
```

- Ignore cache directory:

```
pip install --no-cache-dir <pkg>
```

- Upgrading packages (e.g., pip):

```
pip install --upgrade pip
```

- In general, safer to do:

```
/path/to/my_env/bin/python3 -m pip install …
```

The `cray-python` Module: Workflow Example pt. I

```
# Load cray-python module (default version), swap to GNU
$ module load cray-python
$ module swap PrgEnv-cray PrgEnv-gnu

# Create a directory to hold my environments
$ mkdir $HOME/my_envs

# Create a virtual environment called “mpi4py_env” in my environments folder
$ python3 -m venv $HOME/my_envs/mpi4py_env

# Activate the virtual environment
$ source $HOME/my_envs/mpi4py_env/bin/activate

# Install mpi4py
(mpi4py_env)$ MPICC="cc -shared" pip install --no-cache-dir --no-binary=mpi4py mpi4py
```

The `cray-python` Module: Workflow Example pt. II

```
(mpi4py_env)$ pip list
Package    Version
---------- -------
mpi4py     3.1.4
pip        22.0.4
setuptools 58.1.0

(mpi4py_env) $ salloc -A PROJ_ID -N 1 -t 00:05:00

(mpi4py_env) $ srun --pty python3

Python 3.9.12 (main, Apr 18 2022, 21:29:31)
[GCC 9.3.0 20200312 (Cray Inc.)] on linux
Type "help", "copyright", "credits" or "license" for more information.

>>> from mpi4py import MPI
>>> MPI.Get_library_version()
'MPI VERSION    : CRAY MPICH version 8.1.17.7 (ANL base 3.4a2)\nMPI BUILD INFO : Fri
May 27  0:04 2022 (git hash 43e4dbe)\n'
```

Installing Miniconda pt. 1

- If your workflow better suits conda environments, you can install your own Miniconda: https://docs.conda.io/en/main/miniconda.html
  - Also, please submit a ticket to help@olcf.ornl.gov saying that conda is better for your workflow
- Install process:

```
$ mkdir miniconda_crusher/
$ cd miniconda_crusher/
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
$ chmod u+x Miniconda3-latest-Linux-x86_64.sh
$ ./Miniconda3-latest-Linux-x86_64.sh -u -p ~/miniconda_crusher
```

-p specifies the prefix path for where to install miniconda

-u updates any current installations at the "-p" location (not necessary if you didn't do a "mkdir" beforehand)

Installing Miniconda pt. 2

- While running the installer you will be prompted with something like this:

```
Do you wish the installer to initialize Miniconda3
by running conda init? [yes|no]
```

- If “yes”, your `.bashrc` (or equivalent shell configuration file) will be updated with something like this:

```
# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
.
.
.
#unset __conda_setup
# <<< conda initialize <<<
```

- **Warning:** By default, this will always initialize conda upon login, which clashes with other Python installations (e.g., if you use the anaconda modules on other OLCF systems**). It is *MUCH SAFER* to say “no” and to just export the PATH manually when on Frontier/Crusher to avoid clashing:**

```
export PATH="/path/to/your/miniconda/bin:$PATH"
```

- **Note:** If your `.bashrc` already has a similar block of code (e.g., from other OLCF modules), then it will ***NOT*** modify your `bashrc`

- Highly recommend this (only needs to be run once): `conda config --set auto_activate_base false`

Using Conda Environments

- Activate your base environment first (if not already active):

  `source activate base`

  (must use "source activate" first, but can use "conda activate" after this step)

- Create Environments:

  `conda create –n my_env python=…`

  `conda create –p /path/to/my_env python=…`

- Activate/Deactivate Environments:

  `source activate /path/to/my_env` **or** `conda activate /path/to/my_env`

  `source deactivate`

- Install packages (can use default channel or explicit channel name):

  `conda install package_name` **or** `conda install –c conda-forge package_name`

More Conda Info

- See our Conda Basics guide with a quick-reference list here: https://docs.olcf.ornl.gov/software/python/conda_basics.html#conda-quick
- Conda's official user guide: https://docs.conda.io/projects/conda/en/latest/user-guide/index.html

Best Practices - I

- Most default environment locations are at `$HOME` on NFS, be careful storing things in `$MEMBERWORK` or `$PROJWORK` because it might get purged.
  - For collaboration, you can instead use "Project Home": `/ccs/proj/<proj_id>`
- Make note of your pip cache location by running: `pip cache info`
  - May need to clean it from time-to-time with: `pip cache purge`
- Similarly, clean your `conda` cache occasionally: `conda clean –a`
- Explicitly use "`python3`" (or "`python2`") instead of the "`python`" alias

Best Practices - II

- In general, most python packages assume use of GCC
  - Recommended to use PrgEnv-gnu , especially when building from source
- Deactivate virtual environments first before switching PrgEnv modules
- Deactivate virtual environments before entering batch/interactive jobs
  - Some deactivation syntax won't work properly if entering a job already activated
  - Always better to enter any form of job with a fresh login shell and module environment
- When submitting a batch job that uses virtual environments, it's good to submit like so:

```
sbatch --export=NONE submit.sl
```

This means you'll have to activate all your modules / your virtual env in the batch script

Best Practices - III

- Similar to Andes and Summit, it's always recommended to "clone" the base environment before trying to install packages.
  - For `venv`:

    ```
    python3 -m venv /path/to/new_env --system-site-packages
    ```

  - Cloning with `conda` (does not really apply to Crusher/Frontier):

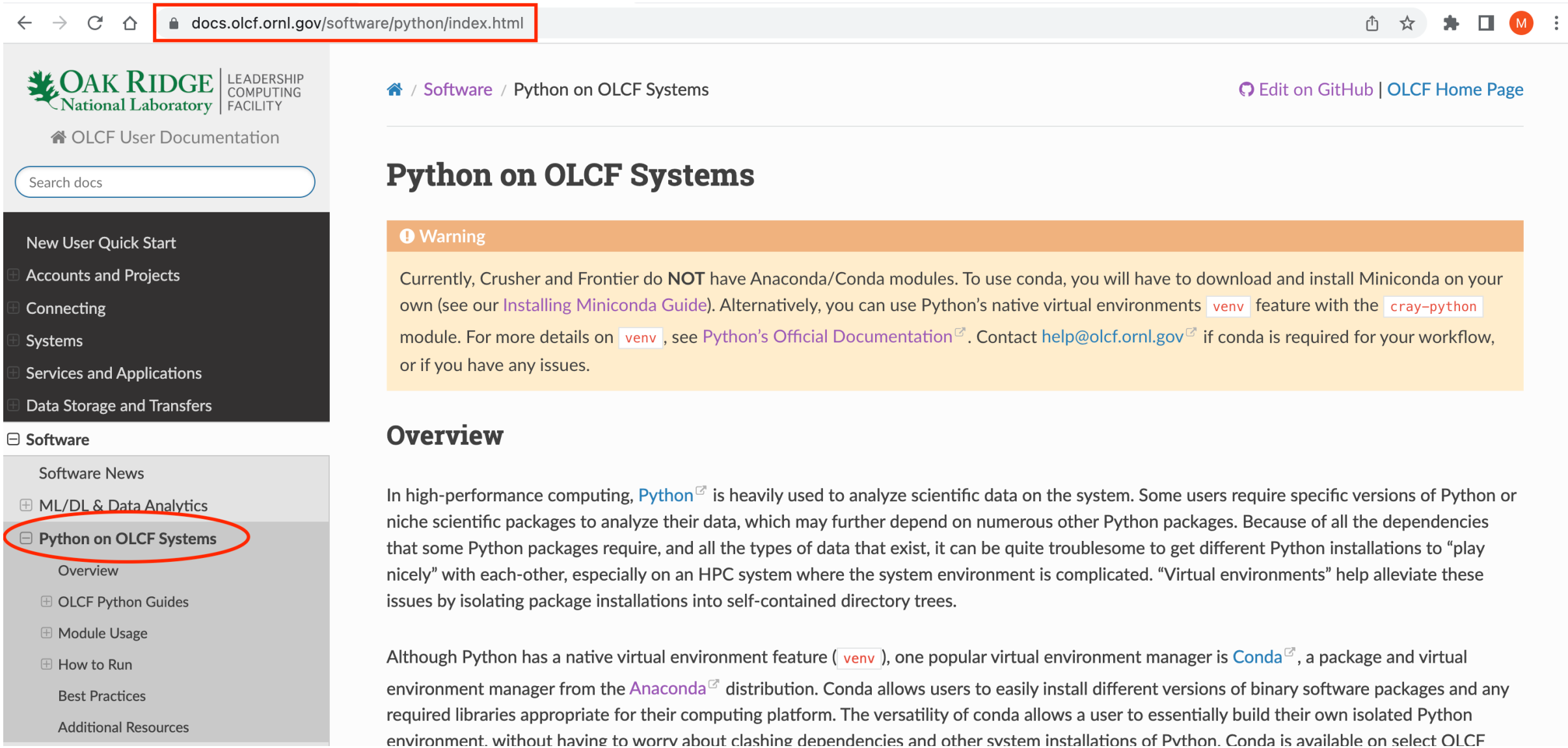
    ```
    conda create -n new_env --clone base
    ```

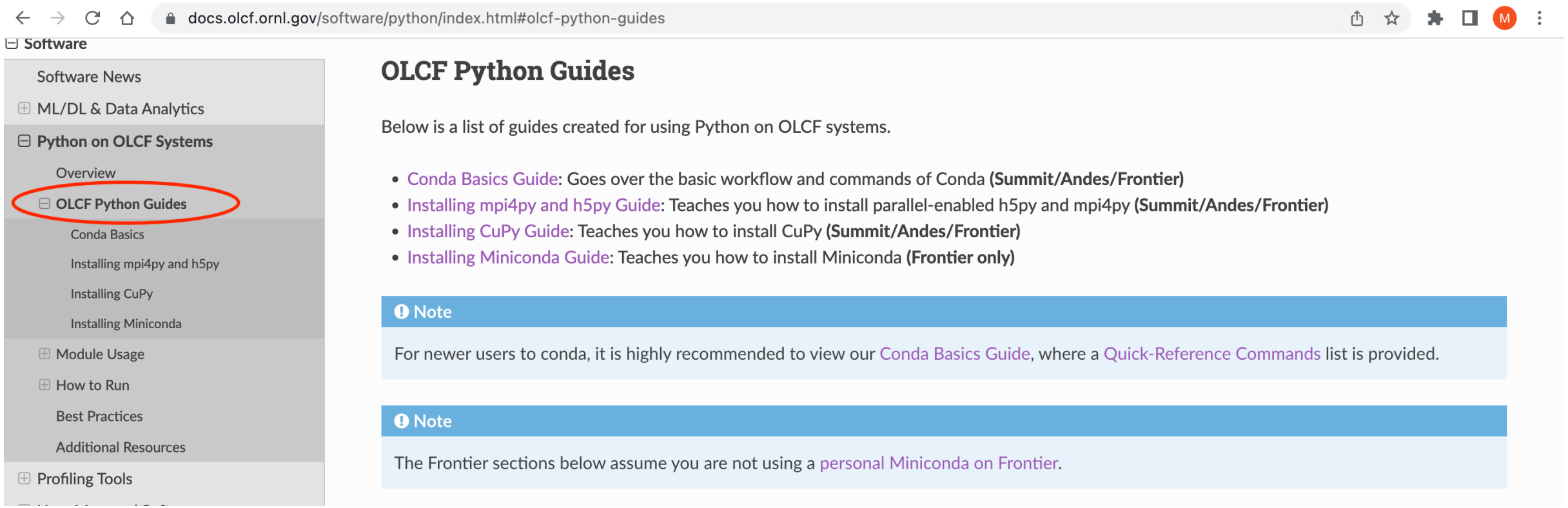
- To "export"/"import" your current environment:
  - For `venv`:

    ```
    python3 -m pip freeze > requirements.txt
    python3 -m pip install -r requirements.txt
    ```

  - For `conda`:

    ```
    conda env export > environment.yml
    conda env create -f environment.yml
    ```


docs.olcf.ornl.gov/software/python/index.html
OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY
OLCF User Documentation
Search docs
New User Quick Start
Accounts and Projects
Connecting
Systems
Services and Applications
Data Storage and Transfers
Software
Software News
ML/DL & Data Analytics
Python on OLCF Systems
Overview
OLCF Python Guides
Module Usage
How to Run
Best Practices
Additional Resources
Software / Python on OLCF Systems
Edit on GitHub | OLCF Home Page
Python on OLCF Systems
Warning
Currently, Crusher and Frontier do NOT have Anaconda/Conda modules. To use conda, you will have to download and install Miniconda on your own (see our Installing Miniconda Guide). Alternatively, you can use Python's native virtual environments venv feature with the cray-python module. For more details on venv, see Python's Official Documentation. Contact help@olcf.ornl.gov if conda is required for your workflow, or if you have any issues.
Overview
In high-performance computing, Python is heavily used to analyze scientific data on the system. Some users require specific versions of Python or niche scientific packages to analyze their data, which may further depend on numerous other Python packages. Because of all the dependencies that some Python packages require, and all the types of data that exist, it can be quite troublesome to get different Python installations to "play nicely" with each-other, especially on an HPC system where the system environment is complicated. "Virtual environments" help alleviate these issues by isolating package installations into self-contained directory trees.
Although Python has a native virtual environment feature (venv), one popular virtual environment manager is Conda, a package and virtual environment manager from the Anaconda distribution. Conda allows users to easily install different versions of binary software packages and any required libraries appropriate for their computing platform. The versatility of conda allows a user to essentially build their own isolated Python environment, without having to worry about clashing dependencies and other system installations of Python. Conda is available on select OLCF


docs.olcf.ornl.gov/software/python/index.html#olcf-python-guides
Software
Software News
ML/DL & Data Analytics
Python on OLCF Systems
Overview
OLCF Python Guides
Conda Basics
Installing mpi4py and h5py
Installing CuPy
Installing Miniconda
Module Usage
How to Run
Best Practices
Additional Resources
Profiling Tools
OLCF Python Guides
Below is a list of guides created for using Python on OLCF systems.
Conda Basics Guide: Goes over the basic workflow and commands of Conda (Summit/Andes/Frontier)
Installing mpi4py and h5py Guide: Teaches you how to install parallel-enabled h5py and mpi4py (Summit/Andes/Frontier)
Installing CuPy Guide: Teaches you how to install CuPy (Summit/Andes/Frontier)
Installing Miniconda Guide: Teaches you how to install Miniconda (Frontier only)
Note
For newer users to conda, it is highly recommended to view our Conda Basics Guide, where a Quick-Reference Commands list is provided.
Note
The Frontier sections below assume you are not using a personal Miniconda on Frontier.

Additional Resources

- Our OLCF Python docs: https://docs.olcf.ornl.gov/software/python/index.html
- Official Python docs: https://docs.python.org/3/library/venv.html
- Official Conda docs: https://docs.conda.io/projects/conda/en/latest/user-guide/index.html
- Submit a ticket to help@olcf.ornl.gov (especially if conda is required for your workflow)
- Questions?