Python and Beyond: Installing Software on Alpine Using Conda, Mamba, and uv

Research Computing
UNIVERSITY OF COLORADO BOULDER

Python and Beyond: Installing Software on Alpine Using Conda, Mamba, and uv

Instructor: John Reiland

- Research Computing, User Support Team

- Website: http://www.colorado.edu/rc
- Documentation: https://curc.readthedocs.io
- Helpdesk: rc-help@colorado.edu
- Survey: http://tinyurl.com/curc-survey18

Slides:
https://github.com/ResearchComputing/python_and_beyond

Research Computing
UNIVERSITY OF COLORADO **BOULDER**

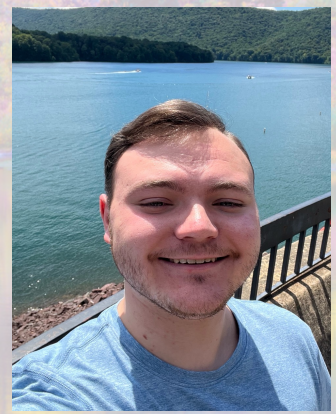
Meet the User Support Team

Layla
Freeborn

Brandon
Reyes

Andy
Monaghan

Michael
Schneider



John
Reiland

Dylan
Gottlieb

Mohal
Khandelwal

Ragan
Lee

Session Overview

- Installing software on CURC systems
- What are Conda and Mamba?
- Using Conda and Mamba on CURC systems
  - Creating software environments
  - Installing packages
  - Useful commands
  - Batch jobs
- Useful strategies for complex Conda/Mamba environments
- Introduction to uv
- Key features of uv
- Comparative analysis of uv with other tools
- Using uv on CURC

Building Software on CURC Systems

- There are numerous ways to install software on Alpine
  - Download and run pre-compiled binaries
  - Compile from source
  - Using containers (Apptainer/Singularity)
  - Using a package manager for HPC systems (Spack)
  - **Within virtual environments (via Conda, Miniconda, Mamba, or uv)**

What are Conda and Mamba?

- They are software package management systems
  - Create, save, load, and switch between virtual environments
  - Install, run, and update packages *and their dependencies*
  - Created for Python programs, but can package and distribute software for any language

Why would we use Mamba?

- Mamba is a faster and more robust package manager in comparison to Conda
  - It is fully compatible with Conda packages
  - Supports **most** of Conda's commands
  - In most cases, Mamba can be used as a drop-in replacement for Conda
    - i.e. replace "conda" with "mamba" in commands

For a more detailed overview of Mamba's capabilities, please see: https://mamba.readthedocs.io

What is a .condarc file?

- The file ".condarc" is a special file that specifies configurations for Conda and Mamba
  - Specifies items such as where to store installed packages and environments
  - Located in your Home directory, **/home/$USER/.condarc**
    - We create it for you, if it doesn't exist, with the content:

```
pkgs_dirs:
  - /projects/$USER/.conda_pkgs
envs_dirs:
  - /projects/$USER/software/anaconda/envs
```

Getting access to Conda and Mamba

On CURC systems, Conda and Mamba are made available through modules

- **We highly recommend using these modules**
  - Redirects output produced by Conda and Mamba
  - Creates useful variables
  - Creates the ".condarc" file, if it doesn't exist
  - They are easier than installing it yourself!

Conda and Mamba modules

- You must be on a compute node to get access to modules!
- Conda is accessible via

```
$ module load anaconda
```

- Mamba is accessible via

```
$ module load miniforge
```

NOTE: We have retired our former "mambaforge" module, as it is deprecated; please use "miniforge".

What is uv?

- Fast, modern Python package manager and environment builder.
- Built by Astral (same team behind Ruff).
- Implemented in Rust.
- All-in-one solution
- Designed to work natively with Python's built-in venv

Image source: https://docs.astral.sh/uv/

Features of uv

- Drop-in Replacement for pip, pip-tools.
- 10-100x faster than traditional tools.
- Reliable and reproducible builds
- Memory-efficient operation, especially for large projects

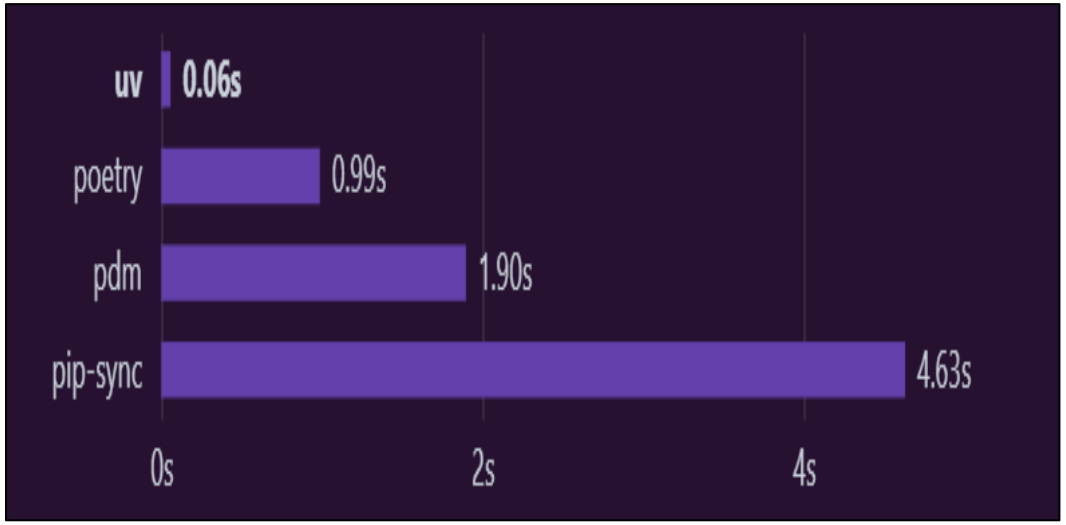


Image source: https://docs.astral.sh/uv/

uv vs. the status quo

Feature	uv	Traditional pip + venv	conda
Speed	10-100x faster than pip	Baseline	Slower than pip
Memory Usage	Very efficient	Higher	High
Reproducibility	Universal lockfile (uv sync).	Requires a separate tool (pip-tools) for locking.	YAML files (often flaky).
Non-Python Packages	No	No	Yes
Dependency Resolution	Fast, modern resolver	Basic	Comprehensive
Ease of use	Simple, single tool	Broader scope with a steeper learning curve.	Requires multiple commands (python -m venv, source, pip).

Getting Started with uv

- To start using uv, you must first load the module in your interactive session:

```
$ module spider uv
$ module load uv
$ uv --version
```

Creating a uv Environment

- uv automates the creation and management of environments using venv
- venv is Python's built-in tool for creating isolated environments
- Each venv:
  - Has its own Python interpreter
  - Has its own installed packages, separate from system Python
  - Avoids dependency conflicts across projects
- It's a lightweight alternative to Conda environments

Demo time!

Start a session and load Conda

Start a session on an Alpine compute node with **acompile**:

```
[johndoe@login-ci3 ~]$ acompile --help
[johndoe@login-ci3 ~]$ acompile --ntasks=4 --time=60:00
…
[johndoe@c3cpu-a5-u28-1 ~]$ module load anaconda
(base) [johndoe@c3cpu-a5-u28-1 ~]$
```

Note: when you login to CURC you'll be on a ***login*** node. You'll need to be on a ***compute*** node to use anaconda. The **acompile** command allows you to quickly start an interactive job on a compute node.

Create your first Conda environment!

- Environments are created and programs are installed in a few simple steps

```
(base) [johndoe@c3cpu-a5-u28-1 ~]$ conda create -n my_first_env python==3.10
(base) [johndoe@c3cpu-a5-u28-1 ~]$ conda activate my_first_env
(my_first_env) [johndoe@c3cpu-a5-u28-1 ~]$ python
```

Don't install packages in your base environment!

Installing packages

- Packages are installed within **activated** environments using **conda install**
- Install latest version available:

```
(my_first_env) [johndoe@c3cpu-a5-u28-1 ~]$ conda install pandas
```

- Install a specific version:

```
(my_first_env) [johndoe@c3cpu-a5-u28-1 ~]$ conda install pandas==2.2.0
```

Installing packages with “pip”

- pip installs should be done within **activated** environments

Using **pip** to install latest version:

```
(my_first_env) [johndoe@c3cpu-a5-u28-1 ~]$ pip install --no-cache-dir pandas
```

`--no-cache-dir` may be crucial on CURC systems

Useful Conda Commands

```
conda env list                              # list all environments
conda list                                  # list packages in active env
conda env remove -n <envname> --all         # remove an environment
conda config --show channels                # view configured channels
conda deactivate                            # deactivate environment
conda create --name <clonedenv> /           # clone an environment
      --clone <envtoclone>
```

Useful Conda file paths on Alpine

```
# location of python libraries
/projects/$USER/software/anaconda/<env>/lib/python3.10/site-packages

# location of package executables
/projects/$USER/software/anaconda/<env>/bin

# location of .condarc file
/home/$USER/.condarc
```

Running Alpine batch jobs with Conda

```
[johndoe@login-ci3 ~]$ nano runconda.sh  #Step 1: open new job script in editor
```

```
#!/bin/bash
# job script name: runconda.sh

#SBATCH --partition=amilan
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --time=10:00
#SBATCH --qos=normal

module purge
module load anaconda
conda activate my_first_env
python my_python_code.py
```

```
# Step 2: Write job script
```

https://curc.readthedocs.io/en/latest/running-jobs/batch-jobs.html

```
[johndoe@login-ci3 ~]$ sbatch runconda.sh  #Step 3: Schedule job
```

Using environments in Open OnDemand

- In Open OnDemand users can utilize their environments in Jupyter sessions. This can be done using two different methods:
  - Specifying the environment name in the "Conda environment" field
    - Allows you to launch the notebook from within your environment
    - Needed for some Jupyter extensions
  - Create a Jupyter Kernel
    - Allows you to switch between different environments while in a notebook
    - May prevent some Jupyter extensions from working
- Both methods are described at https://curc.readthedocs.io/en/latest/open_ondemand/jupyter_session.html

Creating a uv Environment

- Loading the module sets a key environment variable **$UV_ENVS**.

```
$ echo $UV_ENVS
/projects/$USER/software/uv/envs
```

- To create a new environment

```
$ uv venv $UV_ENVS/myenv
$ uv venv $UV_ENVS/myenv1 --python 3.8
```

Activating the Environment

- Packages are installed within activated environments
  - Fully compatible with PyPI
  - Uses caching to speed up repeated installs
  - **Important: not compatible with Jupyter sessions**

```
$ source $UV_ENVS/myenv/bin/activate
$ uv pip install numpy
$ uv pip install pandas
```

Useful Commands

```
uv <command> --help                    # help for specific commands
uv venv $UV_ENVS/<envname>             # create a new environment
uv pip install <packagename>           # install a package in active env
uv pip uninstall <packagename>         # uninstall a package from env
rm -rf $UV_ENVS/<envname>              # remove an environment
uv cache clean                         # clean up unused cache
uv pip freeze                          # list installed packages
deactivate                             # deactivate environment
```

Note: Be very careful with rm -rf when removing environments

Strategies for complex environments

- You may have to use different channels and change channel order
- Conflicts can arise when iteratively installing packages. If this happens create a new environment and install the package that causes conflicts first.
- Use Mamba to accelerate installations

Thank you!

Survey and feedback

http://tinyurl.com/curc-survey18

Slides

https://github.com/ResearchComputing/python_and_beyond