Simplifying Scientific Python Package Installation and Usage

Date: September 13, 2023

Presented by: Amiya Maji (Purdue University)

(The slides are available under “Materials from the Webinar” in the above link.)

Q. Several DOE facilities already provide a Python stack (e.g. Cray Python on Cray platforms, the `python` module at NERSC). Is there a way to streamline `venv`/`conda` usage with these stacks without installing a full Anaconda/Miniconda stack yourself? Or is this really more on the sysadmins’ side to adjust the multi-user config?

A. conda-env-mod can use external conda installation, in fact that is the preferred mode. Support for other environment managers such as venv/mamba is being considered.

Q. We try to teach our PhD students to prevent using any tools by Anaconda Inc. (I would even prevent using `conda`) What do you think about `poetry`?

> **C.** Why? In my experience, I would recommend PDM rather than poetry (since it adheres to standards… Poetry is forging its own standards.)
>
> **C.** We try to prevent any license issues. But you are right about the standards.
>
> **C.** Ah, so it’s like the Docker (commercial license for enterprises) vs Podman situation.
>
> **C.** Do you have some details about potential license issues? If you are not using the conda code base in your project, I don’t understand how their use of the GPL would influence you when used only as packaging tools.
>
> **C.** See https://www.anaconda.com/pricing/organizations for pricing. That’s already a non-GPL thing to do, no?
>
> **C.** But that is only for hosting packages, etc. in the anaconda cloud (i.e, if you are a company, and you want to host a binary repo with your private tools etc. for deployment, then you might pay Anaconda to host these so your clients can install your packages easily.) Using the tools does not require a license.
>
> **C.** So is your issue with anaconda and not conda environments?
>
> **C.** Well, `conda` is developed by Anaconda, Inc.

C. mamba/micromamba are faster, and not developed by Anaconda.

C. I do avoid the anaconda repo and use conda-forge instead due to licensing.

C. You are right, about the licensing, but just in case, we teach poetry.

C. From all my experience, the approach of Anaconda has been "good" in the sense that they develop tools etc. for money, but eventually try to contribute these to the community as open-source projects (i.e. numba). The tools are GPL'd so there will never be a license issue for using them - you do need to pay if you want to use online hosting etc. As for using the anaconda repo rather than conda-forge: I again think that any licensing issue you might have would only be due to individual packages… and you may have these with conda-forge as well. (Of course, IANAL.) I currently cannot see any potential licensing issue using any of the open-source Anaconda tools, but would really like to know if there are any (and any instances of shady business practices… but I currently have a very high opinion of them).

A. I have not used poetry myself, but from the documentation it seems like poetry manages the environment in a manner similar to Spack. You need to activate a "poetry shell" to run Python commands. I would love to learn more about your experiences with Poetry, specially for parallel Python applications. For conda-env-mod, the goal is to make Python package installation similar to traditional HPC package installation (hence, using module files). We are considering using other environment backends (e.g. venv) for people who want to avoid conda.

Q. Installing python on HPC is not too difficult. What is challenging is compatibility between software and python versions. Python does not seem to be backward compatible–or do I miss something?

A. In addition to software and Python versions, none of the environment/package managers seem to track dependency on external packages. Additional complications may come due to how different package managers behave (installation directories/configurations/dependency resolution etc.). For example, consider the scenario that one package is only available from PyPI and another is only available from Conda, it may be non-trivial to identify what dependencies satisfy both. Lack of appropriate documentation would be another challenge. Most documentations work on laptops (or on specific distributions), but may not work on HPC systems.

Q. My general questions here are more along the lines of: Have you considered and made progress with these issues (hopefully to open future discussions). From some of the discussion it seems that there is lots to talk about here.

- **How do you deal with the problem of ensuring a reproducible computing environment? Specifically when we have a need for both Conda and Pip**

(managed with PDM, Poetry, etc.), how do you lock packages and reproduce the environment across different architectures? (Anaconda Project -> Conda Project has some tools, but does not deal nicely with PDM etc.)

- **Installing packages with complex dependencies is tricky. (I.e. how do I add Manim to my project, making sure all necessary components are also included pinned at the right versions for the right platforms etc.) PDM and Poetry currently seem to do the best job (but still highly flawed). Have you thought about a way of dealing with this?**
- **Installing all required tools can be an issue (for conda-env-mod, one needs to install at least Lmod and Conda.) Have you thought about a way to get these? (We are working on developing a cookiecutter generated Makefile that works across platforms (Mac laptop, Linux desktop, HPC cluster, CoCalc, etc.) to bootstrap the appropriate tools (`make tools`), make an appropriate environment with an installed ipykernel (`make init`), and running a shell where users can update dependencies, run jobs, etc. (`make shell`). I am wondering if `conda-env-mod` should be a part of this, or is a parallel effort. It seems like it should be a part of this to capture the module dependencies (but then a way is needed to bootstrap the creation of these modules on non-HPC platforms so that one unified workflow works on all platforms).**

A. Thank you for the thoughtful comments and questions!

A1. conda-env-mod is not an environment manager itself but is expected to work with other environment manager backends. For precise dependency resolution/lockfile, we rely on the export features of conda. As long as the developer can export a working environment to a specification.yaml file with appropriate versions, we can use that in a different computer/cluster.

A2. An environment specification file with proper versioning can achieve most of it. However, we do feel that there needs to be better standardization around these "environment specification files." The specification files need to be more extensible. For example, what system packages does the environment depend on in Redhat/Debian distributions, what modules does it depend on in an HPC cluster, what additional environment variables need to be set when you load this environment etc.

A3. It sounds like the capabilities of the Makefile are similar to what conda-env-mod does. There is definitely some overlap between the two.

We are also working to export an environment as a bash script, so that users don't have to install/configure Lmod on their laptops. There has been some suggestions in the past for pre-packaging conda into conda-env-mod, which we are investigating.

Q. You mention that [conda-env-mod(?)] is preferred to install from source? Why?

A. The current version of conda-env-mod does not require any compilation. It can be installed by cloning the Github repo. As for the Python packages themselves, you can

install them from source or from binary distributions (wheel or conda packages) depending on your requirement. For example, we have found that the binary wheels for Tensorflow and Pytorch are performant since the majority of the computation is done on GPUs. However, most HPC sites recommend building Python packages from source. This is to make sure that they are configured with the correct dependencies (specific versions of MPI, HDF5, etc.) and are optimized for that particular HPC system.

Q. How easily can one export/share an environment defined by conda-env-mod to users who might be using another HPC system or even their own laptops? Including the full env settings which makes conda-env-mod so attractive?

A. You can use `conda-env-mod export -n env_name` to export the package list to YAML files. The module files can be shared easily within the same HPC system. However, when sharing the modules across HPC systems, both centers must use the same module software (Lmod or TCL) and similar application stacks.
We are currently working to export the environment setup as a bash script which individual developers can use on their laptops.

Q. Does module load integrate with lmod?

A. Yes

Q. Can you share a pdf or copy of the related paper by you and colleagues? It is behind a pay-wall presently.

A. Please send me an email and I can share a copy with you. The relevant paper is: https://ieeexplore.ieee.org/abstract/document/9308091

Q. Would you consider transitioning from conda to mamba/micromamba (significantly faster, more reliable dependency resolution)?

> **C.** I have not yet looked to see if it is available, but I second the use of `micromamba`: we are slowly transitioning to a system based on this since it is easy to install, fast, etc.

A. Indeed. We are working on adding support for mamba.

Q. How does conda-env-mod play with other modules users may load before/after their Python environment? Are there any corner cases to be aware of? (E.g., when other modules load/unload particular Pythons that may conflict with your conda-env-mod module.)

A. That is a great question! conda-env-mod adds dependencies to modules that it sees loaded at the time of creating the module. With Lmod, users will see a warning/error if

they try to load a conflicting module. But users can always “--force” their way to a hazardous scenario.

Q. Does `conda-env-mod` support generating Lmod `lua` modules too?

A. Yes.

Q. This isn’t really a question. The DESI project has developed a (conceptually) very similar package management system called desiconda, which is used to manage a base environment, and then we add DESI-authored packages as separate modules on top of that. The base environment contains all of the third-party packages we need. The focus is on NERSC systems. If conda-env-mod had been available ~10 years ago, we probably would have adopted it.

A. Thank you for sharing! Desiconda looks very similar in its approach to Python environments compared to conda-env-mod. There is still scope for consolidating the efforts and sharing the best practices from both projects. :)

Q. Do you do anything to prevent users from stacking incompatible conda environments?

A. We implement some measures to help users avoid such errors. For example, conda-env-mod modules define dependency on the version of anaconda that was used to create that environment. Multiple modules can only be stacked if they use the same anaconda. We also add the Python version in the module file name itself. That way users know the version of Python inside an environment. Similarly, conflicts across non-Python modules will be flagged by the environment module software (see the discussion in the next question). (Expert) users may still be able to mix incompatible environments, but the bar is higher.

Q. How does dependency resolution work across stacked environments? (I.e. What happens if a sub-environment has a non-python dependency that conflicts with the base environment?)

A. This is a great question! We expect the module software to provide some consistency across stacked environments. For example, with the module software Lmod, if the sub-environment loads a (non-python) module that conflicts with the base environment, it will either prohibit that load or unload the base environment. In either scenario, the developer becomes aware of the inconsistency. With other module softwares, such as TCL environment-modules, the module files need to define what modules conflict with them.

Q. What about centrally installing these Python packages using Spack or Easybuild?

A. Centrally installing popular Python packages with Spack is definitely one approach to ensure strong dependency tracking. A lot of sites maintain their own Python stack, but capturing every researcher's custom needs often require significant staff time. Considering the explosion in the number of Python packages, and the constant need for newer versions, users still need to be able to install custom Python packages by themselves. "Conda-env-mod" tries to make that simpler — it is complementary to Spack-based installations.

Q. I don't mean for this to sound confrontational – I'm legitimately curious. It sounds like at least a significant part of the root of the problem you're trying to solve is you want to get people up and running and productive working in Python in their scientific field, but you want that to happen without them learning how to interact well with the Python ecosystem. The conda-env-mod tool is meant to shield them from that so they don't have to think about it and can get back to productive work. I'm wondering what the long-term consequences of that will be. What happens if science in Python progresses at a far greater rate than the scientists' understanding of the language and software ecosystem in which they work? It's probably too soon to answer such a question, and it may be that folks in the ECP community wouldn't be the right people to answer anyway when sufficient time has passed.

> **C.** I think this situation is similar to using modules to load software tools and libraries that aren't python. Those are provided by the vendors and system administrators. If the thought about what happens if scientists don't install their own software and don't know about their underlying dependencies can be applied here, I would expect that much of the progress over the years can attest that scientists can rely on the team of people that provide those.
>
> **C.** Our current approach is to codify our best practices in a well-tested Makefile so that it "just works" when people need it to, but is transparent so users can understand the nitty-gritty as they learn more. I also worry about hiding too many things in custom tools, modules, etc. One approach is to minimize which tools you depend on, but I also like using the latest and greatest.

A. Thank you for raising this concern. The goal of conda-env-mod is not to shield users from learning the basics of Python, but to help them along the way. As the first comment points out, it is similar to how other HPC software are exposed as modules. A user does not need to know every single MPI configuration that is set by the MPI module, but if they are curious they can always try a different configuration.
We have seen that the benefits of using conda-env-mod outweigh its deficiencies. It works great in class environments, for sharing python installations, or even for reproducing existing installations.

Q. Does anyone know a forum etc, where such matters are discussed ... solved?

A. I haven't come across anything that talks about Python package installation in HPC. Maybe we can start an HPC SIG under the Python Foundation.
https://www.python.org/community/sigs/