HPC Day: Advanced Topics (Part 1)

Presented By: Joseph Guzman

Date: 2025-09-19

Link to slides:

http://rcdata.nau.edu/hpcpub/workshops/advanced_topics_p1.pdf

NAU NORTHERN ARIZONA UNIVERSITY

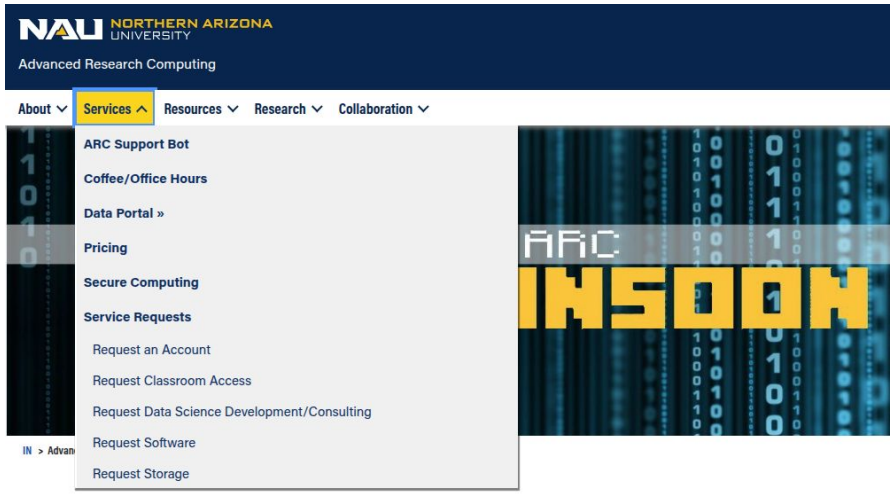
Overview

- 1. ARC Support Bot
- 2. Package Management
- 3. OnDemand Apps
- 4. Free QoS

Section 1: ARC Support Bot

New Experimental Chat Bot

- Navigate to our main website: https://in.nau.edu/arc
- Click Services on the header bar
- Then click the “ARC Support Bot” option
- This webpage will have the main link, which may change



How it works (subject to change)

- The bot was made in Copilot Studio
- No code besides text conversion of docs
- Internally, it has two parts: pre-planned topics and a RAG
- Currently, it has no other knowledge sources
  - If a prompt does not match any topic or any doc, it will fail
- Goal: simulate a customer service rep who has memorized our public-facing documentation
- Disclaimer: The bot relies on LLMs which can make many true-sounding errors. Email ask-arc@nau.edu to talk to a person!

Section 2: Package Management

Packaging Tools Overview

Lmod
- Loadable Module system, includes packages that were created by us, the ARC team
  - Many of these were requested by researchers i.e. davinci
- view available packages by running: module av

Anaconda
- General-purpose package manager, with a focus on Python and R packages
- view available packages here: https://anaconda.org/

pip
- De-facto standard package manager for python packages
- view available packages here: https://pypi.org/

CRAN
- Repository of packages for R
- view packages here: https://cran.r-project.org/index.html

Singularity
- Docker-compatibile containerization tool

Spack
- Curated HPC package manager, created by LLNL

configure+make
- Standard tools for source compilation in Linux

.rpm files
- package recipe file for RHEL-based distros
- The equivalent in debian-based are .deb files

If you are considering one of these tools then you should contact us: ask-arc@nau.edu

How do I install X on monsoon?

- First, check if we have an Lmod module for it
  - you'd examine output of `module av mypkg`
  - if already there and you need a new version let us know
- Use a login node (i.e. wind) for these next steps, we have additional -devel packages
- If there's no Lmod module try the language specific package manager
  - `pip install mypkg` command in python
    - Note: our anaconda3 module bundles many useful python pkgs (i.e. numpy)
  - `install.packages("mypkg")` function in R
- If you get any errors there, then check if the package is available on conda
  - https://anaconda.org/
- You could try to use conda for everything, but there are a couple downsides
  - many more packages exist in PyPI/CRAN
  - conda does not help with source installs for unindexed packages
  - conda environments will be larger on disk
- If still unsure, then create a ticket with us.
- Faculty members can submit a software install request
  - we create new Lmod modules for these

Lmod Tutorial

1. Check what Lmod modules you have loaded:
`module list`

2. Unload a module:
`module unload <pkg>`

3. Unload all modules:
`module purge`
    - OR login again

4. List all modules:
`module av`

5. Search for a module:
`module av <pkg>`

6. Load default module version:
`module load <pkg>`

7. Load specific module version (preferred):
`module load <pkg>/<version>`
    - Why is is it preferred to specify the version? Because we update Lmod packages, especially important for researchers who use conda

8. View module details
`module display <pkg>/<version>`

pip Tutorial

- Its best to use virtual environments to isolate python package on a per-project basis
  - Because python packages tend to have complex dependencies
  - Default is to install packages in ~/.local
- The web interface at https://pypi.org/ is the only way to search for packages
  - Also this is the best way to lookup the module's github repo and documentation

Example:

```
python3 -m venv ~/myenv
. ~/myenv/bin/activate
pip3 install beautifulsoup4
python3 -c "import bs4; print(bs4.__version__)"
```

R install.packages() Tutorial

- R will normally use gcc or g++ in the background to compile your R packages
  - For this reason, you need to execute **`install.packages()`** on a login node
- R will install packages by default into ~/R/<version>
  - Not usually necessary to segment your R packages, but it is possible by simply running mkdir and setting the `R_LIBS_USER` env variable to that path

Example:

```
module load R/4.1.2
R
install.packages("jsonlite")
library(jsonlite)
packageVersion("jsonlite")
```

Miniforge Tutorial (part 1)

- You must create a conda environment before installing any packages
  - We'd again suggest to create one on a per-project basis
- conda packages tend to be quite large, you may quickly run out of space in /home
  - You can request more /home space and we'll give you 20GB
  - or you can try to save space by creating a spec-file.txt and removing the original conda environment
  - or you can configure conda to put things in /scratch (see: https://in.nau.edu/arc/faqs/#configuring)
- You can create a conda environment and then use another package manager like pip to install additional packages
  - But if you try to use conda again, you can create package conflicts, as conda cannot keep track of packages installed by other means
- I'll demo conda with the miniforge3 module
  - miniforge is an alternative distribution of conda, there are only two differences:
    - miniforge has a more minimal base environment (does not come with numpy or matplotlib for example)
    - miniforge comes with the much faster "mamba" installer
  - Once installed you can use the "conda" and "mamba" commands interchangeably, they do **exactly** the same thing, except when it comes to installing something you should use `mamba install -c <channel> <pkg-name>`

Miniforge Tutorial (part 2)

- Example:
  - ```
    module load miniforge3
    mamba create -n myenv
    mamba activate myenv
    mamba install -c conda-forge numpy
    ```
- Now try out the web search ability
  - Visit: https://anaconda.org/search
  - Search for the “hexyl” package (it’s a hex viewer, similar use-case as the `xxd` and `od` commands)
  - Make a new conda environment and try out the installation command it gives you (replace “mamba” with “conda” to use the fast installer)
  - Try out the hexyl command:
    - i.e.
      - read your .bashrc file
        - `hexyl ~/.bashrc | less`
      - read random binary data
        - `hexyl <(head -c 1K /dev/urandom) | less`
    ```

Spack Tutorial

- Search for a package:
  - `spack list mypkg`
  - `spack info mypkg`
- Two workflows for installing packages:
  - direct installs
    - Unlike conda, spack allows for installing and loading packages without an environment. Spack is acutely aware of the dependencies specific to each package.
    - Switch to using environments if you have a complicated set of packages to avoid repetitive loads commands.
    - Example:
      - ```
        module load spack
        spack install gzip
        spack load gzip
        ```
  - install into an environment
    - Example:
      - ```
        module load spack
        spack env create -d ./myenv
        spack env activate ./myenv
        spack add gzip
        spack install
        ```

Section 3: OnDemand Apps

VSCode App for Ondemand

- We're using this implementation for our Open Ondemand (OOD) server: https://github.com/coder/code-server
  - There *may* be some limitations when compared to the desktop version
    - Feel free to reach out via service-now if you encounter any
- Use-Case?
  - Arguably: **OOD's VSCode > OOD's built-in file editor, local VSCode, and terminal-based editors**
  - OOD's VSCode as a cluster-local app will integrate with our local software and hardware stack; and experience less latency when doing cluster-local operations
  - Use the full suite of VSCode features
    - including: code-completion, debuggers, and a terminal app
- Alternatives?
  - VSCode is a generalist editor, not a viable replacement for niche GUI editing environments (such as Jupyter and RStudio)
  - If you've spent the time on configuration, vim/emacs is capable of much the same as VSCode. I'd continue using these if you are familiar with them and do not require significant GUI usage

Getting started with VSCode Plugins

- To install any plugin, press C-S-x to enter the menu
- Code-completion and validation plugins
  - For Python, try: python-lsp-server
  - For C/C++, try: llvm-vs-code-extensions
  - For shell script, try: ShellCheck
- Find more language servers here:
  - https://microsoft.github.io/language-server-protocol/implementors/servers/
- Many, many more, check out this curated list: https://github.com/viatsko/awesome-vscode

Launching the app

- Similar to the other interactive apps:
  - Visit https://ondemand.hpc.nau.edu
  - Click on the “Interactive Apps” Tab
  - Click “VScode”
  - Fill out the form, submit, and click “Connect to VSCode” once it starts
- Note: VSCode plugins can use up significant space in /home
  - Primarily in these directories:
    - ~/.config/code-server
    - ~/.local/share/code-server

Other Notable Apps

- File Explorer
- Job Composer
- Desktop
- **Jupyterlab**
- **RStudio**
- Ansys
- Matlab

What's your favorite programming language or editor/IDE?

Section 4: Free QoS

What is a Slurm QOS?

- The QOS (Quality of Service) value associated with a Slurm job modifies the scheduling parameters
  - By default your QOS value is your group name
    - Generally, there is no scheduling priority difference between groups
  - Other QOS values include:
    - debug
      - set this qos for short low-resource jobs to monitor or debug your running job
        - i.e. to run a small testcase
      - can only run one debug job at a time
    - free
      - set this qos for your low-priority jobs
        - that way you do not impact your group's fairshare usage
        - improves scheduling priority of non-free jobs while the cluster is busy
    - gpu
      - allocated automatically, please use `#SBATCH -G 1` to select a gpu in your job script

Slurm

- Selecting the “free” qos for srun
  - `srun -q free hostname`
  - `srun --qos=free hostname`
- OR in a job script:
  - `#SBATCH -q free`
  - `#SBATCH --qos=free`

- View the QOS of your running jobs:
  - `squeue -a -u $USER -O jobid,jobname,qos,state`
- View the QOS of a past job:
  - `sacct -j <jobid> -o jobid,jobname,qos,state -X`

Why should I use the free qos?

- The purpose of the free qos is to allow users to distinguish between low-priority and high-priority jobs.
  - If everyone uses this for low-priority jobs, then there will generally be less resource contention for high priority jobs
- **Warning**: Your jobs in the free qos are “preemptable”
  - If your job is running and slurm allocates resources your job is using for a non-free qos job, then your job will be preempted and requeued
    - Your job terminates at that time to be restarted later
    - Unsaved progress will be lost
      - To mitigate this behavior, some programs will create checkpoints for the progress so that it can resume from the checkpoint

- Demo on an idle node

END

Need to contact us?

- Email:
  - ask-arc@nau.edu
- Zoom Office Hours
  - Every Wednesday
  - Alternates Between 1:00PM-2:00PM and 2:00PM-3:00PM
  - https://in.nau.edu/arc/office-hours/

Questions?

NAU NORTHERN ARIZONA UNIVERSITY

Section Header/Footer

ALT TITLE

NAU NORTHERN ARIZONA UNIVERSITY