

Using Jupyter Notebooks on HPC systems

UM Fall HPC/Cloud Workshop

Grigory Shamov, October 15, 2025



**University
of Manitoba**

Why the talk is about Jupyter on HPC?

- **There exists a lot of uses for dynamic languages like Python, R, Julia**
 - JuPyteR (formerly iPython) is a popular debugging/Data analysis / collaboration tool
 - <https://jupyter.org/>
- **HPC has high performance hardware and software**
 - NVIDIA (and AMD) GPUs; Fast Interconnect ; Large memory nodes, etc.
 - Scalable, parallel File Systems (VAST, CEPH, DDN Infinia) or local SSD NVMe
- **Uses of JuPyteR in HPC?**
 - Interactive jobs, debugging
 - Visualization, data analysis requiring larger memory, GPUs
 - “Moving compute closer to data”
- **Why not using Jupyter in HPC?**
 - Batch processing is more efficient , can be done concurrently

This talk is about using our HPC infra

- *HPC presents users with its own issues: Batch Jobs and Software Modules*
 - How to run Jupyter on HPC as a job? Interactively?
 - How to access/manage Jupyter software
 - Tools specific for dealing with HPC Lmod modules
 - How to handle Notebooks Kernels (for Python(s), R, Julia, etc.)?
 - Virtual environments
 - Module dependencies
 - Containers?

Working with Notebooks in HPC 1.

- A Jupyter notebook / Jupyter lab is in practice a Web server.
 - Typically users would start it on a PC/laptop as **jupyter lab**
 - Would start a browser and show a notebook
- How to scale it up? How to use it on powerful servers like HPC or cloud
- Can be started manually on a login node and/or interactive job
 - Needs Python packages : **jupyter**, **ipython**, etc.
 - Need to manage software dependencies (modules, **pip**, **virtualenv**)
 - Spans remote and local systems
 - Need to “SSH tunnel” the web server connection
 - Or have a browser on the remote server which is less useful

Working with Notebooks in HPC 1.

- Can be started manually on a login node and/or interactive job

ssh to a node; or start an interactive job with **salloc**

module load python/... ; **i**

install jupyterlab in a virtualenv, or a container ; start the notebook

jupyter-notebook --ip 0.0.0.0 --no-browser --port 8765

ssh -fNL 8765:g333:8765 yourusername@bison.hpc.umanitoba.ca

Then point the browser to a <https://localhost:8765>

Working with Notebooks 2

- A notebook is a Web server. It can be started automatically;
- The connection between the Notebook server can be “proxied” automatically
 - So that computation is done by a computing backend (HPC, cloud, etc.)
- JupyterHub is a web server that starts notebooks and proxies them automatically
 - Google Colab is a powerful cloud platform: <https://colab.research.google.com/>
 - PIMS provides SyZyGy (which uses DRAC cloud) <https://umanitoba.syzygy.ca>
 - Magic Castle and JupyterHub on (some) new DRAC systems
- OpenOnDemand is a Web portal that starts interactive jobs and proxies them automatically
 - Supports Jupyter notebooks too! Provided on Grex and (some) new DRAC systems

JupyterHub and/or OpenOnDemand

System	JupyterHub ?	OpenOnDemand ?
Fir (SFU)	jupyterhub.fir.alliancecan.ca	
Nibi (Sharcnet, UWaterloo)		ondemand.sharcnet.ca
Rorqual (CQ, Montreal)	jupyterhub.rorqual.alliancecan.ca	
Trillium (SciNet, UofT)		ondemand.scinet.utoronto.ca
MagicCastle	jupyter.demo.hpc.umanitoba.ca	
GreX		ood.hpc.umanitoba.ca

Notebooks , kernels and packages

- A notebook is a Web server.
- The server is written in Python and JS; provides interfaces with “Cells”
 - Code cells; Markdown documentation cells
- What does runs in the cells? “Kernels”.
 - Kernels support a particular language (R, Python, Julia, etc.)
 - Many kernels have to be “installed” on user level
 - Some are available as Lmod modules on CC software stack
- How to manage kernels? Per-user command line tools (jupyter and corresp Language).

`pip install ipykernel jupyter && python -m ipykernel install --user --name=my_python3`

`jupyter kernelspec list ; or jupyter kernelspec uninstall my_my_python3 ; etc.`

More information: <https://um-grex.github.io/grex-docs/specific-soft/jupyter-notebook/>

Managing Python dependencies with pip

- Dependencies: Pip handles (mostly) Python dependencies, relying on OS for non-Python ones
 - a. Unlike conda, uv etc, that would package all binary dependencies as well.
 - b. HPC folks like to provide their own optimized binary software : load “modules”!
- Repositories: pip fetches packages from <https://pypi.org/> or a local repository.
 - a. On CCEnv, each and every package must be repackaged to their “wheelhouse”.
 - b. On local software stack SBEnv, manylinux wheels can be used from pypi.org
- Installation destination: in particular when invoked in a Jupyter notebook cell.
 - a. user’s home directory `$HOME/.local/{bin,lib,share}` . (GreX)
 - b. Throw-away virtual environment under `$SLURM_TMPDIR` (Alliance HPC , MagicCastle)
 - c. Explicit virtualenv (create, activate and install) : best method. Needs a Jupyter kernel added.

Exercise: data analysis with Pandas

We will load some Eventbrite registration data (anonymized!) and analyze using the Pandas Python package

- Load necessary modules in Jupyter; using LMod tab.
- Get a Jupyter kernel for Python with JupyterLab launcher
 - a. As an exercise, can create/install kernel in a virtualenv
 - b. Or pick from a module.
- Import packages
- Use a notebook to load the data and analyze
- *Hands-on exercise : copy the materials and open notebooks in JupyterLab notebook*
 - a. On Grex: **`cp -r /global/software/ws-oct2025/jupyter_pandas ~`**
 - b. On Magic Castle: **`cp -r /home/shared/ws-oct2025/jupyter_pandas ~`**



**University
of Manitoba**