Final Report on " A JupyterHub Server to Enhance the Use of Python in Meteorology Coursework at Valparaiso University "

Kevin H. Goebbert

1809 Chapel Dr., Valparaiso, IN 46383

The Valparaiso University Meteorology program attempts to keep up with current meteorological technologies to allow our students to maximize their educational opportunities. Over the past several years of the pandemic-era of teaching, Valpo has benefited from the Unidata maintained instances of a JupyterHub server hosted on the Jetstream platform that allowed a (nearly) seamless laboratory experience for our students as they had to navigate in-class, at-home, and quarantine learning. These server instances allowed for students to only need a browser and internet connection to maintain progress in the course, work on homework assignments, and complete their projects. Without this vital resource we would have had a much harder time teaching and learning through this difficult period. As a result of the success with these hosted instances, we desired to have a resource that would allow for this flexibility, but also link to our internal system and data resources. This equipment grant allowed Valparaiso University to invest in a robust server to host a local JupyterHub instance, which we have locally established as *joanne.valpo.edu* after Joanne Simpson.

This has quickly become the asset that we had hoped as students routinely used this resource to help them work both in the lab and outside to make progress on their various tasks. Due to supply chain issues we did not quite have this resource ready at the start of the Fall 2022 semester, but once it was deployed in early October, students began to rely on this system to the point that they were often the first to know when it went down!

A key element to our setup is that the system authenticates users using our University system, which means students don't need to have another account or a separate password to remember. In addition, whether they log into our local Linux computing lab or through a web browser to the JupyterHub server they have the same access to all of their files/notebooks that were generated initially on either system. This seamlessness has enabled the best aspects of using local resources when convenient, but having a remote access option for other times. This also had the added benefit that one student encountered issues without local system when running Jupyter Lab, but was able to use our server without any hiccups. We look forward to being able to use this resource for years to come and have other departments looking to get accounts on our system to help them learn and adapt their coursework to meet the needs of their disciplines with regard to Python related activities.

We were helped greatly in the installation of our system through notes passed along by Kevin Tyle of the University at Albany who has set up a number of these types of systems at his institution. Due to the ever changing nature of the software that lies behind the JupyterHub server, we had to make modifications to work for our institution. The procedures followed for our institution can be found in Appendix A. These set of instructions are updated as needed based on the learned experience and problems that arise from the use of such systems. In

addition to the notes provided by Kevin Tyle, use of the installation instructions for the Littlest JupyterHub (https://tljh.jupyter.org/en/latest/) were also utilized.

On performance, the machine specifications more than meet our needs, although we have not had a chance to do as extensive testing due to supply chain delays in getting the equipment and then delays in installing the software. However, from over a half semester of use, I did not receive any notice from students about the performance of the machine. The same primary challenges remained for students, which were largely related to remote access of some large datasets that were present for both students working on local machines and on the JupyterHub server. At no point have we experiences "using up" the computer resources during high utilizations times (e.g., during class). We hope to document more usage statistics and report at future committee meetings and/or conference presentations.

There are currently two courses that make extensive use of Python computing resources: Weather Technology and Climatology. The notebooks used in these courses are publicly available at https://github.com/kgoebber/valpo_courses and are updated in the semester those courses are taught. In addition, this fall the senior synoptic meteorology course also used this resource to aid in their analysis and visualization of a case study event. They were provided base Jupyter Notebooks to help in their work and those notebooks are available at https://github.com/kgoebber/synoptic_meteorology/tree/master/case_study . In the future we hope to create a single repository for all University coursework in meteorology to ease the burden of maintenance and allow these resources to outlive individual faculty members. If/when this is accomplished, links to those other resources will be put into the repositories noted in this document.

Finally, this resource has contributed to research projects that have commenced this fall with our staff meteorologist and a few students. Having a single resource to share an environment with access to our long-term data storage has been a real asset to our program and the work that can be generated by our faculty, staff, and students.

Appendix A

JupyterHub Installation Instructions at Valparaiso University

Installation of Jupyterhub on joanne

John Lawson, August 2022

Updated November 2022 (second dev server)

Assistance from Kevin Tyle, Kevin Goebbert, Jon Sanders

- During tests, might want to do something like, "if server is `joanne`, then activate env 2, else 1", to avoid using your own environment upon login by accident)
- Try creating a conda environment that is geopandas + wx2022a + jupyterhub requirements below
- You could do the whole thing in a `screen` or `tmux` window (persistence) - right now, it's not calling the right environment when that happens, so that needs fixing and testing
- We are assuming LTS 20.04 (?) Ubuntu. Older versions of http-configurable-proxy didn't work (see Kevin Tyle docs).
- The Jupyter setup will need testing every semester or year when a new environment is updated, or bug-fix patches are applied. Might want to do this at "weird hours".
- Authentification is "wildcard" at Valpo, so some user guides do not apply regarding SSL, secure tokens, etc, outside of the HTTP server
- LOCK EVERYTHING DOWN HAHA
- Cronjobs to check things like running processes, "cull idle servers", monitor CPU usage, etc. A lot of logging would be good. Data viz reports for Kevin.
- How to make the server user-friendly for faculty to research.
- Maybe we start using Docker.

Installation process

- Log in as a sudo-group user (e.g., jlawson4).
- Run `sudo apt-get update` and `sudo apt-get upgrade` to ensure up-to-date software versions.
- Create new user with `sudo useradd jupyterhq -r -s /bin/false` to prevent logins (no password; no shell)
- Install anaconda (for python):
  - Download anaconda with `wget https://repo.anaconda.com/archive/Anaconda3-2022.05-Linux-x86_64.sh`. This was the latest at time of writing; download a newer anaconda install script by going to the download page online, right-clicking and "copy download link", and replace the URL above with this link.
  - Run the install with `sh ./name` where `name` is the downloaded shell script.
  - Accept all questions. Default installation path works fine. The unpacking/extracting may take a while.
  - I ran `conda init` when asked, and also ignored a warning about the `PYTHONPATH`.
  - Log out and log back in to reload the environment. (This is a good thing to do regularly when testing to check the environments are OK.)
- Set up jupyterhub conda environment:
  - Put `conda-forge` as the preferred installation channel with `conda config --add channels conda-forge`.
  - Create a new environment for serving jupyterhub with the following. Note, at the time of writing, that 3.9 was chosen to be compatible with `geopandas`.

```
conda create python=3.9 jupyterhub jupyterlab jupyterlab_server sudospawner --name jupyterhub_env
```

- Activate the new environment with `conda activate jupyterhub_env`.
  - Add this as the bottom line of your `~/.bashrc` perhaps. Note that `bergeron` and `joanne` share the same bashrc. A user using python on both servers may want to use a `if ... then` block when sourcing the correct environment upon login.

> Note: I am testing moving /home/jlawson4/anaconda3 to /jupyterhq below

- Now allow jupyterhq to spawn servers for users (can also be tweaked to be a group instead; outside this scope):
  - ~~Open the sudo file with `sudo visudo` (NO)~~
  - Create a new file with `sudo vim /etc/sudoers.d/jupyter_sudo` (filename not important). This will be imported along with the "visudo" file so it is safer to edit.
  - Enter the following:

~~# Cmnd_Alias JUPYTER_CMD = /home/jlawson4/anaconda3/envs/jupyterhub_env/bin/sudospawner~~

```
 Runas_Alias JUPYTER_USERS = jupyterhq, jlawson4
Cmnd_Alias JUPYTER_CMD = /jupyterhq/anaconda3/envs/jupyterhub_env/bin/sudospawner
jupyterhq ALL=(JUPYTER_USERS) NOPASSWD:JUPYTER_CMD
```

Notes on each line: 1. All desired jupyterhub users in the first line other than the "hub host/hq" and "admin" Note a group reference could be used instead of JUPYTER_USERS. 2. This alias's path will change depending on where you installed `sudospawner` via conda 3. This allows `jupyterhq` to run `JUPYTER_CMD` for regular users.

Testing the current configuration with sudo

Try the following (using the same path to sudospawner as before)

~~sudo -u jupyterhq sudo -n -u $USER /home/jlawson4/anaconda3/envs/jupyterhub_env/bin/sudospawner --help~~

```
sudo -u jupyterhq sudo -n -u $USER /jupyterhq/anaconda3/envs/jupyterhub_env/bin/sudospawner --help
```

This should work. Now try the following.

```
sudo -u jupyterhq sudo -n -u $USER echo "Valpo is really cool"
```

This should fail by requesting a password (a second one after your current user's).

Limiting resources

TODO - for now, parallelisation should be turned off by default, perhaps?

Jupyterhub directory

- Create a new folder with `sudo mkdir /jupyterhq` .
- ~~Give permissions with sudo chown jupyterhq /srv/jupyterhub - not sure - later~~
- We will initially make this folder universally readable, writable, and executable whilst we set this up. **Do not leave things in this state.**
- Initially, `sudo chmod -R 777 /jupyterhq` .

Now the following should spin up a new instance:

```
sudo -u jupyterhq jupyterhub --JupyterHub.spawner_class=sudospawner.SudoSpawner
```

The issue is that `jupyterhq` cannot be created due to mounting on Bergeron, so we will create the home directory of `jupyterhq` in `/jupyterhq` . **Note I'm not sure what Jon Sanders did here and we should run the guide by him.**

Template `.bashrc` was then put into the new home directory (`/jupyterhq`)

Add to .bashrc of `jupyterhq` user

The following is the minimal `/jupyterhq/.bashrc` file:

```
> Copy here
```

Installing `configurable-http-proxy` .

Run the following:

```
sudo apt-get install node.js
rpm install configurable-http-proxy
```

Move to /jupyterhq.

```
sudo chmod -R 777 /jupyterhq
jupyterhub --generate-config
sudo openssl rand -hex 32 > jupyterhub_cookie_secret
sudo chmod 600 /jupyterhq/jupyterhub_cookie_secret
```

Returning permissions to jupyterhq only (600) is required by `jupyterhub` itself in the name of security. After all, users do not need to modify any element of the jupyterhub configuration once working.

Open `jupyterhub_cookie_secret` and copy that token. Now open jupyterhub_config.py and add

~~c.ConfigurableHTTPProxy.command = '/home/jlawson4/anaconda3/envs/jupyterhub_env/bin/configurable-http-proxy'~~
~~c.SudoSpawner.sudospawner_path = '/home/jlawson4/anaconda3/envs/jupyterhub_env/bin/sudospawner'~~

```
c.JupyterHub.admin_access = True
c.JupyterHub.cookie_secret_file = '/jupyterhq/jupyterhub_cookie_secret
c.ConfigurableHTTPProxy.command = '/jupyterhq/anaconda3/envs/jupyterhub_env/bin/configurable-http-proxy
c.JupyterHub.spawner_class = 'sudospawner.SudoSpawner'
c.SudoSpawner.sudospawner_path = '/jupyterhq/anaconda3/envs/jupyterhub_env/bin/sudospawner'
c.Spawner.http_timeout = 90
c.Spawner.start_timeout = 120
c.Authenticator.admin_users = {"jupyterhq","jlawson4"}
c.ConfigurableHTTPProxy.auth_token = ####
```

and replace the hash marks of the final line with the token you copied earlier.

Running the server

Start it with

~~sudo -u jupyterhq /home/jlawson4/anaconda3/envs/jupyterhub_env/bin/jupyterhub --config=/jupyterhq/~~

```
sudo -u jupyterhq /jupyterhq/anaconda3/envs/jupyterhub_env/bin/jupyterhub --config=/jupyterhq/jupyterh
```

If it fails, try

```
ps aux | grep configurable
```

and kill any existing processes with `kill <ID>` where ID is the running process. This could be automated on reboot or periodically

Stability and security

TODO: * Stress-test * Limit CPU usage * Which users are admin * Aliases and functions to make things easier * Cull servers thing? * Check for http-config running and kill it * Dashboard for checking joanne's vital signs

Supplementary material

TANGENT: easier group management

This is paraphrased from https://github.com/jupyterhub/jupyterhub/wiki/Using-sudo-to-run-JupyterHub-without-root-privileges:

> As an alternative to add every user to the /etc/sudoers file, you can use a group reference at the end of the file, instead of JUPYTER_USERS. The following two lines added to the sudoers file allow 1) the group to execute sudo as user 'rhea', and 2) rhea to run as a designated 'jupyterhub' group user the JUPYTER_CMD with no password prompt. Without the first line, you may get an error like: 'zoe is not in the sudoers file. This incident will be reported.' This error will also appear unless you add the new users to the jupyterhub group, as shown in the next example.

```
%jupyterhub ALL=(jupyterhq) /usr/bin/sudo
jupyterhq ALL=(%jupyterhub) NOPASSWD:JUPYTER_CMD
```

> Provided that the jupyterhub group exists, there will be no need to edit /etc/sudoers again. A new user will gain access to the application easily just getting added to the group:

```
$ adduser -G jupyterhub newuser
```

Kevin looking at auto.conf etc

```
sudo vim /etc/auto.home
```

And add

```
-fstype=nfs,rw,uid=$USER,gid=$USER,soft
```

Next,

```
sudo vim /etc/fstab
```

And add

```
bergeron.valpo.edu:/archive /archive nfs defaults 1 1
```

Then `sudo mount -a`. We will want to add `archive data scratch` to fstab

Need to allow IP address of joanne to connect to Bergeron

`nfsmount.conf` in etc

update

It is best to add the following to the `sudo.d/<whatever>` script:

```
 # group reference to allow all members to metjupyter group
%metjupyter ALL=(jupyterhq) /usr/bin/sudo
jupyterhq ALL=(%metjupyter) NOPASSWD:JUPYTER_CMD
```

We manually add existing users with a script based around

```
sudo adduser <username> metjupyter
```

update

At the start, we need to create new anaconda environment via `sudo -i -u jupyterhq` (might want to alias to the executables so you can type conda, not `/juypterhq/anaconda3...`. Then use `conda install <package> -n jupyterhub_env`. - Manually, I found this 4-part order didn't choke using 3.9 (some of these are redundant but whatever): - geopandas, metpy - ipywidgets, jupyter, jupyterlab, notebook, scikit-image, scikit-learn - bokeh, nose, plotly, wrf-python, arm_pyart, eofs, netcdf4, siphon - jupyterhub, jupyterlab_server, sudospawner Once geopandas works with 3.10, we can try again.

Set `PROJ_LIB` (deprecated) and `PROJ_DATA` to `/jupyterhq/anaconda3/envs/jupyterhub_env/share/proj`.

```
$ sudo -i -u jupyterhq
$ source ~/.bashrc
$ conda update --all
```

To run two different environments side-by-side

I did the following:

- Made a new folder within /jupyterhq called class496 (named for a course)
- Copied the jupyterhub_config.py as a template, but changed the following:
    - `c.JupyterHub.config_file = '/jupyterhq/class496/jupyterhub_config.py'`
        - And so on... Update paths. Find/replace to continue changing the paths like this
    - `c.JupyterHub.hub_bind_url = 'http://127.0.0.1:8082'` and `c.JupyterHub.hub_port = 8082`
        - Hub port bound (?) from 8081 to 8082
    - `c.JupyterHub.bind_url = 'http://:8496'` and `c.JupyterHub.port = 8496`
        - ports from 8000 to 8496 and ask IT to open the port
    - `c.ConfigurableHTTPProxy.api_url = "http://127.0.0.1:8765"`
        - Explicitly set API port to 8765 arbitrarily
    - Change the `auth_token` from the recreated hex code
    - Allowed users just to class members3
        - `c.Authenticator.allowed_users = {"student1","student2"}`
- Create new cookie_secret and put into jupyterhub_config.py file

Change sudoers.d/jupyter_sudo file to add:

```
Cmnd_Alias JUPYTER_CMD_496 = /jupyterhq/anaconda3/envs/class496/bin/sudospawner
jupyterhq ALL=(%metjupyter) NOPASSWD:JUPYTER_CMD_496
```

Edit `jupyterhq/.bashrc` and add:

```
if [ $DEV_ENV -eq 1 ]; then
    conda activate class496
    export PROJ_LIB=/jupyterhq/anaconda3/envs/class496/share/proj
else
    conda activate jupyterhub_env
    export PROJ_LIB=/jupyterhq/anaconda3/envs/jupyterhub_env/share/proj
fi
export PROJ_DATA=$PROJ_LIB
```

To make sure the correct environment is activated, we pass `DEV_ENV=1` into our initial command. Add an alias to make life easier, such as:

```
alias "start_jupy496"="(cd /jupyterhq/class496 && sudo -u jupyterhq DEV_ENV=1 /jupyterhq/anaconda3/env
```