Craig Allen

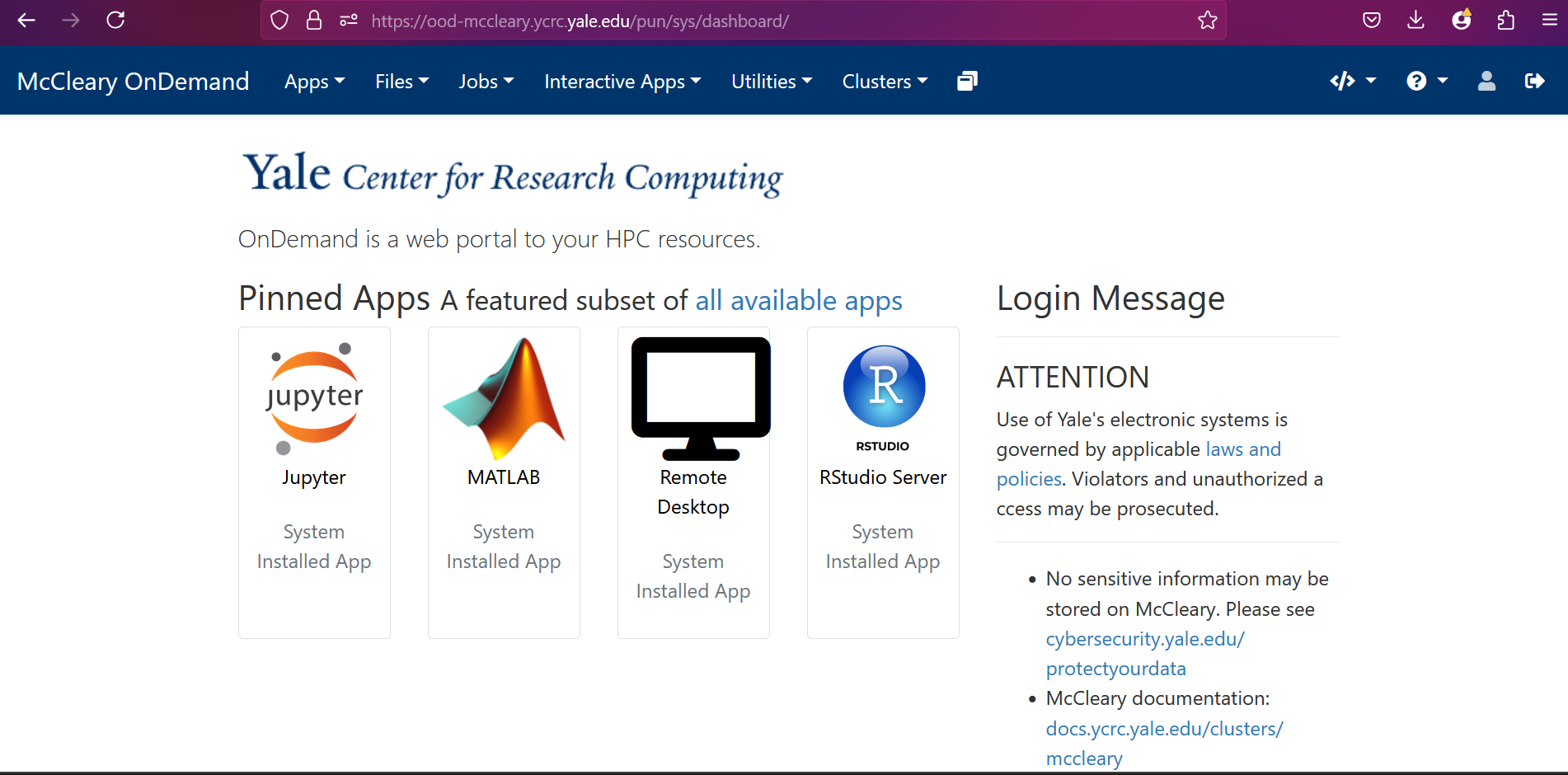
8-2-2024

Stites Lab

HPC Python Usage Guide

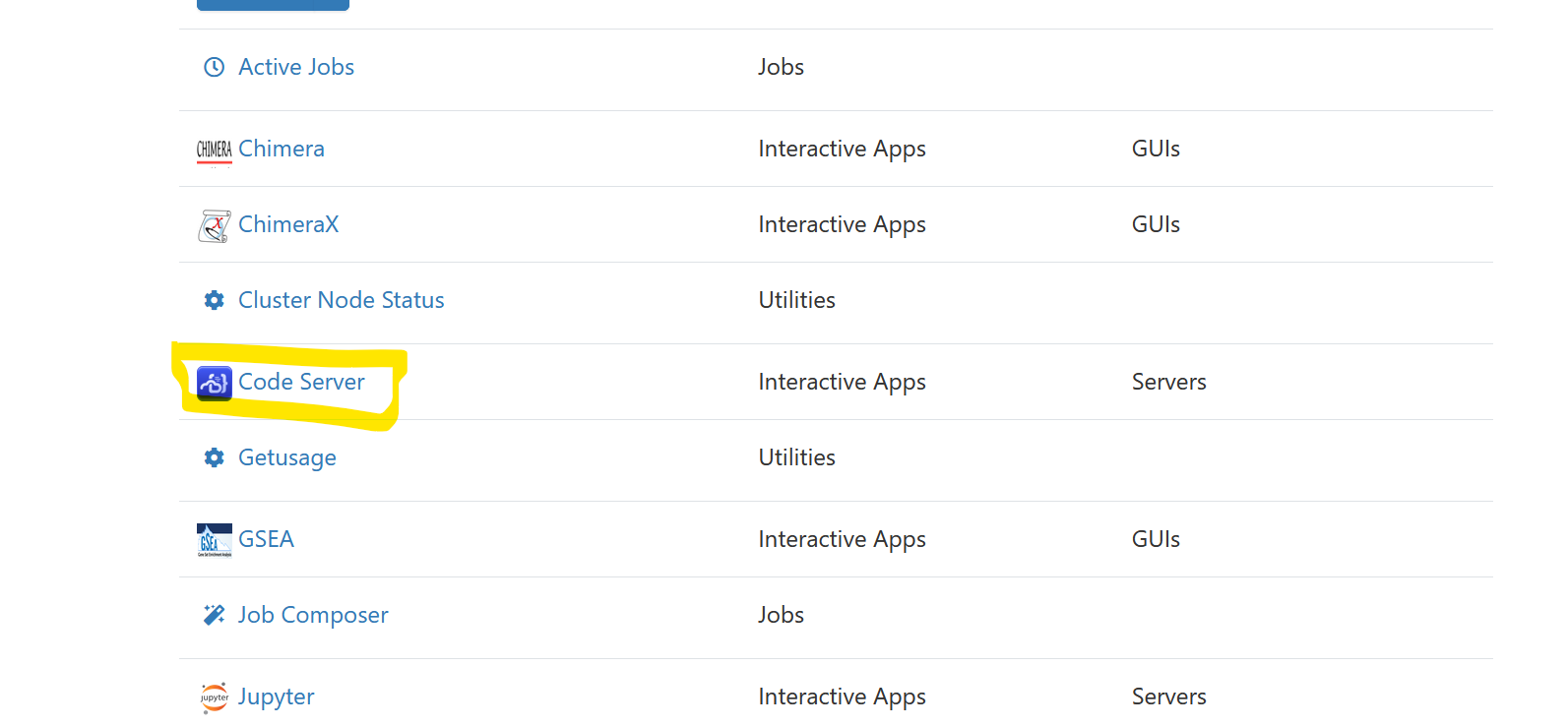
*Code Server*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Once access to the McCleary cluster is granted, visit [ood-mccleary.ycrc.yale.edu](https://ood-mccleary.ycrc.yale.edu) while on Yale network or on Yale VPN. You should be greeted with the following screen / something similar:

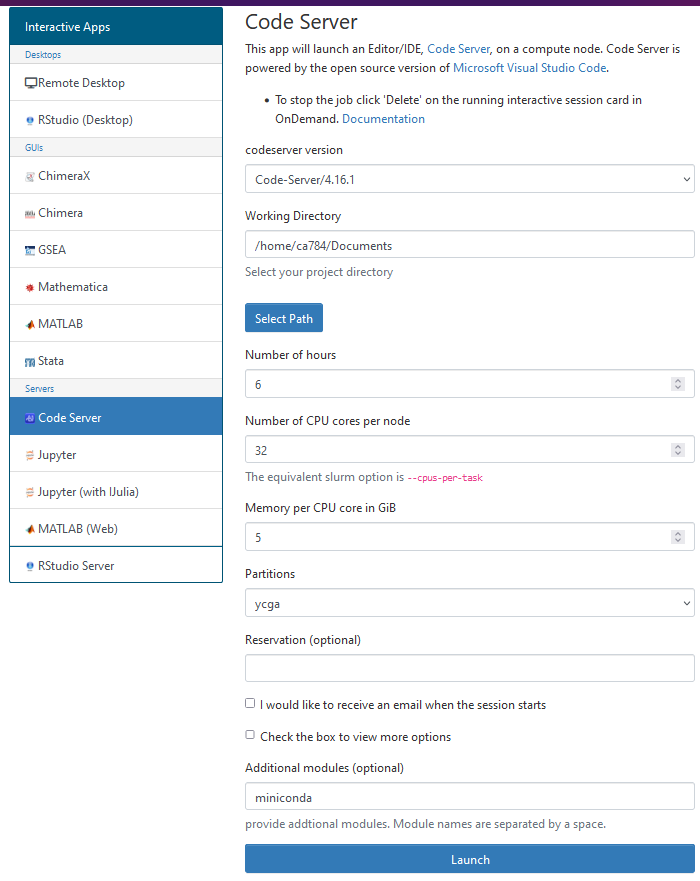


To use Python or another language on the cluster, I find using the VSCode app the best choice. Instead of opening an actual desktop on the cluster, it opens a web instance of VSCode (which normally is an app on your computer desktop) to which any programming can be performed, as it is an industry standard for coding.

1. Navigate to all available apps then select Code Server.

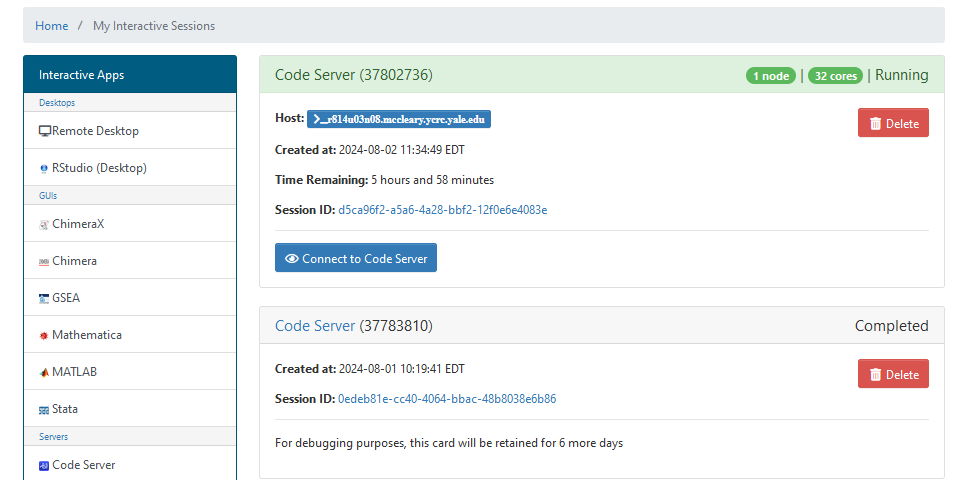


You should now be met with a list of options before starting the code server with the following screen:



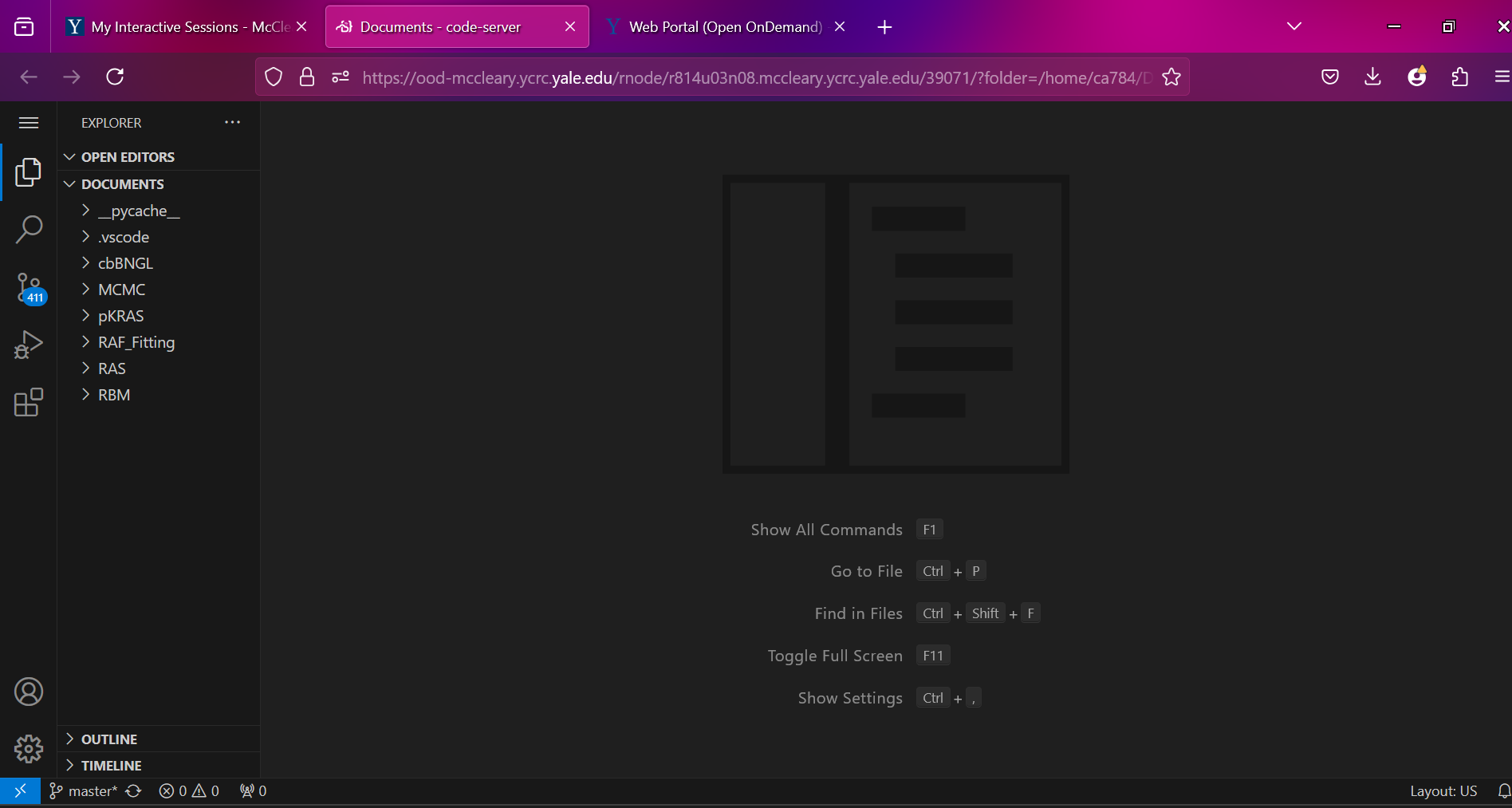
1. Fill in the **Number of hours** you want the server for. Once it reaches those hours it will shut down, so if you plan to work for a while you may want 6 hours or 8 hours etc.
2. The **Number of CPU cores per node** is useful for parallel processing. The MCMC code parallel processes evaluating each rule on each step, so the number of cores you need is really equal to the number of rules. I usually just put 32 which is more than enough.
3. For **Partitions** select ycga for the free computing. Other partitions have different costs associated with them and may take a while to queue to actually get your server, but ycga is usually fast.
4. For **Additional modules** I put miniconda which makes the code server automatically load with that package. This is used for managing Anaconda Environments, a useful way to keep track of which Python packages are installed. (e.g. you may have an anaconda environment you use for ML packages and another you use for bionetgen). **This is optional.**
5. Select **Launch**.

You should be met with the following screen, which lists all of your active sessions. It may take a little while to queue and start the server before you get the large blue connect button.



1. Click Connect to Code Server

Another browser tab should open with your Code Server session like the following:



On the left you can see the navigation screen where you can open up files in Documents. The Documents and entire directory are linked to your HPC account, so outside of this session you can still access these files under the Files tab in the first welcome screen when you log in. For example, if you made a new Python file, if you needed to, you could simply log into the HPC, navigate to files, and find that Python file.

From here using VSCode goes as normal using the high-powered CPUs you had requested initially. 😊

Some minor VSCode notes:

* when you go to run a Python .ipynb notebook file, you have to select a “Kernel” (top right corner and/or on command bar at top). This refers to what Python environment (or Anaconda environment) you would like to use. By default, with Python installed, it should allow you to select a regular Python kernel when you run.
* To view the command line to download Python packages, view console outputs from a regular Python file, etc., a quick way is to click this button in the bottom left, then navigate to “Terminal”:

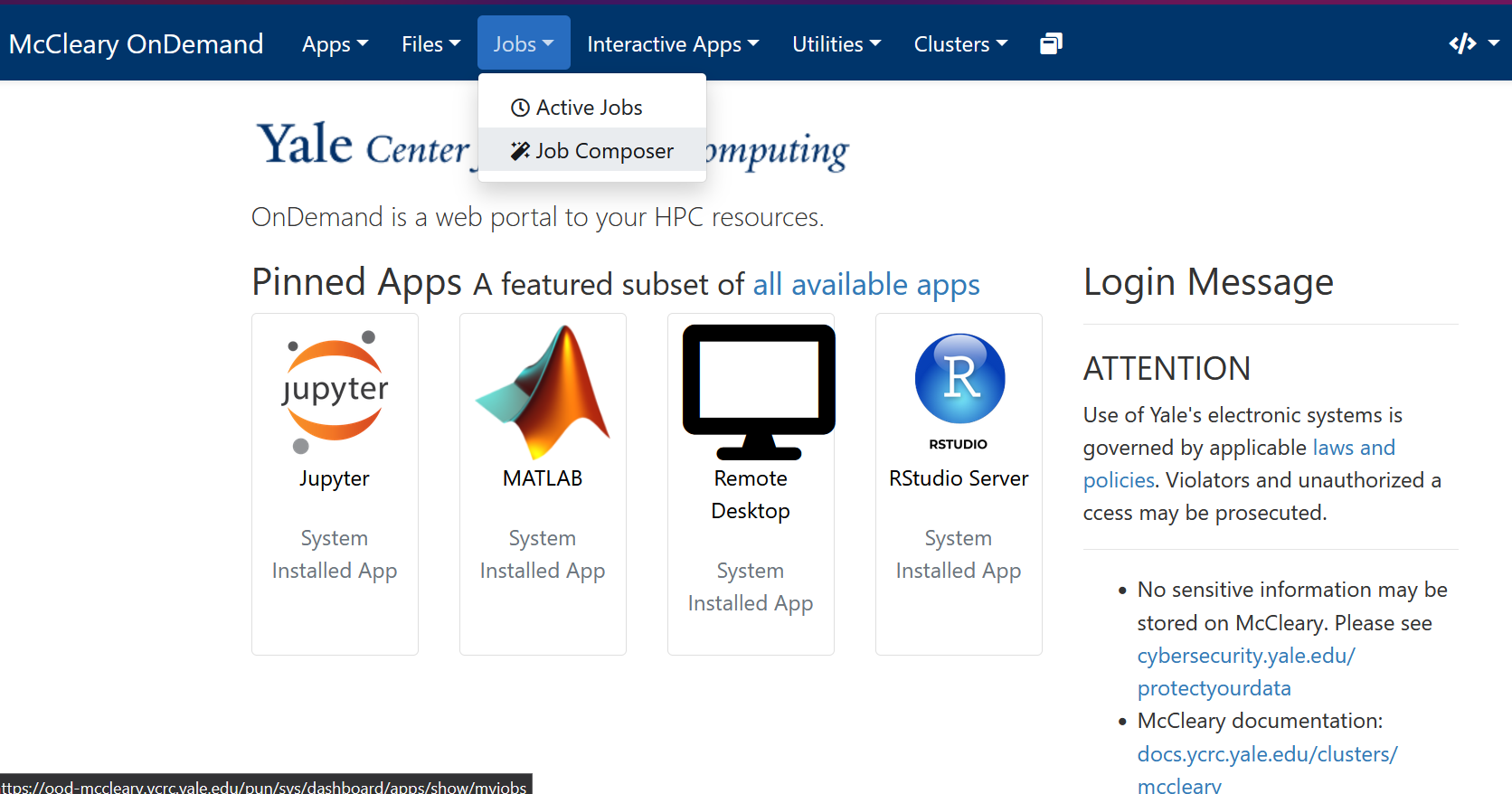


* Python should be installed by default. If it is not, navigate to the packages tab on the left of the screen, search for Python, and install the package.

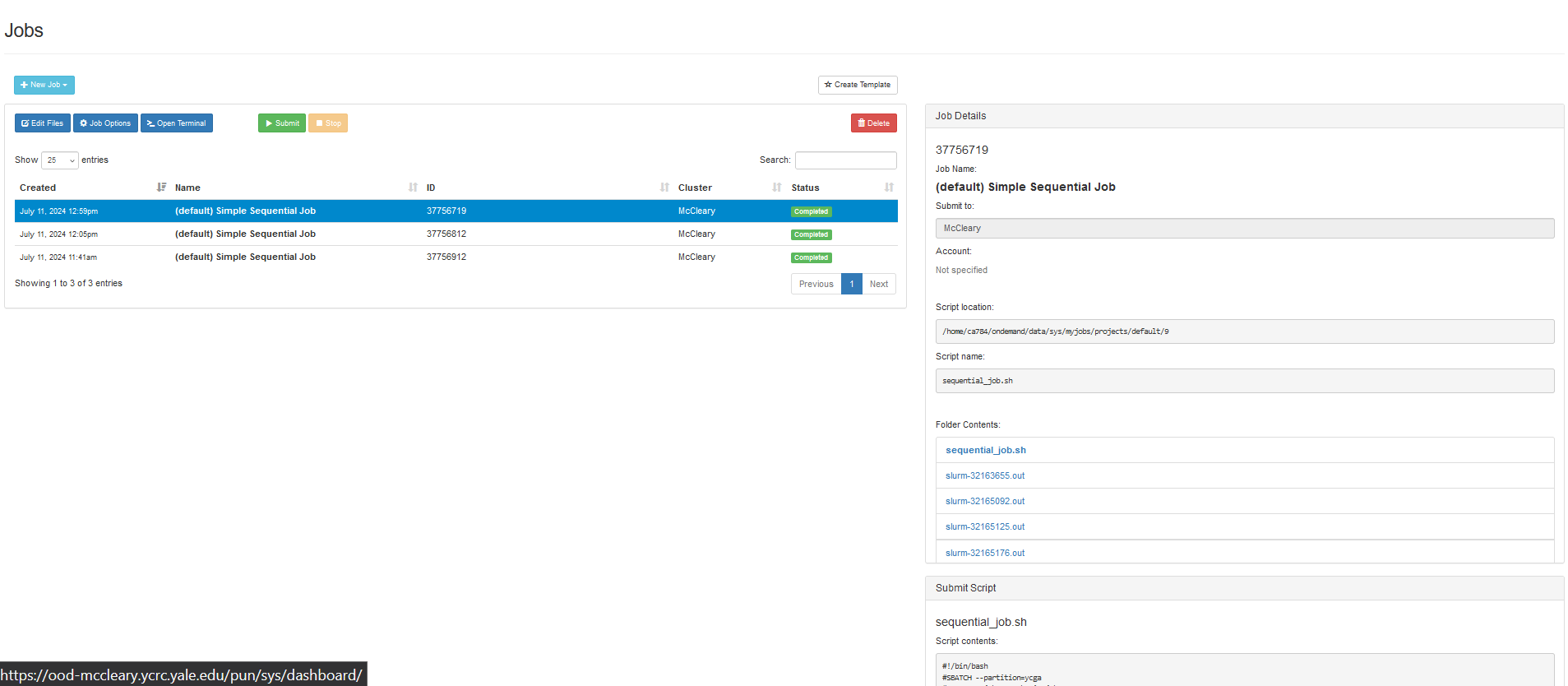
*Running Jobs*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

For Python or other scripts that take too long to run, Jobs can be used instead of opening a Code Server. In this case (e.g. in the MCMC case), a notebook is not used. A Python file with a def \_\_main\_\_(): function must be used (explained more in MCMC guide). The general pipeline of the cluster, is each job has a bash script (script formatting just like you would run commands in the command line of a terminal). The Job Composer application will automatically generate you this file with all the required information to run. It is up to you to modify the arguments they generated (similar to when starting a code server). For example, you may modify the time it needs (for MCMC its set at 24 hours just in case). The job will run until its finished (or failed) or until it hits the maximum time. Outputs are output to a log file which is extremely useful for determining if your program successfully run, any errors, etc.

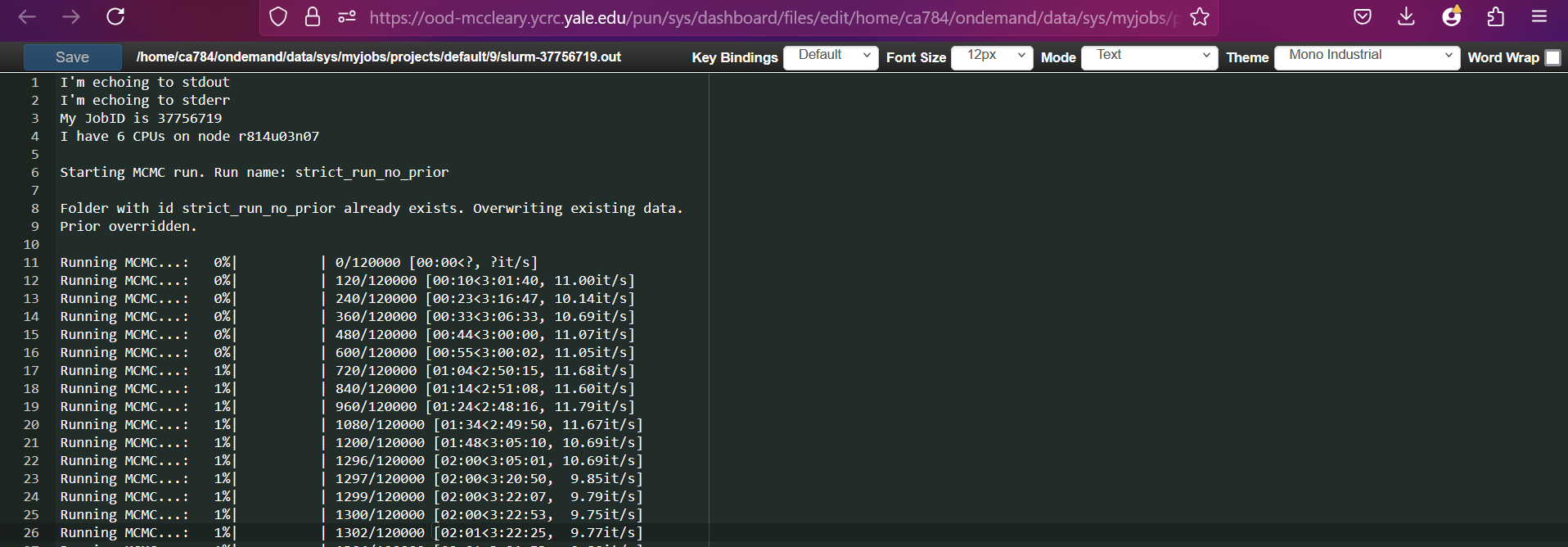
1. First navigate to the Job Composer from the OnDemand screen:



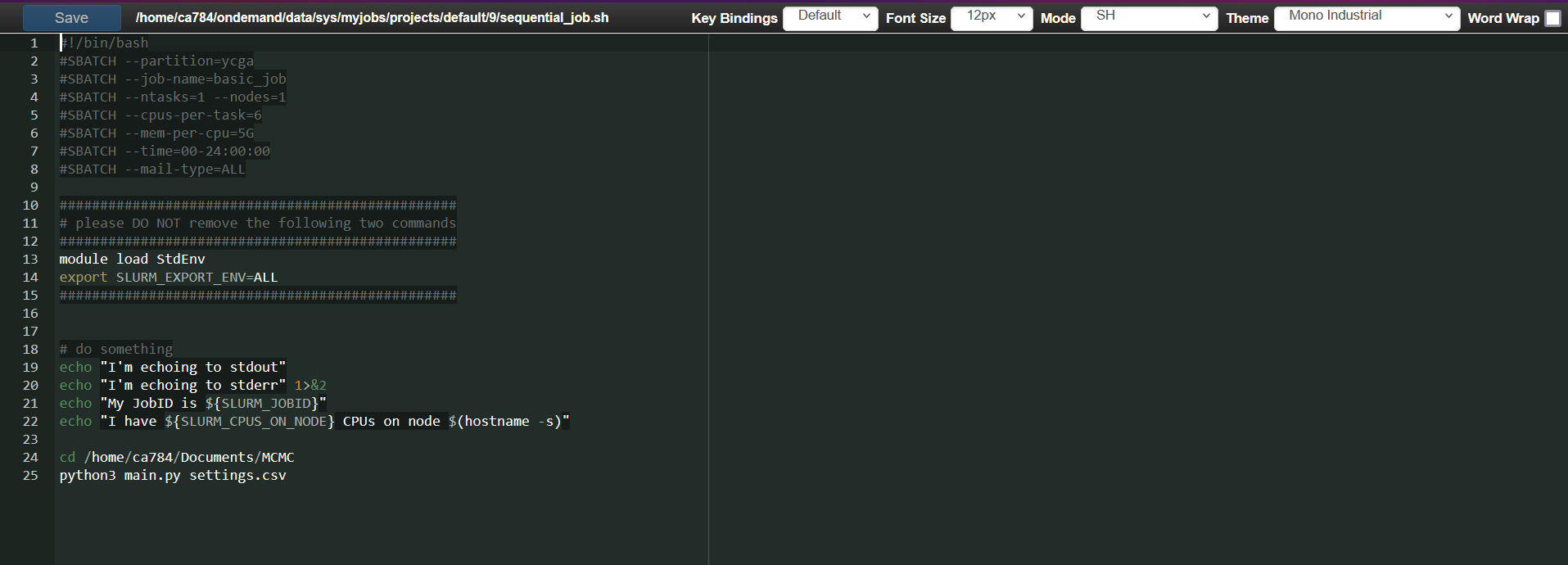
You will see the following tab open:



There is a lot to unpack here. The main screen on the left has a list of jobs you have created and will tell you their status (e.g. Completed, Running, Failed, etc.). The right **Job Details** will tell you basic details about it. You can ignore most of it except **Folder Contents**. This is where you will find the terminal log files from each run. Every time you run the job, and it starts, it will generate a new file here and *live* update the file as the job runs. So as a job is running, you can navigate to the last file in that list (the most recent), click on it, and view the outputs of the terminal as its running the job. You can then refresh that page for any updates.



To edit what the actual job *does* you can navigate to the **Submit Script** area in the bottom right, and it will show the script that the job will run in the command line. Clicking **Open Editor** allows you to open this file in a new page and edit it. Its worth noting, saving this text is a good idea as even if you make a new job you could just copy and paste the old job text to remember everything you need.



Everything until line 23 is automatically generated, and was edited for the settings I want. For example the cpus per task is set at 6, the time is set at 24:00, the partition is ycga. The final two lines (24 and 25) are what I added. Line 24 sets the command line at the directory where the Python file is that I want to run. Then line 25 simply runs that python file using the settings.csv file with the MCMC settings (more on this in MCMC section). **No matter what Python file you run, the Python file must include a function called def \_\_main\_\_(): where you code all the things you want the Python file to do (the MCMC main.py file has this).** Its good practice to put lots of error control and print statements so as this runs you output relevant information to the console.

1. Set up (edit) the job file with all the settings, the Python file and settings you want to run.
2. Select the job you want to run and click > Submit.

The job should Queue and then eventually Run. If the job is stuck in the queue for an excessive time, sometimes it is good to cancel the job then run again.

At this point you can run Python jobs on the cluster 😊! Its up to you (or the MCMC code I already made) to determine what to output, including what files to save (e.g. data or graphs).