A short(ish) guide on how to get Jupyter Notebooks up and running on the Bridges supercomputer.

jupyter-on-a-supercomputer.md

Running Jupyter on a Supercomputer

This quick guide for getting a Jupyter Notebook up and running on Bridges, a supercomputer managed by the Pittsburgh Supercomputing Center. Bridges is a new machine designed to accommodate non-traditional uses of High Performance Computing (HPC) resources like data science and digital humanities. Bridges is available through XSEDE, which is the system that manages access to multiple supercomputing resources. Through XSEDE, Bridges is available researchers or educators at US academic or non-profit research institutions (see the XSEDE eligibility policies) Allocations are free, but there is a somewhat difficult to understand application process filled with jargon and acronyms that take time to understand. See the XSEDE getting started guide for more information about getting access to Bridges. If you need help, check to see if your university has a campus champion, they are available to answer questions and guide you though the XSEDE allocation application process (and cut through all of the acronyms and jargon).

The rest of this guide is going to assume you have gotten an allocation on the Bridges supercomputer with a username and password. This guide is written for users of OS X or Linux who have a good understanding of the command line. That said, those who are just getting started with the command line should still be able to follow along by copying and pasting the commands (except when otherwise noted).

The Setup

Getting a Jupyter Notebook up and running on Bridges involves roughly six steps:

1. Connecting to Bridges
2. Installing Jupyter
3. Launch an Interactive Session
4. Get the Hostname
5. Launch the Jupyter Notebook
6. Reconnect to Bridges with an SSH tunnel

So let's get started!

Step One - Connecting to Bridges

The first step is to connect to Bridges using SSH. SSH is a command line utility for connecting to remote computers (or even "the cloud"). The command below has two parameters, the hostname for the Bridges login server (`bridges.psc.edu`) and a Bridges username. I know my username is `mcburton` so I connect with the following command. Replace `mcburton` with your username to log in with your account. Running this command will connect to the Bridges server and ask for your password (or mine if you didn't change the username).

```
ssh bridges.psc.edu -l mcburton
```

Once you enter your password you should see a message that looks something like this:

```
********************************** W A R N I N G **********************************
You have connected to br006.pvt.bridges.psc.edu

This computing resource is the property of the Pittsburgh Supercomputing Center.
It is for authorized use only.  By using this system, all users acknowledge
notice of, and agree to comply with, PSC polices including the Resource Use
Policy, available at http://www.psc.edu/index.php/policies. Unauthorized or
improper use of this system may result in administrative disciplinary action,
civil charges/criminal penalties, and/or other sanctions as set forth in PSC
policies. By continuing to use this system you indicate your awareness of and
consent to these terms and conditions of use.

LOG OFF IMMEDIATELY if you do not agree to the conditions stated in this warning



Please contact remarks@psc.edu. with any comments/concerns.

********************************** W A R N I N G **********************************
[mcburton@br006 ~]$
```

Congratulations! You are now on the Bridges supercomputer!

Step Two - Installing Jupyter

Step two is the most complicated step in this process, fortunately you only need to do it once. Once you install and configure the system you should never need to do this step again! Because step two is so complicated, it is best to think about it as two sub-steps. The first installs the software and the second configures the system.

Installing the Software

Bridges does not currently have Jupyter installed on the system, so we need to install it. The easiest way to do this is to install the Anaconda Python distribution. Anaconda includes Python and a bunch of third party packages that you might need for doing data science, including Jupyter. Anaconda is by far the easiest way to get everything installed in your home directory so you can start composing Jupyter Notebooks.

While Anaconda has a GUI installer, we need to install it on Bridges (not your local computer) To start, visit the Anaconda download page and click on the "Download for Linux" tab to see the link for downloading the Linux installer.

On this page we are interested in the 64-BIT INSTALLER for Python 3.5 (or Python 2.7 if that is the Python version you prefer). Note, we **don't** want to download the installer to our local machine, we want to download it on Bridges. This means we need to get the download link by right-clicking and selecting "Copy Link Address." Now a URL looking something like `https://repo.continuum.io/archive/Anaconda3-4.2.0-Linux-x86_64.sh` should be in your clipboard. Make sure the URL ends in `.sh`, the Linux installer, and not `.exe` or `.pkg` because those are the installers for Windows and OS X.

Now you need to go back to your terminal where you are logged into Bridges and download the installer using the `wget` command line utility and the URL in your clipboard. It should look something like the command below (this command should work as well, but eventually it will download an older version of the Anaconda installer):

```
wget https://repo.continuum.io/archive/Anaconda3-4.2.0-Linux-x86_64.sh
```

This command will take a while because it has to download a nearly 500 megabyte file to the remote server and for reasons that I don't understand it goes very slowly (this ***IS*** supposed to be a supercomputer right?). Once the download has completed you should see something that looks like this:

```
--2017-01-04 15:40:32--  https://repo.continuum.io/archive/Anaconda3-4.2.0-Linux-x86_64.sh
Resolving repo.continuum.io (repo.continuum.io)... 104.16.18.10, 104.16.19.10,
2400:cb00:2048:1::6810:130a, ...
Connecting to repo.continuum.io (repo.continuum.io)|104.16.18.10|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 478051940 (456M) [application/octet-stream]
Saving to: ‘Anaconda3-4.2.0-Linux-x86_64.sh’

100%[=====================================================================>] 478,051,940
1.71MB/s   in 8m 11s

2017-01-04 15:48:42 (952 KB/s) - ‘Anaconda3-4.2.0-Linux-x86_64.sh’ saved [478051940/478051940]

[mcburton@br006 ~]$
```

All of this gobbly gook means that you've downloaded the Anaconda installer into your home directory on Bridges. Now run the installer using the following command:

```
bash Anaconda3-4.2.0-Linux-x86_64.sh
```

The installer will ask you to hit ENTER and review the license agreement (keep hitting space bar to scroll through the text and then type "yes" to agree to the terms. Next the installer asks where you want to install the Anaconda distribution. The default location is the `anaconda3` directory in your home directory, this is an excellent choice, but you can select something else if you want.

Once you confirm the install location Anaconda starts the installation process. This can take a while, but you can get a rough estimate of the progress because it installs the packages in alphabetical order. Once the installation finishes you will be asked to prepend the Anaconda install location in your PATH. You should type "yes" to make it easier to run Jupyter without mucking around with explicit file paths. If you accidentally hit "no" (the default) don't worry, we will fix it below.

Configuring the System

Once Anaconda and Jupyter are installed there is one last thing to configure. By default Jupyter tries to use the directory specified by `XDG_RUNTIME_DIR` environment variable for writing some files. However, for the compute nodes on Bridges that directory doesn't exist, which causes Jupyter to crash when you run it on a compute node. To solve this problem, we need to set an environment variable `JUPYTER_RUNTIME_DIR` to point to a different folder that does exist. There is a bit more information about the runtime environment in the Jupyter documentation. In our case, we are going to create a new folder in our home directory.

First, create a folder in your home directory for jupyter runtime stuff. The following command creates a folder in your home directory called `runtime`. Note, you should only have to do this the first time you try and run Jupyter. Once you have created the directory you can just set the `JUPYTER_RUNTIME_DIR` environment variable.

```
mkdir ~/runtime
```

Next, add a command to set `JUPYTER_RUNTIME_DIR` to the path of that directory to your `.bashrc`. By adding this command to the `.bashrc` it will automatically set these variables every time you log into the system.

```
echo 'export JUPYTER_RUNTIME_DIR=~/runtime' >> .bashrc
```

OK, now everything *should* be ready to go!

To check and see if the install and configuration run the following command (or log out and log back in again):

```
source .bashrc
```

Now check and see if Jupyter is installed on the system. Run the following command:

```
jupyter
```

You should hopefully see output that looks something like this:

```
usage: jupyter [-h] [--version] [--config-dir] [--data-dir] [--runtime-dir]
               [--paths] [--json]
               [subcommand]
jupyter: error: one of the arguments --version subcommand --config-dir --data-dir --runtime-dir --
paths is required
```

If you see this then it means Jupyter is installed. Congratulations! If instead you see something like

```
-bash: jupyter: command not found
```

It means the installation of Anaconda didn't work quite right. This can happen if you accidentally said "no" to the last question in the Anaconda installer. If this is the case, you probably need to modify your `.bashrc` to modify your PATH by running this command:

```
echo 'export PATH="/home/mcburton/anaconda3/bin:$PATH"' >> .bashrc
```

Now either log out and back in again or run the `source .bashrc` command listed above and then try running the `jupyter` command again. Hopefully it will work.

Step two is probably the most complicated step of this process, thankful ***you never need to do this again***. In the future you can skip over step two and go directly to step three, connecting to a compute node so you can run Jupyter. Now we need to venture deeper into Bridges infrastructure to find a nook where we can launch the Jupyter Notebook server.

Step Three - Launch an Interactive Session

When you first connect to Bridges you connect what is called the "login node." We don't actually want to perform any of our computational work on the login node of Bridges, instead we need to connect to a "compute node." Compute nodes are much more powerful and have far fewer users connected to them than the login node. Think of the login node as hote lobby of the supercomputer where you handle administrative tasks like installing Anaconda or transferring files. A compute node is like your private hotel room where you can run Jupyter and do your computational analysis.

Bridges uses a tool (unfortunately) called Slurm (no connection to the other, far more popular, slurm) to run processes, o "jobs," on compute nodes. Slurm is like the reception desk (or maybe concierge) in that it gives you access to compute nodes on Bridges. It is also the process that tracks your usage in the XSEDE allocation system.

There are a lot options for running jobs using SLURM on Bridges, but for the purposes of this tutorial we are going to focus just on running what is called an "interactive session." This just connects us to a compute node so we can run commands on the command line. It is essentially the same as SSH'ing into a compute node. Run the following command to start an interactive session:

```
interact
```

This will will use SLURM (specifically a command called srun) to locate an available compute node. Sometimes this process can take a while Bridges is very busy, but it usually shouldn't take more than 10 seconds. You should see some output that looks like this:

```
A command prompt will appear when your session begins
"Ctrl+d" or "exit" will end your session

srun: job 653500 queued and waiting for resources
srun: job 653500 has been allocated resources
[mcburton@r001 ~]$
```

If a command prompt, here it is `[mcburton@r001 ~]$` , pops up it means you are now connected to a compute node and ready to start supercomputing! But first, we need some information so we can tunnel into the compute node.

Step Four - Get the Hostname

Because the compute node is not publicly accessible, we will need to establish a "tunnel" to connect to it through the login node. In order to establish this tunnel we need to know the hostname of the compute node. To continue with the hotel analogy, he hostname is like the hotel room number. The "tunnel" is as if someone wanted to call you using your hotel room phone, to reach you they'd need to know your room number.

To get the hostname of the compute node you are using, you can run the following command:

```
        hostname
```

This should spit out something that looks like this:

```
r001.pvt.bridges.psc.edu
```

Now that we have the hostname we can create an SSH tunnel through the login node to this compute node (step six) and access the Jupyter Notebook server we will launch in the next step.

Step Five - Launch the Jupyter Notebook

Once everything is configured, we can now launch the Jupyter Notebook on the compute node. Note the two options specified in this command. The first command, `--no-browser`, tells Jupyter not to launch a web browser when it launches. We specify this because we are running Jupyter on a remote machine, which means it can't launch a web browser on your local computer. Instead it launches a text based browser in your terminal that isn't very useful. The second command, `ip=0.0.0.0`, tells Jupyter to serve the notebook on a public IP address. The default behavior is to only serve the notebook locally (for security reasons), but that doesn't work because we are trying to connect to a remote computer.

```
jupyter notebook --no-browser --ip=0.0.0.0
```

Once you launch this command you should see the following output:

```
[I 09:38:03.033 NotebookApp] [nb_conda_kernels] enabled, 2 kernels found
[I 09:38:03.050 NotebookApp] Writing notebook server cookie secret to
/home/mcburton/runtime/notebook_cookie_secret
[I 09:38:10.159 NotebookApp] ✓ nbpresent HTML export ENABLED
[W 09:38:10.160 NotebookApp] ✗ nbpresent PDF export DISABLED: No module named 'nbbrowserpdf'
[I 09:38:11.584 NotebookApp] [nb_anacondacloud] enabled
[I 09:38:11.645 NotebookApp] [nb_conda] enabled
[I 09:38:11.649 NotebookApp] Serving notebooks from local directory: /home/mcburton
[I 09:38:11.649 NotebookApp] 0 active kernels
[I 09:38:11.649 NotebookApp] The Jupyter Notebook is running at: http://0.0.0.0:8888/
[I 09:38:11.649 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to
skip confirmation).
```

Pay special attention to the second to last line in that output:

```
[I 09:38:11.649 NotebookApp] The Jupyter Notebook is running at: http://0.0.0.0:8888/
```

This tells us that the Jupyter Notebook Server is running on publicly on the compute node on port 8888 (http://0.0.0.0:8888). Make sure to take note of the port number, usually port 8888. We need that port number when we create the SSH tunnel in the next step.

Step Six - Reconnect to Bridges with an SSH Tunnel

Congratulations, you are at the final step in the process of running Jupyter on a supercomputer. Well, technically you already have the notebook running but you can't connect to it directly. What we are going to do is open up a *new* termina and re-connect to Bridges establishing an SSH Tunnel that allows us to connect your local web browser to the Jupyter Notebook server running deep inside the bowels of Bridges. Note, ***keep the old terminal open*** since that is the connection that is running the Jupyter Notebook server, if you close it you will shut down the server.

```
ssh -L 8888:r001.pvt.bridges.psc.edu:8888 bridges.psc.edu -l mcburton
```

This command will connect you to the Bridges login node, but it will also create a tunnel that connects TCP port 8888 on your local machine to the remote TCP port on the Bridges compute node, `r001.pvt.bridges.psc.edu`. The hostname might look a bit different when you work your way through the steps, but the end result will be the same. You should now be able to visit http://localhost:8888 and connect to the Jupyter Notebook web server running on Bridges. If it doesn't work, and you get a "This site can't be reached" error, double check that you have the right hostname for the compute node and the right port specified for the Jupyter Notebook server.

Now what?

Now you are ready to start composing Jupyter notebooks on a supercomputer! For general information about using the Bridges supercomputer, read the Bridges User Guide. This guide contains useful information about the various file spaces, how to get files into your workspace, and how to use advanced data science tools like Hadoop and Spark.

Next Steps

- Update documentation to deal with the new Notebook token authentication
- Install JupyterHub on Bridges so we don't have to do any crazy SSH tunneling.

dsaffo commented on Jun 26, 2017

bless you, this was extremely helpful

nealmcb commented on Jun 30, 2017

NCAR has installed Jupyter on their supercomputers along with a script that helps users set up the tunnels, by specifying command copy/paste. See https://www2.cisl.ucar.edu/resources/computational-systems/yellowstone/software/python/jupyter - it may be wort adapting or emulating. I figure there are other ways this could be made yet more convenient, but it is a wide-ranging set of tools to t together in a convenient way.....

amogh112 commented on Jun 30, 2018

Thanks a lot!
Updating this for how the cluster works now,
jupyter is found in any of the anaconda modules
On XSEDE system(remote)
interact #this starts a compute node
hostname #gets a hostname of the format - r001.pvt.bridges.psc.edu
jupyter notebook --no-browser --ip=0.0.0.0 #you may add a port also using --port=9999
On personal computer
ssh -L 8888:hostname_as found_earlier:8888 bridges.psc.edu -l username -p 2222 #replace the hostname that you had found
rest is the same.