The Emory Rollins School of Public Health

High Performance Computing Getting Started Guide

(V.2, Rev. 11/3/23)

“Welcome to the RSPH Compute Cluster!”

This guide is to help new users of the (new) RSPH High Performance Computing (HPC) Cluster quickly get up and running on the cluster. This document is meant to be a gentle introduction to HPC cluster usage and intended to be supplemented by other information sources that cover topics in greater depth. For other resources, **please see the final section on “Additional Resources.”**

About the HPC Cluster

The RSPH HPC cluster is a dedicated scientific computational system that combines the computing power of 120+ compute servers (in total at the time of this writing) for an aggregate total of nearly four thousand cores. It consists of 25 *general use compute nodes,* 24 of which have 32 compute cores and at least 196GB of RAM each. The remaining general use node is a “large memory node” with 1.5 TB of RAM. The other cluster systems consist of 70+ reserved use faculty nodes, which are also available for general, preemptive use when they are not in active use by their research groups. These systems are all connected together via 25GB Ethernet high-speed network, and all have access to a shared 1.2 Petabyte Panasas parallel file system. The system can support thousands of concurrent running jobs by dozens of simultaneous users.

In addition to the hardware, the system runs the Rocky Linux operating system (currently version 8), which is a “white-box” implementation of the Red Hat Enterprise Linux OS that purports to be 100% binary compatible with the commercial version. Job scheduling is handled by the SLURM job scheduler, which is an application that currently runs on the majority of the Top 500 supercomputing sites in the world.

(*NB: The above two paragraphs may be used as a “HPC Cluster description” for grant application purposes.)*

Access to the HPC cluster is free for all members of the RSPH community, although some fees may be required for storage beyond the initial allotment of 25GB per user. All RSPH faculty may request access to the cluster for themselves, and non-faculty may request accounts via sponsorship by an RSPH faculty member. Accounts are requested by emailing “help@sph.emory.edu". An active Emory NetID is required for all account recipients and should be included in the account request.

Important Cluster Policies

Please read all of this section before using the RSPH HPC Cluster. By using the RSPH HPC Cluster, you understand and agree to abide by the following policies as well as those policies enforced at the University level (see https://provost.emory.edu/faculty/policies-guidelines/handbook/information-technology.html for more details**).**

No "backups" of cluster user data are provided by the system or HPC staff. ***All users of the cluster are responsible for maintaining appropriate backups of their own data.*** Please consult with members of the HPC team if there are questions regarding best practices for maintaining and storing data long term. Suggestions for cluster data management follow later in this document.

The HPC Cluster has not been *certified* as HIPAA-Compliant. While a great deal of attention is paid to providing a secure computing environment for research computation, the HPC cluster is a shared computing environment and as such it is incumbent upon users of the cluster to protect data and computations from non-authorized access. Users are also encouraged to de-identify any potentially sensitive or protected data before transferring it to the HPC cluster, and additionally any responsibility for compliance with data usage agreements and university data policies falls upon the PI. If there are any questions or concerns about security features of the HPC Cluster, please contact the HPC Staff for more information.

Users may not share their cluster credentials with anyone, including colleagues who may already have access to the HPC cluster. Failure to abide by this rule can result loss of HPC cluster access. Users seeking help with access to the HPC cluster should consult the HPC staff.

Users agree to not run any sort of server or service from the HPC cluster. This includes, but is not limited to, data-copying servers, web servers, access point servers, crypto-mining servers or unsupported Graphical User Interface (GUI) services. If you have any questions about the appropriateness of an application for the HPC Cluster, please consult a member of the HPC staff.

Any attempts to circumvent the supported login/access and security features of the cluster will result in having the account closed and possibly any future access denied. Attempts to breach any security features of the cluster may also be referred to appropriate University officials and law enforcement.

HPC cluster accounts may only be requested by RSPH faculty. Faculty may "self-sponsor" accounts, but all others (students, staff and RSPH-external collaborators) must have their accounts sponsored by an active, primary-appointed RSPH faculty member. Access to the cluster for sponsored accounts will remain available while the sponsoring relationship is active (e.g, while the student is still actively pursuing relevant research with the faculty member, or while the sponsoring faculty member is still at Emory, etc.).

All computational jobs must be submitted to SLURM, the system resource manager and job scheduler. If you need to run a process on the login node (such as large file copying) please advise the HPC staff in advance for consultation. Any computational jobs not accounted for in SLURM may be killed without notice.

One may not run computations on the login node. Only processes for transfer of data and light file editing are appropriate on the login node. For other activities, please use a compute node via and interactive session (more information on interactive sessions follows in this document).

The above policies may be amended as needed before this document can be updated and republished. Please consult with the HPC staff if you have any questions regarding RSPH HPC cluster policies.

1 - Connecting to the HPC Cluster

To connect the the HPC cluster, one first requires access to the **Emory VPN** HIPAA-core. All users can self manage general access to the Emory VPN by following the instructions at (http://it.emory.edu/vpntools/access.html).

Once general VPN access has been configured by the user and VPN connection has been successfully made, the secondary access to the HIPAA-core will be granted that will allow access to the HPC cluster login node. **This access is requested on behalf of the user by the HPC team from the Office of Information Technology (OIT).**

With a successful VPN connection established, connections to the cluster are made via an an additional encrypted connection using the **Secure Shell**, or more commonly, **ssh.** To use ssh to access the cluster, one must use a local ssh *client.* Fortunately, all major operating systems now come with natively supported ssh clients by default based on the OpenSSH standard. The following operating systems have ssh clients that can be accessed in the following ways:

1) On Mac OS X system, open a terminal app and type **`ssh`**
2) Under Linux, open a terminal and type **`ssh`**
3) Under Windows 10 and later, open up a Powershell (search for "Powershell" in the search tool next to the Start menu) or CMD terminal and type **`ssh.`**

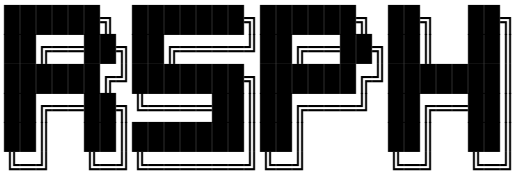
All user accounts on the cluster have userIDs that resemble their Emory NetID (in lower case) as the account name. To login to the cluster, type in the respective terminal

```
ssh <your_Emory_NetID>@clogin01.sph.emory.edu
```

(The '0' above in "clogin01" is a zero, not a capital 'O'.) You will be prompted for a password, and then immediately prompted to change that password to a new hard-to-guess password. **All passwords are required to be at least 10 characters long and contain at least one character each from lower case letters, upper case letters, digits and punctuation.** If your new password is rejected for any reason, the system will end your login attempt and you will have to try again.

Once you have successfully logged into the system, you will see the banner screen and a command prompt:

```
$ ssh testuser@clogin01.sph.emory.edu
testuser@clogin01.sph.emory.edu's password:

Welcome to the
```



```
High Performance Computing (HPC) Cluster

***  AUTHORIZED USE ONLY ***


.
Last login: Tue Aug 11 16:02:19 2020 from 10.110.100.86
[testuser@clogin01 ~]$
```

The Login Node (`clogin01`) and Compute Nodes (`node1-node94,...`)

Once logged in, you may then start submitting commands to the system. Please note that when one "logs in" to the HPC cluster, the user session is present on the *login node*, which is named "clogin01" as mentioned before and displayed by default in your shell prompt (also shown above).

This node is different from the remaining nodes of the cluster in that its sole purpose is that of being the gateway to accessing the cluster itself. For this reason, **no computations are permitted on the login node**. All computations must be submitted via the job scheduler, SLURM, to a partition of *compute nodes*. Jobs that run on the login node have the potential of causing interference to other's access to the cluster, and therefore may be killed without warning or notice when discovered by the HPC staff. (To work interactively on the HPC cluster within a terminal, please submit a SLURM request for an *interactive-cpu* session. Details of interactive-cpu sessions are covered later in this document.).

2 - Copying Data to the Cluster

There are multiple ways of getting data to the HPC cluster for computations. Below are some reliable suggestions; for more detailed information on linux commands, please consult one of the additional resources at the end of the guide.

scp

The most commonly supported method for copying data to the cluster is with the **`scp`** or "secure copy" terminal command, which provides encrypted transport of data to and from the cluster. All three major client operating systems (in their up-to-date versions) have a command-line scp client installed alongside ssh by default. To use the **`scp`** command you may either *push* or *pull* data to and from the cluster securely over remote networks. The general format for copying a file to a remote system takes the format of

scp <path_to_file> <username>@<remote_system_name>:<destination_path>

For example, to copy a file "testscript.sh" from your current directory on your local computer to the cluster as the cluster user "testuser", you'd use

```
scp testfile.sh testuser@clogin01.sph.emory.edu:.
```

which after authentication would copy the file testfile.sh to testuser's home directory (designated here by the dot "." after the colon) on the cluster. (Substitute your cluster userID for "testuser" in the above example for your own copying, of course.)

For more detailed treatment of the **`scp`** command, please consult one of the "Additional Resources" at the end of the Guide.

wget and curl

Two Unix command line options exist for downloading web-based data. These programs are often used in the context of downloading software packages. For instance, while logged in at the command line on the cluster, one can use **`wget`** to download the miniconda package to one's project directory :

```
$ mkdir -p /projects/my_lab/bin/miniconda3
$ cd /projects/my_lab/bin/miniconda3
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-
x86_64.sh -O miniconda.sh
```

```
--2023-11-08 14:56:28--  https://repo.anaconda.com/miniconda/
Miniconda3-latest-Linux-x86_64.sh
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.131.3,
104.16.130.3, 2606:4700::6810:8303, ...
Connecting to repo.anaconda.com (repo.anaconda.com)|
104.16.131.3|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 120771089 (115M) [application/x-sh]
Saving to: 'miniconda.sh'

miniconda.sh              100%[=====================================>]
115.18M  35.5MB/s    in 3.4s

2023-11-08 14:56:31 (34.4 MB/s) - 'miniconda.sh' saved
[120771089/120771089]
```

Curl can alternatively be used in a similar fashion. For a more in-depth treatment of the wget and curl programs, please either consult the on cluster man pages ('man curl') or one of the "additional resources" at the end of the guide.

git

One can also use the git command to clone code repositories to the HPC Cluster. This is a common way to download application software to the cluster, as software is often made available to the general public by sharing the location of git repositories (or "repos"). For instance, if one wanted to install the source code for the open-source Julia programming language, you could use the command

```
git clone https://github.com/JuliaLang/julia.git
```

The output is a bit lengthy, so it is not included here, but what would follow is a visual output of the downloading of the JuliaLang repository.

aws-cli

Data stored in the cloud at Amazon Web Services (AWS) can be easily access from the cluster with the use of the aws-cli command module. To use the aws-cli, one needs the set of access keys associated with their AWS S3 location (please consult documentation at AWS for details on acquiring access keys). One the keys are in hand, one can then load the AWS-CLI module and configure the CLI to communicate with the user's AWS resources, for example:

```
[testuser@clogin01 ~]$ module load aws-cli
[testuser@clogin01 ~]$ aws configure
AWS Access Key ID [None]: AK************7
AWS Secret Access Key [None]: c**********************v
Default region name [None]: us-east-2
Default output format [None]:
```

Now that the aws-cli is configured, it is ready for aws-cli commands:

```
[testuser@clogin01 ~]$ aws s3 ls
2023-06-12 11:00:56 emorydata
[testuser@clogin01 ~]$ aws s3 cp testsfile.txt s3://emorydata/testfile.txt
upload: ./testsfile.txt to s3://emorydata/testfile.txt
[testuser@clogin01 ~]$ aws s3 ls emorydata
2023-07-14 00:50:18         49 testfile.txt
```

This is just a very short example of the capabilities of the aws-cli. For more information about the use of the AWS Command Line Interface (AWS-CLI), please the AWS-CLI website (https://aws.amazon.com/cli/).

Globus Connect Personal

More and more often, users are wanting to transfer very large datasets from one institution to another. One of the emerging standards for such transfers is the use of the Globus data transfer network, a research-oriented service that aspires to reduce overhead and transfer times of large data repositories. While as of this writing Emory is not currently a member of Globus, users who have access to Globus at other institutions may install the Globus Connect Personal (https://www.globus.org/globus-connect-personal) client locally in their home directories and utilize the high-speed technology to copy very large data sets between institutions. The Globus Connect Personal toolkit does not require administrative privileges to install, and users are welcome install it in their computing environments. It is recommended, however, that the HPC staff be notified in advance of the import any really large data sets in oder to ensure the availability of sufficient storage space.

3 - Storage on the Cluster

The storage is presented via three volumes for access by users on the cluster:

- The `/home` file system, where every cluster user has a directory for their account
- And the `/projects` file system, where groups may store project data for a period of time and also run computations.
- The `/scratch` file system, where users may load data for immediate computation

There is also a fourth volume for applications, `/apps`, which is readable by cluster users. This is where binaries for cluster-provided software can be found, although it is preferred that applications be accessed via the **`module`** command.

Where do I put my data?

By default, all users of the cluster have a 25GB quota for their home directory located in `/home/<userid>`. There are no time restrictions on data in your home directory, so data may reside in your home directory as long as the account is active. The home directories reside in the PanFS file system which is mounted across all nodes of the cluster, so it is suitable to run computations from your home directory location provided you have enough space in your quota to write your output files.

For computations where a larger cluster-wide computation space is needed, the `/scratch` file system is available. This shared space is usable for large computations that are limited in time scope, as there is a **two-week maximum retention policy** for files in `/scratch`. Space in /scratch is available on a first-come-first-use basis for users that request quota space there. (Please contact the HPC management for possible policy exceptions.)

For reserved space exceeding the home directory quota, and for shared project space, the `/projects` directory is recommended. Faculty may request quota in `/projects` for use for their research or their sponsored accounts in increments of 1 TB. Files stored in `/projects` may remain for one year, depending on the quota arrangement.

T**here is no difference between the /home, /scratch and /projects volumes in filesystem performance**. All are PanFS volumes mounted across the cluster, and all take advantage of the features of the Panasas storage. The major differences are in the storage policies.

Quotas and Storage Volumes

Below is a table of the user-writable storage volumes on the HPC Cluster.

Volume	Quota	Purge Policy	Suitable for Job I/O
/home	25 GB/user	None	Yes
/project	1 TB/PI or Group*, additional fee for use option $108/TB/year, billed monthly ($9/TB/month)	Annual renewal	Yes
/scratch	100GB/usr default quota*	2-week	Yes

*Increases available via request

Monitoring your Storage Quota

The parallel file system allows users to self-monitor their quota via the **pan_quota** command:

```
[user@clogin01 MyTest]$ pan_quota

  <GB> <soft> <hard> : <files>    <soft>    <hard> :     <path to volume> <pan_identity(name)>
  9.30  24.00  25.00 :  121845 unlimited unlimited : /home/user/MyTest uid:99999(user)
```

The first number listed above is the current amount of storage consumed by the user. This number is computed on the fly by the storage system and may change if storage is currently being written to the system. The third number, under <hard>, is the actual limit on the storage. Upon consuming this amount, the user will no longer be able to write to the file system. The second number (<soft>), is the amount at which the system will send the user email warnings.

NOTICE: **Data on the PanFS file systems on the cluster are not backed up.** *All users are encouraged to keep and manage their own backups of any vital data.* While a great deal of effort and expense have gone to provide a very redundant and robust cluster file system, accidents and system failures do sometimes occur. If you overwrite a file you may be able to recover it if discovered immediately via the system snapshots feature (see the section on **Snapshots** below). If you have further questions about how to best protect data used on the HPC Cluster, please contact the HPC Cluster management.

Data Archiving Options

Users have a number of options for archiving data from the cluster filesystem, and once a project or job run is completed it is highly recommended to migrate data off of the cluster file system. **Researchers are reminded to follow all University guidelines with regards to research data management, especially with respect to Federally-protected and sensitive data.**

Git

For keeping track of source code and small amounts of data, a code repository service such as **GitHub, GitLab** or **Bitbucket** is often a good idea. Regular "commits" to a git repository not only help document changes to a code base, but using offsite services (often for free) make copies of vital code where it can be available should the files become lost or unavailable for some unforeseen reason. The command line client for git is installed on the cluster and available for use. (Type "`man git`" at the command line for on-cluster documentation or https://git-scm.com/ for more general information.)

AWS S3 and similar cloud-based Storage

Researchers may also independently manage off-site or cloud-based storage such as AWS S3 "buckets". The AWS Command Line Interface (CLI) is available on the cluster as a cluster software module, which may be used to directly access AWS buckets from the HPC Cluster. This storage provides for some of the fastest uploading and downloading options to and from the cluster. Data stored in the cloud also provides the greatest flexibility in terms of multiple-location access and sharing. The downside of cloud storage is that it is not free, although users who access AWS through AWS@Emory (https://it.emory.edu/catalog/cloud-services/aws-at-emory.html) may receive discounts on egress charges.

Emory-based Isilon Storage

A University-supported archive location is the OIT-supported Isilon storage service (https://it.emory.edu/catalog/technical-infrastructure/storage-management.html). Access to space on the Isilon service can be negotiated with the RSPH IT Systems group for suitable long term archiving of data in a school-supported secure, HIPAA-compliant and redundant storage environment. Storage charges may apply, which can be billed to a University SpeedType. Please open a Help Ticket (email

“help@sph.emory.edu") for more information about the availability of this kind of storage.

Snapshots

The cluster file system currently generates and retains two snapshots of data in the /home file system in the hidden directory “`.snapshot`”. For instance:

```
[user@clogin01 ~]$ ls /home/.snapshot
2020.07.28.23.59.01.Daily_Home  2020.07.29.23.59.01.Daily_Home
```

Shows the two most recent snapshots (as of the date in this example) of the home directory file system. If a user accidentally deletes a file, they can restore one of two possible copies from the snapshots of their home directories in the above locations.

4 - Cluster Software

There are many available software packages on the cluster, not including those that you may wish to install locally in your home directory. Some of the larger, more popular software we have configured to use a system called **modules**. The modules system allows users to select both software packages and their particular versions in a way that automatically sets the path and other environment variables for that application.

Assume, for instance, that a user would like to use the R programming language:

```
testuser@node15 ~]$ R
bash: R: command not found…
```

By using the **`module spider`** command, we can see what packages are installed cluster-wide:

```
[testuser@node15 ~]$ module spider

----------------------------------------------------------------------------------------
The following is a list of the modules and extensions currently available:
----------------------------------------------------------------------------------------
  GEM: GEM/1.1, GEM/1.4.5, GEM/1.5.2
    GEM (Gene-Environment interaction analysis for Millions of samples) is a software
program for large-scale gene-environment
    interaction testing in samples from unrelated individuals.

  Go: Go/1.15.6
    Go is a statically typed, compiled programming language.

  MATLAB: MATLAB/R2020a
    MATLAB is a multi-paradigm numerical computing environment and proprietary
programming language developed by MathWorks.

  R: R/4.0.2, R/4.0.3, R/4.2.2
    R is a free software environment for statistical computing and graphics.

  SAS: SAS/9.4
    SAS is a statistical software suite for data management, advanced analytics,
multivariate analysis, business intelligence, criminal
    investigation and predictive analytics.


   aws-cli: aws-cli/2.1.29
    The AWS Command Line Interface (aws-cli) is the preferred method for accessing AWS
services from the HPC Cluster.

...
```

We can then use the **`module load`** command to load the R module, and then run R:

```
[testuser@node15 ~]$ module load R
[testuser@node15 ~]$ module list
```

```
Currently Loaded Modules:
  1) R/4.2.2


[testuser@node15 ~]$ R

R version 4.2.2 (2022-10-31) -- "Innocent and Trusting"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
…
```

In this example, we simply loaded the 'R' module and the latest version listed was loaded by default. If for instance we had wanted to use a different version of R listed, such as R/4.0.3, we could have specified the version, e.g., "**`module load R/4.0.3`**". Additional versions of software as well as additional modules may be added by request.

Additional Software

Users are permitted to install applications to be used for the cluster on their own, either in their home directories or in project spaces. Due to limits on home directory space (25GB) and the fact that many software packages can get quite large, the use of projects spaces is usually preferable. This is especially true if the application is to be shared amongst members of a lab or project.

Sometimes a programming project requires a particular configuration or version of software, and one would like to effectively "freeze" that version rather than potentially break functionality by having it subject to system-wide upgrades, for instance. One method of solving potential versioning issues is to create and use *virtual environments*. Virtual environments involve the setting of environment variables to create paths to separate software trees, the management of which is somewhat automatic. The most popular feature is that virtual environments permit the installation of software without the requirement of additional administrative privileges (since the packages are installed in the user's space and not at the system level).

Below are two recommended options for the creation and use of virtual environments.

Python Virtual Environments

Python virtual environments are standard methods of creating a separate installation tree of software for use in the Python programming language. In this way, one can install any python library or package to their python installation without concern about overwriting or colliding with a system-wide installation of Python, for instance. In addition, one can also create a text file summary of installed packages for recreation of the environment elsewhere, making the project more portable.

To begin the creation of the Python virtual environment, we first open an interactive session on a compute node. There we load the latest Python module so that our base environment is using the lastest version:

```
testuser@clogin01 ~]$ srun --pty bash
srun: job 14434696 queued and waiting for resources
srun: job 14434696 has been allocated resources
[testuser@node6 ~]$ python3 --version
Python 3.6.8
[testuser@node6 ~]$ module load python
[testuser@node6 ~]$ python3 --version
Python 3.11.3
```

At this point, we create the virtual environment, which we will call “VEtest” just for this example (you can call yours anything you want). We then activate the environment by sourcing the <env>/bin/activate file, which establishes the environment and therefore any changes madeto the Python installation will now apply to this environment:

```
[testuser@node6 ~]$ python3 -m venv VEtest
[testuser@node6 ~]$ . VEtest/bin/activate
(VEtest) [testuser@node6 ~]$ ls -l VEtest/
total 144
drwxr-xr-x 2 testuser hpcusers 4096 Nov 10 15:42 bin
drwxr-xr-x 3 testuser hpcusers 4096 Nov 10 15:41 include
drwxr-xr-x 3 testuser hpcusers 4096 Nov 10 15:41 lib
lrwxrwxrwx 1 testuser hpcusers    3 Nov 10 15:41 lib64 -> lib
-rw-r--r-- 1 testuser hpcusers  204 Nov 10 15:41 pyvenv.cfg
```

We can now install packages in our environment. We are first encouraged to update the Python package installer, pip, after which we install the Jupyter notebook environment as an example:

```
(VEtest) [testuser@node6 ~]$ pip install --upgrade pip
Requirement already satisfied: pip in ./VEtest/lib/python3.11/site-packages (22.3.1)
Collecting pip
  Downloading pip-23.3.1-py3-none-any.whl (2.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 30.8 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 22.3.1
    Uninstalling pip-22.3.1:
      Successfully uninstalled pip-22.3.1
Successfully installed pip-23.3.1
(VEtest) [testuser@node6 ~]$ pip3 install jupyter
Collecting jupyter
  Downloading jupyter-1.0.0-py2.py3-none-any.whl (2.7 kB)
Collecting notebook (from jupyter)
  Downloading notebook-7.0.6-py3-none-any.whl.metadata (10 kB)
Collecting qtconsole (from jupyter)
.....
```

```
Successfully installed anyio-4.0.0 argon2-cffi-23.1.0 argon2-cffi-bindings-21.2.0
arrow-1.3.0 asttokens-2.4.1 async-lru-2.0.4 attrs-23.1.0 babel-2.13.1
beautifulsoup4-4.12.2 bleach-6.1.0 certifi-2023.7.22 cffi-1.16.0 charset-
normalizer-3.3.2 comm-0.2.0 debugpy-1.8.0 decorator-5.1.1 defusedxml-0.7.1
executing-2.0.1 fastjsonschema-2.18.1 fqdn-1.5.1 idna-3.4 ipykernel-6.26.0
ipython-8.17.2 ipywidgets-8.1.1 isoduration-20.11.0 jedi-0.19.1 jinja2-3.1.2
json5-0.9.14 jsonpointer-2.4 jsonschema-4.19.2 jsonschema-specifications-2023.7.1
jupyter-1.0.0 jupyter-client-8.6.0 ….
```

Please refer to online information sources for more information about Python virtual environments.

Anaconda Virtual Environments

Anaconda (or its more frugal companion, “miniconda”) is similar in function to Python virtual environments. It is developed independently by a private company that provides the basic functionality of their software to the public for free. The big differences are that Anaconda comes with its own package management command, **`conda`** (replacing **`pip`**), and that very large software libraries for different programming aspects of Data Science are made available, including a distribution of the R programming language. Thus, in this way one can create one’s own virtual environment for R programming using Anaconda or Miniconda.

To get started with Anaconda, one has to first download the package from the anaconda website. We included an example of downloading the smaller miniconda package in the above section on **`wget.`**

To install the package in the local directory (preferably in /projects space, as all of the *condas will consume lots of space), use the command

```
bash miniconda.sh
```

And follow the online instructions. At the end of the installation, you will be asked to add some lines to your shell’s .rc files to automate the activating of the environment when you login in the future. This is ok, unless you’d rather activate the environment manually when you login. (To manually activate the environment, you can source the anaconda-related entries in your .rc files.)

To add R programming functionality to miniconda, use the following conda commands :

1. Create a conda environment for R and install the essentials:

```
conda create -n r_env r-essentials r-base
```

2. Activate the environment and list the packages. You may additionally install Jupyter Notebooks in this fashion:

```
conda activate r_env
conda list
```

```
conda install jupyter
```

3. Deactivate the environment when you are done.

```
conda deactivate
```

The use of Jupyter Notebooks on the cluster is covered in a section of the Guide below. Please refer to only resources for more information about using the Anaconda/ miniconda environment.

4 - SLURM and Submitting Jobs

In order to run a computation on the HPC cluster, you must submit the work as a “job”. This is in contrast to work done on a traditional computer like a laptop or an office workstation, where work is done primarily interactively and via a graphical user interface (GUI). Large jobs or numbers of computations are submitted in “batches”.

The job scheduler SLURM does the work of running all jobs submitted to the cluster. It is very similar in function to the Grid Engine scheduler seen on other clusters. SLURM is a fault-tolerant cluster management and job scheduling system that allocates resources to compute nodes so that they can run programs or jobs. It also manages contention for cluster resources by managing a queue of pending work.

A List of Basic SLURM Commands

sbatch command used to submit jobs (like 'qsub' in Grid Engine)

squeue used to display information about the run queue

scancel used to cancel or kill a job

scontrol used to show information about running or pending jobs

srun used to run an interactive instance

sinfo used to report the state of the cluster partition and nodes

Once logged into the system via ssh, you can do a quick check on the system via the **sinfo** command:

```
[testuser@clogin01 ~]$ sinfo
PARTITION       AVAIL  TIMELIMIT  NODES  STATE NODELIST
month-long-cpu     up 31-00:00:0     24   idle node[1-24]
week-long-cpu      up 7-00:00:00     24   idle node[1-24]
day-long-cpu       up 1-00:00:00     24   idle node[1-24]
short-cpu*         up      30:00     24   idle node[1-24]
interactive-cpu    up 2-00:00:00     24   idle node[1-24]
```

This display shows all of the partitions (i.e., “queues” in GE) on the system with the number nodes, their availability and status. The cluster isn’t being used at all at the moment in the above example. Here is a more typical **sinfo** output:

```
$ sinfo
PARTITION       AVAIL  TIMELIMIT  NODES  STATE NODELIST
month-long-cpu     up 31-00:00:0      1   comp node13
month-long-cpu     up 31-00:00:0     18    mix node[1-12,14-19]
month-long-cpu     up 31-00:00:0      5   idle node[20-24]
week-long-cpu      up 7-00:00:00      1   comp node13
week-long-cpu      up 7-00:00:00     18    mix node[1-12,14-19]
week-long-cpu      up 7-00:00:00      5   idle node[20-24]
day-long-cpu       up 1-00:00:00      1   comp node13
day-long-cpu       up 1-00:00:00     18    mix node[1-12,14-19]
day-long-cpu       up 1-00:00:00      5   idle node[20-24]
short-cpu*         up      30:00      1   comp node13
….
```

The command **sbatch** is used to submit jobs to the job scheduler. There are a number of parameters that can be passed to **sbatch** via the command line or separately in a shell script. Both methods can be used simultaneously, but any settings on the command line will prevail.

Here is a basic example of a python script that simply prints a line containing the hostname of the compute node on which it ran:

```
$ cat HelloWorld.py
#! /apps/bin/python3
import socket

hostname = socket.gethostname()

print("Hello World from " + hostname + "!\n")
```

If we submit this job without any additional arguments, we can see the results almost immediately in the directory from which we submitted it:

```
$ sbatch HelloWorld.py
Submitted batch job 3026
$ ls
HelloWorld.py  sleeper.sh  slurm-3026.out
$ cat slurm-3026.out
Hello World from node1!
```

When a job is submitted, **sbatch** returns a job ID, which is then associated with all aspects of the job. In this case, the output of the HelloWorld.py program was written to the output file “slurm-3026.out”, which is the default behavior. To change the default behavior, we can add arguments to the sbatch command:

```
$ sbatch --partition=short-cpu --output=hello-%j.out
HelloWorld.py
Submitted batch job 3027
$ ls
hello-3027.out  HelloWorld.py  sleeper.sh  slurm-3026.out
$ cat hello-3027.out
Hello World from node1!
```

Here we told the job scheduler to change the name of the output file to “hello-<jobid>.out” and designated the “short-cpu” queue as the place to run the very short program. It is a good idea to specify the appropriate queue for running jobs based on the anticipated time to complete the computation.

To check on the system details of a submitted job, we can use the **scontrol** command:

```
$ scontrol show job 3027
JobId=3027 JobName=HelloWorld.py
   UserId=testuser(3004) GroupId=hpcusers(3001) MCS_label=N/A
   Priority=4294901740 Nice=0 Account=(null) QOS=(null)
   JobState=COMPLETED Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:00 TimeLimit=00:30:00 TimeMin=N/A
   SubmitTime=2020-08-17T11:42:38 EligibleTime=2020-08-17T11:42:38
   AccrueTime=2020-08-17T11:42:38
   StartTime=2020-08-17T11:42:38 EndTime=2020-08-17T11:42:38 Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2020-08-17T11:42:38
   Partition=short-cpu AllocNode:Sid=clogin01:556227
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=node1
   BatchHost=node1
   NumNodes=1 NumCPUs=1 NumTasks=0 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=1,mem=192079M,node=1,billing=1
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:1 CoreSpec=*
   MinCPUsNode=1 MinMemoryNode=192079M MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=/home/testuser/HelloWorld.py
   WorkDir=/home/testuser
   StdErr=/home/testuser/hello-3027.out
   StdIn=/dev/null
   StdOut=/home/testuser/hello-3027.out
   Power=
   MailUser=(null) MailType=NONE
```

This output is sometimes useful in determining what may have happened to a job during its submission.

You can also use **scontrol** to display the configuration of a partition:

```
$ scontrol show partition day-long-cpu
```

```
PartitionName=day-long-cpu
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
   AllocNodes=ALL Default=NO QoS=N/A
   DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=1-00:00:00 MinNodes=0 LLN=NO
MaxCPUsPerNode=UNLIMITED
   Nodes=node[1-24]
   PriorityJobFactor=20000 PriorityTier=20000 RootOnly=NO ReqResv=NO
OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=768 TotalNodes=24 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
```

To cut down on typing and for more complicated parameters, one can set the arguments to **sbatch** in a wrapper or *job submission script*, which uses a combination of shell commands and SLURM directives, which begin with "#SBATCH". These can get very complex, but here is an example that covers some basic parameters:

```
#!/bin/bash

#SBATCH --job-name=normal.R
#SBATCH —partition=day-long-cpu

## This puts all output files in a separate directory.
#SBATCH --output=Out/normal.%A_%a.out
#SBATCH —error=Err/normal.%A_%a.err

## Submitting 100 instances of srun commands listed below
#SBATCH —array=0-100

## For notification purposes.  Use your Emory email address only!
#SBATCH —mail-user=<your_email_address>@emory.edu.
#SBATCH --mail-type=END,FAIL


module purge
module load R

srun /home/<user>/normal.R
```

This example shows how one might submit an R computation named "normal.R". This job submission script not only sets the values for several parameters such as job name, run partition, and output file names, but it allows for submission of an "array" of jobs, which is useful if one is running a large number of identical or similar computations. This script also demonstrates the use of modules in a script to ensure that your program has the appropriate path set.

Once we have our job submission script in place, we can use sbatch to submit the job (assuming we have named the above script "normal.sh"):

```
$ sbatch normal.sh
Submitted batch job 3028
```

And use the command **squeue** to monitor the progress:

```
$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
          3028_22 day-long- normal.R  testuse CG       0:01      1 node23
          3028_24 day-long- normal.R  testuse CG       0:01      1 node10
           3028_2 day-long- normal.R  testuse CG       0:02      1 node3
           3028_4 day-long- normal.R  testuse CG       0:02      1 node5
          3028_11 day-long- normal.R  testuse CG       0:02      1 node12
          3028_14 day-long- normal.R  testuse CG       0:02      1 node15
          3028_19 day-long- normal.R  testuse CG       0:02      1 node20
          3028_20 day-long- normal.R  testuse CG       0:02      1 node21
     3028_[36-100] day-long- normal.R  testuse PD       0:00      1
(Resources)
          3028_25 day-long- normal.R  testuse  R       0:01      1 node13
          3028_26 day-long- normal.R  testuse  R       0:01      1 node17
          3028_27 day-long- normal.R  testuse  R       0:01      1 node18
          3028_28 day-long- normal.R  testuse  R       0:01      1 node11
          3028_29 day-long- normal.R  testuse  R       0:01      1 node22
          3028_30 day-long- normal.R  testuse  R       0:01      1 node4
```

Here, under “ST” for job state, we see that some array jobs are running (R), others are completing (CG), and one is still pending (PD).

Finally, but perhaps most importantly, to cancel a job submission, one uses the **scancel** command:

```
$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              3231 short-cpu  sleeper testuser  R       0:44      1 node1
              3232 short-cpu  sleeper testuser  R       0:36      1 node1
              3233 short-cpu  sleeper testuser  R       0:33      1 node1
              3234 short-cpu  sleeper testuser  R       0:33      1 node1
[testuser@clogin01 ~]$ scancel 3232
[testuser@clogin01 ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              3231 short-cpu  sleeper testuser  R       0:52      1 node1
              3233 short-cpu  sleeper testuser  R       0:41      1 node1
              3234 short-cpu  sleeper testuser  R       0:41      1 node1
```

The **scancel** command takes a job id as its argument, and is pretty straight forward.

For more detailed options and explanations about SLURM commands, please consult the referenced documentation in “Additional Resources” below.

The “preemptable” Partition

The preemptable partition is a partition that aggregates underutilized CPUs on faculty-owned systems. The partition has access to more than 2200 associated CPUs and 70+ member nodes.

With the benefits of being able to access potentially large number of underused CPUs, there is a tradeoff, of course: jobs submitted to the preempt able partition are scheduled at a lower priority and may be "preempted" by jobs run by owners of the system when they need to run their group’s work. When preemption occurs, your job is killed on the assigned node and requeued on the job scheduler. The job(s) will therefore start again, perhaps on a different available node. Your program should be able to account for the possibility that it could be killed and restarted/etc.

There is currently a three day time limit on jobs submitted to the preemptable partition.

Interactive Terminal Sessions (via `srun`)

Sometimes one would like to test code before submitting jobs widely to the cluster. In this case and interactive terminal sessions may be useful. To run an application on the cluster in an interactive session you can use **srun** on the **interactive-cpu queue**, which will start a terminal session for you on a compute node. Here is a simple example of running R in an interactive session:

```
[user@clogin01 ~]$ module load R/4.0.2
[user@clogin01 ~]$ srun -p interactive-cpu --pty bash
[user@node1 ~]$ R

R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale
```

```
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> quit()
Save workspace image? [y/n/c]: n
```

When you have finished with your session, simply exit the shell from the execution node:

```
[user@node1 ~]$ exit
exit
[user@clogin01 ~]$
```

Interactive MATLAB and SAS

Both MATLAB and SAS are programs that are commonly used on personal computers via their Graphical User Interface, or "GUI". This creates a challenge on the HPC cluster, as access to the compute nodes is presented via the Linux Command Line interface (CLI) or text-based interface.

Fortunately, both programs have methods for suppressing the GUI and allowing for command line interactive use. For SAS, the method is to use the **`-nodms`** and **`-nonews`** flags when calling the program from the command line:

```
testuser@clogin01 ~]$ srun -p interactive-cpu --pty bash
srun: job 14567300 queued and waiting for resources
srun: job 14567300 has been allocated resources
[testuser@node1 ~]$ module load SAS
[testuser@node1 ~]$ sas -nodms -nonews
NOTE: Copyright (c) 2016 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software 9.4 (TS1M6)
      Licensed to EMORY UNIVERSITY - T&R - SFA - AMAZON CLOUD, Site 70127129.
NOTE: This session is executing on the Linux 4.18.0-477.27.1.el8_8.x86_64
      (LIN X64) platform.



NOTE: Analytical products:

      SAS/STAT 15.1
      SAS/ETS 15.1
      SAS/OR 15.1
      SAS/IML 15.1
      SAS/QC 15.1
```

```
NOTE: SAS initialization used:
      real time           1.24 seconds
      cpu time            0.03 seconds

  1?
```

Similarly, MATLAB has command line options, **-nosplash** and **-nodisplay**, that allow for text-based interactive sessions:

```
[testuser@clogin01 ~]$ srun -p interactive-cpu --pty bash
srun: job 14567308 queued and waiting for resources
srun: job 14567308 has been allocated resources
[testuser@node1 ~]$ module load MATLAB
[testuser@node1 ~]$ matlab -nosplash -nodisplay

                                    < M A T L A B (R) >
                          Copyright 1984-2020 The MathWorks, Inc.
                       R2020a Update 5 (9.8.0.1451342) 64-bit (glnxa64)
                                      August 6, 2020


To get started, type doc.
For product information, visit www.mathworks.com.

>>
```

Jupyter Notebooks

While the main business of the HPC cluster is large-scale serial-batch processing, it is sometimes useful to be able to have a functional visual environment for doing code development while using the interactive session on a compute node of the cluster. The issue then becomes how to access a visual editing environment for a service whose interface is the command line and that also requires running computation on a designated compute node on a particular partition.

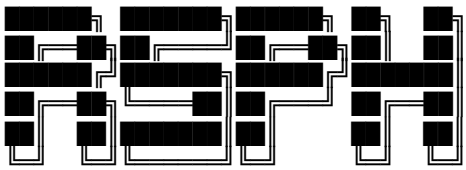
In our HPC environment, we have chosen to support the use of Jupyter Notebooks for this purpose. Jupyter Notebooks are an open source tool that allows one to use a language kernel (such as Python, R, Julia and SAS) in an interactive web-browsing environment that allows for both code editing and graphics. Notebooks can also be saved and shared, which has led to them becoming a standard for sharing computational results in a reproducible way.

The key to using Jupyter Notebooks on the HPC cluster is the use of some command line wizardry called “SSH tunneling”. Through ssh tunneling, we can relay the traffic between the end user and a compute node without concern for creating a computational load on the the intermediary login node.

This process involves several steps:

1. Create an ssh tunnel from a port from your local system to the HPC cluster to relay web-based Jupyter Notebook traffic. You want to be careful to use an unused networking port. (Just for consistency, use the same port number throughout each of the steps.)

```
mymachine$ ssh -L 8999:localhost:8999 cluster

Welcome to the
```



```
High Performance Computing (HPC) Cluster

***  AUTHORIZED USE ONLY ***


-->>>  DO NOT RUN APPLICATIONS ON THE LOGIN NODE <<<---
       Please submit ALL computations, including small
       interactive ones, to compute nodes.


Last login: Wed Nov  5 18:31:17 2023 from 10.110.122.27
```

```
[testuser@clogin01 ~]$
```

2. Reserve resources on a compute node for your computation and Jupyter kernel using salloc. In this case, we are using the general use interactive-cpu partition, and arbitrarily requesting 1 CPU core and 1 GB of memory as an example.

```
[testuser@clogin01 ~]$ salloc -p interactive-cpu -n1 --mem=1000
salloc: Pending job allocation 14309753
salloc: job 14309753 queued and waiting for resources
salloc: job 14309753 has been allocated resources
salloc: Granted job allocation 14309753
```

3. Create a tunnel from the login node to the node reserved in Step 2 (which will then be "tunneled" back to your computer using the other tunnel created in Step 1.) Unlike the previous ssh tunnel command, this tunnel will run in the background and return the access back to the shell on clogin01. Also note that the host that this tunnel will connect to is determined by the outcome of the above scheduling request, the name of which is stored temporarily in the shell variable **SLURM_NODELIST**:

```
[testuser@clogin01 ~]$ ssh -N -L 8999:localhost:8999 "$SLURM_NODELIST" &
[1] 3779883
testuser@clogin01 ~]$
```

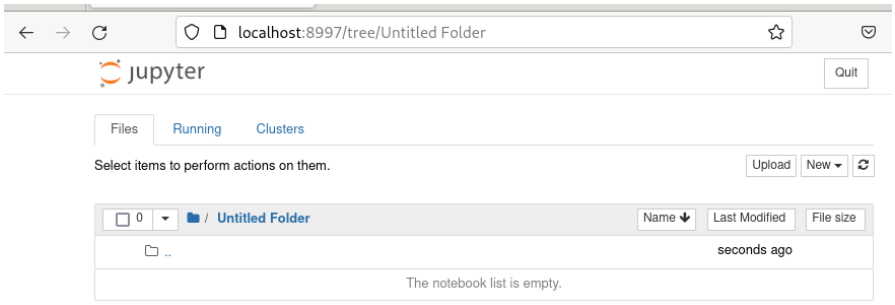
4. Access the interactive-cpu SLURM reservation by executing the **srun** command. This will open a shell on the compute node through the second tunnel, where you can access your virtual environment for your project and run the Jupyter Notebook.

```
testuser@clogin01 ~]$ srun --pty bash; kill %1
```

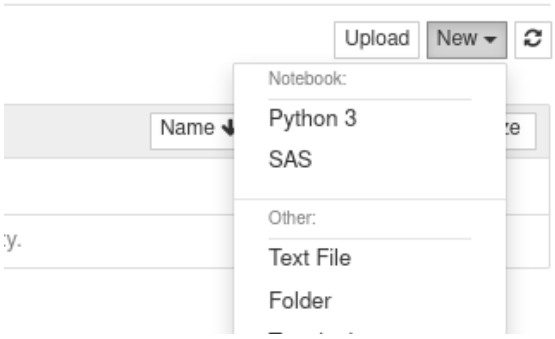
5. Execute the Jupyter notebook command at the command line. In this example, we are using a Python virtual environment created beforehand that has the Jupyter Notebook package installed (an Anaconda environment can similarly have the Jupyter package installed).

```
[testuser@node24 ~]$ . ./jupyterenv/bin/activate

(jupyterenv) [testuser@node24 ~]$ jupyter notebook --NotebookApp.token='' --no-browser
--port=8999

[I 19:27:44.830 NotebookApp] Serving notebooks from local directory: /home/testuser
[I 19:27:44.830 NotebookApp] Jupyter Notebook 6.3.0 is running at:
[I 19:27:44.830 NotebookApp] http://localhost:8999/
[I 19:27:44.830 NotebookApp] Use Control-C to stop this server and shut down all
kernels (twice to skip confirmation).
```

6. Then, in a web browser on your local system, open the provided URL ("http://localhost:8999" in this case.



Opening the Jupiter Notebook on the local computer



Selecting the Python3 kernel under the "New" button

7. Click on the "New" button in the upper right hand corner to start your kernel. In this case, Python and SAS kernels are available in this virtual environment. (Anaconda and Miniconda environments can be installed with R kernels.)

8. When you are finished, exit the Jupyter application in the browser, and then exit from the compute node. Then exit finally from the **`salloc`** command to end the job and return resources back to the scheduler.

```
I 22:34:10.673 NotebookApp] Kernel shutdown: 207f85ae-0b35-4ffd-b482-999d36c854f3
[I 22:34:13.754 NotebookApp] Shutting down on /api/shutdown request.
[I 22:34:13.755 NotebookApp] Shutting down 0 kernels
[I 22:34:13.755 NotebookApp] Shutting down 0 terminals
(jupyterenv) [testuser@node24 ~]$
(jupyterenv) [testuser@node24 ~]$ exit
exit
[testuser@clogin01 ~]$ jobs
[1]+  Done                    ssh -N -L 8997:localhost:8997 "$SLURM_NODELIST"
[testuser@clogin01 ~]$ exit
exit
salloc: Relinquishing job allocation 14309753
[testuser@clogin01 ~]$
```

At this point, you are done.

NB: Occasionally when attempting to use an ssh tunnel, one finds the chosen port is actually occupied by another process:

```
[testuesr@clogin01 ~]$ ssh -N -L 8999:localhost:8999 "$SLURM_NODELIST" &
[1] 4001239
[testuesr@clogin01 ~]$ bind [127.0.0.1]:8999: Address already in use
channel_setup_fwd_listener_tcpip: cannot listen to port: 8999
Could not request local forwarding.
```

If this happens, the best course of action is to just try another port number, like 8998. You may also find it simpler to start the process over and just use the new port number throughout the tunnel chain. On unix-based systems, like Linux and Mac OS, you can check the file /etc/services to see if a port is reserved on that system for a particular application.

```
user@laptop ~ % grep 8999  /etc/services
bctp            8999/udp     # Brodos Crypto Trade Protocol
bctp            8999/tcp     # Brodos Crypto Trade Protocol
user@laptop ~ % grep 8998  /etc/services
#               8955-8998   Unassigned
#               8990-8998   Unassigned
```

Not all services listed in the services file are always active on the system, of course.

Getting Help

To receive cluster assistance or request an account, please email help@sph.emory.edu with the phrase “RSPH HPC cluster: “ preceding the actual issue in the subject line. This will help the Help Desk route the request to the correct HPC cluster team (there are several different clusters on campus).

Additional Resources

While instructions on how to use the Linux operating system are outside the scope of this document, we can suggest the following helpful Linux tutorials and other guides:

The Linux Command Line, a popular free online book:
https://www.linuxcommand.org/tlcl.php

Learning the Linux Command Line, a video training module accessible via LinkedIn Learning, is particularly good and provides guidance in getting ssh working under Windows. While Emory University discontinued its sponsored access to LinkedIn Learning, many people may access the service via their local library. (Residents of Dekalb County, for instance, may access the service via dekalblibrary.org. An active library account is required.)

The Emory VPN website: http://it.emory.edu/vpntools/access.html

SLURM manual: https://slurm.schedmd.com/quickstart.html

Anaconda: https://docs.anaconda.com/free/

Python: https://docs.python.org/3/tutorial/index.html