Connecting

To connect to the DCC simply enter the following into a terminal with SSH installed:

```
username@dcc-login.oit.duke.edu
```

Replace `username` with your DukeID. This connects the current terminal to a login node.

DO NOT RUN SIMULATIONS ON THE LOGIN NODE.

The login node is shared by all users and intended for minor tasks only. To run intensive tasks, set up a job script as described in the user manual.

Important directories

A file system overview can be found here.

Your home directory has limited storage. There is more storage in the labs folder, `/hpc/group/dolbowlab/`. Make a directory there with the name of you DukeID. Use this for storage.

Additionally, make a folder in the `/work/` directory with the same name. This is a high speed file system with limited storage. When running simulations, write your outputs here, and then transfer them to the lab directory. The `/work/` file system is purged every 75 days.

The dolbowlab Partition

Our lab owns a node on the DCC that is only accessible by lab members. It only has 78 cores so it can easily get clogged up if multiple lab members are using it. The reset of the DCC is still accessible if the lab node is full.

Installing RACCOON on the DCC

Normally on an HPC, you would use the https://mooseframework.inl.gov/getting_started/installation/hpc_install_moose.html to install MOOSE on an HPC. Due to some missing packages, RACCOON needs to be installed using the typical conda installation.

Full DCC Job Scheduling and SLURM User Guide

This document describes the process for submitting and running jobs to the DCC under the Slurm Workload Manager.

What is a cluster, and why do I need SLURM?

Our HPC cluster is made up of a number of compute nodes, each with a varyling level of scratch space, processors, memory, and sometimes GPUs. Users submit jobs to **SLURM** that specify the instructions (code) and software they want to run along with a description of the computing resources needed. SLURM, a combined scheduler and resource manager, then runs those jobs on the DCC.

Cluster Basics and Terminology

account: Your DCC group is your SLURM account. A SLURM account grants users rights and manages limits in relation to SLURM partitions. You can see your DCC group(s) in Research Toolkits.

batch: Instructions run in a file without user interaction, typically referred to as 'run in batch' or 'batch processing'. This is the most powerful way to use the DCC.

batch script: A text file containing a series of resource specifications, such as the number of nodes, amount of memory, and other requirements needed to run, and commands to be run. This script is submitted to the Slurm workload manager, which schedules and executes the job on available compute nodes within a high-performance computing (HPC) cluster.

core: A processing unit within a computer chip (CPU). The CPU in a node performs computations.

GPU: A graphics processing unit (GPU) is a specialized processor which can generate computer graphics, but in HPC, designed to accelerate computation-intensive tasks by performing parallel operations, making it ideal for tasks such as simulations, deep learning, and scientific computations.

interactive session or job: SLURM allows users to request a real time interactive session on a compute node through OnDemand or via SSH using `srun` so you can use a computing node real-time to code, debug, and/or watch your request run in real-time.

job: A task or set of tasks, submitted to the cluster scheduler to be executed.

node: A physical machine in a cluster, including login, compute, and transfer nodes

Type	Description
Login nodes	The login nodes are a place where users can login, edit files, view job results and submit new jobs. Login nodes are a shared resource and should not be used to run application workloads. There are limits on the login nodes.
Data Transfer node / Globus	The data transfer nodes are available for moving data to be accessible to and from the cluster. On the DCC, is done primarily via Globus.

Type	Description
Compute node	The compute nodes are the computers where jobs are run. To run jobs on the compute nodes, users need to access a head node and schedule their program to be run on the compute nodes once the requested resources are available.

partition: A partition, also called a job queue, is a subset of compute nodes on the cluster. Each partition is configured with with a set of limits (includind allowed users) that specify the requirements for jobs run in that partition. The DCC has as set of partitions for all users as well as thoughs for specific use cases or lab groups.

SLURM: Is the open source, fault tolerant, scalable cluster management and job scheduling system we run on the Duke Compute Cluster.

- When: It finds and allocates the computing resources that fulfill the job's request at the soonest available time. The time to start the job will be faster if the batch script is written well for what the job will need (i.e. not asking for just the right amount of resources).
- How: When a job is scheduled to run, the scheduler instructs the resource manager to launch the application(s) across the job's allocated resources. This is also known as "running the job".
- Results: When a job has completed, you can check the job logs for informational updates or errors and the resources used via sacct or sreport. For long term usage, see our usage reports.

Logging into the DCC

As a shared computing resource, all work done on the DCC must be done by submitting jobs to the scheduler. Login nodes must not be used to execute computing tasks.

Acceptable use of login nodes include:

- lightweight file transfers,
- script and configuration file editing,
- job submission and monitoring,
- BaseSpace BaseMount - users can use basemount to mount BaseSpace into their home directories.

To minimize disruption and ensure a comfortable working environment for users, resource limits are enforced on login nodes, and processes started there will automatically be terminated if their resource usage (including CPU time, memory and run time) exceed those limits.

Web Access

Users who are not comfortable using the command line can access the DCC through Open OnDemand, which provides a access to interactive sessions on the DCC through your browser. The Open OnDemand service. provides acesss to the DCC through RStudio, Jupyter, or Linux desktop sessions.

SSH Access

Login to the DCC using `ssh netid@dcc-login.oit.duke.edu`, (note: VPN is not required, but MFA **is required**).

```
kk338@CDSS-5630 ~ % ssh kk338@dcc-login.oit.duke.edu
Password:
Duo two-factor login for kk338

Enter a passcode or select one of the following options:

1. Duo Push to XXX-XXX-4007
2. Phone call to XXX-XXX-4007
3. Phone call to XXX-XXX-9784
4. SMS passcodes to XXX-XXX-4007 (next code starts with: 2)

Passcode or option (1-4): 1
Success. Logging you in...
Last login: Tue Dec 21 11:07:41 2021 from xxx
################################################################################
# MOTD                                                                         #
# My patch window is wednesday 03:00                                           #
################################################################################
kk338@dcc-login-03  ~ $
```

A word on ssh clients

Linux and macOS systems generally come with a SSH client already installed and you can `ssh` directly from a terminal window. For Windows users you will need an ssh client such as PuTTY or MobaXterm. Advanced users may consider installing WSL2 (Windows Subsystem for Linux 2), see installation instructions here, but beware DCC support staff do not

offer support for your local computer and software installations. Other ssh clients may work or not, but beware of clients that create persistent sessions to the DCC.

Users may connect to the DCC login nodes using VSCode to troubleshoot code and manage scripts. However, **VSCode SHOULD NOT BE USED** for computationally intense tasks on login nodes or on compute nodes via common or scavenger partitions. If you would like to execute code using VSCode, connect to OnDemand and run your code interactively using the CodeServer option.

SSH Keys

Setting up ssh keys from your workstation will greatly simplify the login and file transfer process by creating a secure key based session between your workstation and the DCC instead of using your password with MFA.

Step 1. To generate a key pair run the following (on your local machine) from your command shell prompt:

```
ssh-keygen -t ed25519
```

You should get prompted for what file to save the key in. The default should be fine (just hit enter).

Next you should be prompted for a passphrase. Pick something secure that you can remember.

The public key generated will be stored in the file:

```
~/.ssh/id_ed25519.pub
```

Step 2. View and copy your ssh key:

```
cat ~/.ssh/id_ed25519.pub
```

Please be sure that you do not accidentally leave off the .pub. The file ~/.ssh/id_ed25519 (without the .pub) is your private key file, which you should NEVER share with ANYONE.

Note that the contents of this file (what is displayed by cat) should look something like this:

ssh-ed25519 AAUAC3NzaC1lZDI1NTE5AAAAIDhkOqqB+kRDIDp86EGo0hNvogOxwJmM5kxx/aYCzOrG kk338@CDSS-CPP0H34NX2

Step 3. Add your public key to your Duke profile by copying to your “SSH Public keys” under “Advanced User Options” at: idms-web.oit.duke.edu/portal.

Find the section titled "Manage Your Public SSH Keys." In that section there should be clickable text that says "+ See More about SSH keys". Click that to expand this section, and you should see a textbox for "New Public Key" where you can paste your Public Key (copied in the previous step).

Step 4. Test your connection. Note `Enter passphrase for key` should display instead of `Password`

```
kk338@CDSS-5630 ~ % ssh kk338@dcc-login.oit.duke.edu
Enter passphrase for key '/Users/kk338/.ssh/id_ed25519':
 Last login: Tue Dec 21 11:13:41 2021 from xxx
###############################################################################
# MOTD                                                                        #
# My patch window is wednesday 03:00                                          #
###############################################################################
kk338@dcc-login-03  ~ $
```

Key Management

An "authentication agent" for SSH keys is typically referred to as "ssh-agent" which is a built-in tool on most operating systems that stores your SSH private keys in memory, allowing you to authenticate without re-entering your passphrase

every time you connect to a server; while a "password manager" like 1Password helps with management of passwords as well as SSH keys. **It will likely makes sense for you use either a password manager or ssh-agent, but not both**:

1Password (password manager)

- To get started:1Password
- Key Management: 1Password Manage Keys

ssh-agent (Linux)

Most modern Linux desktop environments will automatically prompt for and manage your ssh private key passphrase, if not,run the following: `ssh-add ~/.ssh/id_ed25519`

ssh-agent (Mac)

macOS 12.0 or newer, run:

```
ssh-add --apple-use-keychain ~/.ssh/id_ed25519
```

macOS older than 12.0, run:

```
ssh-add -K ~/.ssh/id_ed25519
```

(and then type the password you just used) which will add add the key to the authentication agent, and also save the password in your keychain.

Then edit `~/.ssh/config` and add these lines to it:

```
Host *
UseKeychain yes
AddKeysToAgent yes
IdentityFile ~/.ssh/id_ed25519
```

ssh-agent (Windows):

- Documentation Windows ssh-agent

Running the following works for Windows, as well: `ssh-add ~/.ssh/id_ed25519`

View more general information about ssh public key authentication at ssh.com.

VSCode

Users may connect to the DCC login nodes using VSCode to troubleshoot code and manage scripts. However, **VSCode SHOULD NOT BE USED** for computationally intense tasks on login nodes or on compute nodes via common or scavenger partitions. If you would like to execute code using VSCode, connect to OnDemand and run your code interactively using the CodeServer option or follow the below instructions for connecting to an interactive session using VSCode.

Before starting this process, make sure you have generated SSH keys for the DCC. See the SSH Keys section above. Using MFA via Duo Mobile will not work.

Step 1: SSH Config

In order to access the compute nodes, you'll first need to create an SSH config file. Navigate to where you store your SSH keys (by default on macOS, it's ~/.ssh) and create a text file named 'config'.

```
(base) rm145@CDSS-LGQFWT4QR4 ~ % cd ~
(base) rm145@CDSS-LGQFWT4QR4 ~ % cd ~/.ssh
(base) rm145@CDSS-LGQFWT4QR4 .ssh % touch config
```

Within this file, you'll want to add the following:

```
# SSH for DCC login node
Host dcc-login
  IdentityFile /path/to/privateKey
  HostName dcc-login.oit.duke.edu
  User netID

# SSH for DCC compute node
Host dcc-compute
  IdentityFile /path/to/privateKey
  HostName dcc-core-[x]
  User netID
  ProxyJump dcc-login
```

You'll need to replace the following with your information:

- IdentityFile path
- User netID
- HostName core number for the compute node; you won't know this until you create the interactive application job

Save the file.

Step 2: Create Interactive Job

You can do this using Open OnDemand. Once created, check the coompute node that you're connected to; this will be the compute node that you need to use in your SSH config file. Edit your config file under `HostName` for the dcc-compute.

Step 3: Connect with VSCode Now, we'll use VSCode to connect directly to the compute node reserved via the interactive job. You'll only have access to this compute node for the walltime you requested.

- Start VSCode and (macOS) hit command+shift+p, then select 'Remote-SSH: Connect to Host...'
- Select `dcc-login`
- You'll be prompted for the SSH key's passphrase; enter that and you should be connected to the DCC login nodes
- Go to 'File > New Window', then hit command+shift+p again and select 'Remote-SSH: Connect to Host...'
- This time, select `dcc-compute`
- You should be prompted to enter the SSH key passphrase again; do so, and you should be connected to the compute node

Overview

The DCC has several shared file systems available for all users of the cluster. General partitions are on Isilon, 40Gbps or 10Gbps network attached storage arrays.

Sensitive data is not permitted on cluster storage.

Path	Size	Description	Backups
`/work/<netid>`	650 TB	Unpartitioned, high speed volume, shared across all users. Files older than 75 days are purged automatically.	None
`/cwork/<netid>`	830 TB	high speed volume, availabe to all users. Files older than 90 days are purged automatically.	None
`/hpc/home/<netid>`	25 GB	Use for personal scripts, and other environment setup. When a users Duke ID is deactivated, their access and home directory is automatically removed from the cluster.	None
`/hpc/group/<groupname>`	1 TB, Expandable for a fee in Research Toolkits	Private to each lab group, persistent working space for applications and projects that last longer than 75 days.	7 day snapshot
`/hpc/dctrl/<netid>`	500 GB	Private to each PhD student, persistent working space for applications and projects.	7 day snapshot
`/datacommons/<groupname>`	Fee-based	Archival storage that can be mounted to the DCC to ease transfer of data to computational space and results out to long term storage. Because I/O will not be as performant as with cluster storage, job file access should not be configured that will cause excessive read/write to Data Commons storage.	Optional 30 day backup

OIT storage rates are set annually. Group storage is "Standard" and Data Commons is "Archive".

How should I use DCC storage?

To optimize the performance of the cluster and make your utilization efficient, we recommend the following:

- **/home** -> personal scripts and configuration files, environment setup information
- **/group** -> software installations, lab specific scripts, moderately sized data sets or intermediate results that are needed for longer than 75 days.
- **/work** -> large data sets under analysis, intermediate results. In the root folder, create your own folder for your use with: `mkdir <netid>` ***Remember: Files older than 75 days are automatically purged!***
- **/cwork/** -> experimental version of /work for large data sets under analysis, intermediate data, and preliminary results.

Remember: Files older than 90 days are automatically purged!

- **/datacommons** -> long term storage for source data and results data

DCC compute nodes also have a /scratch volume that is local to the compute node. This can be used when highly performant storage is needed during a job, but data should be deleted at the completion of the job. Sizes vary, so use of this requires strong working knowledge of the nodes on the DCC.

Viewing Usage

Viewing usage in...

.../home

View your current utilization with: `du -hd1 /hpc/home/<netid>`

```
$ du -hd1 /hpc/home/kk338
1.5M    /hpc/home/kk338/.config
32K /hpc/home/kk338/Desktop
175K    /hpc/home/kk338/.vnc
138K    /hpc/home/kk338/.dbus
32K /hpc/home/kk338/.singularity
173K    /hpc/home/kk338/.ipython
56K /hpc/home/kk338/.java
80K /hpc/home/kk338/.ssh
80K /hpc/home/kk338/bin
6.0K    /hpc/home/kk338/R
104K    /hpc/home/kk338/danai
698K    /hpc/home/kk338/.cache
11M /hpc/home/kk338/ondemand
3.4M    /hpc/home/kk338/.local
195K    /hpc/home/kk338/.jupyter
22M /hpc/home/kk338/.comsol
28K /hpc/home/kk338/.conda
160K    /hpc/home/kk338/.gnupg
200K    /hpc/home/kk338/tutorial
383K    /hpc/home/kk338/.matlab
243M    /hpc/home/kk338
```

.../group

View your current group volume size and amount used with: `df -h /hpc/group/<groupname>`

```
$ df -h /hpc/group/rescomp
Filesystem                                Size  Used Avail Use% Mounted on
oit-nas-fe13.dscr.duke.local:/hpc-rescomp  1.0T  390G  635G  39% /hpc/group/rescomp
```

.../work

View your current usage of /work with: `quota -s -f /work`

```
$ quota -s -f /work
Disk quotas for user rm145 (uid 1032945):
     Filesystem   space   quota   limit   grace   files   quota   limit   grace
vast.oit-vast-01.oit.duke.edu:/vwork
                  1024M      0K  16384G               3       0       0
```

.../datacommons

View your current volume size and amount used with: `df -h /datacommons/<groupname>`

```
$ df -h /datacommons/plusds
```

```
Filesystem                                              Size  Used Avail Use% Mounted on
oit-nas-fe13dc.dscr.duke.local:/ifs/datacommons/plusds   12T   12T  716G  95% /datacommons/plusds
```

Permissions

Directory and file permissions

Standard Unix file permissions are supported on the DCC, ACLs are not supported.

By default when users create a new directory the default permissions are "755" (non-group-writable). To automatically create new directories or files that are group writable, run the command:

```
umask 002;echo "umask 002" >> ~/.bashrc
```

This will change the default permissions to "775" (group-writable).

File Transfers

Globus is the recommended method to transfer data to and from the DCC and can be used with any OIT provided research storage.

- New users: Globus Web User Guide.
- Advanced users: Globus CLI User Guide

Globus Quick Links

- Duke Compute Cluster (DCC) Data Transfer Node – This endpoint can support any OIT provided storage.
- Duke Box Storage – Allows researchers to easily move data to and from Duke Box.
- HARDAC to DCC Instructions

File transfer considerations and other methods

Bundle your files before transferring

.tar your files before transferring. Transferring a large number of small files is very inefficient and will take much longer than transferring a single large file.

There are many ways to transfer files to and from the DCC. When choosing consider your network path and try to optimize transfers through directly connected (wired) systems.

For SCP and RSYNC the initiating system must be connected to the Duke network, or off campus use VPN. When executing SCP for files to or from the DCC, MFA is required, and will default to your first available option. If you are having trouble with MFA and SCP, get help with MFA, or bypass MFA all together by setting up and using SSH Keys to access the DCC.

File transfers within the DCC

SCP (Secure Copy) a single file

Command Format

```
scp source destination
```

Push

```
$scp <localpath.file> <netid>@dcc-login.oit.duke.edu:<dccpath>
```

Pull

```
$scp <netid>@dcc-login.oit.duke.edu:<dccpath.filename> <localpath>
```

Sample Execution

```
# Note use of ssh keys to simplfy login
# Sample command and output pushing a file from my workstation to my group directory:

kk338@CDSS-5630 ~ % scp jobs.txt kk338@dcc-login.oit.duke.edu:/hpc/group/rescomp/kk338
Enter passphrase for key '/Users/kk338/.ssh/id_rsa':
jobs.txt                                      100% 3847KB   8.4MB/s   00:00

# Sample command and output pulling a file from my group directory to my local workstation (note the use of
`.` to denote current working directory):

kk338@CDSS-5630 ~ % scp kk338@dcc-login.oit.duke.edu:/hpc/group/rescomp/kk338/DailyUsage.xlsx .
Enter passphrase for key '/Users/kk338/.ssh/id_rsa':
DailyUsage.xlsx                                              100%   40KB 361.1KB/s   00:00
```

While you can use `scp -r` to recursively copy all of the files in a directory, we recommend the use of `rsync` for a large number of files.

RSYNC for a larghe number of files

Command Format

```
# -r syncs recursively
# -P updates the modify time, useful for /work purge script analysis
rsync –rP source destination
```

Push

```
$rsync –rP dir1/ netid@dcc-login-02.oit.duke.edu:.
```

Pull

```
$rsync –rP netid@dcc-login.oit.duke.edu:~/dir1 .
```

Best Practices for Managing Files

When using cluster storage and running parallel workloads, it is important to optimize your work to take advantage of available storage to ensure the speed and efficiency of your work as well as to avoid stressing the shared filesystem and negatively impacting performance for all users.

Storage Tiers explained

The DCC uses a hierachical organanization of storage media (called storage tiers) that differ in performance, cost, and capacity. Each option on the DCC has different intended use, performance, capabilities, and cost implications. See the full list of options here.

Using /scratch

Every DCC node has a local /scratch directory on NVMe, which is the highest throughput and fastest response time storage available on the DCC. Not only will this speed up your workloads, but it will also take pressure off the DCC shared filesystems. But beware: scratch space is limited, isolated per node, and is not backed up, so you should only use it for temporary file storage.

Examples for use: - When working with many small files, consider copying them to /scratch at the start of a run and reading them from there. - For STAR (mapping reads to genome indexes using STAR) mkdir -p /scratch/projectname/ --outTmpDir /projectname/${sample} --

Storing Files

- Avoid storing many files in a single directory – hundreds of files are ok, thousands are not. Our filesystem manages file locks (allows multiple nodes to access the same data concurrently safely) at the subdirectory level. Plan your data structure to use multiple sub-directories by default to minimize contention and locking events.
- Avoid operating on many small files – the performance may depend on how you handle the data on disk. Create a smaller number of larger files.

Be aware of I/O load

- when running multiple instance of the same job consider reading in all of the data at the beginning of the workflow, perform the entire analysis, then write the results at the end.
- Avoid opening/closing or creating/deleting files repeatedly in tight loops. Creating and deleting files requires locking certain filesystem structures, and these “synchronization points” are computationally expensive to enforce. Try to create files “as needed” through your workflow, and minimize the number created within a single directory. Similarly, delete files “as needed” (which is less “expensive” for directories that are “properly sized” in terms of number of files). Try not to re-use filenames in the same directory, when creating and deleting files in a loop. If possible, open files once at the beginning of your workflow/program, then close them at the end.

Watch your quotas

You are limited in capacity and exceeding your storage quotas will impact performance.

Choose the right command(s) for the jobs

- Avoid using ls -l in scripts – consider ‘find’ which is more appropriate for scripts, if you use ‘ls’ use it by itself or use ls -l filename if you want the long listing of a specific file.

A The NYU HPC center has published some suggestions you can try for your projects.

About

The DCC support most commonly used open-source scientific software and tools for cluster and high performance computing.

Widely used software are installed as modules DCC adminstrators. Applications can be installed on request, email rescomputing@duke.edu. Generally, applications should be open source and recommended for use on a computing cluster.

Users may also install software in their group directories.

Modules

Software is provided on the DCC under the form of loadable environment modules. The use of a module system means that most software is not accessible by default and has to be loaded using the module command. This mechanism allows us to provide multiple versions of the same software concurrently, and gives users the possibility to easily switch between software versions. The modules system helps setting up the user's shell environment to give access to applications, and make running and compiling software easier.

When you first login, you'll be presented with a default, bare bones environment with minimal software available. The module system is used to manage the user environment and to activate software packages on demand. In order to use software installed, you must first load the corresponding software module.

When you load a module, the system will set or modify your user environment variables to enable access to the software package provided by that module. For instance, the `$PATH` environment variable might be updated so that appropriate executables for that package can be used.

Run `module avail` to see the list of installed applications

Run `module load (module name)` to use the application

You can add the `module load` command to your job scripts, or to the end of the .bash_profile file in your home directory.

User installed software

Lab groups are welcome to install software in their `hpc/group/<groupname>` space if `sudo` access is not required. This can be helpful for lab specific software or for when a group wants to strongly control software versions and packages.

Miniconda installation sample

Login to the DCC using SSH:

```
mkdir -p /hpc/group/<groupname>/<netid>
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
sh Miniconda3-latest-Linux-x86_64.sh
```

and then follow the instructions. The place to give as Miniconda install location should be `/hpc/group/<groupname>/<netid>/miniconda3`. It will offer to update your ~/.bashrc, (init) which is what you want. Log out log back in and then you can run conda install, pip install, create environments, etc.

Python

Users who will be using Python through SSH and OnDemand should setup virtual environments using miniconda with the desired Python version and packages. After installing Minconda, create a virtual environment. In the example below, `test` is the name of the environment and `python=3.9` is the python version number.

```
conda create -n test python=3.9
conda activate test
conda install ipykernel
python -m ipykernel install --user --name test
```

In addition to using at the command line (using `conda activate test`), you will also see this new environment in OnDemand as an option on the Launcher when creating a new notebook. To switch to this new environment on a running notebook: - Go to Kernel menu on the left to 'change kernel' and select the environment. - click on the present kernel name on the top right corner and select the environment.

To deactivate the environment, you can use conda deactivate command.

Note: You can use conda or pip to install packages in this virtual environment.

For more information on using conda virtual environments, see Managing Environments in the conda user guide.

Version Control with Git

New to version control and Git? Visit: info on Duke's GitLab. Git is fully supported on the DCC and highly recommended for version control of your bash scripts, code, and to ease automation and deployment of your software environments using containers.

DCC Lab groups can easily create a GitLab group through Research Toolkits to easily share repos within your group. See instructions.

HTTPS and SSH are both supported and can be used to connect to Git from the DCC. HTTPS is faster and easier to setup for quickly connecting and copying a repo to the DCC. When working with repos where you regularly push and pull updates, SSH will be easier to use as SSH keys will vastly simplify your login experience.

Using HTTPS

1. Open a web browser and navigate to the Git repository you want to connect to.
2. Click on the "Clone" button in the top right corner of the repository page.
3. In the "Clone with HTTPS" section, copy the URL by clicking the "Copy" button.
4. Open a terminal or command prompt and navigate to the directory where you want to clone the repository.
5. Run `git clone <url>` to clone the repository. If prompted, enter your GitLab username and password to authenticate the clone.

Once the repository is cloned, you can make changes, commit, and push as normal.

Using SSH

Generating SSH keys

1. Open the terminal
2. Type `ssh-keygen` and press enter
3. Press enter to use the default file location
4. Enter a passphrase and press enter

You should now have a private key (id_rsa) and a public key (id_rsa.pub) in the default ~/.ssh directory

Adding an SSH key to Git

1. Go to your GitHub/GitLab profile settings
2. Select "SSH Keys" from the left sidebar
3. Click the "Add SSH Key" button
4. Give your key a title (e.g. "DCC")
5. Open the id_rsa.pub file in a text editor and copy the contents
6. Paste the contents into the "Key" field in GitLab
7. Click the "Add key" button

You should now be able to use SSH to access GitLab Note: if you encounter any problem with ssh-keygen, you may want to run ssh-keygen -t rsa -b 4096

Connecting to GitLab/GitHub Using SSH

1. Open a web browser and navigate to the Git repository you want to connect to.
2. Click on the "Clone" button in the top right corner of the repository page.
3. In the "Clone with SSH" section, copy the URL by clicking the "Copy" button.
4. Open a terminal or command prompt and navigate to the directory where you want to clone the repository.
5. Run `git clone <url>` to clone the repository. If prompted, enter your ssh key passphrase to authenticate the clone.

Once the repository is cloned, you can make changes, commit, and push as normal.

Containers

Slides from our workshop: Singularity and GitLab CI

Singularity from Syslabs.io is available on the DCC for users who would like to run a containerized environment. Singularity is a secure HPC alternative to Docker and all Docker images can be imported into Singularity. Singularity was designed with high performance computing in mind and can securely run on a HPC cluster and fully support MPI, GPUs, and other specialty high performance computing resources.

Why use containers?

Containers are used to get software to run reliably across multiple computing platforms. They can also be used to simplify software installations across different groups or even members of the same group by packing all dependencies of an application within a single image. Since the entire user space portion of the Linux environment, including programs, custom configurations, and environment, are bundled into a single file, providing the following benefits:

- **portability** - the single container file is easy to transport and can be used on a variety of stand-alone and cluster platforms.
- **reproducibility** - as software dependencies continue to grow, bundling software together with its dependencies, data, scripts, adn documentation help make results easier to reproduce.
- **shareability** - you can host singularity images on web sites and in common repositories to make it easy for colleagues to download and use.
- **usability** - you can bring a singularity container to the DCC and run it without the need for support from a systems administrator to install something for you.

Getting and building Singularity containers

Pre-built Docker and Singularity containers are often available within scientific repositories and on GitHub. Images can be moved to the DCC using the same methods as any other file. Because many images are quite large, we recommend storing them in `/hpc/group`.

Docker containers

Most Docker containers are fully supported with singularity and can be run using Singularity.

Useful Singularity Container repositories

Pre-built containers are also available in external repositories and can be loaded to the DCC using `singularity pull` or `singularity build` commands.

- Sylab's Container Library
- Docker Hub
- Red Hat's Quay
- NVIDIA GPU Cloud

Creating your own Singularity containers

Singularity containers cannot be created on the DCC directly because you need root access to the build system. Generally to build a container, you will need root access to a Linux machine. If you do not have a Linux VM, you can obtain one through Duke Virtual Computing Manager, or lab groups may have access to a RAPID VM through their Faculty/PIs allocation. Helpful documentation:

- Install Singularity on Linux
- Singularity Definition Files
- Instructions to Build a Container

Once you build your container image, you can move it to the Duke Compute Cluster for use.

Using OIT GitLab CI

Users may also create their own Singularity containers by using the OIT Gitlab CI process. Using GitLab CI, your build process for the Linux environment is automated through GitLab by providing a singularity definition file and a .gitlab-ci.yml file. Using this process, the basic steps are:

1. Create a project in GitLab. Each singularity container should have its own project and your Container image will be named after the project.
2. Create a singularity.def file in your project. You can create from scratch or start from a copy from a repository.
3. Copy the .gitlab-ci.yml file from the sample project into your project. Once this file is added, every time you commit to your project, the GitLab Continuous Integration pipeline will be initiated and a singularity image is created for your project.
4. Pull your image down to the DCC from GitLab.

For more details see these sample projects: General use or in PACE , DCC sample project

Running Singularity on the DCC

Singularity is available on the DCC and can be used as part of SLURM interactive or batch jobs. Note: you may need to bind in file system mount points (such as /datacommons) using the -B option.

Sample interactive session

In this session, jmnewton requests an interactive session through SLURM and executes a shell using the singularity image for Detectron2. Using singularity in this way lets you interact with the environment like it is a virtual machine.

```
jmnewton@dcc-login-03  /hpc/group/oit/jmnewton $ srun -p common --pty bash
srun: job 1518869 queued and waiting for resources
srun: job 1518869 has been allocated resources
jmnewton@dcc-core-315  /hpc/group/oit/jmnewton $ singularity shell detectron2.sif
Singularity> cat /etc/os-release
NAME="Ubuntu"
VERSION="18.04.3 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.3 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
```

```
Singularity> exit
exit
jmnewton@dcc-core-315  /hpc/group/oit/jmnewton $ exit
exit
jmnewton@dcc-login-03  /hpc/group/oit/jmnewton $
```

Sample batch session

The sample below illustrates how to use singularity as part of a batch session.

Sample batch script(`slurm_singularity.sh`):

```
#!/bin/bash
# Submit to a random node with a command e.g.
#   sbatch slurm_singularity.sh
#SBATCH --job-name=slurm_singularity
#SBATCH --partition=common
singularity exec /hpc/group/oit/jmnewton/detectron2.sif cat /etc/os-release
```

Submitting the batch script:

```
jmnewton@dcc-login-03  /hpc/group/oit/jmnewton $ sbatch slurm_singularity.sh
Submitted batch job 1518992
jmnewton@dcc-login-03  /hpc/group/oit/jmnewton $ cat slurm-1518992.out
NAME="Ubuntu"
VERSION="18.04.3 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.3 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
jmnewton@dcc-login-03  /hpc/group/oit/jmnewton $
```

Index

Users must have an existing account on the Duke Compute Cluster to access DCC OnDemand.

Login

Login to DCC OnDemand at dcc-ondemand-01.oit.duke.edu

The DCC OnDemand **Dashboard** will then open. From there, you can use the menus across the top of the page to manage files, get a shell on the DCC, submit jobs or open interactive applications.

To end your OnDemand session, click on the Log Out link at the top right of the Dashboard window and close your browser.

Troubleshooting Connection

If you're encountering issues with the OnDemand web interface, try clearing your browser cache. This can often resolve issues with outdated or corrupted cached files Clearing Your Browser Cache

Managing your files

To create, edit or move files, click on the **Files** menu from the **Dashboard** page. A dropdown menu will appear, listing your home directory and the work directory on the DCC. Choosing one of the file spaces opens the **File Explorer** in a new browser tab. The files in the selected directory are listed.

You can navigate to any directory mounted on the DCC by manually entering the full file system path.

Once you are in the correct directory, you can transfer files to and from the DCC using the DCC OnDemand web interface.

Working with Jobs

You can view current DCC jobs, create new job scripts, edit existing scripts, and submit them to the scheduler through the DCC OnDemand interface.

To access the job management tools, use the "Jobs" menu. For more information about using the tools, please visit the OSC job management page.

Getting a shell

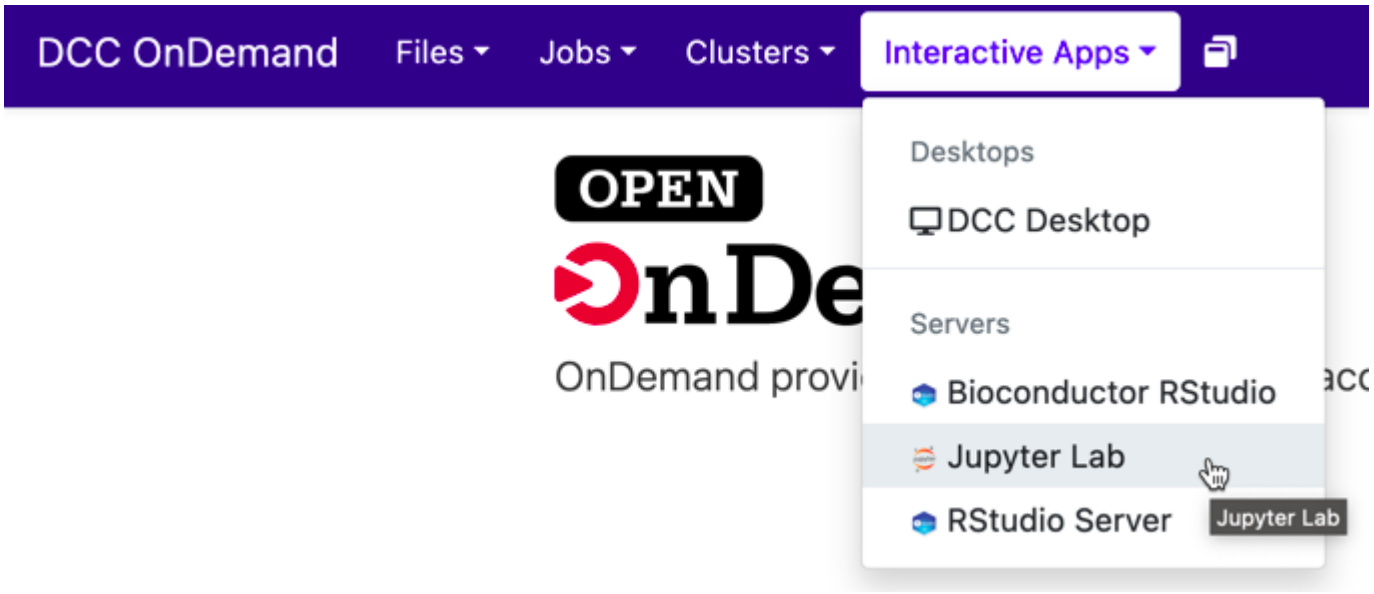
You can get shell access to the DCC by choosing **Clusters > DCC Shell Access** from the top menu in the OnDemand **Dashboard**.

In the window that will open, you'll be logged in to one of the DCC's login nodes, exactly as if you were using SSH to connect. Except you don't need to install any SSH client on your local machine.

Interactive applications

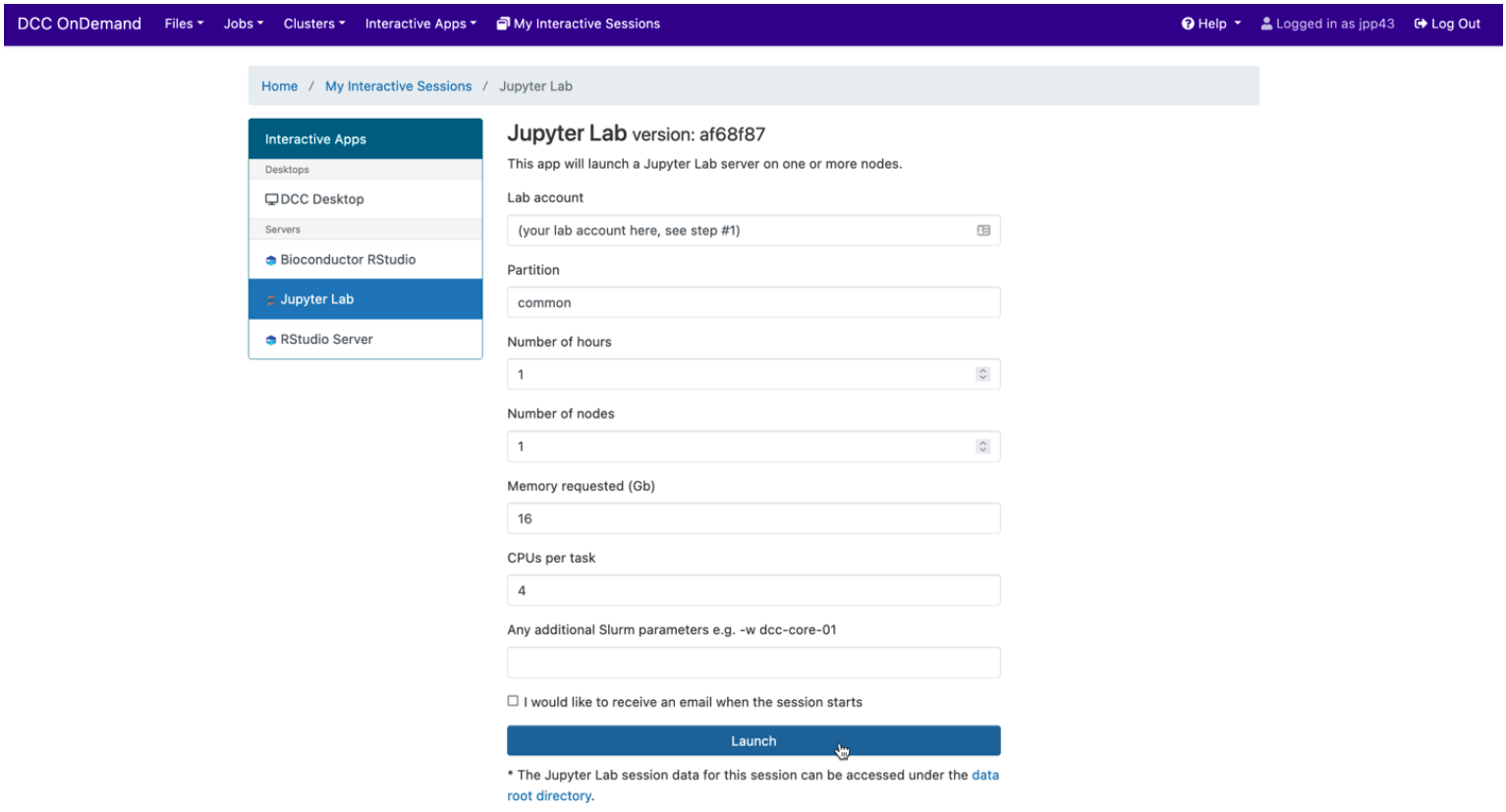
One of the main features of DCC OnDemand is the ability to run interactive applications directly from the web interface, without leaving your web browser.

Using the Jupyter Lab server



1. Click on Interactive Apps in the top navigation menu
2. Click on Jupyter Lab

Launching a Jupyter Lab server

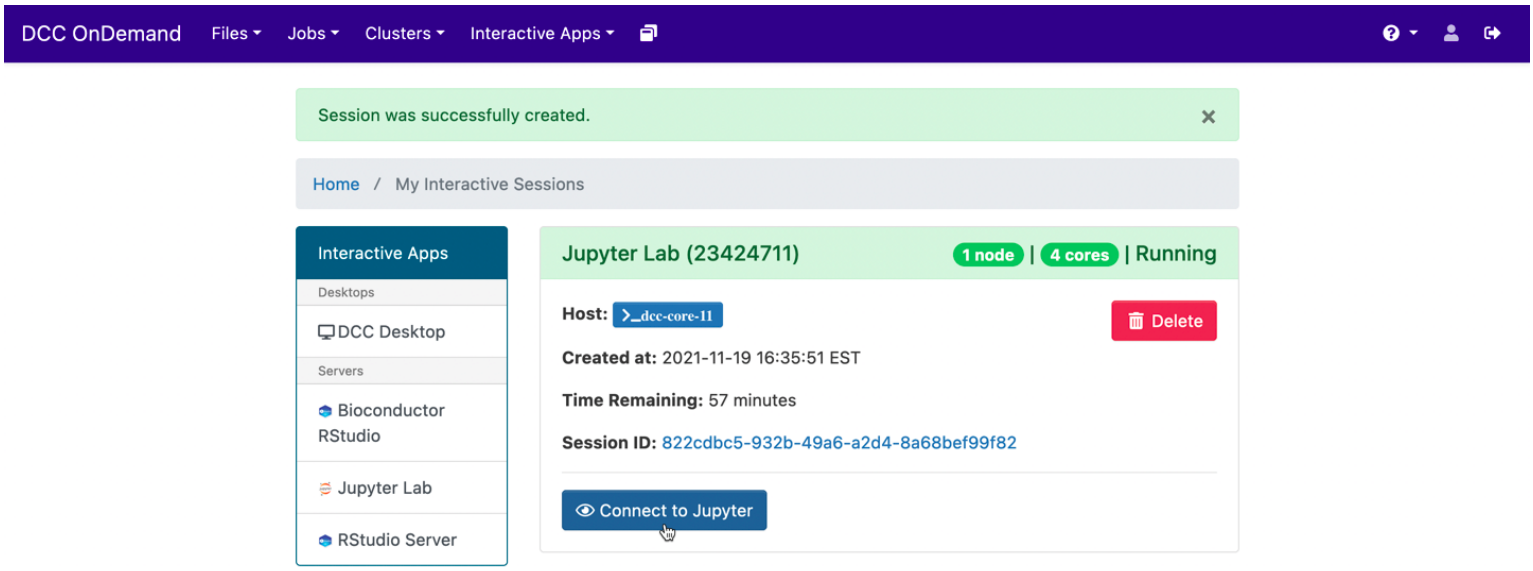


1. For lab account, input the name of your DCC group (list of all groups can be found here)
2. Under partition, type in "common", or if your lab has dedicated resources, add your own partition. You may also use common-gpu or scavenger-gpu if you need GPU resources. (remember, if a GPU is not available you may not get your interactive session in a timely fashion)
3. Input the number of hours you would like the server to remain active (please try to remain small, as it will continue

running even if you are not using it)

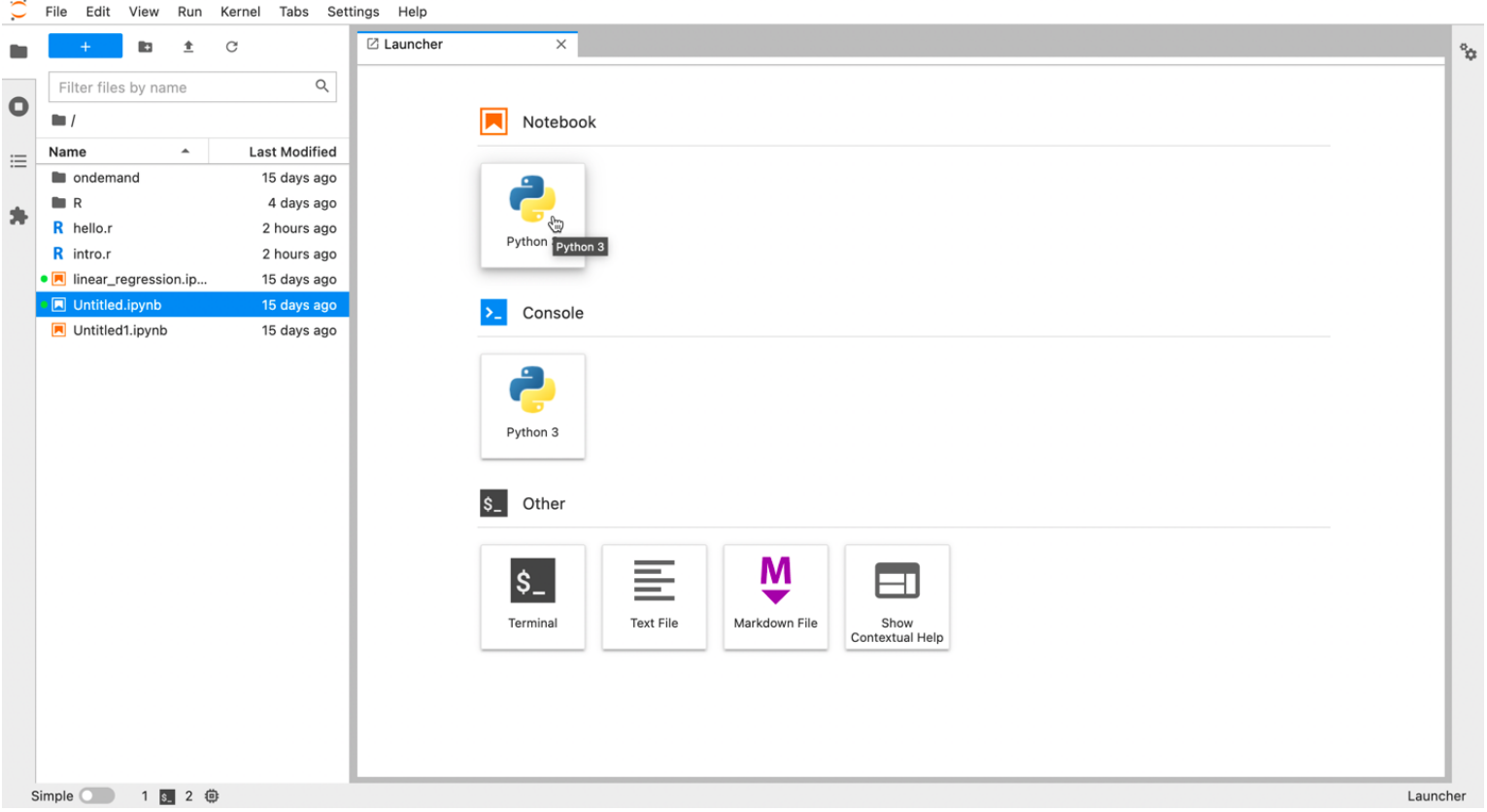
4. Input the desired amount of nodes, memory, and CPUs (try to start small with only a few gigabytes of memory and cores)
5. Enter any additional Slurm parameters (this is optional). If you would like to request a GPU, make sure the partition you have selected has GPU resources, and add `--gres=gpu:1` under “additional slurm parameters”
6. Press the blue "Launch" button on the bottom of the page

Connecting to Jupyter

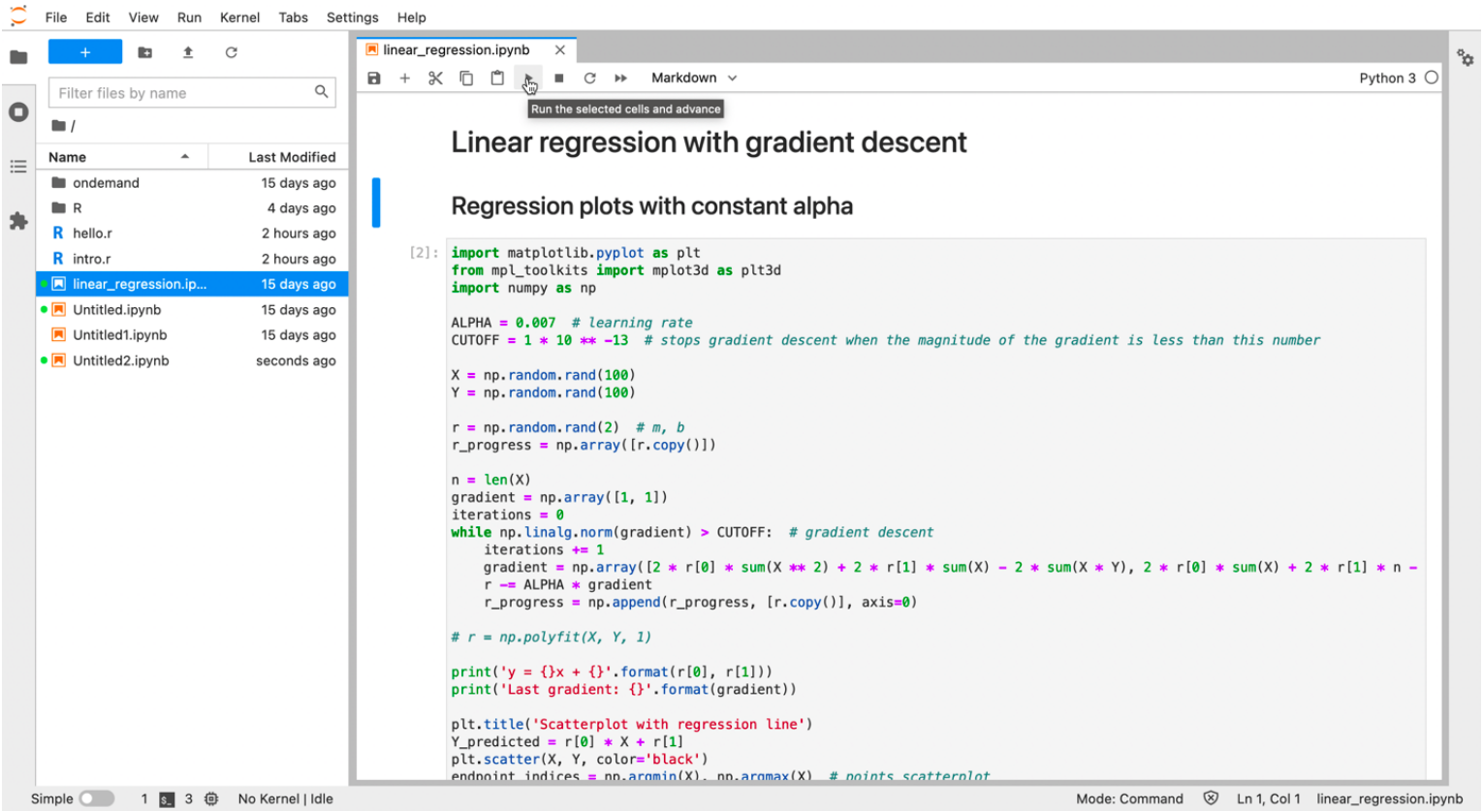


1. After pressing the blue "launch" button, your job will be queued to start a Jupyter Lab server. You should see this automatically
2. Wait a few seconds to a few minutes for the Jupyter Lab server to finish launching. The status will automatically change from "Starting" to "Running" when the server is ready
3. Press the blue "Connect to Jupyter" button when the server is running to access your Jupyter Lab server

Using Jupyter Lab



1. Click on Python3 under "Notebook" to create a new .ipynb notebook
2. Alternatively, upload your existing .ipynb files using the pane on the left-hand side
3. You can drag-and-drop or press the upward facing arrow to upload files. Note: the Jupyter session defaults to file browsing in your home directory. To browse to your group directory, first (in a terminal window) create a symbolic link in your home directory. `ln -s /hpc/group/<groupname> <groupname>`
4. When you are ready, you can run your Jupyter Notebook by pressing the run button at the top of the .ipynb file window



Using the RStudio server

1. Click on Interactive Apps in the top navigation menu
2. Click on RStudio

Launching an RStudio server

1. Select your partition from the dropdown
2. Select the name of your DCC group. If you are a member of multiple groups, make sure the group you select has access to the partition you select.
3. Input the number of hours you would like the server to remain active (please try to remain small, as it will continue running even if you are not using it)
4. Input the desired amount of nodes, memory, CPUs, and GPUs (try to start small with only a few gigabytes of memory and cores). The higher the requirements you select, the longer your wait times.
5. Optionally, enter a Datacommons mount path and/or any additional Slurm parameters
6. Use the "Select Path" button to select the base of installed R packages
7. Press the blue "Launch" button on the bottom of the page

Connecting to RStudio

1. After pressing the blue "launch" button, your job will be queued to start an RStudio server. You should see this automatically
2. Wait a few seconds to a few minutes for the RStudio server to finish launching. The status will automatically change from "Starting" to "Running" when the server is ready
3. Press the blue "Connect to RStudio Server" button when the server is running to access your RStudio server

Using RStudio Server

1. In the top left of the interface, click on "File" > "New File" > "R Script" to create a new R Script
2. To save this file, use Ctrl+S (Command+S on macOS) and choose a file path
3. Alternatively, upload your existing .r files using the "Upload" button toward the top of the pane in the bottom left corner
4. When you are ready, you can run your R Script by pressing the "Run" button toward the top right of the top left pane

Packages available in RStudio

Several pre-built options are available through RStudio server to support a variety of package installations. Once you select your option, you may also install additional packages.

Image Name	R Version	About	Build Repo

Image Name	R Version	About	Build Repo
`jags.sif`	4.1.1	JAGS is Just Another Gibbs Sampler. It is a program for the statistical analysis of Bayesian hierarchical models by Markov Chain Monte Carlo.	JAGS Singularity Recipe
`rstudio-rstan.sif`	4.1.1	RStan and supporting/common packages	RSTAN Singularity Recipe
`ondemand_bioconductor.sif`	Bioconductor is version 3.14; R version 4.1.0	This image is based on the Microsoft Docker Bioconductor image, more about Bioconductor	Bioconductor Singularity Recipe

Installing your own packages

Coming Soon

How to use containers in the DCC and with OnDemand

You can use Apptainer/Singularity containers to encapsulate software that you require to run your analyses. These containers can be used with normal batch and interactive Slurm jobs and they can also be used with the OnDemand server (provided they are built to support the OnDemand application).

Some basic information and links to further documentation for containers can be found at https://dcc.duke.edu/software/singularity/.

A suggested method to create containers for use with Jupyter Lab and RStudio follow.

In both cases you will be using the OIT Gitlab instance to create a project that will build your container.

- Open https://gitlab.oit.duke.edu in your browser and click on the "Duke Shibboleth Login" button. If you don't have a GitLab account already one will be created for you.
- You should the click on the "New Project" button and create a project. The project name will also be the name of the container image file. The project can be owned by you or a GitLab group. You may want the ownership to be a group if the image will be shared with lab members.
- You will then need to add two files to your project: .gitlab-ci.yml and Singularity.def. You will also want to update the README.md to reflect what your container is used for. The content of the .gitlab-ci.yml file should be an exact copy of the one in the following two projects. The Singularity.def contents will vary as this is where you customize what packages you want installed in your image.

Jupyterlab project

Use the project https://gitlab.oit.duke.edu/OIT-DCC/dcc-container-example as an example for creating an image that will contain Jupyterlab and any additional Python modules you need to use.

- Copy the contents of the .gitlab-ci.yml file to your own projects .gitlab-ci.yml file.
- Create your own Singularity.def file using the one in the project as a template. Change the %label to reflect you or your groups information, and in the %post section after the mamba install of jupyterlab add lines that install the modules you wish to use.
- When you push your changes to your project the .gitlab-ci.yml files precense will cause a GitLab CI process to kick off. This will build a container with the name of your project and deploy it to the DCC. It will be stored at /opt/apps/containers/community//.sif or /opt/apps/containers/community/<GitLab_Group/.sif.
- To use your image, launch the "Jupyter Lab Singularity" app under "Interactive Apps" at https://dcc-ondemand-01.oit.duke.edu. In the "Singularity Container File" field enter the stored location of your image e.g. /opt/apps/containers/community//.sif and launch the job. Your Jupyter Lab job will be able to use the modules you built your container with.

RStudio project

Use the project https://gitlab.oit.duke.edu/OIT-DCC/dcc-container-rstudio-example as an example for creating an image that will contain RStudio and any additional R modules you need to use.

- Copy the contents of the .gitlab-ci.yml file to your own projects .gitlab-ci.yml file.
- Create your own Singularity.def file using the one in the project as a template. Change the %label to reflect you or your groups information, and in the %post section There is a section commented "R: Install Additional R Packages"

that shows installing additional R packages.

- When you push your changes to your project the .gitlab-ci.yml files precense will cause a GitLab CI process to kick off. This will build a container with the name of your project and deploy it to the DCC. It will be stored at /opt/apps/containers/community//.sif or /opt/apps/containers/community/<GitLab_Group/.sif.
- To use your image, launch the "RStudio Singularity" app under "Interactive Apps" at https://dcc-ondemand-01.oit.duke.edu. In the "Singularity Container File" field enter the stored location of your image e.g. /opt/apps/containers/community//.sif and launch the job. Your RStudio job will be able to use the modules you built your container with.

Using the DCC Desktop with Matlab

This procedure demonstrates how to load Matlab, but can be done with any applications installed on the DCC that have a GUI interface (for example: Matlab, Mathematica, Comsol).

1. Click on Interactive Apps in the top navigation menu
2. Click on DCC Desktop

Launching a DCC Desktop

1. For lab account, input the name of your DCC group (list of all groups can be found here)
2. Under partition, type in "common", or if your lab has dedicated resources, add your own partition
3. Input the number of hours you would like the session to remain active (please try to remain small, as it will continue running even if you are not using it)
4. Input the desired amount of nodes, memory, and CPUs (try to start small with only a few gigabytes of memory and cores)
5. Press the blue "Launch" button on the buttom of the page
6. After pressing the blue "launch" button, your job will be queued to start
7. Wait a few seconds to a few minutes for the Desktop to finish launching. The status will automatically change from "Starting" to "Running" when it is ready. If you have requested a large number of resources, your session will wait to start until the resources are available on the DCC.
8. Press the blue "Launch DCC Desktop" button when it is ready

Using DCC Desktop to launch Matlab

In the bottom center launch a terminal window:

```
$ module load Matlab/R2022b
$ matlab
```

Your Matlab GUI should now launch within your browser window.

DCC Community Software

Do you have or use software on the DCC that would benefit the larger DCC community? We are happy to host you! We will provide the hosting location and storage, give you access to manage the install/files, and post your info to this site. Contact us for information.

OnDemand

RStudio Interactive Servers

Additional RStudio Interactive Servers can be created by forking the base OnDemand RStudio project.

Microbiome

Created by Josh Granek at the Duke Center for Human Systems Immunology to assist with bioinformatics support for the microbiome community at Duke. This app will launch an RStudio server from a Microbiome focused Singularity image on one or more nodes.

OnDemand GitLab Project

1. RStudio Microbiome Singularity Recipe

DCC

Alphafold2

Thanks to the Duke Human Vaccine Institute, Alphafold 2.2.0 from Deep Mind is installed inside of a singularity container on the DCC.

Sample job submission scripts for Alphafold Monomer and Alphafold Multimer predictions are provided at:

```
/opt/apps/community/alphafold2/alphafoldv2.2/scripts
```

To use Alphafold, first copy the sample submission script to your group directory, then using the comments, edit the submission script.

To connect with the Duke Alphafold2 Community enroll in the community listserve.

H200 Partition

The H200 partition is comprised of 6 nodes with 54 H200 GPUs. Access to this partition is limited and is only available by direct request from a faculty PI.

General use of an H200 GPU

We have enabled MIG on some of the GPUs within the H200ea partition.

Slurm settings:

```
Account: h200ea
Partition: h200ea
--gres=gpu:h200
```

Fractionalized GPUs

Two GPUs on dcc-h200-gpu-05 (0 and 1) have been fractionalized (one into 7 parts and one into 2 parts). Fractional GPUs can be requested with:

```
--gres=gpu: h200_1g.18gb:1
--gres= gpu:h200_4g.71gb:1
--gres= gpu:h200_3g.71gb:1
```

Submission

All h200ea jobs will need to specify the “gres” with the “h200” GPU type, e.g.

`#SBATCH --gres=gpu:h200:1`

This is to accommodate the MIG settings on dcc-h200-gpu-05.

Due to the increased usage of the h200ea partition, we ask that jobs that will require more than 72 hours to complete be limited to a single H200 GPU (7-day partition limit). If the jobs will require less than 72 hours, 2-GPU jobs can be submitted with the line

`#SBATCH --time=72:00:00`

added to the job script. There is currently a per user max GPU limit of 12 H200s allocated at one time.

DCC Partitions

SLURM partitions are separate queues that divide up a cluster's nodes based on specific attributes. Each partition has its own constraints, which control which jobs can run in it. There are many DCC partitions. Access to each partition is restricted based on your slurm account. Users may have multiple slurm accounts and must specify the correct slurm account to gain access to restricted partitions.

General use DCC Partitions

- **common** for jobs that will run on the DCC core node
- **gpu-common** for jobs that will run on DCC GPU nodes
- **scavenger** for jobs that will run on lab-owned nodes in “low priority” (kill and requeue preemption).
- **scavenger-gpu** for GPU jobs that will run on lab-owned nodes in “low priority” (kill and requeue preemption)
- **courses** and **courses-gpu** are special paritions used to support Duke courses. You must have an `account=coursess25` to access these partitions. If you are only part of the DCC through a course, you will not be able to use the common or gpu-common partitions.
- **interactive** for debugging and testing scripts. This should be limited to short interactive sessions or short-duration tests and is not intended for use in general analyses. Default resources are low. See below for partition limits per user.
- **h200ea** for access to h200 GPUs. Limited access, by request only from faculty PI. Send requests to rescomputing@duke.edu.

Note: If a partition is not specified, the default partition is the common partition.

Duke Compute Account Limits

These limits are subject to change.

Duke Compute Cluster Partition Limits	Default Value
Max running Mem per user account	1.5TB
Max running CPUs per user account	400
Max queued jobs per user per account	400

Parition Limits Per User

Partition Name	Max CPU	Max Memory (GB)
interactive	10	64

Configuration

Hardware

Partition Name	Number of Nodes	Processors	RAM (GB)	GPU	Max Walltime (days-hours:minutes:seconds)
common	57	4844	37791	--	90-00:00:00
common-gpu	32	424	3354	34	2-00:00:00
scavenger	88	7502	48078	--	90-00:00:00
scavenger-gpu	187	3643	24283	215	7-00:00:00
courses	50	4200	23334	--	7-00:00:00
courses-gpu	10	840	4666	20	2-00:00:00
compalloc	4	496	3975	--	30-00:00:00
interactive	61	5340	42673	--	1-00:00:00
h200ea	6	1116	11780	54	7-00:00:00

GPUs

GPU Type	GPU Full Name	VRAM (GB)	GPU per Node	Number of Nodes with GPU Config	Partitions
2080	nvidia_geforce_rtx_2080_ti		1	7	common-gpu
2080	nvidia_geforce_rtx_2080_ti		3	1	common-gpu
5000_ada	nvidia_rtx_5000_ada_generation		1	24	common-gpu
2080	nvidia_geforce_rtx_2080_ti		1	67	scavenger-gpu
2080	nvidia_geforce_rtx_2080_ti		4	2	scavenger-gpu

GPU Type	GPU Full Name	VRAM (GB)	GPU per Node	Number of Nodes with GPU Config	Partitions
5000_ada	nvidia_rtx_5000_ada_generation		1	4	scavenger-gpu
6000_ada	nvidia_rtx_6000_ada_generation		1	24	scavenger-gpu
6000_ada	nvidia_rtx_6000_ada_generation		4	1	scavenger-gpu
a5000	nvidia_rtx_a5000		1	68	scavenger-gpu
a5000	nvidia_rtx_a5000		4	3	scavenger-gpu
a6000	nvidia_rtx_a6000		1	8	scavenger-gpu
p100	tesla_p100-pcie-16gb		2	10	scavenger-gpu

Useful SLURM commands for resources

These commands can be used to view cluster resources.

General partition info

- `scontrol show partition partitionName` will give you general information about a specific partition.

Example for the common partition:

```
(base) rm145@dcc-login-02  ~ $ scontrol show partition common
PartitionName=common
   AllowGroups=ALL DenyAccounts=courses,coursess25,panlab AllowQos=ALL
   AllocNodes=ALL Default=YES QoS=N/A
   DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO ExclusiveTopo=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=90-00:00:00 MinNodes=0 LLN=NO MaxCPUsPerNode=UNLIMITED
MaxCPUsPerSocket=UNLIMITED
   Nodes=dcc-comp-[01-10],dcc-core-[01-12,14-41,43-49]
   PriorityJobFactor=10 PriorityTier=10 RootOnly=NO ReqResv=NO OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=GANG,REQUEUE
   State=UP TotalCPUs=4844 TotalNodes=57 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
   TRES=cpu=4844,mem=38698635M,node=57,billing=4844
```

Specific GPU info per partition

```
scontrol show node `sinfo -h -r -p partitionName -o %N` | grep Gres | sort | uniq -c
```

Example for GPUs in the common-gpu partition:

```
rm145@dcc-login-05  ~ $ scontrol show node `sinfo -h -r -p gpu-common -o %N` | grep Gres | sort | uniq -c
      7    Gres=gpu:2080:1
      1    Gres=gpu:2080:3
     24    Gres=gpu:5000_ada:1
```

Current version of the intro to the DCC slides.

Introduction to SLURM

Most DCC partitions are lab-owned machines. These can only be used by members of the group. Submitting to a group partition gives “high-priority”.

Submit to partitions with

```
sbatch –p (partition name) --account=(account name)
```

(in a job script) or

```
srun –p (partition name) --account=(account name) --pty bash –i
```

(interactively) In general, the partition name and account name will be the same for most lab-owned machines.

Partitions

SLURM partitions are separate queues that divide up a cluster's nodes based on specific attributes. Each partition has its own constraints, which control which jobs can run in it. There are many DCC Partitions, if a partition is not specified, the default partition is the common partition.

Running an interactive job

Reserve a compute node by typing `srun –pty bash -i`

```
tm103@dcc-login-02 ~ $ srun --pty bash -i
srun: job 186535 queued and waiting for resources
srun: job 186535 has been allocated resources
tm103@dcc-core-11 ~ $
tm103@dcc-core-11 ~ $ squeue -u tm103
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
186535 common bash tm103 R 0:14 1 dcc-core-11
```

I now have an interactive session in the common partition on node dcc-core-11

SLURM commands

`sbatch` – Submit a batch job

`#SBATCH` – Specify job parameters in a job script

`squeue` – Show lists of jobs

`scancel` – Delete one or more batch jobs

`sinfo` – Show info about machines

`scontrol` – Show cluster configuration information

Use `sbatch` (all lower case) to submit text file job scripts, e.g. test.sh

```
sbatch test.sh
```

Use `#SBATCH` (upper case) in your scripts for scheduler directives, e.g.

```
#SBATCH --mem=1G
#SBATCH --output=matlab.out
```

All SLURM directives can be given on the command line instead of the script. slurm.schedmd.com

Slurm memory directives

The default memory request (allocation) is 2 GB RAM. This is a hard limit, always request a little more. To request a total amount of memory for the job, use one of the following: * `--mem=<MB>` additional memory * `--mem=<Gigabyte>G` the amount of memory required per node, or * `--mem-per-cpu=<MB>` the amount of memory per CPU core, for multi-threaded jobs

Note: –mem and –mem-per-cpu are mutually exclusive

Slurm parallel directives

All parallel directives have defaults of 1

`-N <number>` How many nodes (machines)

`-n <number>` or `--ntasks=<number>` How many parallel jobs (“tasks”)

`-c <number>` or `--cpus-per-task=<number>`

Use `-n` and `-N` for multi-node jobs (e.g. MPI)

Use `-c` (–cpus-per-task) for multi-threaded jobs

Job script examples

```
#!/bin/bash
#SBATCH ---output=test.out
hostname # print hostname
```

This prints the name of the compute node in the file “test.out”

```
tm103@dcc-login-02 ~/slurm $ sbatch simple.sh
Submitted batch job 186554tm103@dcc-login-02 ~/slurm $ cat test.out
dcc-core-14
```

Long-form commands example

```
#SBATCH --output=slurm.out
#SBATCH --error=slurm.err
#SBATCH --mem=100 # 100 MB RAM
#SBATCH --partition=scavenger#
hostname 1>&2  #prints hostname to the error file
```

This job will run in low priority on a lab node in the “scavenger” partition

Short-form commands example.

SLURM short commands don’t use “=“ signs

```
#!/bin/bash
#SBATCH -o slurm.out
#SBATCH -e slurm.err
#SBATCH --mem=4G # 4 GBs RAM
#SBATCH –p scavenger
hostname 1>&2  #prints hostname to the error file
```

R example script

```
#!/bin/bash
#SBATCH –e slurm.err
#SBATCH --mem=4G # 4 GB RAM
module load R/3.6.0
R CMD BATCH Rcode.R
```

This loads the environment module for R/3.6.0 and runs a single R script (“Rcode.R”)

The `#SBATCH –mem=4G` requests additional RAM

Multi-threaded (multi-core) example

```
!/bin/bash
#SBATCH –J test
#SBATCH –o test.out
#SBATCH –c 4
#SBATCH –-mem-per-cpu=500 #(500 MB)
myApplication –n $SLURM_CPUS_PER_TASK
```

The value of `$SLURM_CPUS_PER_TASK` is the number after `-c`. This example starts a single, multi-threaded job that uses 4 CPU cores and 2 GB (4x500MB) of RAM

OpenMP multicore example

```
!/bin/bash
#SBATCH –J openmp-test
#SBATCH –o slurm.out
#SBATCH –c 4
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
myOpenMPapp # will run on 4 CPU cores
```

This sets `$OMP_NUM_THREADS` to the value of `$SLURM_CPUS_PER_TASK`

Slurm job arrays

Slurm job arrays are a mechanism for submitting and managing collections of similar jobs using one job script and one application program.

- Add `--array` or `-a` option to the job script
- Each job task will inherit a `SLURM_ARRAY_TASK_ID` environment variable with a different integer value

- Each job array can be up 100,000 job tasks on the DCC
- Job arrays are only supported for batch jobs
- Job array "tasks" must be independent: slurm.schedmd.com/job_array.html

For example, in a job script, add the line `#SBATCH --array=1-30` or, alternatively, `#SBATCH -a 1-30` to submit 30 job tasks. The job array indices can also be specified on the command line, e.g.

```
sbatch -a 1-30 myjob.sh
```

The index values can be continuous, e.g.

```
-a 0-31 (32 tasks, numbered from 0,1,2,…,31)
```

or discontinuous, e.g.

```
-a 3,5,7-9,12 (6 tasks, numbers 3,5,7,8,9,12)
```

It can also be a single job task, e.g.

```
-a 7
```

The discontinous notation is useful for resubmitting specific job tasks that had previously failed. Each job task is assigned the enviromental variable `$SLURM_ARRAY_TASK_ID` set to it's index value.

```
tm103@dcc-login-02  ~/misc/jobarrays $ cat array-test.sh
#!/bin/bash
echo $SLURM_ARRAY_TASK_ID
tm103@dcc-login-02  ~/misc/jobarrays $ sbatch -a 1-3 array-test.sh
Submitted batch job 24845830
tm103@dcc-login-02  ~/misc/jobarrays $ ls slurm-24845830*
slurm-24845830_1.out  slurm-24845830_2.out  slurm-24845830_3.out
tm103@dcc-login-02  ~/misc/jobarrays $ cat slurm-24845830*
1
2
3
tm103@dcc-login-02  ~/misc/jobarrays $
```

Python job array example

```
#!/bin/bash
#SBATCH -e slurm_%A_%a.err
#SBATCH -o slurm_%A_%a.out
#SBATCH --array=1-5000
python myCode.py

$ cat test.py
import os
taskID=int(os.environ['SLURM_ARRAY_TASK_ID'])
...
```

Start 5000 Python jobs, each with a different "taskID", initialized from `$SLURM_ARRAY_TASK_ID`

Importing Environmental Variables

Python:

```
numCPUs=int(os.environ['SLURM_CPUS_PER_TASK'])
taskID=int(os.environ['SLURM_ARRAY_TASK_ID'])
```

R:

```
numCPUs <- as.integer(Sys.getenv(SLURM_CPUS_PER_TASK))
taskID <- as.integer(Sys.getenv(SLURM_ARRAY_TASK_ID))
```

MATLAB:

```
numCPUs = str2num(getenv('SLURM_CPUS_PER_TASK'))
taskID = str2num(getenv('SLURM_ARRAY_TASK_ID'))
```

Processing separate input files

Process an existing file list, e.g. files.txt

```
#!/bin/bash
readarray -t FILES < files.txt
FILENAME=${FILES[(($SLURM_ARRAY_TASK_ID - 1))]}
myapp $FILENAME
```

Dynamically generate a file list from “ls”

```
#!/bin/bash
export FILES=($(ls -1 myfile*))
FILENAME=${FILES[(($SLURM_ARRAY_TASK_ID - 1))]}
myapp $FILENAME
```

Example: Using the taskID as part of the file name and output directory for the case with input file names of the form input1,input2,...,inputN for -a 1-N, e.g.

```
#!/bin/bash
#SBATCH -e slurm_%A_%a.err
#SBATCH -o slurm_%A_%a.out
mkdir out_${SLURM_ARRAY_TASK_ID}
cd out_${SLURM_ARRAY_TASK_ID}
myapp ../input_${SLURM_ARRAY_TASK_ID}.txt
```

where output directories out1, out2, ... are created for input files input1.txt, input2.txt,...

“Unrolling” for loops example

Original “serial” code (Python)

```
fibonacci = [0,1,1,2,3,5,8,13,21]
for i in range(len(fibonacci)):
    print(i,fibonacci[i])
```

Job array version

```
import os
i=int(os.environ['SLURM_ARRAY_TASK_ID'])
fibonacci = [0,1,1,2,3,5,8,13,21]
```

```
#for i in range(len(fibonacci)):
    print(i,fibonacci[i])
```

where the for loop is commented-out and each job task is doing a single “iteration”

```
tm103@dcc-login-02  ~/misc/jobarrays $ cat fib-array.sh
#!/bin/bash
#SBATCH -e slurm.err
module load Python/2.7.11
python fibonacci.py
tm103@dcc-login-02  ~/misc/jobarrays $ sbatch –a 1-8 fib-array.sh
Submitted batch job 24856052
tm103@dcc-login-02  ~/misc/jobarrays $ ls slurm-24856052_*
slurm-24856052_1.out  slurm-24856052_3.out  slurm-24856052_5.out  slurm-24856052_7.out
slurm-24856052_2.out  slurm-24856052_4.out  slurm-24856052_6.out  slurm-24856052_8.out
tm103@dcc-login-02  ~/misc/jobarrays $ cat slurm-24856052*
(1, 1)
(2, 1)
(3, 2)
(4, 3)
(5, 5)
(6, 8)
(7, 13)
(8, 21)
tm103@dcc-login-02  ~/misc/jobarrays $
```

Running MPI jobs

Supported MPI versions are Intel MPI and OpenMPI

Compiling with OpenMPI

```
tm103@dcc-login-02  ~ $ module load OpenMPI/4.0.5-rhel8
OpenMPI 4.0.5-rhel8
tm103@dcc-login-03  ~ $ mpicc -o openhello hello.c
tm103@dcc-login-02  ~ $ ls -l openhello
-rwxr-xr-x. 1 tm103 scsc 9184 Sep  1 16:08 openhello`
```

OpenMPI job script

```
#!/bin/bash
#SBATCH -o openhello.out
#SBATCH -e slurm.err
#SBATCH -n 20
module load OpenMPI/4.0.5-rhel8
mpirun -n $SLURM_NTASKS openhello
```

OpenMPI example output

```
tm103@dcc-login-02 ~/misc/slurm/openmpi $ cat openhello.out
dcc-core-01, rank 0 out of 20 processors
dcc-core-01, rank 1 out of 20 processors
dcc-core-01, rank 2 out of 20 processors
dcc-core-01, rank 3 out of 20 processors
dcc-core-01, rank 4 out of 20 processors
dcc-core-03, rank 13 out of 20 processors
dcc-core-03, rank 14 out of 20 processors
dcc-core-03, rank 10 out of 20 processors
dcc-core-03, rank 11 out of 20 processors
dcc-core-03, rank 12 out of 20 processors
dcc-core-02, rank 8 out of 20 processors
dcc-core-02, rank 9 out of 20 processors
```

```
dcc-core-02, rank 5 out of 20 processors
```

GPU jobs

To run a GPU batch job, add the job script lines

```
#SBATCH -p gpu-common
#SBATCH --gres=gpu:1
#SBATCH --exclusive
```

To get an interactive GPU node session, type the command line `srun -p gpu-common --gres=gpu:1 --pty bash -i` tm103@dcc-clogin-02 ~ $ srun -p gpu-common --gres=gpu:1 --pty bash -i tm103@dcc-gpu-01 ~ $ /usr/local/cuda-7.5/samples/1_Utilities/deviceQuery/deviceQuery ... Detected 1 CUDA Capable device(s)

```
Device 0: "Tesla K80"
    CUDA Driver Version / Runtime Version          7.5 / 7.5
    CUDA Capability Major/Minor version number:    3.7
    Total amount of global memory:                 11520 MBytes (12079136768 bytes)
    (13) Multiprocessors, (192) CUDA Cores/MP:     2496 CUDA Cores
```

Cluster GPU Types

Request a specific GPU type and GPU device number with

```
#SBATCH --gres=gpu:(GPU type):(GPU number)
```

where "GPU number" was a number from 1 to 4 and "GPU type" was from the list below.

GPU type	GPU full name	VRAM (GB)	availability notes
2080	nvidia_geforce_rtx_2080_ti	11	gpu-common, researcher owned, and scavenger-gpu
5000_ada	nvidia_rtx_5000_ada_generation	32	gpu-common, researcher owned, and scavenger-gpu
a6000	nvidia_rtx_a6000	30	researcher owned and scavenger-gpu
6000_ada	nvidia_rtx_6000_ada_generation	48	researcher owned and scavenger-gpu
p100	tesla_p100-pcie-16gb	16	courses, scavenger-gpu
a5000	nvidia_rtx_a5000	24	researcher owned and scavenger-gpu
a100	nvidia_a100_80gb_pcie	80	researcher owned
6000	quadro_rtx_6000	24	researcher owned
h100	nvidia_h100_nvl	80	researcher owned

To see all GPU types and count on the cluster, run `gpuavail` from a login node. This command will also provide a

realtime listing of GPUs available.

```
kk338@dcc-login-04  ~ $ gpuavail

     CONFIGURATION
  NODE                          NODE
  TYPE                         COUNT
  ---------------------
  gpu:a5000:1                   76
  gpu:2080:1                    74
  gpu:5000_ada:1                28
  gpu:6000_ada:1                24
  gpu:p100:2                    10
  gpu:a6000:1                    8
  gpu:a5000:4                    6
  gpu:2080:4                     4
  gpu:6000_ada:4                 1
  gpu:2080:3                     1
  gpu:5000_ada:4                 1


                AVAILABILITY
  NODE                          GPU       GPU     GPUs    CPUs    GB MEM
  NAME                          TYPE      COUNT   AVAIL   AVAIL   AVAIL
  ------------------------------------------------------------------
  dcc-allenlab-gpu-01           a6000     1       1       9       114
  dcc-allenlab-gpu-02           a6000     1       1       9       114
  dcc-allenlab-gpu-03           a6000     1       1       9       114
  dcc-allenlab-gpu-04           a6000     1       1       9       114
  dcc-allenlab-gpu-05           6000_ada  1       1       11      118
  dcc-allenlab-gpu-06           6000_ada  1       1       11      118
  dcc-allenlab-gpu-07           6000_ada  1       1       11      118
  dcc-allenlab-gpu-08           6000_ada  1       1       11      118
  dcc-allenlab-gpu-09           6000_ada  1       1       11      118
  dcc-allenlab-gpu-10           6000_ada  1       1       11      118
  dcc-allenlab-gpu-11           6000_ada  1       1       11      118
  dcc-brunellab-gpu-01          2080      1       1       13      51
  dcc-brunellab-gpu-02          2080      1       1       21      83
  dcc-brunellab-gpu-03          2080      1       1       21      83
  dcc-brunellab-gpu-04          2080      1       1       21      83
  dcc-carlsonlab-gpu-01         2080      1       1       21      83
```

Job dependencies

Submit a job that waits for another job to finish.

```
$ sbatch dep1.q
Submitted batch job 666898
```

Make a note of the assigned job ID of dep1

```
$ sbatch --dependency=afterok:666898 dep2.q
Job dep2 will not start until dep1 finishes
```

Job dependencies with arrays

Wait for specific job array elements

```
sbatch --depend=after:123_4 my.job sbatch --depend=afterok:123_4:123_8 my.job2
```

Wait for entire job array to complete

```
sbatch --depend=afterany:123 my.job
```

Wait for entire job array to complete successfully

```
sbatch --depend=afterok:123 my.job
```

Wait for entire job array to complete and at least one task fails

```
sbatch --depend=afternotok:123 my.job
```

More information: hcc-docs.unl.edu/display/HCCDOC/Job+Dependencies

Additional Resources

http://schedmd.com

Usage Reports

Cluster Usage Reporting

View real-time job accounting on the cluster with sacct or job reporting for the last 30 days with sreport.

Historical job accounting is available to DCC users through Tableau.

All job data is reported on the start day of the job (important for long running jobs).

登录到 Tableau at Duke

If you are member of multiple lab groups and need to update your default SLURM account for reporting information, please use:

```
sacctmgr modify user u=(NetID) set DefaultAccount=(account name)
```