The Foundry

System Information

Software

The Foundry was built and managed with Puppet. The underlying OS for the Foundry is Ubuntu 18.04 LTS. With the Foundry we made the conversion from Centos to Ubuntu, and made the jump from a 2.6.x kernel to a 5.3.0 kernel build. For resource management and scheduling we are using SLURM Workload manager version 17.11.2

Hardware

Management nodes

The head nodes are virtual servers, the login nodes match that of one of the compute node types.

Compute nodes

The newly added compute nodes are Dell C6525 nodes configured as follows.

Dell C6525: 4 node chassis with each node containing dual 32 core AMD EPYC Rome 7452 CPUs with 256 GB DDR4 ram and 6 480GB SSD drives in raid 0.

As of 06/17/21 we currently have over 11,000 cores of compute capacity on the Foundry.

GPU nodes

The newly added GPU nodes are Dell C4140s configured as follows.

Dell C4140: 1 node chassis with 4 Nvidia V100 GPUs connected via NV-link and interconnect with other nodes via HDR-100 infiniband. Each has dual 20 core intel processors and 192GB of DDR4 ram.

As of 06/17/21 we currently have 24 V100 GPUs available for use.

Storage

General Policy Notes

None of the cluster attached storage available to users is backed up in any way by us, this means that if you delete something and don't have a copy somewhere else, it is gone. Please note the data
stored on cluster attached storage is limited to Data Class 1 and 2 as defined by UM System Data Classifications. If you have need to store things in DCL3 or DCL4 please contact us so we may find a solution for you.

**Home Directories**

The Foundry home directory storage is available from an NFS share backed by our enterprise SAN, meaning your home directory is the same across the entire cluster. This storage will provide 10 TB of raw storage, limited to 50GB per user. **This volume is not backed up, we do not provide any data recovery guarantee in the event of a storage system failure.** System failures where data loss occurs are rare, but they do happen. All this to say, you **should not** be storing the only copy of your critical data on this system. If you find you need more storage we provide a couple options. The first is scratch space which holds no quota, but is regularly cleaned which means that if your data goes stale in the scratch space it will be deleted to make room for new. The second is a storage lease.

**Scratch Directories**

Each user will get a scratch directory created for them at /lustre/scratch/$USER an alias of `cdsc` has also been made for users to cd directly to this location. As with all storage scratch space is not backed up, and even more to the fact of data impermanence in this location it is actively cleaned in an attempt to prevent the storage from filling up. The intent for this storage is for your programs to create temporary files which you may need to keep after the calculation completes for a short time only. The volume is a high speed network attached scratch space, there currently are no quotas placed on the directories in this scratch space, however if the 60TB volume filling up becomes a problem we will have to implement quotas.

Along with the networked scratch space, there is always local scratch on each compute node for use during calculations in /tmp. There is no quota placed on this space, and it is cleaned regularly as well, but things stored in this space will only be available to processes executing on the node in which they were created. Meaning if you create it in /tmp in a job, you won't be able to see it on the login node, and other processes won't be able to see it if they are on a different node than the process which created the file.

**Leased Space**

If home directory, and scratch space availability aren't enough for your storage needs we also lease out quantities of cluster attached space. If you are interested in leasing storage please contact us. If you already are leasing storage, but need a reference guide on how to manage the storage please go here.

**Policies**

**Under no circumstances should your code be running on the login node.**

You are allowed to install software in your home directory for your own use. Know that you will *NOT* be given root/sudo access, so if your software requires it you will not be able to use that software.
without our help. Contact ITRSS about having the software installed as modules for large user groups.

User data on the Foundry is not backed up meaning it is your responsibility to back up important research data to a location off site via any of the methods in the moving data section.

If you are a student your jobs can run on any compute node in the cluster, even the ones dedicated to researchers, however if the researcher who has priority access to that dedicated node then your job will stop and go back into the queue. You may prevent this preemption by specifying to run on just the non-dedicated nodes in your job file, please see the documentation on how to submit this request.

If you are a researcher who as purchased a priority lease, you will need to submit your job to your priority partition, otherwise your job will fall into the same partition which contains all nodes. Jobs submitted to your priority partition will requeue any job running on the node you need in a lower priority partition. This means that even your own jobs, if running in the requeue partition, are subject to being requeued by your higher priority job. This also means that other users with access to your priority partition may submit jobs that will compete with yours for resources, but not bump yours into requeued status. If you submit your job to your priority partition it will run to completion, failure, or until it runs through the entire execution time you've given it.

If you are a researcher who has purchased an allocation of CPU hours you will run on all nodes at the same priority as the students. Your job will not run on any dedicated nodes and will be susceptible to preemption by any other user unless you submit it to the non-dedicated pool of nodes.

In all publications or products resulting from work performed using the Foundry the NSF Grant which provided funding for the Foundry must be acknowledged. This can be done by adding this sentence to the publication “This work was supported in part by the National Science Foundation under Grant No. OAC-1919789.” Or something to this effect.

**Partitions**

The Hardware in the Foundry is split up into separate groups, or partitions. Some hardware is in more than one partition, if you do not define which partition to use, it will fall into the default partition requeue. However there are a few cases that you will want to assign a job to a specific partition. Please see the table below for a list of the limits or default values given to jobs based on the partition. The important thing to note is how long you can request your job to run.

<table>
<thead>
<tr>
<th>Partition</th>
<th>Time Limit</th>
<th>Default Memory per CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>requeue</td>
<td>7 days</td>
<td>800MB</td>
</tr>
<tr>
<td>general</td>
<td>14 days</td>
<td>800MB</td>
</tr>
<tr>
<td>any priority partition</td>
<td>30 days</td>
<td>varies by hardware</td>
</tr>
</tbody>
</table>

**Quick Start**

We have created a quick start video, it can be found at. https://www.youtube.com/watch?v=AqaRbovceCk&feature=youtu.be

We also have provided written instruction below which you may use for quick reference if needed.
Logging in

SSH (Linux)

Open a terminal and type

```bash
ssh username@foundry.mst.edu
```

replacing username with your campus sso username, Enter your sso password

Logging in places you onto the login node. **Under no circumstances should you run your code on the login node.**

If you are submitting a batch file, then your job will be redirected to a compute node to be computed.

However, if you are attempting use a GUI, ensure that you do not run your session on the login node (Example: username@login-44-0). Use an interactive session to be directed to a compute node to run your software.

```bash
sinteractive
```

For further description of sinteractive, read the section in this documentation titled **Interactive Jobs.**

Putty (Windows)

Open Putty and connect to foundry.mst.edu using your campus SSO.
Off Campus Logins

Our off campus logins use public key authentication only, password authentication is disabled for off campus users unless they are connected to the campus VPN. To learn how to connect from off campus please see our how to on setting up public key authentication. After setting up your public key, you may still use the host foundry.mst.edu to connect without using the VPN.

Submitting a job

Using SLURM, you need to create a submission script to execute on the backend nodes, then use a command line utility to submit the script to the resource manager. See the file contents of a general submission script complete with comments.

Example Job Script

```
#!/bin/bash
#SBATCH --job-name=Change_ME
#SBATCH --ntasks=1
#SBATCH --time=0-00:10:00
#SBATCH --mail-type=begin,end,fail,requeue
#SBATCH --export=all
#SBATCH --out=Foundry-%j.out

# %j will substitute to the job's id
#now run your executables just like you would in a shell script, Slurm will set the working directory as the directory the job was submitted from.
#e.g. if you submitted from /home/blspcy/softwaretesting your job would run in that directory.

#echo "this is a general submission script"
#echo "I've submitted my first batch job successfully"
```

Now you need to submit that batch file to the scheduler so that it will run when it is time.

```
sbatch batch.sub
```

You will see the output of sbatch after the job submission that will give you the job number, if you would like to monitor the status of your jobs you may do so with the squeue command.

Common SBATCH Directives
<table>
<thead>
<tr>
<th>Directive</th>
<th>Valid Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-job-name=</td>
<td>string value no spaces</td>
<td>Sets the job name to something more friendly, useful when examining the queue.</td>
</tr>
<tr>
<td>-ntasks=</td>
<td>integer value</td>
<td>Sets the requested CPUS for the job</td>
</tr>
<tr>
<td>-nodes=</td>
<td>integer value</td>
<td>Sets the number of nodes you wish to use, useful if you want all your tasks to land on one node.</td>
</tr>
<tr>
<td>-time=</td>
<td>D-HH:MM:SS, HH:MM:SS</td>
<td>Sets the allowed run time for the job, accepted formats are listed in the valid values column.</td>
</tr>
<tr>
<td>-mail-type=</td>
<td>begin,end,fail,requeue</td>
<td>Sets when you would like the scheduler to notify you about a job running. By default no email is sent.</td>
</tr>
<tr>
<td>-mail-user=</td>
<td>email address</td>
<td>Sets the mailto address for this job</td>
</tr>
<tr>
<td>-export=</td>
<td>ALL, or specific variable names</td>
<td>By default Slurm exports the current environment variables so all loaded modules will be passed to the environment of the job.</td>
</tr>
<tr>
<td>-mem=</td>
<td>integer value</td>
<td>number in MB of memory you would like the job to have access to, each queue has default memory per CPU values set so unless your executable runs out of memory you will likely not need to use this directive.</td>
</tr>
<tr>
<td>-mem-per-cpu=</td>
<td>integer</td>
<td>Number in MB of memory you want per cpu, default values vary by queue but are typically greater than 1000Mb.</td>
</tr>
<tr>
<td>-nice=</td>
<td>integer</td>
<td>Allows you to lower a jobs priority if you would like other jobs set to a higher priority in the queue, the higher the nice number the lower the priority.</td>
</tr>
<tr>
<td>-constraint=</td>
<td>please see sbatch man page for usage</td>
<td>Used only if you want to constrain your job to only run on resources with specific features, please see the next table for a list of valid features to request constraints on.</td>
</tr>
<tr>
<td>-gres=</td>
<td>name:count</td>
<td>Allows the user to reserve additional resources on the node, specifically for our cluster gpus. e.g. -gres=gpu:2 will reserve 2 gpus on a gpu enabled node.</td>
</tr>
<tr>
<td>-p</td>
<td>partition_name</td>
<td>Not typically used, if not defined jobs get routed to the highest priority partition your user has permission to use. If you were wanting to specifically use a lower priority partition because of higher resource availability you may do so.</td>
</tr>
</tbody>
</table>

### Valid Constraints

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intel</td>
<td>Node has intel CPUs</td>
</tr>
<tr>
<td>amd</td>
<td>Node has amd CPUs</td>
</tr>
<tr>
<td>EDR</td>
<td>Node has an EDR (100Gbit/sec) infiniband interconnect</td>
</tr>
<tr>
<td>FDR</td>
<td>Node has a FDR (56Gbit/sec) infiniband interconnect</td>
</tr>
<tr>
<td>QDR</td>
<td>Node has a QDR (36Gbit/sec) infiniband interconnect</td>
</tr>
<tr>
<td>DDR</td>
<td>Node has a DDR (16Gbit/sec) Infiniband interconnect</td>
</tr>
<tr>
<td>serial</td>
<td>Node has no high speed interconnect</td>
</tr>
<tr>
<td>gpu</td>
<td>Node has GPU acceleration capabilities</td>
</tr>
<tr>
<td>cpucodename*</td>
<td>Node is running the codename of cpu you desire e.g. rome</td>
</tr>
</tbody>
</table>
Note if some combination of your constraints and requested resources is unfillable you will get a submission error when you attempt to submit your job.

**Monitoring your jobs**

```
squeue -u username
```

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>STATE</th>
<th>TIME</th>
<th>CPUS</th>
<th>NODES</th>
</tr>
</thead>
<tbody>
<tr>
<td>719</td>
<td>requeue</td>
<td>Submiss</td>
<td>blspcy</td>
<td>RUNNING</td>
<td>00:01</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Cancel your job**

scancel - Command to cancel a job, user must own the job being cancelled or must be root.

```
scancel <jobnumber>
```

**Viewing your results**

Output from your submission will go into an output file in the submission directory, this will either be slurm-jobnumber.out or whatever you defined in your submission script. In our example script we set this to Foundry-jobnumber.out, this file is written asynchronously so it may take a bit after the job is complete for the file to show up if it is a very short job.

**Moving Data**

Moving data in and out of the Foundry can be done with a few different tools depending on your operating system and preference.

**Globus**

The Foundry has a globus endpoint configured, which will allow you access to move data in and out if you sign in to https://app.globus.org using the UMSystem identity provider and your UMSystem account.

From sign in you will need to find the end point you are going to move data to/from. If you are moving data from global endpoint to global endpoint, e.g. Forge to Foundry, you will need to find both of them using the search tool. They are named appropriately for you to find them easily.

Once you have connected your account with these endpoints/collections you may move data from one to the other and vice versa using the globus web page.

You can install Globus software to create a personal endpoint on any number of your personal devices to move data back and forth from them to The Foundry as well.

Predrag has made a short video on using globus if you'd like to get a better idea of how this all looks. https://youtu.be/fOfZJncPqx0
DFS volumes

Missouri S&T users can mount their web volumes and S Drives with the `mountdfs` command. This will mount your user directories to the login machine under `/mnt/dfs/$USER`. The data can be copied over with command line tools to your home directory, **your data will not be accessible from the compute nodes so do not submit jobs from these directories.** Aliases “cds” and “cdwww” have been created to allow you to cd into your s drive and web volume quickly and easily.

Windows

WinSCP

Using WinSCP connect to foundry.mst.edu using your SSO just as you would with ssh or putty and you will be presented with the contents of your home directory. Now you will be able to drag files into the winscp window and drop them in the folder you want them in and the copying process should begin. It should also work the same in the opposite direction to get data back out.

Filezilla

Using Filezilla you connect to foundry.mst.edu using your SSO and you will have the contents of your home directory displayed, drag and drop works with Filezilla as well.

Git

Git is installed on the cluster and is recommended to keep track of code changes across your research. See [getting started with git](https://git.mst.edu) for usage guides, Campus offers a hosted private git server at [https://git.mst.edu](https://git.mst.edu) at no additional cost.

Linux

Filezilla

See windows instructions

scp

scp is a command line utility that allows for secure copies from one machine to another through ssh, scp is available on most Linux distributions. If I wanted to copy a file in using scp I would open a
scp /home/blspcy/batch.sub blspcy@foundry.mst.edu:/home/blspcy/batch.sub

It will then ask me to authenticate using my campus SSO, then copy the file from my local location of /home/blspcy/batch.sub to my foundry home directory. If you have questions on how to use scp I recommend reading the man page for scp, or check it out online at SCP man page

rsync

rsync is a more powerful command line utility than scp, it has a simpler syntax, and checks to see if the file has actually changed before performing the copy. See the man page for usage details or online documentation

git

See git for windows for instruction, it works the same way.

Modules

An important concept for running on the cluster is modules. Unlike a traditional computer where you can run every program from the command line after installing it, with the cluster we install the programs to a main “repository” so to speak, and then you load only the ones you need as modules. To see which modules are available for you to use you would type “module avail”. Once you find which module you need for your file, you type “module load <module>” where <module> is the module you found you wanted from the module avail list. You can see which modules you already have loaded by typing “module list”.

Here is the output of module avail as of 03/18/2020

```bash
blspcy@login-14-42:/share/apps/modulefiles/common/mpich/3.3.2$ module avail

-------------------------------------------------------------------------
/usr/share/lmod/lmod/modulefiles

-------------------------------------------------------------------------
Core/lmod/6.6   Core/settarg/6.6

-------------------------------------------------------------------------
/share/apps/modulefiles/common

-------------------------------------------------------------------------
R/3.6.3         intelmpi/2020.0       mpich/3.3.2/intel/2020.0
ansys/2019r2    lammmps/03Mar2020   openmpi/3.1.5/gnu/9.2.0
CST/2020        matlab/2019b        openmpi/3.1.5/intel/2020.0
gnu/9.2.0       molpro/2019.2.3    openmpi/4.0.3/gnu/9.2.0
hpl/2.3         moose/1.1          openmpi/4.0.3/intel/2020.0
intel/2020.0    mpich/3.3.2/gnu/9.2.0 quantum-espresso/6.5

Use "module spider" to find all possible modules.
Use "module keyword key1 key2 ..." to search for all possible modules
```
matching any of the "keys".

**Compiling Code**

There are several compilers available through modules, to see a full list of modules run

```
module avail
```

the naming scheme for the compiler modules are as follows.

```
MPI_PROTOCOL/MPI_VERSION/COMPILER/COMPILER_VERSION e.g openmpi/3.1.4/gnu/9.2.0 is the openmpi libraries version 3.1.4 built with the gnu compiler version 9.2.0. All mpi libraries are set to communicate over the high speed infiniband interface. The exception to this rule of naming is the intelmpi, which is just intelmpi/INTEL_VERSION since it is installed only with the intel compiler.
```

After you have decided which compiler you want to use you need to load it.

```
module load openmpi/4.0.3/gnu/9.2.0
```

Then compile your code, use mpicc for c code and mpif90 for fortran code. Here is an MPI hello world C code.

```
helloworld.c

/* C Example */
#include <stdio.h>
#include <mpi.h>

int main (argc, argv)
   int argc;
   char *argv[];
{
   int rank, size;

   MPI_Init (&argc, &argv); /* starts MPI */
   MPI_Comm_rank (MPI_COMM_WORLD, &rank); /* get current process id */
   MPI_Comm_size (MPI_COMM_WORLD, &size); /* get number of processes */
   printf("Hello world from process %d of %d\n", rank, size);
   MPI_Finalize();
return 0;
}
```

Use mpicc to compile it.
mpicc ./helloworld.c

Now you should see a a.out executable in your current working directory, this is your mpi compiled code that we will run when we submit it as a job.

**IMPORTANT NOTE!!**

The openmpi based mpi libraries will throw errors when you use mpirun on your compiled code about not being able to initialize the fabric. These errors are *false errors*, your job is running, and running fine. The error is a bug introduced by having the newest infiniband cards, which haven't been fully accounted for in the openmpi's code base. We have done several tests and have determined that even though the job throws this error it is indeed communicating with other mpi launched processes in the job, and the job will run to completion without further errors caused by openmpi.

### Submitting an MPI job

You need to be sure that you have the same module loaded in your job environment as you did when you compiled the code to ensure that the compiled executables will run correctly, you may either load them before submitting a job and use the directive

```bash
#SBATCH --export=all
```

in your submission script, or load the module prior to running your executable in your submission script. Please see the sample submission script below for an mpi job.

**helloworld.sub**

```bash
#!/bin/bash
#SBATCH -J MPI_HELLO
#SBATCH --ntasks=8
#SBATCH --export=all
#SBATCH --out=Foundry-%j.out
#SBATCH --time=0-00:10:00
#SBATCH --mail-type=begin,end,fail,requeue
module load openmpi/4.0.2/gnu/9.2.0
mpirun ./a.out
```

Now we need to submit that file to the scheduler to be put into the queue.

```bash
sbatch helloworld.sub
```

You should see the scheduler report back what job number your job was assigned just as before, and you should shortly see an output file in the directory you submitted your job from.

### Interactive jobs
Some things can't be run with a batch script because they require user input, or you need to compile some large code and are worried about bogging down the login node. To start an interactive job simply use the

```
sinteractive
```

command and your terminal will now be running on one of the compute nodes. The hostname command can help you confirm you are no longer running on a login node. Now you may run your executable by hand without worrying about impacting other users. The sinteractive script by default will allot a 1 cpu for whatever the default time is of the partition you submit against (for requeue that is 10 minutes). You may request more by using SBATCH directives, e.g.

```
sinteractive --time=02:00:00 --ntasks=2 --nodes=1
```

will start a job with 2 CPUs on one node for 2 hours. You can still run mpi based jobs inside the interactive job and it will pick up your job's resources and use them just like it would in a batch job.

If you will need a GUI Window for whatever you are running inside the interactive job you will need to connect to The Foundry with X forwarding enabled. For Linux this is simply adding the -X switch to the ssh command.

```
ssh foundry.mst.edu -X
```

For Windows there are a couple X server software's available for use, x-ming and x-win32 that can be configured with putty. Here is a simple guide for configuring putty to use xming.

### Job Arrays

If you have a large number of jobs you need to start I recommend becoming familiar with using job arrays, basically it allows you to submit one job file to start up to 10000 jobs at once.

One of the ways you can vary the input of the job array from task to task is to set a variable based on which array id the job is and then use that value to read the matching line of a file. For instance the following line when put into a script will set the variable PARAMETERS to the matching line of the file data.dat in the submission directory.

```
PARAMETERS=$(awk -v line=${SLURM_ARRAY_TASK_ID} '{if (NR == line) { print $0; };};'} ./data.dat)
```

You can then use this variable in your execution line to do whatever you would like to do, you just have to have the appropriate data in the data.dat file on the appropriate lines for the array you are submitting. See the sample data.dat file below.

```
data.dat

"I am line number 1"
"I am line number 2"
"I am line number 3"
```
you can then submit your job as an array by using the \-array directive, either in the job file or as an argument at submission time, see the example below.

```
array_test.sub

#!/bin/bash
#SBATCH -J Array_test
#SBATCH --ntasks=1
#SBATCH --out=Foundry-%j.out
#SBATCH --time=00:10:00
#SBATCH --mail-type=begin,end,fail,requeue

PARAMETERS=$(/bin/awk -v line=${SLURM_ARRAY_TASK_ID} '{if (NR == line) {print $0; };}') ./data.dat

echo $PARAMETERS
```

I prefer to use the array as an argument at submission time so I don't have to touch my submission file again, just the data.dat file that it reads from.

```
sbatch --array=1-2,4 array_test.sub
```

Will execute lines 1, 2, and 4 of data.dat which echo out what line number they are from my data.dat file.

You may also add this as a directive in your submission file and submit without any switches as normal. Adding the following line to the header of the submission file above will accomplish the same thing as supplying the array values at submission time.

```
#SBATCH --array=1-2,4
```

Then you may submit it as normal

```
sbatch array_test.sub
```

**Checking your account usage**

If you have purchased a number of CPU hours from us you may check on how many hours you have used by issuing the

```
usereport
```

command from a login node, this will show your account's CPU hour limit and the total amount used.
Note this is usage for your account, not your user.

Applications

The applications portion of this wiki is currently a Work in progress, not all applications are currently here, nor will they ever be as the applications we support continually grows.

Anaconda

If you would like to install python modules via conda, you may load the anaconda module to get access to conda for this purpose. After loading the module you will need to initialize conda to work with your shell.

```
module load anaconda
conda init
```

This will ask you what shell you are using, and after it is done it will ask you to log out and back in again to load the conda environment. After you log back in your command prompt will look different than it did before. It should now have (base) on the far left of your prompt. This is the virtual environment you are currently in. Since you do not have permissions to modify base, you will need to create and activate your own virtual environment to build your software inside of.

```
conda create --name myenv
conda activate myenv
```

Now instead of (base) it should say (myenv) or whatever you have named your environment in the create step. These environments are stored in your home directory so they are unique to you. If you are working together with a group, everyone in your group will either need a copy of the environment you've built in $HOME/.conda/envs/

Once you are inside your virtual environment you can run whatever conda installs you would like and it will install them and dependencies inside this environment. If you would like to execute code that depends on the modules you install you will need to be sure that you are inside your virtual environment. (myenv) should be shown on your command prompt, if it is not, activate it with `conda activate`.

Ansys

- Default Version = 2019r2
- Other versions available: none yet

Running the Workbench

Be sure you are connected to the Foundry with X forwarding enabled, and running inside an interactive job using command
sinteractive

before you attempt to launch the workbench. Running sinteractive without any switches will give you 1 cpu for 1 hour, if you need more time or resources you may request it. See Interactive Jobs for more information.

Once inside an interactive job you need to load the ansys module.

module load ansys

Now you may run the workbench.

runwb2

Job Submission Information

Fluent is the primary tool in the Ansys suite of software used on the Foundry. Most of the fluent simulation creation process is done on your Windows or Linux workstation. The 'Solving' portion of a simulation is where the Foundry is utilized. Fluent will output a lengthy file, based on the simulation being run and that lengthy output file would be used on your Windows or Linux Workstation to do the final review and analysis of your simulation.

The basic steps

1. Create your geometry
2. Setup your mesh
3. Setup your solving method
4. Use the .cas and .dat files, generated from the first three steps, to construct your jobfile
5. Copy those files to the Foundry, to your home folder
6. Create your jobfile using the slurm tools on the Foundry Documentation page
7. Load the Ansys module
8. Submit your newly created jobfile with sbatch

Serial Example.

I used the Turbulent Flow example from Cornell's SimCafe. On the Foundry, I have this directory structure for this example. Please create your own structure that makes sense to you.

TurbulentFlow/
|-- flntgz-48243.cas
|-- flntgz-48243.dat
|-- output.dat
|-- slurm-8731.out
|-- TurbulentFlow_command.txt
|-- TurbulentFlow.sbatch

The .cas file is the CASE file that contains the parameters define by you when creating the model.
The .dat file is the data result file used when running the simulation.
The .txt file, is the actual, command equivalent, of your model, in a form that the Foundry
understands.
The .sbatch file, is the slurm job file that you will use to submit your model for analysis.
The .out file is the output from the run.
The .dat file is the binary (ansys specific) file created during the solution, that could be imported into
Ansys back on the Windows/Linux workstation for further analysis.

**Jobfile Example.**

TurbulentFlow.sbatch

```bash
#!/bin/bash
#SBATCH --job-name=TurbulentFlow.sbatch
#SBATCH --ntasks=1
#SBATCH --nodes=1
#SBATCH --time=01:00:00
#SBATCH -o foundry-%j.out
fluent 2ddp
```

The SBATCH commands are explained in the Foundry Documentation.

The job-name is a name given to help you determine which job is which.
This job will be in the — **partition=requeue** queue.
It will use 1 node — **nodes=1**.
It will use 4 processors in one node — **ntasks=4**.
It has a wall clock time of 1 hour — **time=01:00:00**.
It will email the user when it begins, ends, or if it fails. **-mail-type & -mail-user**

`fluent` is the command we are going to run.

**2ddp** is the mode we want fluent to use

* **Modes** The `[mode]` option must be supplied and is one of the following:
  * 2d runs the two-dimensional, single-precision solver
  * 3d runs the three-dimensional, single-precision solver
  * 2ddp runs the two-dimensional, double-precision solver
  * 3ddp runs the three-dimensional, double-precision solver

`-g` turns off the GUI
Path to the command file we are calling in fluent. `<

/home/rlhaffer/unittests/ANSYS/TurbulentFlow/TurbulentFlow_command.txt

Contents of command file
This file can get long. As it contains the .cas file & .dat file information as well as saving frequency
and iteration count

**NOTE**, this is all in one line when creating the command file
When the simulation is finished, you will have a foundry-#####.out file that looks something like this:

```
/share/apps/ansys_inc/v150/fluent/fluent15.0.7/bin/fluent -r15.0.7 2ddp -g
/share/apps/ansys_inc/v150/fluent/fluent15.0.7/cortex/lnamd64/cortex.15.0.7
-f fluent -g (fluent "2ddp -alnamd64 -r15.0.7 -
path/share/apps/ansys_inc/v150/fluent")
Loading
"/share/apps/ansys_inc/v150/fluent/fluent15.0.7/lib/fluent.dmp.114-64"
Done.
/share/apps/ansys_inc/v150/fluent/fluent15.0.7/bin/fluent -r15.0.7 2ddp -
alnamd64 -path/share/apps/ansys_inc/v150/fluent -cx
edrcompute-43-17.local:56955:53521
Starting
/share/apps/ansys_inc/v150/fluent/fluent15.0.7/lnamd64/2ddp/fluent.15.0.7 -
cx edrcompute-43-17.local:56955:53521

Welcome to ANSYS Fluent 15.0.7
Copyright 2014 ANSYS, Inc.. All Rights Reserved.
Unauthorized use, distribution or duplication is prohibited.
This product is subject to U.S. laws governing export and re-export.
For full Legal Notice, see documentation.

Build Time: Apr 29 2014 13:56:31 EDT  Build Id: 10581

Loading
"/share/apps/ansys_inc/v150/fluent/fluent15.0.7/lib/flprim.dmp.1119-64"
Done.

--------------------------------------------------------------
This is an academic version of ANSYS FLUENT. Usage of this product license is limited to the terms and conditions specified in your ANSYS license form, additional terms section.
--------------------------------------------------------------

Cleanup script file is /home/rlhaffer/unittests/ANSYS/TurbulentFlow/cleanup-fluent-edrcompute-43-17.local-17945.sh

> Reading "/home/rlhaffer/unittests/ANSYS/TurbulentFlow/flntgz-48243.cas"...
  3000 quadrilateral cells, zone 2, binary.
  5870 2D interior faces, zone 1, binary.
  30 2D velocity-inlet faces, zone 5, binary.
  30 2D pressure-outlet faces, zone 6, binary.
  100 2D wall faces, zone 7, binary.
```
100 2D axis faces, zone 8, binary.
3131 nodes, binary.
3131 node flags, binary.

Building...
  mesh
  materials,
  interface,
  domains,
  mixture
  zones,
  pipewall
  outlet
  inlet
  interior-surface_body
  centerline
  surface_body
Done.

Reading "/home/rlhaffer/unittests/ANSYS/TurbulentFlow/flntgz-48243.dat"...
Done.

<table>
<thead>
<tr>
<th>iter</th>
<th>continuity</th>
<th>x-velocity</th>
<th>y-velocity</th>
<th>k</th>
<th>epsilon</th>
<th>time/iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>389</td>
<td>9.7717e-07</td>
<td>1.0711e-07</td>
<td>2.9115e-10</td>
<td>5.2917e-08</td>
<td>3.4788e-07</td>
<td>0:00:00</td>
</tr>
</tbody>
</table>

Writing "/home/rlhaffer/unittests/ANSYS/TurbulentFlow/output.dat"...
Done.

**Parallel Example**

To use fluent in parallel please you need set the PBS_NODEFILE envrionment variable inside your job.
Please see example submission file below.

**TurbulentFlow.sbatch**

```bash
#!/bin/bash

#SBATCH --job-name=TurbulentFlow.sbatch
#SBATCH --ntasks=32
#SBATCH --time=01:00:00
#SBATCH -o foundry-%j.out

#generate a node file
export PBS_NODEFILE=`generate_pbs_nodefile`
```
Interactive Fluent

If you would like to run the full GUI you may do so inside an interactive job, make sure you’ve connected to The Foundry with X Forwarding enabled. Start the job with:

```
sinteractive
```

This will give you 1 processor for 1 hour, to request more processors or more time please see the documentation at [Interactive Jobs](#). Once inside the interactive job you will need to load the ansys module.

```
module load ansys
```

Then you may start fluent from the command line.

```
fluent 2ddp
```

will start the 2d, double precision version of fluent. If you’ve requested more than one processor you need to first run

```
export PBS_NODEFILE=`generate_pbs_nodefile`
```

Then you need to add some switches to fluent to get it to use those processors.

```
fluent 2ddp -t## -pethernet -cnf=$PBS_NODEFILE -ssh
```

You need to replace the ## with the number of processors requested.

Comsol

Comsol Multiphysics is available for general usage through a comsol/5.6_gen module. Please see the sample submission script below for running comsol in parallel on the Foundry.

```
#!/bin/bash
#SBATCH -J Comsol_job
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=64
#SBATCH --mem=0
#SBATCH --time=1-00:00:00
```

Cuda

Our login nodes don't have the CUDA toolkit installed so to compile your code you will need to start an interactive job on these nodes to do your compilation.

```bash
sinteractive -p cuda --time=01:00:00 --gres=gpu:1
```

This interactive session will start on a cuda node and give you access to one of the GPUs on the node, once started you may compile your code and do whatever testing you need to do inside this interactive session.

To submit a job for batch processing please see this example submission file below.

```bash
#!/bin/bash
#SBATCH -J Cuda_Job
#SBATCH -p cuda
#SBATCH -o Forge-%j.out
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --gres=gpu:1
#SBATCH --time=01:00:00
./a.out
```

This file requests 1 cpu and 1 gpu on 1 node for 1 hour, to request more cpus or more gpus you will need to modify the values related to ntasks and gres=gpu. It is recommended that you at least have 1 cpu for each gpu you intend to use, we currently only have 2 gpus available per node. Once we incorporate the remainder of the GPU nodes we will have 7 gpus available in one chassis.

Gaussian

Gaussian has 2 different versions on the Foundry, the sample submission file below uses the g09 executable however if you load the version 16 module you will need to use g16.
#!/bin/bash
#SBATCH --job-name=gaussian
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=1000
module load gaussian/09e01
$09 < Fe_C05.inp

You will need to replace the file name of the input file in the sample provided with your own.

Matlab

IMPORTANT NOTE Currently campus has 100 Matlab seat licenses to be shared between the Foundry and research desktops. There are certain times of the year where Matlab usage is quite high. License check out is on a first come, first served basis. If you are not able to get a Matlab license, you might consider using GNU Octave. This is available on the Foundry and will do much of what Matlab will do.

Matlab is available to run in batch form or interactively on the cluster.

- Default version = 2021a
- Other installed version(s): 2019b, 2020a, 2020b (run “module avail” to see what versions are currently available)

Interactive Matlab

To get started with Matlab, run the following sequence of commands from the login node. This will start an interactive job on a backend node, load the default module for Matlab, and then launch Matlab. If you have connected with X forwarding, you will get the full Matlab GUI to use however you would like. By default, this limits you to 1 core for 4 hours maximum on one of our compute nodes. To use more than 1 core, or to run for longer than 4 hours, you will need to either add additional parameters to the “sinteractive” command or submit a batch submission job that configures all of the job parameters you require.

sinteractive
module load matlab
matlab

Please note that by default Matlab does not parallelize your code so unless you use parallelized calls in your code. If you have parallelized your code you will need to first open a parallel pool to run your code in.

Batch Submit

If you want to use Batch Submissions for Matlab you will need to create a submission script similar to
the ones above in quick start, but you will want to limit the nodes your job runs on 1, please see the sample submission script below.

matlab.sub

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --ntasks=12
#SBATCH -J Matlab_job
#SBATCH -o Foundry-%j.out
#SBATCH --time=01:00:00
#sbatch --mem-per-cpu=4000

module load matlab
matlab < helloworld.m
```

This submission asks for 12 processors on 1 node for an hour, the maximum per node we currently have is 64. Without using the distributive computing engine, which is outside the scope of this tutorial, you will only be able to use 64 processors in a 'local' parallel pool.

To make use of this new found power you must implement opening a parallel pool

```
parpool('local',12)
```

in your matlab code then run it using either the interactive method or batch submit. The specific parpool command opens a local pool with 12 matlab workers.

**Python**

Python versions 2.7.17 and 3.6.9 are available, and users may install python modules for themselves via the pip modules available in python. Please note that the pip and pip3 commands are links to old wrapper scripts which come packaged in the OS and may not be able to install the newest version of whatever module you are trying to install. Because of this I will include instructions for how to use the pip utilities and also how to upgrade them for your user and use them with the new syntax.

For the old standard pip and pip3 utilities you would simply call them from the command line to install, uninstall, search for, or list installed modules. In the following examples you may swap pip with pip3 and it will perform operations on python3 instead of python2.

```
pip list #This lists all available modules and their versions.
pip install --user numpy #This will install the newest available version of the numpy module for your user.
pip uninstall --user numpy #This will uninstall the numpy module for your user.
pip install --user --upgrade numpy==1.18.5 #This will uninstall the old version and install the specified version of numpy.
pip search numpy #This will perform a search for all python modules that
```
you could install that match the search term numpy.

Again, using pip3 the syntax is all the same but it will install modules for python3 instead.

Now to get the newest version of pip or pip3 you will need to run pip the new way and have it perform the upgrade on itself. Just like pip and pip3 were interchangeable in the examples above, in the following examples python and python3 will be interchangeable as well. python will perform operations on version 2, and python3 will perform operations on version 3.

```
python -m pip install --upgrade --user pip #This will upgrade pip to the newest version for your user.
python -m pip install --user numpy #This will install the newest available version of the numpy module for your user.
```

All of the syntax for pip is the same after calling python -m pip as it was for the pip and pip3 wrapper scripts. Also if you upgrade pip, you must use the new method of pip installing modules unless you uninstall your user's pip module.

If your user's python environment gets broken by pip installing anything for your user, you may start over by removing or moving the `~/.local/` folder in your home directory. This is the location the python stores all your modules and python environments by default and is the location which gets checked when trying to install/uninstall/list modules.

**Singularity**

With the Foundry we've introduced the ability to build your own software in a singularity container, or use publicly available containers inside a Foundry job. Please keep in mind that you will need to still abide by the rules of running either through interactive jobs, or through batch submissions.

Singularity does not automatically create a job environment for you, it, like most other executables runs where it is called from.

No module is needed to call singularity, and the singularity container should have all needed libraries in it for the software you are trying to run as it adopts the container's environment for many of the environment modules that are modified by our modules. However Slurm environment variables are passed in, so you would be able to run any mpi based software inside a container which would rely on these variables to set up the mpi run.

I highly suggest reading the documents on singularity at https://sylabs.io/guides/3.5/user-guide/ and understanding what you need to do to create the environment your software needs.

An important thing to understand is that you may create these containers anywhere and then move them into the Foundry for execution which would give you the ability to configure the container as you see fit on your own computer where you have administrative privileges and execute your code as a user on the Foundry.

Running interactive in a singularity container inside an interactive session would look like the following set of commands.

```
sinteractive
singularity shell library://centos
```
The first puts you interactively on a compute node. The second loads a singularity shell on the remote image from the singularity library. The singularity command has a large amount of help built into the command. I suggest starting with.

```bash
singularity --help
```

Which will give you a list of commands available and a description of what each could do for you. For example, if you know what command you need to run inside the container you don’t need to drop into the shell of the container, you can simply run the command.

```bash
singularity exec library://centos pwd
```

Which will run the command `pwd` inside the container.

Another thing to note is the flexibility of singularity. It can run containers from its own library, docker, dockerhub, singularityhub, or a local file in many formats.

**StarCCM+**

Engineering Simulation Software

Default version = 2021.2

Other working versions:

- 2020.1
- 12.02.010

Job Submission Information

Copy your `.sim` file from the workstation to your cluster home profile. Once copied, create your job file.

Example job file:

```bash
#!/bin/bash
#SBATCH --job-name=starccm_test
#SBATCH --nodes=1
#SBATCH --ntasks=12
#SBATCH --mem=40000
#SBATCH --partition=requeue
#SBATCH --time=12:00:00
#SBATCH --mail-type=BEGIN
#SBATCH --mail-type=FAIL
#SBATCH --mail-type=END
#SBATCH --mail-user=username@mst.edu
module load starccm/2021.2
```
It's preferred that you keep the ntasks and -np set to the same processor count.

Breakdown of the script:
This job will use 1 node, asking for 12 processors, 40,000 MB of memory for a total wall time of 12 hours and will email you when the job starts, finishes or fails.

The StarCCM commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-batch</td>
<td>tells Star to utilize more than one processor</td>
</tr>
<tr>
<td>-np</td>
<td>number of processors to allocate</td>
</tr>
<tr>
<td>/path/to/your/starccm/simulation/example.sim</td>
<td>use the true path to your .sim file in your cluster home directory</td>
</tr>
</tbody>
</table>

From: https://wiki.mst.edu/itrst/ - IT Research Support Team
Permanent link: https://wiki.mst.edu/itrst/pub/foundry
Last update: 2022/02/02 12:56