MILA Docs Documentation

Release latest

Jun 16, 2021

CONTENTS

1 Purpose of this documentation **3**
- 1.1 Intended audience . . . 3
- 1.2 Contributing . . . 3

2 General Cluster theory **7**
- 2.1 What is a computer Cluster ? . . . 7
- 2.2 Parts of a computing cluster . . . 7
- 2.3 UNIX . . . 9
- 2.4 The batch scheduler . . . 9
- 2.5 Processing data . . . 10
- 2.6 Software dependency management and associated challenges . . . 10

3 Mila research computing infrastructure information and policies **13**
- 3.1 Roles and authorisations . . . 13
- 3.2 Overview of available computing resources at Mila . . . 13
- 3.3 Node profile description . . . 13
- 3.4 Data Sharing Policies . . . 14
- 3.5 Monitoring . . . 15
- 3.6 Storage . . . 18

4 1 Users Guide **21**
- 4.1 1.1 Welcome to the machine; Logging in to the cluster . . . 21
- 4.2 1.2 Running your code . . . 22
- 4.3 1.3 Portability concerns and solutions . . . 27
- 4.4 1.4 Using containers . . . 40
- 4.5 1.5 Contributing datasets . . . 42
- 4.6 1.6 Notebooks . . . 44
- 4.7 1.7 Advanced SLURM usage and Multiple GPU jobs . . . 44
- 4.8 1.8 Frequently asked questions (FAQs) . . . 48

5 AI tooling and methodology handbook **51**

6 Computational resources outside of Mila **53**
- 6.1 Compute Canada Clusters . . . 53

7 Audio and video resources at Mila **59**

8 Who, what, where is IDT **61**

Welcome to Mila's technical documentation. See contents below.

CHAPTER

ONE

PURPOSE OF THIS DOCUMENTATION

This documentation aims to cover the information required to run scientific and data-intensive computing tasks at Mila and the available resources for its members.

It also aims to be an outlet for sharing know-how, tips and tricks and examples from the IDT team to the Mila researcher community.

1.1 Intended audience

This documentation is mainly intended for Mila researchers having access to the Mila cluster. This access is determined by your researcher status. See *Roles and authorisations* for more information. The core of the information with this purpose can be found in the following section : *Mila research computing infrastructure information and policies*.

However, we also aim to provide more general information which can be useful outside the scope of using the Mila cluster. For instance, more general theory on computational considerations and such. In this perspective, we hope the documentation can be of use for all of Mila members.

1.2 Contributing

See the following file for contribution guidelines :

```
# Contributing to the Mila Docs

Thank you for your interest into making a better documentation for all at Mila. Here are␣
↪some gidelines to help bring your contribbutions to life.

## What could be included

* Mila cluster usage
* Compute Canada cluster usage
* Job management tips / tricks
* Research good practices
* Software development good practices
* Useful tools

## Issues / Pull Requests

### Issues
```

(continues on next page)

(continued from previous page)

```
Issues can be used to report any error in the documentation, missing or unclear sections,
↪ broken tools or other suggestions to improve the overall documentation.

### Pull Requests

PRs are welcome! Reference the related issues like this:

```
Resolves: #123
See also: #456, #789
```

You can attempt to build the docs yourself to see if the formating is right:

```console
python3 -m pip install -r docs/requirements.txt
sphinx-build -b html docs/ docs/_build/
```

This will produce the html version of the documentation which you can navigate by␣
↪opening `docs/_build/index.html`.

If you have any touble building the docs, don't hesitate to open an issue to request␣
↪help or simply provide the content you would like to add in markdown if that is␣
↪simpler for you.

Sphinx / reStructuredText (reST)

The markup language used for the Mila Docs is [reStructuredText](http://docutils.
↪sourceforge.net/rst.html) and we follow the [Python's Style Guide for␣
↪documenting](https://docs.python.org/devguide/documenting.html#style-guide).

Here are some of reST syntax useful to know (more can be found in [Sphinx's reST␣
↪Primer](https://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html)):

Inline markup

* one asterisk: `*text*` for emphasis (italics),
* two asterisks: `**text**` for strong emphasis (boldface), and
* backquotes: ` ``text`` ` for code samples, and
* external links: `` `Link text <http://target>`_ ``.

Lists

```reST
* this is
* a list

  * with a nested list
  * and some subitems

* and here the parent list continues
```

(continues on next page)
```

(continued from previous page)

````
```

### Sections

```reST
=================
This is a heading
=================
```

There are no heading levels assigned to certain characters as the structure is␣
↪determined from the succession of headings. However, the Python documentation is␣
↪suggesting the following convention:

    * `#` with overline, for parts
    * `*` with overline, for chapters
    * `=`, for sections
    * `-`, for subsections
    * `^`, for subsubsections
    * `"`, for paragraphs

### Note box

```reST
.. note::
   This is a long
   long long note
```
````
````

CHAPTER

TWO

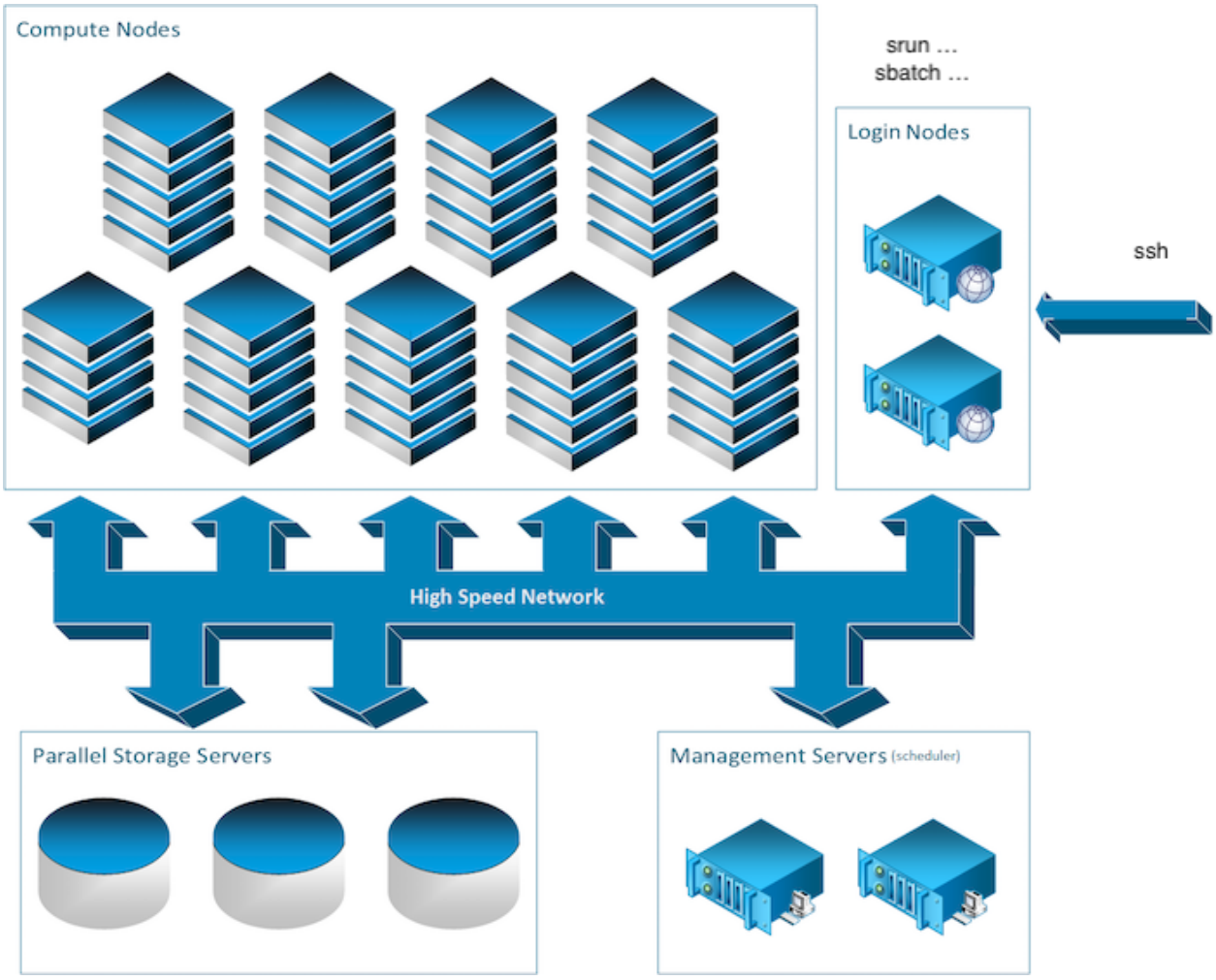
GENERAL CLUSTER THEORY

2.1 What is a computer Cluster ?

> A computer cluster is a set of loosely or tightly connected computers that work together so that, in many respects, they can be viewed as a single system.
>
> Wikipedia

2.2 Parts of a computing cluster

In order to provide high performance computation capabilities, clusters can combine hundreds to thousands of computers, called *nodes*, which are all inter-connected with a high-performance communication network. Most nodes are designed for high-performance computations, but clusters can also use specialized nodes to offer parallel file systems, databases, login nodes and even the cluster scheduling functionality as pictured in the image below.



We will overview the different types of nodes which you can encounter on a typical cluster.

2.2.1 The Login Nodes

To execute computing processes on a cluster, you must first connect to a cluster and this is accomplished through a *login node*. These so-called login nodes are the entry point to most clusters.

Another entry point to some clusters such as the Mila cluster is the JupyterHub WEB interface, but we'll read about that later. For now let's return to the subject of this section; Login nodes. To connect to these, you would typically use a remote shell connection. The most usual tool to do so is SSH. You'll hear and read a lot about this tool. Imagine it as a very long (and somewhat magical) extension cord which connects the computer you are using now, such as your laptop, to a remote computer's terminal shell. You might already know what a terminal shell is if you ever used the command line.

2.2.2 The Compute Nodes

In the field of artificial intelligence, you will usually be on the hunt for GPUs. In most clusters, the compute nodes are the ones with GPU capacity.

While there is a general paradigm to tend towards a homogeneous configuration for nodes, this is not always possible in the field of artificial intelligence as the hardware evolve rapidly as is being complemented by new hardware and so on. Hence, you will often read about computational node classes. Some of which might have different GPU models or even no GPU at all. It is important to keep this in mind as you'll have to be aware of *which* nodes you are working on. More on that later.

2.2.3 The Storage nodes

Some computers on a cluster will have one function only which is to serve files. While the name of these computers might matter to some, as a user, you'll only be concerned about the path to the data. More on that in the *Processing data* section.

2.2.4 Different nodes for different uses

It is important to note here the difference in intended uses between the compute nodes and the login nodes. While the Compute Nodes are meant for heavy computation, the Login Nodes are not.

The login nodes however are used by everyone who uses the cluster and care must be taken not to overburden these nodes. Consequently, only very short and light processes should be run on these otherwise the cluster may become inaccessible. In other words, please refrain from executing long or compute intensive processes on login nodes because it affects all other users. In some cases, you will also find that doing so might get you into trouble.

2.3 UNIX

All clusters typically run on GNU/Linux distributions. Hence a minimum knowledge of GNU/Linux and BASH is usually required to use them. See the following tutorial for a rough guide on getting started with Linux.

2.4 The batch scheduler

Once connected to a login node, presumably with SSH, you can issue a job execution request to what is called the job scheduler. The job scheduler used at Mila and Compute Canada clusters is called SLURM (*slurm*). The job scheduler's main role is to find a place to run your program in what is simply called : *a job*. This "place" is in fact one of many computers synchronised to the scheduler which are called : *Compute Nodes*.

In fact it's a bit trickier than that, but we'll stay at this abstraction level for now.

2.4.1 Slurm

Resource sharing on a supercomputer/cluster is orchestrated by a resource manage/job scheduler. Users submit jobs, which are scheduled and allocated resources (CPU time, memory, GPUs, etc.) by the resource manager, if the resources are available the job can start otherwise it will be placed in queue.

On a cluster, users don't have direct access to the compute nodes but instead connect to a login node to pass the commands they would like to execute in a script for the workload manager to execute.

Mila as well as **Compute Canada** use the workload manager Slurm to schedule and allocate resources on their infrastructure.

Slurm client commands are available on the login nodes for you to submit jobs to the main controller and add your job to the queue. Jobs are of 2 types: *batch* jobs and *interactive* jobs.

For practical examples of SLURM commands on the Mila cluster, see *1.2 Running your code*.

2.5 Processing data

Clusters have different types of file systems to support different data storage use cases. We differentiate them by name. You'll hear or read about file systems such as "home", "scratch" or "project" and so on.

Most of these file systems are are provided in a way which is globally available to all nodes in the cluster. Software or data required by jobs can be accessed from any node on the cluster. (See *Mila* or *CC* for more information on available file systems)

Different file systems have different performance levels. For instance, backed up file-systems (such as `$PROJECT`) provide more space and can handle large files but cannot sustain highly parallel accesses typically required for high speed model training.

Each compute node has local file systems (of which `$SLURM_TMPDIR`) that are usually more efficient but any data remaining on these will be erased at the end of the job execution for the next job to come along.

2.6 Software dependency management and associated challenges

This section aims to raise awareness to problems one can encounter when trying to run a software on different computers and how this is dealt with on typical computation clusters.

2.6.1 Python Virtual environments

TODO

2.6.2 Cluster software modules

Both Mila and Compute Canada clusters provides various software through the `module` command. Modules are small files which modify your environment variables to register the correct location of the software you wish to use. To learn practical examples of module uses, see *1.3.3.1 The module command*.

2.6.3 Containers

Containers are a special form of isolation of software and it's dependencies. It does not only create a separate file system, but can also create a separate network and execution environment. All software you have used for your experiments is packaged inside one file. You simply copy the image of the container you built on every environment without the need to install anything.

CHAPTER

THREE

MILA RESEARCH COMPUTING INFRASTRUCTURE INFORMATION AND POLICIES

This section seeks to provide factual information and policies on the Mila cluster computing environments.

3.1 Roles and authorisations

There are mainly two types of researchers statuses at Mila :

1. Core researchers
2. Affiliated researchers

This is determined by Mila policy. Core researchers have access to the Mila computing cluster. See your supervisor's Mila status to know what is your own status.

3.2 Overview of available computing resources at Mila

The Mila cluster is to be used for regular development and relatively small number of jobs (< 5). It is an heterogeneous cluster. It uses *SLURM* to schedule jobs.

3.3 Node profile description

		12	GPU Arch and Memory

Model #	Model #								
kepler[2-3]	k80	8	16	2	4	2	256	3.6	x86_64 tesla,12GB
kepler4	m40	4	16	2	4	2	256	3.6	x86_64 maxwell,24GB
kepler5	v100	2	m40	1	16	2	4	2	256 3.6 x86_64 volta,12GB
mila01	v100	8	80	2	20	2	512	7	x86_64 tesla,16GB
mila02	v100	8	80	2	20	2	512	7	x86_64 tesla,32GB
mila03	v100	8	80	2	20	2	512	7	x86_64 tesla,32GB
power9[1-2]	v100	4	128	2	16	4	586	0.88	power9 tesla,nvlink,16gb
rtx[6,9]	titanrtx	2	20	1	10	2	128	3.6	x86_64 turing,24gb
rtx[1-5,7-8]	titanrtx	2	20	1	10	2	128	0.93	x86_64 turing,24gb
cn-a[01-11]	rtx8000	8	80	2	20	2	380	3.6	x86_64 turing,48gb
cn-b[01-05]	v100	8	80	2	20	2	380	3.6	x86_64 tesla,nvlink,32gb
cn-c[01-40]	rtx8000	8	64	2	32	1	386	3	x86_64 turing,48gb
cn-d[01-02]	A100	8	256	8	16	2	1032	1.4	x86_64 ampere,40gb

3.3. Node profile description

3.3.1 Special Nodes and outliers

Power9

Power9 servers are using a different processor instruction set than Intel and AMD (x86_64). As such you need to setup your environment again for those nodes specifically.

- Power9 Machines have 128 threads. (2 processors / 16 cores / 4 way SMT)
- 4 x V100 SMX2 (16 GB) with NVLink
- In a Power9 machine GPUs and CPUs communicate with each other using NVLink instead of PCIe.

This allow them to communicate quickly between each other. More on LMS

Power9 have the same software stack as the regular nodes and each software should be included to deploy your environment as on a regular node.

AMD

Warning: As of August 20 the GPUs had to return back to AMD. Mila will get more samples. You can join the amd slack channels to get the latest information

Mila has a few node equipped with MI50 GPUs.

```
$ srun --gres=gpu -c 8 --reservation=AMD --pty bash

# first time setup of AMD stack
$ conda create -n rocm python=3.6
$ conda activate rocm

$ pip install tensorflow-rocm
$ pip install /wheels/pytorch/torch-1.1.0a0+d8b9d32-cp36-cp36m-linux_x86_64.whl
```

3.4 Data Sharing Policies

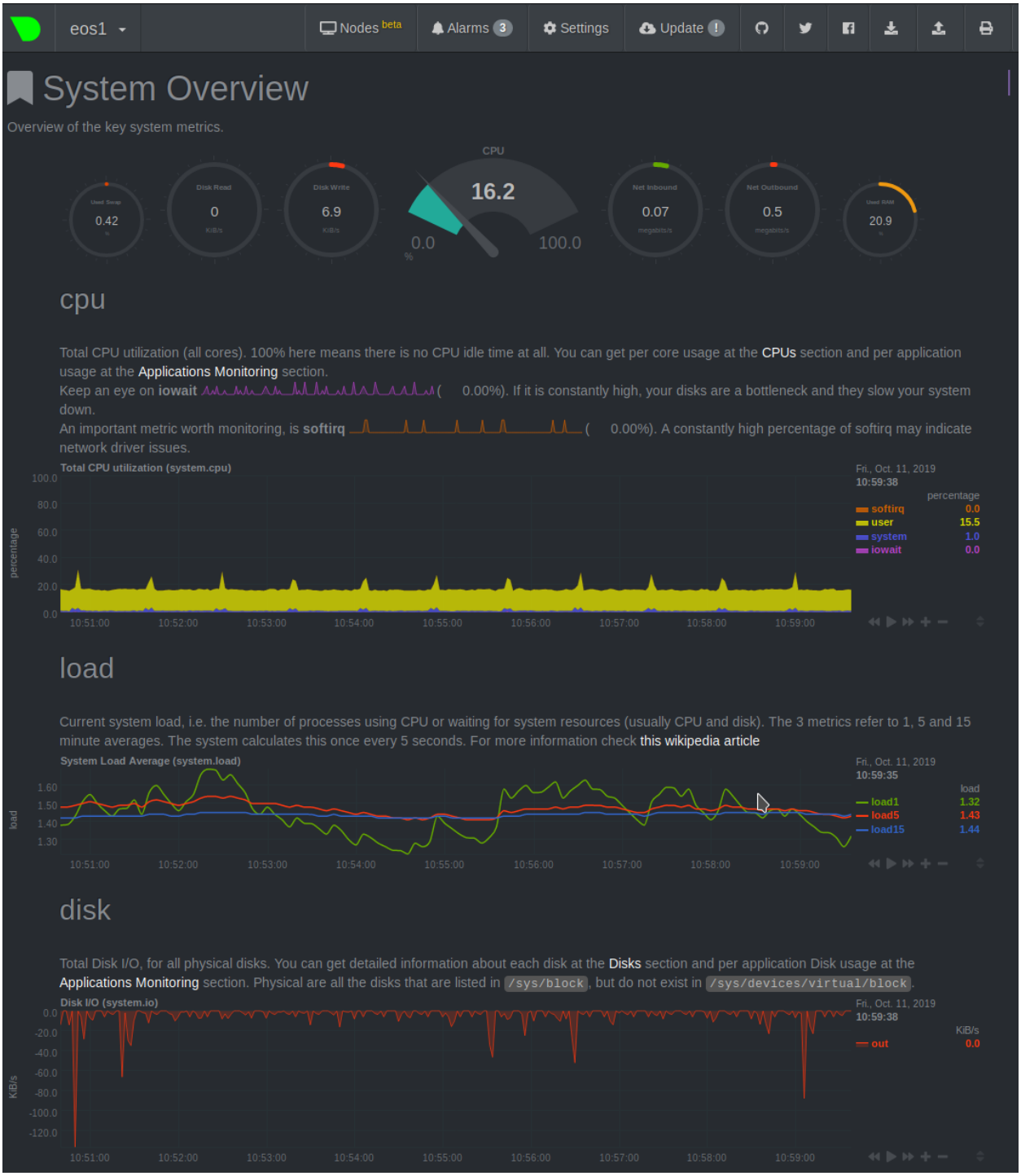
/miniscratch supports ACL to allows collaborative work on rapidly changing data, i.g. work in process datasets, model checkpoints, etc…

/network/projects aims to offer a collaborative space for long-term projects. Data that should be kept for a longer period then 90 days can be stored in that location but first a request to Mila's helpdesk has to be made.

3.5 Monitoring

Every compute node on the Mila cluster has a monitoring daemon allowing you to check the resource usage of your model and identify bottlenecks. You can access the monitoring web page by typing in your browser: `<node>.server.mila.quebec:19999`.

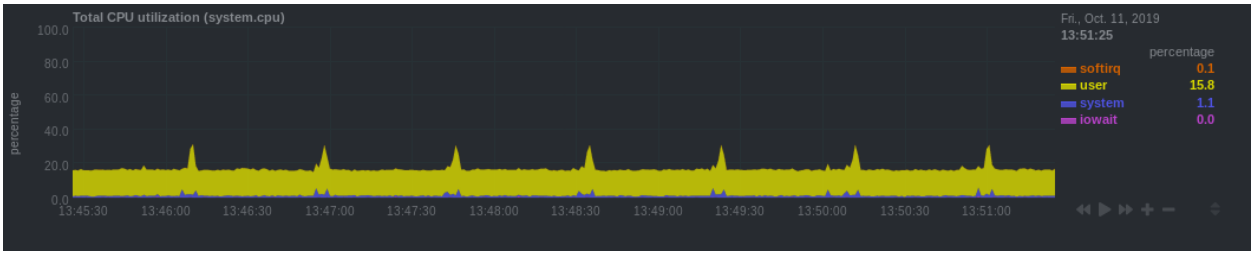
For example, if I have a job running on `eos1` I can type `eos1.server.mila.quebec:19999` and the page below should appear.



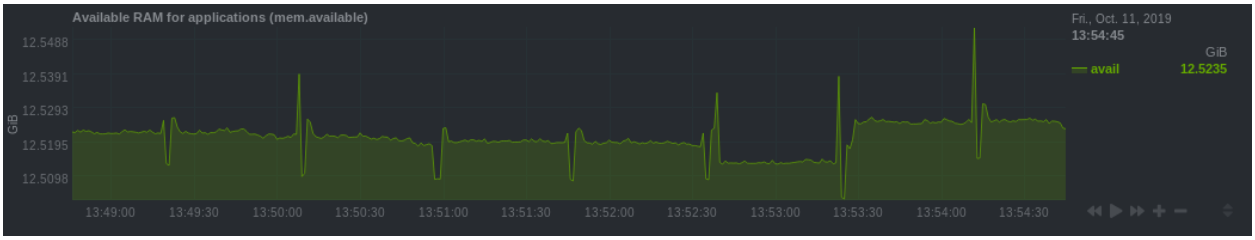
3.5.1 Notable Sections

You should focus your attention on the metrics below

- CPU
  - iowait (pink line): High values means your model is waiting on IO a lot (disk or network)

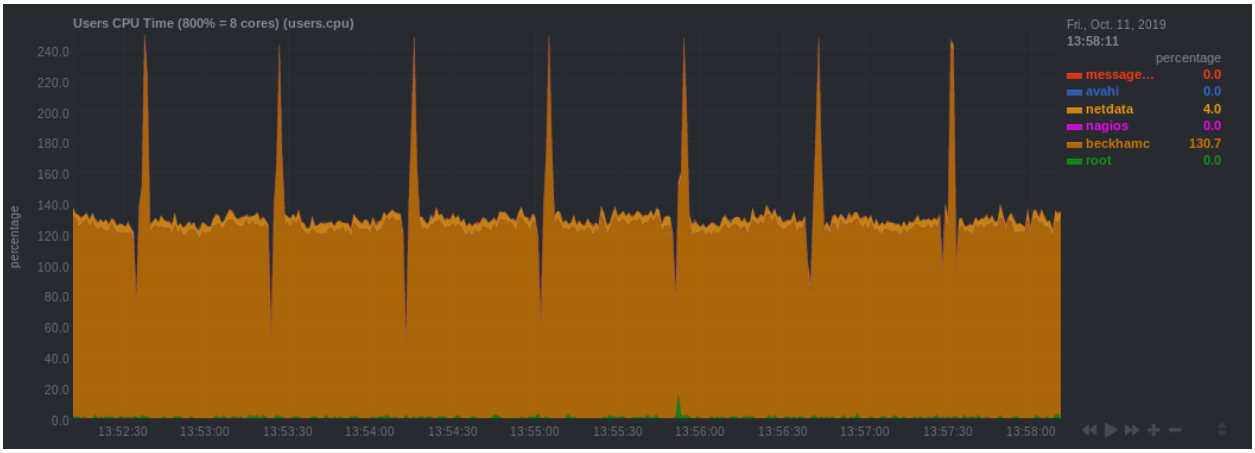


- RAM
  - Make sure you are only allocating enough to make your code run and not more otherwise you are wasting resources.



- NV
  - Usage of each GPU
  - You should make sure you use the GPU to its fullest
    - Select the biggest batch size if possible
    - Spawn multiple experiments

- Users
  - In some cases the machine might seem slow, it may be useful to check if other people are using the machine as well



3.6 Storage

Path	Performance	Usage	Quota (Space/Files)	Auto-cleanup
`$HOME` or `/home/mila/<u>/<username>/`	Low	• Personal user space • Specific libraries, code, binaries	200G/1000K	
`/network/projects/<groupname>/`	Fair	• Shared space to facilitate collaboration between researchers • Long-term project storage	200G/1000K	
`/network/data1/`	High	• Raw datasets (read only)		
`/network/datasets/`	High	• Curated raw datasets (read only)		
`/miniscratch/`	High	• Temporary job results • Processed datasets • Optimized for small Files • Supports ACL to help share the data with others		90 days
`$SLURM_TMPDIR`	Highest	• High speed disk for temporary job results	4T/-	at job end

- `$HOME` is appropriate for codes and libraries which are small and read once, as well as the experimental results that would be needed at a later time (e.g. the weights of a network referenced in a paper).
- `projects` can be used for collaborative projects. It aims to ease the sharing of data between users working on a long-term project. It's possible to request a bigger quota if the project requires it.
- `datasets` contains curated datasets to the benefit of the Mila community. To request the addition of a dataset or

a preprocessed dataset you think could benefit the research of others, you can fill this form.

- `data1` should only contain **compressed** datasets. *Now deprecated and replaced by the* `datasets` *space.*
- `miniscratch` can be used to store processed datasets, work in progress datasets or temporary job results. Its blocksize is optimized for small files which minimizes the performance hit of working on extracted datasets. It supports ACL which can be used to share data between users. This space is cleared weekly and files older then 90 days will be deleted.
- `$SLURM_TMPDIR` points to the local disk of the node on which a job is running. It should be used to copy the data on the node at the beginning of the job and write intermediate checkpoints. This folder is cleared after each job.

Note: **Auto-cleanup** is applied on files not read or modified during the specified period

---

Warning: Currently there are no backup system in the lab. Storage local to personal computers, Google Drive and other related solutions should be used to backup important data

CHAPTER

FOUR

1 USERS GUIDE

or IDT's list of opinionated howtos.

This section seeks to provide users of the Mila infrastructure with practical knowledge, tips and tricks and example commands.

4.1 1.1 Welcome to the machine; Logging in to the cluster

To access the Mila Cluster clusters, you will need an account. Please contact Mila systems administrators if you don't have it already. Our IT support service is available here: https://it-support.mila.quebec/

4.1.1 1.1.1 SSH Login

You can access the Mila cluster via ssh:

```
$ ssh <user>@login.server.mila.quebec -p 2222
```

Four login nodes are available and accessible behind a Load-Balancer. At each connection, you will be redirected to the least loaded login-node. Each login node can be directly accessed via: `login-X.login.server.mila.quebec` on port `2222`.

The login nodes support the following authentication mechanisms: `publickey,keyboard-interactive`. If you would like to set an entry in your `.ssh/config` file, please use the following recommendation:

```
Host HOSTALIAS
    User YOUR-USERNAME
    Hostname login.server.mila.quebec
    PreferredAuthentications publickey,keyboard-interactive
    Port 2222
    ServerAliveInterval 120
    ServerAliveCountMax 5
```

The RSA, DSA and ECDSA fingerprints for Mila's login nodes are:

```
SHA256:baEGIa311fhnxBWsIZJ/zYhq2WfCttwyHRKzAb8zlp8 (ECDSA)
SHA256:XvukABPjV75guEgJX1rNxlDlaEg+IqQzUnPiGJ4VRMM (DSA)
SHA256:Xr0/JqV/+5DNguPfiN5hb8rSG+nBAcfVCJoSyrR0W0o (RSA)
SHA256:gfXZzaPiaYHcrPqzHvBi6v+BWRS/lXOS/zAjOKeoBJg (ED25519)
```

4.2 1.2 Running your code

4.2.1 1.2.1 SLURM commands guide

1.2.1.1 Basic Usage

The SLURM documentation provides extensive information on the available commands to query the cluster status or submit jobs.

Below are some basic examples of how to use SLURM.

1.2.1.2 Submitting jobs

1.2.1.2.1 Batch job

In order to submit a batch job, you have to create a script containing the main command(s) you would like to execute on the allocated resources/nodes.

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --output=job_output.txt
#SBATCH --error=job_error.txt
#SBATCH --ntasks=1
#SBATCH --time=10:00
#SBATCH --mem=100Gb

module load python/3.5
python my_script.py
```

Your job script is then submitted to SLURM with `sbatch` (ref.)

```
$ sbatch job_script
sbatch: Submitted batch job 4323674
```

The *working directory* of the job will be the one where your executed `sbatch`.

Tip: Slurm directives can be specified on the command line alongside `sbatch` or inside the job script with a line starting with `#SBATCH`.

1.2.1.2.2 Interactive job

Workload managers usually run batch jobs to avoid having to watch its progression and let the scheduler run it as soon as resources are available. If you want to get access to a shell while leveraging cluster resources, you can submit an interactive jobs where the main executable is a shell with the `srun/salloc` (srun/salloc) commands

```
$ salloc
```

will start an interactive job on the first node available with the default resources set in SLURM (1 task/1 CPU). `srun` accepts the same arguments as `sbatch` with the exception that the environment is not passed.

Tip: To pass your current environment to an interactive job, add `--preserve-env` to `srun`.

`salloc` can also be used and is mostly a wrapper around `srun` if provided without more info but it gives more flexibility if for example you want to get an allocation on multiple nodes.

1.2.1.3 Job submission arguments

In order to accurately select the resources for your job, several arguments are available. The most important ones are:

Argument	Description
-n, –ntasks=<number>	The number of task in your script, usually =1
-c, –cpus-per-task=<ncpus>	The number of cores for each task
-t, –time=<time>	Time requested for your job
–mem=<size[units]>	Memory requested for all your tasks
–gres=<list>	Select generic resources such as GPUs for your job: `--gres=gpu:GPU_MODEL`

Tip: Always consider requesting the adequate amount of resources to improve the scheduling of your job (small jobs always run first).

1.2.1.4 Checking job status

To display *jobs* currently in queue, use `squeue` and to get only your jobs type

```
$ squeue -u $USER
JOBID   USER          NAME    ST  START_TIME         TIME NODES CPUS TRES_PER_NMIN_MEM NODELIST (REASON
133     my_username   myjob   R   2019-03-28T18:33   0:50     1    2        N/A  7000M c1-8g-tiny1 (Non
```

1.2.1.5 Removing a job

To cancel your job simply use `scancel`

```
$ scancel 4323674
```

4.2.2 1.2.2 Partitioning

Since we don't have many GPUs on the cluster, resources must be shared as fairly as possible. The `--partition=/-p` flag of SLURM allows you to set the priority you need for a job. Each job assigned with a priority can preempt jobs with a lower priority: `unkillable > main > long`. Once preempted, your job is killed without notice and is automatically re-queued on the same partition until resources are available. (To leverage a different preemption mechanism, see the *Handling preemption*)

Flag	Max Resource Usage	Max Time	Note
–partition=unkillable	1 GPU, 6 CPUs, mem=32G	2 days	
–partition=main	2 GPUs, 8 CPUs, mem=48G	2 days	
–partition=long	no limit of resources	7 days	

For instance, to request an unkillable job with 1 GPU, 4 CPUs, 10G of RAM and 12h of computation do:

```
$ sbatch --gres=gpu:1 -c 4 --mem=10G -t 12:00:00 --partition=unkillable <job.sh>
```

You can also make it an interactive job using `salloc`:

```
$ salloc --gres=gpu:1 -c 4 --mem=10G -t 12:00:00 --partition=unkillable
```

The Mila cluster has many different types of nodes/GPUs. To request a specific type of node/GPU, you can add specific feature requirements to your job submission command.

To access those special nodes you need to request them explicitly by adding the flag `--constraint=<name>`. The full list of nodes in the Mila Cluster can be accessed *Node profile description*.

Example:

To request a Power9 machine

```
$ sbatch -c 4 --constraint=power9
```

To request a machine with 2 GPUs using NVLink, you can use

```
$ sbatch -c 4 --gres=gpu:2 --constraint=nvlink
```

Feature	Particularities
x86_64 (Default)	Regular nodes
Power9	*Power9* CPUs (incompatible with x86_64 software)
12GB/16GB/24GB/32GB/48GB	Request a specific amount of *GPU* memory
maxwell/pascal/volta/tesla/turing/kepler	Request a specific *GPU* architecture
nvlink	Machine with GPUs using the NVLink technology

Note: You don't need to specify *x86_64* when you add a constraint as it is added by default (`nvlink -> x86_64&nvlink`)

1.2.2.1 Information on partitions/nodes

`sinfo` (ref.) provides most of the information about available nodes and partitions/queues to submit jobs to.

Partitions are a group of nodes usually sharing similar features. On a partition, some job limits can be applied which will override those asked for a job (i.e. max time, max CPUs, etc…)

To display available *partitions*, simply use

```
$ sinfo
PARTITION AVAIL TIMELIMIT NODES STATE  NODELIST
batch     up     infinite     2 alloc  node[1,3,5-9]
batch     up     infinite     6 idle   node[10-15]
cpu       up     infinite     6 idle   cpu_node[1-15]
gpu       up     infinite     6 idle   gpu_node[1-15]
```

To display available *nodes* and their status, you can use

```
$ sinfo -N -l
NODELIST    NODES PARTITION STATE  CPUS MEMORY TMP_DISK WEIGHT FEATURES REASON
node[1,3,5-9]   2 batch     allocated 2    246    16000     0  (null)   (null)
node[2,4]       2 batch     drain     2    246    16000     0  (null)   (null)
node[10-15]     6 batch     idle      2    246    16000     0  (null)   (null)
...
```

and to get statistics on a job running or terminated, use `sacct` with some of the fields you want to display

```
$ sacct --format=User,JobID,Jobname,partition,state,time,start,end,elapsed,nnodes,ncpus,nodelist,workdi
 User        JobID    JobName  Partition      State  Timelimit               Start                 End
--------- ------------ ---------- ---------- ---------- ---------- ------------------- -------------
```

```
username 2398         run_extra+ azureComp+    RUNNING 130-05:00+ 2019-03-27T18:33:43           Unknou
username 2399         run_extra+ azureComp+    RUNNING 130-05:00+ 2019-03-26T08:51:38           Unknou
```

or to get the list of all your previous jobs, use the `--start=####` flag

```
$ sacct -u my_username --start=20190101
```

`scontrol` (ref.) can be used to provide specific information on a job (currently running or recently terminated)

```
$ scontrol show job 43123
JobId=43123 JobName=python_script.py
UserId=my_username(1500000111) GroupId=student(1500000000) MCS_label=N/A
Priority=645895 Nice=0 Account=my_username QOS=normal
JobState=RUNNING Reason=None Dependency=(null)
Requeue=1 Restarts=3 BatchFlag=1 Reboot=0 ExitCode=0:0
RunTime=2-10:41:57 TimeLimit=130-05:00:00 TimeMin=N/A
SubmitTime=2019-03-26T08:47:17 EligibleTime=2019-03-26T08:49:18
AccrueTime=2019-03-26T08:49:18
StartTime=2019-03-26T08:51:38 EndTime=2019-08-03T13:51:38 Deadline=N/A
PreemptTime=None SuspendTime=None SecsPreSuspend=0
LastSchedEval=2019-03-26T08:49:18
Partition=slurm_partition AllocNode:Sid=login-node-1:14586
ReqNodeList=(null) ExcNodeList=(null)
NodeList=node2
BatchHost=node2
NumNodes=1 NumCPUs=16 NumTasks=1 CPUs/Task=16 ReqB:S:C:T=0:0:*:*
TRES=cpu=16,mem=32000M,node=1,billing=3
Socks/Node=* NtasksPerN:B:S:C=1:0:*:* CoreSpec=*
MinCPUsNode=16 MinMemoryNode=32000M MinTmpDiskNode=0
Features=(null) DelayBoot=00:00:00
OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
WorkDir=/home/mila/my_username
StdErr=/home/mila/my_username/slurm-43123.out
StdIn=/dev/null
StdOut=/home/mila/my_username/slurm-43123.out
Power=
```

or more info on a node and its resources

```
$ scontrol show node node9
NodeName=node9 Arch=x86_64 CoresPerSocket=4
CPUAlloc=16 CPUTot=16 CPULoad=1.38
AvailableFeatures=(null)
ActiveFeatures=(null)
Gres=(null)
NodeAddr=10.252.232.4 NodeHostName=mila20684000000 Port=0 Version=18.08
OS=Linux 4.15.0-1036 #38-Ubuntu SMP Fri Dec 7 02:47:47 UTC 2018
RealMemory=32000 AllocMem=32000 FreeMem=23262 Sockets=2 Boards=1
State=ALLOCATED+CLOUD ThreadsPerCore=2 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
Partitions=slurm_partition
BootTime=2019-03-26T08:50:01 SlurmdStartTime=2019-03-26T08:51:15
CfgTRES=cpu=16,mem=32000M,billing=3
AllocTRES=cpu=16,mem=32000M
CapWatts=n/a
CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

4.2.3 1.2.3 Useful Commands

Command	Description
salloc	Get an interactive job and give you a shell. (ssh like) CPU only
salloc –gres=gpu -c 2 –mem=12000	Get an interactive job with one GPU, 2 CPUs and 12000 MB RAM
sbatch	start a batch job (same options as salloc)
sattach –pty <jobid>.0	Re-attach a dropped interactive job
sinfo	status of all nodes
sinfo -Ogres:27,nodelist,features -tidle,mix,alloc	List GPU type and FEATURES that you can request
savail	(Custom) List available gpu
scancel <jobid>	Cancel a job
squeue	summary status of all active jobs
squeue -u $USER	summary status of all YOUR active jobs
squeue -j <jobid>	summary status of a specific job
squeue -Ojobid,name,username,partition, state,timeused,nodelist,gres,tres	status of all jobs including requested resources (see the SLURM squeue doc for all output options)
scontrol show job <jobid>	Detailed status of a running job
sacct -j <job_id> -o NodeList	Get the node where a finished job ran
sacct -u $USER -S <start_time> -E <stop_time>	Find info about old jobs
sacct -oJobID,JobName,User,Partition,Node,State	List of current and recent jobs

4.2.4 1.2.4 Special GPU requirements

Specific GPU *architecture* and *memory* can be easily requested through the `--gres` flag by using either

- `--gres=gpu:architecture:memory:number`
- `--gres=gpu:architecture:number`
- `--gres=gpu:memory:number`
- `--gres=gpu:model:number`

Example:

To request a Tesla GPU with *at least* 16GB of memory use

```
$ sbatch -c 4 --gres=gpu:tesla:16gb:1
```

The full list of GPU and their features can be accessed *here*.

4.2.5 1.2.5 CPU-only jobs

Since the priority is given to the usage of GPUs, CPU-only jobs have a low priority and can only consume **4 cpus maximum per node**. The partition for CPU-only jobs is named `cpu_jobs` and you can request it with `-p cpu_jobs` or if you don't specify any GPU, you will be automatically rerouted to this partition.

4.2.6 1.2.6 Example script

Here is a `sbatch` script that follows good practices on the Mila cluster:

```
#!/bin/bash
#SBATCH --partition=unkillable                      # Ask for unkillable job
#SBATCH --cpus-per-task=2                           # Ask for 2 CPUs
#SBATCH --gres=gpu:1                                # Ask for 1 GPU
#SBATCH --mem=10G                                   # Ask for 10 GB of RAM
#SBATCH --time=3:00:00                              # The job will run for 3 hours
#SBATCH -o /network/tmp1/<user>/slurm-%j.out  # Write the log on tmp1

# 1. Load the required modules
module --quiet load anaconda/3

# 2. Load your environment
conda activate <env_name>

# 3. Copy your dataset on the compute node
cp /network/data/<dataset> $SLURM_TMPDIR

# 4. Launch your job, tell it to save the model in $SLURM_TMPDIR
#    and look for the dataset into $SLURM_TMPDIR
python main.py --path $SLURM_TMPDIR --data_path $SLURM_TMPDIR

# 5. Copy whatever you want to save on $SCRATCH
cp $SLURM_TMPDIR/<to_save> /network/tmp1/<user>/
```

4.3 1.3 Portability concerns and solutions

4.3.1 1.3.1 Creating a list of your software's dependencies

TODO

4.3.2 1.3.2 Managing your envs

1.3.2.1 Pip/Virtualenv

Pip is the preferred package manager for Python and each cluster provides several Python versions through the associated module which comes with pip. In order to install new packages, you will first have to create a personal space for them to be stored. The preferred solution (as it is the preferred solution on Compute Canada clusters) is to use virtual environments.

First, load the python module you want to use:

```
$ module load python/3.6
```

Then, create a virtual environment in your `home` directory:

```
$ virtualenv $HOME/<env>
```

where `<env>` is the name of your environment. Finally, activate the environment:

```
$ source $HOME/<env>/bin/activate
```

You can now install any python package you wish using the `pip` command, e.g. pytorch:

```
(<env>)$ pip install torch torchvision
```

or Tensorflow:

```
(<env>)$ pip install tensorflow-gpu
```

1.3.2.2 Conda

Another solution for Python is to use miniconda or anaconda which are also available through the `module` command: (the use of conda is not recommended for Compute Canada Clusters due to the availability of custom-built packages for pip)

```
$ module load miniconda/3
[=== Module miniconda/3 loaded ===]
To enable conda environment functions, first use:

$ conda-activate
```

Then like advised, if you want to enable `conda activate/deactivate` functions, start the following alias once

```
$ conda-activate
```

To create an environment (see here for details) do:

```
$ conda create -n <env> python=3.6
```

where `<env>` is the name of your environment. You can now activate it by doing:

```
$ conda activate <env>
```

You are now ready to install any python package you want in this environment. For instance, to install pytorch, you can find the conda command of any version you want on pytorch's website, e.g:

```
(<env>)$ conda install pytorch torchvision cudatoolkit=10.0 -c pytorch
```

Don't forget to clean the environment after each install:

```
(<env>)$ conda clean --all
```

4.3.3 1.3.3 Using Modules

Many software, such as Python and Conda, are already compiled and available on the cluster through the `module` command and its sub-commands. In particular, if you with to use `Python 3.7` you can simply do:

```
$ module load python/3.7
```

1.3.3.1 The module command

For a list of available modules, simply use:

```
$ module avail
-------------------------------------------------------------------------------------------------------------
   cuda/10.0 -> cudatoolkit/10.0    cuda/9.2      -> cudatoolkit/9.2                                    py
   cuda/10.1 -> cudatoolkit/10.1    mujoco-py     -> python/3.7/mujoco-py/2.0                           py
   cuda/10.2 -> cudatoolkit/10.2    mujoco-py/2.0 -> python/3.7/mujoco-py/2.0                           py
   cuda/11.0 -> cudatoolkit/11.0    pytorch       -> python/3.7/cuda/10.2/cudnn/7.6/pytorch/1.5.1       te
   cuda/9.0  -> cudatoolkit/9.0     pytorch/1.4.0 -> python/3.7/cuda/10.2/cudnn/7.6/pytorch/1.4.0       te
```

```
---------------------------------------------------------------------------------------------------- /cv
   Mila       (S,L)    anaconda/3 (D)    go/1.13.5        miniconda/2        mujoco/1.50        python/
   anaconda/2          go/1.12.4         go/1.14   (D)    miniconda/3 (D)    mujoco/2.0  (D)    python/

---------------------------------------------------------------------------------------------------- /cvmf
   python/3.7/mujoco-py/2.0

---------------------------------------------------------------------------------------------------- /cv
   cuda/10.0/cudnn/7.3        cuda/10.0/nccl/2.4         cuda/10.1/nccl/2.4     cuda/11.0/nccl/2.7
   cuda/10.0/cudnn/7.5        cuda/10.1/cudnn/7.5        cuda/10.2/cudnn/7.6    cuda/9.0/cudnn/7.3
   cuda/10.0/cudnn/7.6 (D)    cuda/10.1/cudnn/7.6 (D)    cuda/10.2/nccl/2.7     cuda/9.0/cudnn/7.5 (D)

---------------------------------------------------------------------------------------------------- /cvmf
   python/3.7/cuda/10.1/cudnn/7.6/pytorch/1.4.1    python/3.7/cuda/10.1/cudnn/7.6/pytorch/1.5.1 (D)    p
   python/3.7/cuda/10.1/cudnn/7.6/pytorch/1.5.0    python/3.7/cuda/10.2/cudnn/7.6/pytorch/1.4.1        p

---------------------------------------------------------------------------------------------------- /cvmfs
   python/3.7/tensorflow/1.15    python/3.7/tensorflow/2.0    python/3.7/tensorflow/2.2 (D)
```

Modules can be loaded using the `load` command:

```
$ module load <module>
```

To search for a module or a software, use the command `spider`:

```
$ module spider search_term
```

E.g.: by default, `python2` will refer to the os-shipped installation of `python2.7` and `python3` to `python3.6`. If you want to use `python3.7` you can type:

```
$ module load python3.7
```

1.3.3.2 Available Software

Modules are divided in 5 main sections:

Section	Description
Core	Base interpreter and software (Python, go, etc…)
Compiler	Interpreter-dependent software (*see the note below*)
Cuda	Toolkits, cudnn and related libraries
Pytorch/Tensorflow	Pytorch/TF built with a specific Cuda/Cudnn version for Mila's GPUs (*see the related paragraph*)

Note: Modules which are nested (../../..) usually depend on other software/module loaded alongside the main module. No need to load the dependent software, the complex naming scheme allows an automatic detection of the dependent module(s):

i.e.: Loading `cudnn/7.6/cuda/9.0/tensorrt/7.0` will load `cudnn/7.6` and `cuda/9.0` alongside

`python/3.X` is a particular dependency which can be served through `python/3.X` or `anaconda/3` and is not automatically loaded to let the user pick his favorite flavor.

1.3.3.3 Default package location

Python by default uses the user site package first and packages provided by `module` last to not interfere with your installation. If you want to skip packages installed in your site package (in your /home folder), you have to start Python with the `-s` flag.

To check which package is loaded at import, you can print `package.__file__` to get the full path of the package.

Example:

```
$ module load pytorch/1.5.0
$ python -c 'import torch;print(torch.__file__)'
/home/mila/my_home/.local/lib/python3.7/site-packages/torch/__init__.py   <== package from your own sit
```

Now with the `-s` flag:

```
$ module load pytorch/1.5.0
$ python -s -c 'import torch;print(torch.__file__)'
/cvmfs/ai.mila.quebec/apps/x86_64/debian/pytorch/python3.7-cuda10.1-cudnn7.6-v1.5.0/lib/python3.7/site-p
```

4.3.4 1.3.4 On using containers

Another option for portable code might also be *1.4 Using containers*.

One popular mechanism for containerisation on a computational cluster is called *singularity*. This is the recommended approach for running containers on the Mila cluster.

Singularity is a software container system designed to facilitate portability and reproducibility of high performance computing (HPC) workflows. It performs a function similar to docker, but with HPC in mind. It is compatible with existing docker containers, and provides tools for building new containers from recipe files or ad-hoc commands.

Building a container is like creating a new environment except that containers are much more powerful since they are self-contain systems. With singularity, there are two ways to build containers.

The first one is by yourself, it's like when you got a new Linux laptop and you don't really know what you need, if you see that something is missing, you install it. Here you can get a vanilla container with Ubuntu called a sandbox, you log in and you install each packages by yourself. This procedure can take time but will allow you to understand how things work and what you need. This is recommended if you need to figure out how things will be compiled or if you want to install packages on the fly. We'll refer to this procedure as singularity sandboxes.

The second one way is more like you know what you want, so you write a list of everything you need, you sent it to singularity and it will install everything for you. Those lists are called singularity recipes.

1.3.4.1 First way: Build and use a sandbox

You might ask yourself; *On which machine should I build a container ?*

First of all, you need to choose where you'll build your container. This operation requires **memory and high cpu usage**.

> **Warning:** Do NOT build containers on any login nodes !

- (Recommended for beginner) If you need to **use apt-get**, you should **build the container on your laptop** with sudo privileges. You'll only need to install singularity on your laptop. Windows/Mac users can look there and Ubuntu/Debian users can use directly:

  ```
  $ sudo apt-get install singularity-container
  ```

- If you **can't install singularity** on your laptop and you **don't need apt-get**, you can reserve a **cpu node on the mila cluster** to build your container.

In this case, in order to avoid too much I/O over the network, you should define the singularity cache locally:

```
$ export SINGULARITY_CACHEDIR=$SLURM_TMPDIR
```

- If you **can't install singularity** on your laptop and you **want to use apt-get**, you can use singularity-hub to build your containers and read *Recipe_section*.

1.3.4.1.1 Download containers from the web

Hopefully, you may not need to create containers from scratch as many have been already built for the most common deep learning software. You can find most of them on dockerhub.

Tip: (Optional) You can also pull containers from nvidia cloud see nvidia

---

Go on dockerhub and select the container you want to pull.

For example, if you want to get the latest pytorch version with gpu support (Replace *runtime* by *devel* if you need the full CUDA toolkit):

```
$ singularity pull docker://pytorch/pytorch:1.0.1-cuda10.0-cudnn7-runtime
```

or the latest tensorflow:

```
$ singularity pull docker://tensorflow/tensorflow:latest-gpu-py3
```

Currently the pulled image `pytorch.simg` or `tensorflow.simg` is read only meaning that you won't be able to install anything on it. Starting now, pytorch will be taken as example. If you use tensorflow, simply replace every **pytorch** occurrences by **tensorflow**.

1.3.4.1.2 How to add or install stuff in a container

The first step is to transform your read only container `pytorch-1.0.1-cuda10.0-cudnn7-runtime.simg` in a writable version that will allow you to add packages.

Warning: Depending of the version of singularity you are using, singularity will build a container with the extension .simg or .sif. If you got .sif files, replace every occurences of .simg by .sif.

Tip: If you want to use **apt-get** you have to put **sudo** ahead of the following commands

---

This command will create a writable image in the folder `pytorch`.

```
$ singularity build --sandbox pytorch pytorch-1.0.1-cuda10.0-cudnn7-runtime.simg
```

Then you'll need the following command to log inside the container.

```
$ singularity shell --writable -H $HOME:/home pytorch
```

Once you get into the container, you can use pip and install anything you need (Or with `apt-get` if you built the container with sudo).

Warning: Singularity mount your home, so if you install things into the $HOME of your container, they will be installed in your real $HOME !

You should install your stuff in /usr/local instead.

1.3.4.1.3 Creating useful directory

One of the benefit of containers is that you'll be able to use them across different clusters. However for each cluster the dataset and experiment folder location can be different. In order to be invariant to those locations, we will create some useful mount points inside the container:

```
<Singularity_container>$ mkdir /dataset
<Singularity_container>$ mkdir /tmp_log
<Singularity_container>$ mkdir /final_log
```

From now, you won't need to worry anymore when you write your code to specify where to pick up your dataset. Your dataset will always be in `/dataset` independently of the cluster you are using.

1.3.4.1.4 Testing

If you have some code that you want to test before finalizing your container, you have two choices. You can either log into your container and run python code inside it with

```
$ singularity shell --nv pytorch
```

or you can execute your command directly with

```
$ singularity exec --nv pytorch python YOUR_CODE.py
```

Tip: —nv allows the container to use gpus. You don't need this if you don't plan to use a gpu.

Warning: Don't forget to clear the cache of the packages you installed in the containers.

1.3.4.1.5 Creating a new image from the sandbox

Once everything you need is installed inside the container, you need to convert it back to a read-only singularity image with:

```
$ singularity build pytorch_final.simg pytorch
```

1.3.4.2 Second way: Use recipes

A singularity recipe is a file including specifics about installation software, environment variables, files to add, and container metadata. It is a starting point for designing any custom container. Instead of pulling a container and install your packages manually, you can specify in this file the packages you want and then build your container from this file.

Here is a toy example of a singularity recipe installing some stuff:

```
################## Header: Define the base system you want to use ################
# Reference of the kind of base you want to use (e.g., docker, debootstrap, shub).
Bootstrap: docker
# Select the docker image you want to use (Here we choose tensorflow)
From: tensorflow/tensorflow:latest-gpu-py3

################## Section: Defining the system ####################################
# Commands in the %post section are executed within the container.
%post
        echo "Installing Tools with apt-get"
        apt-get update
        apt-get install -y cmake libcupti-dev libyaml-dev wget unzip
        apt-get clean
        echo "Installing things with pip"
        pip install tqdm
        echo "Creating mount points"
        mkdir /dataset
        mkdir /tmp_log
        mkdir /final_log


# Environment variables that should be sourced at runtime.
%environment
        # use bash as default shell
        SHELL=/bin/bash
        export SHELL
```

A recipe file contains two parts: the `header` and `sections`. In the `header` you specify which base system you want to use, it can be any docker or singularity container. In `sections`, you can list the things you want to install in the subsection `post` or list the environment's variable you need to source at each runtime in the subsection `environment`. For a more detailed description, please look at the singularity documentation.

In order to build a singularity container from a singularity recipe file, you should use:

```
$ sudo singularity build <NAME_CONTAINER> <YOUR_RECIPE_FILES>
```

> **Warning:** You always need to use sudo when you build a container from a recipe.

1.3.4.2.1 Build recipe on singularity hub

Singularity hub allows users to build containers from recipes directly on singularity-hub's cloud meaning that you don't need anymore to build containers by yourself. You need to register on singularity-hub and link your singularity-hub account to your github account, then

1) Create a new github repository.
2) Add a collection on singularity-hub and select the github repository your created.
3) Clone the github repository on your computer.
4) Write the singularity recipe and save it as a file nammed **Singularity**.
5) Git add **Singularity**, commit and push on the master branch.

At this point, robots from singularity-hub will build the container for you, you will be able to download your container from the website or directly with:

```
$ singularity pull shub://<github_username>/<repository_name>
```

1.3.4.2.2 Example: Recipe with openai gym, mujoco and miniworld

Here is an example on how you can use singularity recipe to install complex environment as opanai gym, mujoco and miniworld on a pytorch based container. In order to use mujoco, you'll need to copy the key stored on the mila cluster in */ai/apps/mujoco/license/mjkey.txt* to your current directory.

```
#This is a dockerfile that sets up a full Gym install with test dependencies
Bootstrap: docker

# Here we ll build our container upon the pytorch container
From: pytorch/pytorch:1.0-cuda10.0-cudnn7-runtime

# Now we'll copy the mjkey file located in the current directory inside the container's␣
↪root
# directory
%files
        mjkey.txt

# Then we put everything we need to install
%post
        export PATH=$PATH:/opt/conda/bin
        apt -y update && \
        apt install -y keyboard-configuration && \
        apt install -y \
        python3-dev \
        python-pyglet \
        python3-opengl \
        libhdf5-dev \
        libjpeg-dev \
        libboost-all-dev \
        libsdl2-dev \
        libosmesa6-dev \
        patchelf \
        ffmpeg \
```

(continues on next page)

(continued from previous page)

```
        xvfb \
        libhdf5-dev \
        openjdk-8-jdk \
        wget \
        git \
        unzip && \
        apt clean && \
        rm -rf /var/lib/apt/lists/*
        pip install h5py

        # Download Gym and Mujoco
        mkdir /Gym && cd /Gym
        git clone https://github.com/openai/gym.git || true && \
        mkdir /Gym/.mujoco && cd /Gym/.mujoco
        wget https://www.roboti.us/download/mjpro150_linux.zip  && \
        unzip mjpro150_linux.zip && \
        wget https://www.roboti.us/download/mujoco200_linux.zip && \
        unzip mujoco200_linux.zip && \
        mv mujoco200_linux mujoco200

        # Export global environment variables
        export MUJOCO_PY_MJKEY_PATH=/Gym/.mujoco/mjkey.txt
        export MUJOCO_PY_MUJOCO_PATH=/Gym/.mujoco/mujoco150/
        export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/Gym/.mujoco/mjpro150/bin
        export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/Gym/.mujoco/mujoco200/bin
        export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/bin
        cp /mjkey.txt /Gym/.mujoco/mjkey.txt
        # Install python dependencies
        wget https://raw.githubusercontent.com/openai/mujoco-py/master/requirements.txt
        pip install -r requirements.txt
        # Install Gym and Mujoco
        cd /Gym/gym
        pip install -e '.[all]'
        # Change permission to use mujoco_py as non sudoer user
        chmod -R 777 /opt/conda/lib/python3.6/site-packages/mujoco_py/
        pip install --upgrade minerl

# Export global environment variables
%environment
        export SHELL=/bin/sh
        export MUJOCO_PY_MJKEY_PATH=/Gym/.mujoco/mjkey.txt
        export MUJOCO_PY_MUJOCO_PATH=/Gym/.mujoco/mujoco150/
        export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/Gym/.mujoco/mjpro150/bin
        export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/Gym/.mujoco/mujoco200/bin
        export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/bin
        export PATH=/Gym/gym/.tox/py3/bin:$PATH

%runscript
        exec /bin/sh "$@"
```

Here is the same recipe but written for TensorFlow.

```
#This is a dockerfile that sets up a full Gym install with test dependencies
Bootstrap: docker

# Here we ll build our container upon the tensorflow container
From: tensorflow/tensorflow:latest-gpu-py3

# Now we'll copy the mjkey file located in the current directory inside the container's␣
↪root
# directory
%files
        mjkey.txt

# Then we put everything we need to install
%post
        apt -y update && \
        apt install -y keyboard-configuration && \
        apt install -y \
        python3-setuptools \
        python3-dev \
        python-pyglet \
        python3-opengl \
        libjpeg-dev \
        libboost-all-dev \
        libsdl2-dev \
        libosmesa6-dev \
        patchelf \
        ffmpeg \
        xvfb \
        wget \
        git \
        unzip && \
        apt clean && \
        rm -rf /var/lib/apt/lists/*

        # Download Gym and Mujoco
        mkdir /Gym && cd /Gym
        git clone https://github.com/openai/gym.git || true && \
        mkdir /Gym/.mujoco && cd /Gym/.mujoco
        wget https://www.roboti.us/download/mjpro150_linux.zip  && \
        unzip mjpro150_linux.zip && \
        wget https://www.roboti.us/download/mujoco200_linux.zip && \
        unzip mujoco200_linux.zip && \
        mv mujoco200_linux mujoco200

        # Export global environment variables
        export MUJOCO_PY_MJKEY_PATH=/Gym/.mujoco/mjkey.txt
        export MUJOCO_PY_MUJOCO_PATH=/Gym/.mujoco/mujoco150/
        export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/Gym/.mujoco/mjpro150/bin
        export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/Gym/.mujoco/mujoco200/bin
        export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/bin
        cp /mjkey.txt /Gym/.mujoco/mjkey.txt

        # Install python dependencies
```

(continues on next page)

(continued from previous page)

```
        wget https://raw.githubusercontent.com/openai/mujoco-py/master/requirements.txt
        pip install -r requirements.txt
        # Install Gym and Mujoco
        cd /Gym/gym
        pip install -e '.[all]'
        # Change permission to use mujoco_py as non sudoer user
        chmod -R 777 /usr/local/lib/python3.5/dist-packages/mujoco_py/

        # Then install miniworld
        cd /usr/local/
        git clone https://github.com/maximecb/gym-miniworld.git
        cd gym-miniworld
        pip install -e .

# Export global environment variables
%environment
        export SHELL=/bin/bash
        export MUJOCO_PY_MJKEY_PATH=/Gym/.mujoco/mjkey.txt
        export MUJOCO_PY_MUJOCO_PATH=/Gym/.mujoco/mujoco150/
        export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/Gym/.mujoco/mjpro150/bin
        export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/Gym/.mujoco/mujoco200/bin
        export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/bin
        export PATH=/Gym/gym/.tox/py3/bin:$PATH

%runscript
        exec /bin/bash "$@"
```

Keep in mind that those environment variables are sourced at runtime and not at build time. This is why, you should also define them in the `%post` section since they are required to install mujoco.

1.3.4.3 Using containers on clusters

On every cluster with SLURM, dataset and intermediate results should go in `$SLURM_TMPDIR` while the final experiments results should go in `$SCRATCH`. In order to use the container you built, you need to copy it on the cluster you want to use.

> **Warning:** You should always store your container in $SCRATCH !

Then reserve a node with srun/sbatch, copy the container and your dataset on the node given by slurm (i.e in `$SLURM_TMPDIR`) and execute the code <YOUR_CODE> within the container <YOUR_CONTAINER> with:

```
$ singularity exec --nv -H $HOME:/home -B $SLURM_TMPDIR:/dataset/ -B $SLURM_TMPDIR:/tmp_log/ -B $SCRATCH
```

Remember that `/dataset`, `/tmp_log` and `/final_log` were created in the previous section. Now each time, we'll use singularity, we are explicitly telling it to mount `$SLURM_TMPDIR` on the cluster's node in the folder `/dataset` inside the container with the option `-B` such that each dataset downloaded by pytorch in `/dataset` will be available in `$SLURM_TMPDIR`.

This will allow us to have code and scripts that are invariant to the cluster environment. The option `-H` specify what will be the container's home. For example, if you have your code in `$HOME/Project12345/Version35/` you can specify `-H $HOME/Project12345/Version35:/home`, thus the container will only have access to the code inside `Version35`.

If you want to run multiple commands inside the container you can use:

```
$ singularity exec --nv -H $HOME:/home -B $SLURM_TMPDIR:/dataset/ -B $SLURM_TMPDIR:/tmp_log/ -B $SCRATCH
```

1.3.4.3.1 Example: Interactive case (srun/salloc)

Once you get an interactive session with slurm, copy <YOUR_CONTAINER> and <YOUR_DATASET> to $SLURM_TMPDIR

```
# 0. Get an interactive session
$ srun --gres=gpu:1
# 1. Copy your container on the compute node
$ rsync -avz $SCRATCH/<YOUR_CONTAINER> $SLURM_TMPDIR
# 2. Copy your dataset on the compute node
$ rsync -avz $SCRATCH/<YOUR_DATASET> $SLURM_TMPDIR
```

then use `singularity shell` to get a shell inside the container

```
# 3. Get a shell in your environment
$ singularity shell --nv          -H $HOME:/home          -B $SLURM_TMPDIR:/dataset/          -B $SLURM
```

```
# 4. Execute your code
<Singularity_container>$ python <YOUR_CODE>
```

or use `singularity exec` to execute <YOUR_CODE>.

```
# 3. Execute your code
$ singularity exec --nv          -H $HOME:/home          -B $SLURM_TMPDIR:/dataset/          -B $SLURM_
```

You can create also the following alias to make your life easier.

```
$ alias my_env='singularity exec --nv          -H $HOME:/home          -B $SLURM_TMPDIR:/dataset/
```

This will allow you to run any code with:

```
$ my_env python <YOUR_CODE>
```

1.3.4.3.2 Example: sbatch case

You can also create a `sbatch` script:

```
#!/bin/bash
#SBATCH --cpus-per-task=6         # Ask for 6 CPUs
#SBATCH --gres=gpu:1              # Ask for 1 GPU
#SBATCH --mem=10G                 # Ask for 10 GB of RAM
#SBATCH --time=0:10:00            # The job will run for 10 minutes

# 1. Copy your container on the compute node
rsync -avz $SCRATCH/<YOUR_CONTAINER> $SLURM_TMPDIR
# 2. Copy your dataset on the compute node
rsync -avz $SCRATCH/<YOUR_DATASET> $SLURM_TMPDIR
# 3. Executing your code with singularity
singularity exec --nv \
        -H $HOME:/home \
        -B $SLURM_TMPDIR:/dataset/ \
        -B $SLURM_TMPDIR:/tmp_log/ \
        -B $SCRATCH:/final_log/ \
        $SLURM_TMPDIR/<YOUR_CONTAINER> \
```

(continues on next page)

(continued from previous page)

```
        python <YOUR_CODE>
# 4. Copy whatever you want to save on $SCRATCH
rsync -avz $SLURM_TMPDIR/<to_save> $SCRATCH
```

1.3.4.3.3 Issue with PyBullet and OpenGL libraries

If you are running certain gym environments that require `pyglet`, you may encounter a problem when running your singularity instance with the Nvidia drivers using the `--nv` flag. This happens because the `--nv` flag also provides the OpenGL libraries:

```
libGL.so.1 => /.singularity.d/libs/libGL.so.1
libGLX.so.0 => /.singularity.d/libs/libGLX.so.0
```

If you don't experience those problems with `pyglet`, you probably don't need to address this. Otherwise, you can resolve those problems by `apt-get install -y libosmesa6-dev mesa-utils mesa-utils-extra libgl1-mesa-glx`, and then making sure that your `LD_LIBRARY_PATH` points to those libraries before the ones in `/.singularity.d/libs`.

```
%environment
        # ...
        export LD_LIBRARY_PATH=/usr/lib/x86_64-linux-gnu/mesa:$LD_LIBRARY_PATH
```

1.3.4.3.4 Mila cluster

On the Mila cluster `$SCRATCH` is not yet defined, you should add the experiment results you want to keep in `/network/tmp1/$USER/`. In order to use the sbatch script above and to match other cluster environment's names, you can define `$SCRATCH` as an alias for `/network/tmp1/$USER` with:

```
$ echo "export SCRATCH=/network/tmp1/$USER" >> ~/.bashrc
```

Then, you can follow the general procedure explained above.

1.3.4.3.5 Compute Canada

Using singularity on Compute Canada is similar except that you need to add Yoshua's account name and load singularity. Here is an example of a `sbatch` script using singularity on compute Canada cluster:

> **Warning:** You should use singularity/2.6 or singularity/3.4. There is a bug in singularity/3.2 which makes gpu unusable.

```
#!/bin/bash
#SBATCH --account=rpp-bengioy     # Yoshua pays for your job
#SBATCH --cpus-per-task=6         # Ask for 6 CPUs
#SBATCH --gres=gpu:1              # Ask for 1 GPU
#SBATCH --mem=32G                 # Ask for 32 GB of RAM
#SBATCH --time=0:10:00            # The job will run for 10 minutes
#SBATCH --output="/scratch/<user>/slurm-%j.out" # Modify the output of sbatch

```

(continues on next page)

(continued from previous page)

```
# 1. You have to load singularity
module load singularity
# 2. Then you copy the container to the local disk
rsync -avz $SCRATCH/<YOUR_CONTAINER> $SLURM_TMPDIR
# 3. Copy your dataset on the compute node
rsync -avz $SCRATCH/<YOUR_DATASET> $SLURM_TMPDIR
# 4. Executing your code with singularity
singularity exec --nv \
        -H $HOME:/home \
        -B $SLURM_TMPDIR:/dataset/ \
        -B $SLURM_TMPDIR:/tmp_log/ \
        -B $SCRATCH:/final_log/ \
        $SLURM_TMPDIR/<YOUR_CONTAINER> \
        python <YOUR_CODE>
# 5. Copy whatever you want to save on $SCRATCH
rsync -avz $SLURM_TMPDIR/<to_save> $SCRATCH
```

4.4 1.4 Using containers

Docker containers are now available on the local cluster with a root-less system called Shifter integrated into Slurm. *It is still in beta and be careful with this usage*

4.4.1 1.4.1 Initialising your Containers

To first use a container, you have to pull it to the local registry to be converted to a Shifter-compatible image.

```
$ shifterimg pull docker:image_name:latest
```

You can list available images with

```
$ shifterimg images
```

DO NOT USE IMAGES WITH SENSITIVE INFORMATION yet, it will soon be possible. For now, every image is pulled to a common registry but access-control will soon be implemented.

4.4.2 1.4.2 Using in Slurm

1.4.2.1 Containerized Batch job

You must use the `--image=docker:image_name:latest` directive to specify the container to use. Once the container is mounted, you are not yet inside the container's file-system, you must use the `shifter` command to execute a command in the chroot environment of the container.

e.g.:

```
#!/bin/bash
#SBATCH --image=docker:image_name:latest
#SBATCH --nodes=1
#SBATCH --partition=low

shifter python myPythonScript.py args
```

1.4.2.2 Container Interactive job

Using the salloc command, you can request the image while getting the allocation

```
$ salloc -c2 --mem=16g --image=docker:image_name:latest
```

and once in the job, you can activate the container's environment with the `shifter` command

```
$ shifter /bin/bash
```

4.4.3 1.4.3 Command line

`shifter` support various options on the command line but you should be set with the image name and the command to execute:

```
shifter [-h|--help] [-v|--verbose] [--image=<imageType>:<imageTag>]
    [--entrypoint[=command]] [--workdir[=/path]]
    [-E|--clearenv] [-e|--env=<var>=<value>] [--env-file=/env/file
    [-V|--volume=/path/to/bind:/mnt/in/image[:<flags>[,...]][;...]]
    [-m|--module=<modulename>[,...]]
    [-- /command/to/exec/in/shifter [args...]]
```

4.4.4 1.4.4 Volumes

`/home/yourusername`, `/Tmp`, `/ai` and all `/network/..` sub-folders are mounted inside the container.

4.4.5 1.4.5 GPU

To access the GPU inside a container, you need to specify `--module=nvidia` on the `sbatch/salloc/shifter` command line

```
$ shifter --image=centos:7 --module=nvidia bash
```

Following folders will be mounted in the container:

Host	Container	Comment
/ai/apps/cuda/10.0	/cuda	Cuda libraries and bin, added to `PATH`
/usr/bin	/nvidia/bin	To access `nvidia-smi`
/usr/lib/x86_64-linux-gnu/	/nvidia/lib	`LD_LIBRARY_PATH` will be set to `/nvidia/lib`

Note:

- Use image names in 3 parts to avoid confusion: `_type:name:tag_`
- Please keep in mind that root is squashed on Shifter images, so the software should be installed in a way that is executable to someone with user-level permissions.
- Currently the `/etc` and `/var` directories are reserved for use by the system and will be overwritten when the image is mounted
- The container is not isolated so you share the network card and all hardware from the host, no need to forward ports

4.4.6 1.4.6 Example

```
username@login-2:~$ shifterimg pull docker:alpine:latest
2019-10-11T20:12:42 Pulling Image: docker:alpine:latest, status: READY

username@login-2:~$ salloc -c2 --gres=gpu:1 --image=docker:alpine:latest
salloc: Granted job allocation 213064
salloc: Waiting for resource configuration
salloc: Nodes eos20 are ready for job

username@eos20:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="18.04.2 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.2 LTS"
VERSION_ID="18.04"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic

username@eos20:~$ shifter sh
~ $ cat /etc/os-release
NAME="Alpine Linux"
ID=alpine
VERSION_ID=3.10.2
PRETTY_NAME="Alpine Linux v3.10"

~ $
```

Note: Complete Documentation: https://docs.nersc.gov/programming/shifter/how-to-use/

4.5 1.5 Contributing datasets

4.5.1 1.5.1 Add a dataset for Mila users

If a dataset could help the research of others at Mila, this form can be filled to request it's addition to /network/datasets.

Those datasets can be mirrored to the Béluga cluster in `~/projects/rrg-bengioy-ad/data/curated/` if they follow Compute Canada's good practices on data.

4.5.2 1.5.2 Publicly share a Mila dataset

Mila offers two ways to publicly share a Mila dataset:

- Academic Torrent
- Google Drive

Note that these options are not mutually exclusive and both can be used.

We do host Mila datasets using Academic Torrent and we also offer Google Drive space.

1.5.2.1 Academic Torrent

Mila hosts/seeds some datasets created by the Mila community through Academic Torrent. The first step is to create an account and a torrent file.

Then drop the dataset in `/miniscratch/transit_datasets` and send the Academic Torrent url to Mila's helpdesk. If the dataset does not reside on the Mila clusterm, only the Academic Torrent url would be needed to proceed with the initial download. Then you can delete / stop sharing your copy.

1.5.2.2 Google Drive

Only a member of the staff team can upload to Mila's Google Drive which requires to first drop the dataset in `/miniscratch/transit_datasets`. Then, contact Mila's helpdesk and provide the following informations:

- directory containing the dataset in `/miniscratch/transit_datasets`
- the name of the dataset
- the arXiv and GitHub urls (those can be sent later if the article is still in the submission process)
- instructions to know if the dataset needs to be `unzip`ed, `untar`ed or else before uploading to Google Drive

1.5.2.3 Digital Object Identifier (DOI)

It is recommended to get a DOI to reference the dataset. A DOI is a permanent id/url which prevents losing references of online scientific data. https://figshare.com can be used to create a DOI:

- Go in *My Data*
- Create an item by clicking *Create new item*
- Check *Metadata record only* at the top
- Fill the metadata fields

Then reference the dataset using https://doi.org like this: https://doi.org/10.6084/m9.figshare.2066037

4.6 1.6 Notebooks

4.6.1 1.6.1 JupyterHub

JupyterHub is a platform connected to SLURM to start a **JupyterLab** session as a batch job then connects it when the allocation has been granted. It does not require any ssh tunnel or port redirection, the hub acts as a proxy server that will redirect you to a session as soon as it is available.

It is currently available for Mila clusters and some Compute Canada clusters (under PBS)

Cluster	Address	Login type
Mila Local	https://jupyterhub.server.mila.quebec	Google Oauth
Mila Cloud GCP	https://jupyterhub.gcp.mila.quebec	Google Oauth
Compute Canada	https://docs.computecanada.ca/wiki/JupyterHub	CC login

Warning: Do not forget to close the JupyterLab session! Closing the window leaves running the session and the SLURM job it is linked to.

To close it, use the hub menu and then `Control Panel > Stop my server`

Note: For Mila Clusters:

mila.quebec account's credential should be used to login and start a **JupyterLab** session.

1.6.1.1 Access Mila Storage in JupyterLab

Unfortunatly, JupyterLab does not allow the navigation to parent directories of `$HOME`. This makes some file systems like `/network/datasets` or `$SLURM_TMPDIR` unavailable through their absolute path in the interface. It is however possible to create symbolic links to those resources. To do so, you can use the `ln -s` command:

```
ln -s /network/datasets $HOME
```

Note that `$SLURM_TMPDIR` is a directory that is dinamycaly created for each job so you would need to recreate the symbolic link everytime you start a JupyterHub session:

```
ln -sf $SLURM_TMPDIR $HOME
```

4.7 1.7 Advanced SLURM usage and Multiple GPU jobs

4.7.1 1.7.1 Handling preemption

There are 2 types of preemption:

- **On the local cluster:** jobs can preempt one-another depending on their priority (unkillable>high>low) (See the Slurm documentation)
- **On the cloud clusters:** virtual machines can be preempted as a limitation of less expensive virtual machines (spot/low priority)

On the local cluster, the default preemption mechanism is to killed and re-queue the job automatically without any notice. To allow a different preemption mechanism, every partition have been duplicated (i.e. have the same characteristics as their counterparts) allowing a **120sec** grace period before killing your job *but don't requeue it automatically*: those partitions are referred by the suffix: `-grace` (`main-grace, low-grace, cpu_jobs-grace`).

When using a partition with a grace period, a series of signals consisting of first `SIGCONT` and `SIGTERM` then `SIGKILL` will be sent to the SLURM job. It's good practice to catch those signals using the Linux `trap` command to properly terminate a job and save what's necessary to restart the job. On each cluster, you'll be allowed a *grace period* before SLURM actually kills your job (`SIGKILL`).

The easiest way to handle preemption is by trapping the `SIGTERM` signal

```
#SBATCH --ntasks=1
#SBATCH ....

exit_script() {
    echo "Preemption signal, saving myself"
    trap - SIGTERM # clear the trap
    # Optional: sends SIGTERM to child/sub processes
    kill -- -$$
}

trap exit_script SIGTERM

# The main script part
python3 my_script
```

Note:

Requeuing:
The local Slurm cluster does not allow a grace period before preempting a job while requeuing it automatically, therefore your job will be cancelled at the end of the grace period.
To automatically requeue it, you can just add the `sbatch` command inside your `exit_script` function.

The following table summarizes the different preemption mode and grace periods:

Cluster	Signal(s)	Grace Period	Requeued
local	SIGCONT/SIGTERM	120s	No
Google Gloud (GCP)	SIGCONT/SIGTERM	30s	Yes
Amazon (AWS)	SIGCONT/SIGTERM	120s	Yes
Azure	•	•	•

4.7.2 1.7.2 Packing jobs

1.7.2.1 Sharing a GPU between processes

`srun`, when used in a batch job is responsible for starting tasks on the allocated resources (see srun) SLURM batch script

```
#SBATCH --ntasks-per-node=2
#SBATCH --output=myjob_output_wrapper.out
#SBATCH --ntasks=2
#SBATCH --gres=gpu:1
#SBATCH --cpus-per-task=4
#SBATCH --mem=18G
srun -l --output=myjob_output_%t.out python script args
```

this will run python 2 times, each process with 4 CPUs with the same arguments `--output=myjob_output_%t.out` will create 2 output files appending the task id (`%t`) to the filename and 1 global log file for things happening outside the `srun` command.

Knowing that, if you want to have 2 different arguments to the python program, you can use a multi-prog configuration file: `srun -l --multi-prog silly.conf`

```
0  python script firstarg
1  python script secondarg
```

or by specifying a range of tasks

```
0-1  python script %t
```

%t being the taskid that your python script will parse. Note the `-l` on the `srun` command: this will prepend each line with the taskid (0:, 1:)

1.7.2.2 Sharing a node with multiple GPU 1process/GPU

On Compute Canada, several nodes, especially nodes with `largeGPU` (P100) are reserved for jobs requesting the whole node, therefore packing multiple processes in a single job can leverage faster GPU.

If you want different tasks to access different GPUs in a single allocation you need to create an allocation requesting a whole node and using `srun` with a subset of those resources (1 GPU).

Keep in mind that every resource not specified on the `srun` command while inherit the global allocation specification so you need to split each resource in a subset (except –cpu-per-task which is a per-task requirement)

Each `srun` represents a job step (`%s`).

Example for a GPU node with 24 cores and 4 GPUs and 128G of RAM Requesting 1 task per GPU

```
#!/bin/bash
#SBATCH --nodes=1-1
#SBATCH --ntasks-per-node=4
#SBATCH --output=myjob_output_wrapper.out
#SBATCH --gres=gpu:4
#SBATCH --cpus-per-task=6
srun --gres=gpu:1 -n1 --mem=30G -l --output=%j-step-%s.out --exclusive --multi-prog␣
↪python script args1 &
srun --gres=gpu:1 -n1 --mem=30G -l --output=%j-step-%s.out --exclusive --multi-prog␣
↪python script args2 &
```

(continues on next page)

(continued from previous page)

```
srun --gres=gpu:1 -n1 --mem=30G -l --output=%j-step-%s.out --exclusive --multi-prog␣
↪python script args3 &
srun --gres=gpu:1 -n1 --mem=30G -l --output=%j-step-%s.out --exclusive --multi-prog␣
↪python script args4 &
wait
```

This will create 4 output files:

- JOBID-step-0.out
- JOBID-step-1.out
- JOBID-step-2.out
- JOBID-step-3.out

1.7.2.3 Sharing a node with multiple GPU & multiple processes/GPU

Combining both previous sections, we can create a script requesting a whole node with four GPUs, allocating 1 GPU per `srun` and sharing each GPU with multiple processes

Example still with a 24 cores/4 GPUs/128G RAM Requesting 2 tasks per GPU

```
#!/bin/bash
#SBATCH --nodes=1-1
#SBATCH --ntasks-per-node=8
#SBATCH --output=myjob_output_wrapper.out
#SBATCH --gres=gpu:4
#SBATCH --cpus-per-task=3
srun --gres=gpu:1 -n2 --mem=30G -l --output=%j-step-%s-task-%t.out --exclusive --multi-
↪prog silly.conf &
srun --gres=gpu:1 -n2 --mem=30G -l --output=%j-step-%s-task-%t.out --exclusive --multi-
↪prog silly.conf &
srun --gres=gpu:1 -n2 --mem=30G -l --output=%j-step-%s-task-%t.out --exclusive --multi-
↪prog silly.conf &
srun --gres=gpu:1 -n2 --mem=30G -l --output=%j-step-%s-task-%t.out --exclusive --multi-
↪prog silly.conf &
wait
```

`--exclusive` is important to specify subsequent step/srun to bind to different cpus.

This will produce 8 output files, 2 for each step:

- JOBID-step-0-task-0.out
- JOBID-step-0-task-1.out
- JOBID-step-1-task-0.out
- JOBID-step-1-task-1.out
- JOBID-step-2-task-0.out
- JOBID-step-2-task-1.out
- JOBID-step-3-task-0.out
- JOBID-step-3-task-1.out

Running `nvidia-smi` in silly.conf, while parsing the output, we can see 4 GPUs allocated and 2 tasks per GPU

```
$ cat JOBID-step-* | grep Tesla
0: |   0  Tesla P100-PCIE...  On   | 00000000:04:00.0 Off |                    0 |
1: |   0  Tesla P100-PCIE...  On   | 00000000:04:00.0 Off |                    0 |
0: |   0  Tesla P100-PCIE...  On   | 00000000:83:00.0 Off |                    0 |
1: |   0  Tesla P100-PCIE...  On   | 00000000:83:00.0 Off |                    0 |
0: |   0  Tesla P100-PCIE...  On   | 00000000:82:00.0 Off |                    0 |
1: |   0  Tesla P100-PCIE...  On   | 00000000:82:00.0 Off |                    0 |
0: |   0  Tesla P100-PCIE...  On   | 00000000:03:00.0 Off |                    0 |
1: |   0  Tesla P100-PCIE...  On   | 00000000:03:00.0 Off |                    0 |
```

4.8 1.8 Frequently asked questions (FAQs)

4.8.1 1.8.1 Connection/SSH issues

1.8.1.1 I'm getting `connection refused` while trying to connect to a login node

Login nodes are protected against brute force attacks and might ban your IP if it detects too many connections/failures. You can try to unban yourself by using the following web page: https://unban.server.mila.quebec/

4.8.2 1.8.2 Shell issues

1.8.2.1 How do I change my shell

By default you will be assigned `/bin/bash` as a shell. If you would like to change for another one, please submit a support ticket.

4.8.3 1.8.3 SLURM issues

1.8.3.1 How can I get an interactive shell on the cluster ?

Use `salloc [--slurm_options]` without any executable at the end of the command, this will launch your default shell on an interactive session. Remember that an interactive session is bound to the login node where you start it so you could risk loosing your job if the login node becomes unreachable.

1.8.3.2 srun: error: –mem and –mem-per-cpu are mutually exclusive

You can safely ignore this, `salloc` has a default memory flag in case you don't provide one.

1.8.3.3 How can I see where and if my jobs are running ?

Use `squeue -u YOUR_USERNAME` to see all your job status and locations. To get more info on a running job, try `scontrol show job #JOBID`

1.8.3.4 Unable to allocate resources: Invalid account or account/partition combination specified

Chances are your account is not setup properly. You should file a ticket in our helpdesk: https://it-support.mila.quebec/ .

1.8.3.5 How do I cancel a job?

Use the `scancel #JOBID` command with the job ID of the job you want cancelled. In the case you want to cancel all your jobs, type `scancel -u YOUR_USERNAME`. You can also cancel all your pending jobs for instance with `scancel -t PD`.

1.8.3.6 How can access a node on which one of my job is running ?

You can ssh into a node on which you have a job running, your ssh connection will be adopted by your job, i.e. if your job finishes your ssh connection will be automatically terminated. In order to connect to a node, you need to have password-less ssh either with a key present in your home or with an `ssh-agent`. You can generate a key on the login node for password-less like this:

```
$ ssh-keygen (3xENTER)
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 600 ~/.ssh/authorized_keys
$ chmod 700 ~/.ssh
```

1.8.3.7 I'm getting `Permission denied (publickey)` while trying to connect to a node ?

See previous question

1.8.3.8 Where do I put my data during a job ?

Your `/home` as well as the datasets are on shared file-systems, it is recommended to copy them to the `$SLURM_TMPDIR` to better process them and leverage higher-speed local drives. If you run a low priority job subject to preemption, it's better to keep any output you want to keep on the shared file systems because the `$SLURM_TMPDIR` is deleted at the end of each job.

1.8.3.9 slurmstepd: error: Detected 1 oom-kill event(s) in step #####.batch cgroup

You exceeded the amount of memory allocated to your job, either you did not request enough memory or you have a memory leak in your process. Try increasing the amount of memory requested with `--mem=` or `--mem-per-cpu=`.

1.8.3.10 fork: retry: Resource temporarily unavailable

You exceeded the limit of 2000 tasks/PIDs in your job, it probably means there is an issue with a sub-process spawning too many processes in your script. For any help with your software, please contact the helpdesk.

CHAPTER

FIVE

AI TOOLING AND METHODOLOGY HANDBOOK

This section seeks to provide researchers with insightful articles pertaining to aspects of methodology in their work.

CHAPTER

SIX

COMPUTATIONAL RESOURCES OUTSIDE OF MILA

This section seeks to provide insights and information on computational resources outside the Mila cluster itself.

6.1 Compute Canada Clusters

The clusters Beluga, Cedar, Graham, Helios and Niagara are clusters provided by Compute Canada on which we have allocations. These are to be used for many jobs, multi-nodes and/or multi-GPU jobs as well as long running jobs

6.1.1 Current allocation description

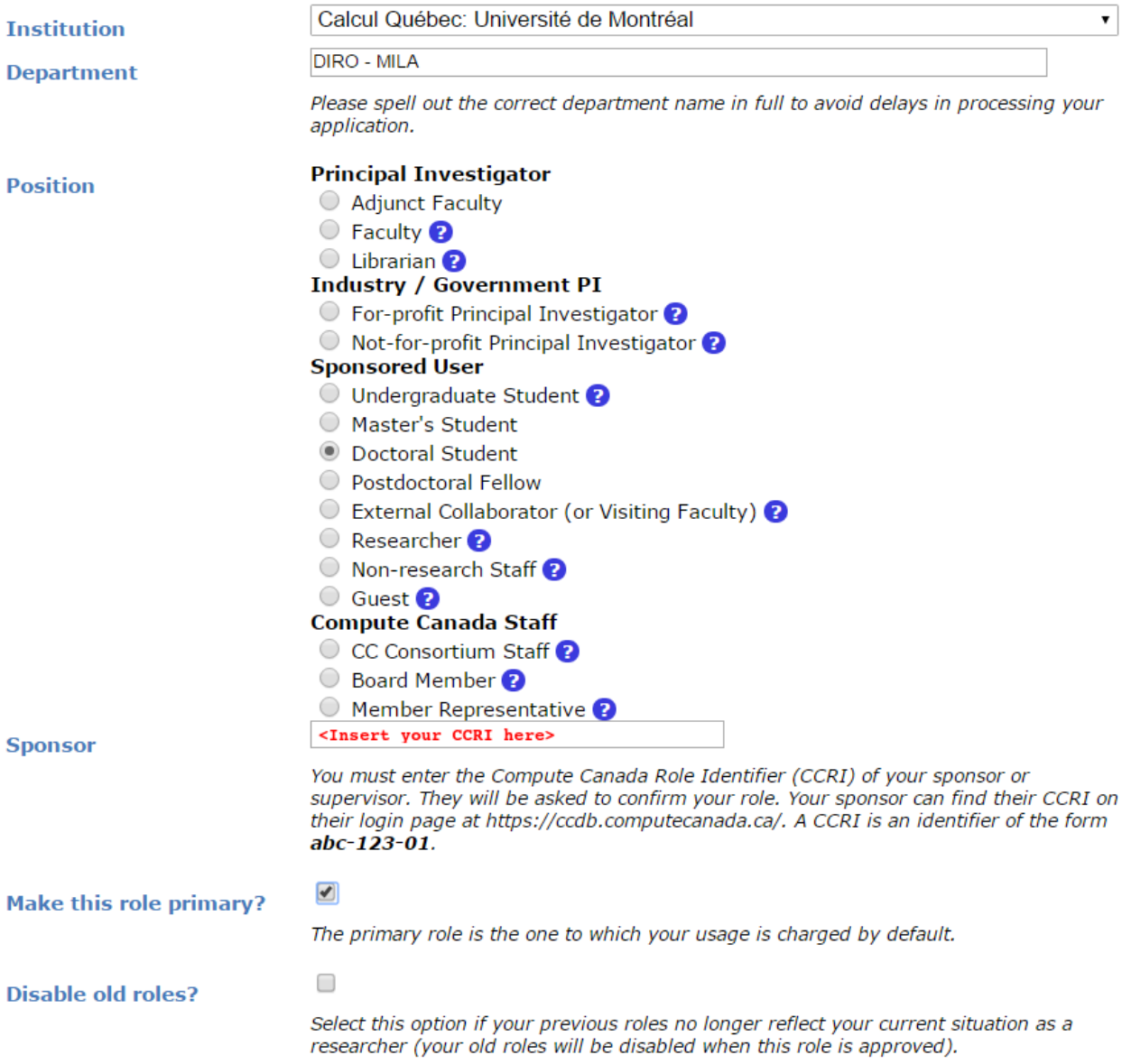
TODO

6.1.2 Account Creation

To access the Compute Canada (CC) clusters you have to first create an account at https://ccdb.computecanada.ca. Use a password with at least 8 characters, mixed case letters, digits and special characters. Later you will be asked to create another password with those rules, and it's really convenient that the two password are the same.

Then, you have to apply for a `role` at https://ccdb.computecanada.ca/me/add_role, which basically means telling CC that you are part of the lab so they know which cluster you can have access to, and track your usage.

You will be asked for the CCRI (See screenshot below). Please reach out to your sponsor to get the CCRI.



You will need to **wait** for your sponsor to accept before being able to login to the CC clusters.

6.1.3 Clusters

Beluga

Beluga is a cluster located at ETS in Montreal. It uses *Slurm* to schedule jobs. Its full documentation can be found here, and its current status here.

You can access Beluga via ssh:

```
$ ssh <user>@beluga.computecanada.ca
```

Where `<user>` is the username you created previously (see *Account Creation*).

Launching Jobs

Users must specify the resource allocation Group Name using the flag `--account=rrg-bengioy-ad`. To launch a CPU-only job:

```
$ sbatch --time=1:0:0 --account=rrg-bengioy-ad job.sh
```

To launch a GPU job:

```
$ sbatch --time=1:0:0 --account=rrg-bengioy-ad --gres=gpu:1 job.sh
```

And to get an interactive session, use the `salloc` command:

```
$ salloc --time=1:0:0 --account=rrg-bengioy-ad --gres=gpu:1
```

The full documentation for jobs launching on Beluga can be found here.

Beluga Nodes description

The GPU nodes consist of:

- 40 CPU cores
- 186 GB RAM
- 4 GPU NVIDIA V100 (16GB)

Tip: You should ask for max 10 CPU cores and 32 GB of RAM per GPU you are requesting (as explained here), otherwise, your job will count for more than 1 allocation, and will take more time to get scheduled.

Beluga Storage

Storage	Path	Usage
$HOME	/home/<user>/	• Code • Specific libraries
$HOME/projects	/project/rpp-bengioy	• Compressed raw datasets
$SCRATCH	/scratch/<user>	• Processed datasets • Experimental results • Logs of experiments
$SLURM_TMPDIR		• Temporary job results

They are roughly listed in order of increasing performance and optimized for different uses:

- The `$HOME` folder on NFS is appropriate for codes and libraries which are small and read once. **Do not write experiemental results here!**
- The `/projects` folder should only contain **compressed raw** datasets (**processed** datasets should go in `$SCRATCH`). We have a limit on the size and number of file in `/projects`, so do not put anything else there. If you

add a new dataset there (make sure it is readable by every member of the group using `chgrp -R rpp-bengioy <dataset>`).

- The `$SCRATCH` space can be used for short term storage. It has good performance and large quotas, but is purged regularly (every file that has not been used in the last 3 months gets deleted, but you receive an email before this happens).
- `$SLURM_TMPDIR` points to the local disk of the node on which a job is running. It should be used to copy the data on the node at the beginning of the job and write intermediate checkpoints. This folder is cleared after each job.

When an experiment is finished, results should be transferred back to Mila servers.

More details on storage can be found here.

Modules

Many software, such as Python or MATLAB are already compiled and available on Beluga through the `module` command and its subcommands. Its full documentation can be found here.

module avail	Displays all the available modules
module load <module>	Loads <module>
module spider <module>	Shows specific details about <module>

In particular, if you with to use `Python 3.6` you can simply do:

```
$ module load python/3.6
```

Tip: If you wish to use Python on the cluster, we strongly encourage you to read CC Python Documentation, and in particular the Pytorch and/or Tensorflow pages.

The cluster has many python packages (or `wheels`), such already compiled for the cluster. See here for the details. In particular, you can browse the packages by doing:

```
$ avail_wheels <wheel>
```

Such wheels can be installed using pip. Moreover, the most efficient way to use modules on the cluster is to build your environnement inside your job. See the script example below.

Script Example

Here is a `sbatch` script that follows good practices on Beluga:

```
#!/bin/bash
#SBATCH --account=rrg-bengioy-ad         # Yoshua pays for your job
#SBATCH --cpus-per-task=6                # Ask for 6 CPUs
#SBATCH --gres=gpu:1                     # Ask for 1 GPU
#SBATCH --mem=32G                        # Ask for 32 GB of RAM
#SBATCH --time=3:00:00                   # The job will run for 3 hours
#SBATCH -o /scratch/<user>/slurm-%j.out  # Write the log in $SCRATCH

# 1. Create your environement locally
module load python/3.6
```

(continues on next page)

(continued from previous page)

```
virtualenv --no-download $SLURM_TMPDIR/env
source $SLURM_TMPDIR/env/bin/activate
pip install --no-index torch torchvision

# 2. Copy your dataset on the compute node
# IMPORTANT: Your dataset must be compressed in one single file (zip, hdf5, ...)!!!
cp $SCRATCH/<dataset.zip> $SLURM_TMPDIR

# 3. Eventually unzip your dataset
unzip $SLURM_TMPDIR/<dataset.zip> -d $SLURM_TMPDIR

# 4. Launch your job, tell it to save the model in $SLURM_TMPDIR
#    and look for the dataset into $SLURM_TMPDIR
python main.py --path $SLURM_TMPDIR --data_path $SLURM_TMPDIR

# 5. Copy whatever you want to save on $SCRATCH
cp $SLURM_TMPDIR/<to_save> $SCRATCH
```

Using CometML and Wandb

The compute nodes for Beluga don't have access to the internet, but there is a special module that can be loaded in order to allow training scripts to access some specific servers, which includes the necessary servers for using CometML and Wandb ("Weights and Biases").

```
$ module load httpproxy
```

More documentation about this can be found at *https://docs.computecanada.ca/wiki/Weights_%26_Biases_(wandb)*.

Graham

Graham is a cluster located at University of Waterloo. It uses SLURM to schedule jobs. Its full documentation can be found here, and its current status here.

You can access Graham via ssh:

```
$ ssh <user>@graham.computecanada.ca
```

Where `<user>` is the username you created previously (see *Account Creation*).

Since its structure is similar to *Beluga*, please look at the *Beluga* documentation, as well as relevant parts of the Compute Canada Documentation.

Note: For GPU jobs the ressource allocation Group Name is the same as Beluga, so you should use the flag `--account=rrg-bengioy-ad` for GPU jobs.

Cedar

Cedar is a cluster located at Simon Fraser University. It uses SLURM to schedule jobs. Its full documentation can be found here, and its current status here.

You can access Cedar via ssh:

```
$ ssh <user>@cedar.computecanada.ca
```

Where `<user>` is the username you created previously (see *Account Creation*).

Since its structure is similar to *Beluga*, please look at the *Beluga* documentation, as well as relevant parts of the Compute Canada Documentation.

Note: However, we don't have any CPU priority on Cedar, in this case you can use `--account=def-bengioy` for CPU. Thus, it might take some time before they start.

6.1.4 FAQ

What to do with *ImportError: /lib64/libm.so.6: version GLIBC_2.23 not found*? The structure of the file system is different than a classical Linux, so your code has trouble finding libraries. See how to install binary packages.

Disk quota exceeded error on `/project` file systems You have files in `/project` with the wrong permissions. See how to change permissions.

CHAPTER

SEVEN

AUDIO AND VIDEO RESOURCES AT MILA

This section seeks to provide information on audio and video systems made available at Mila.

CHAPTER

EIGHT

WHO, WHAT, WHERE IS IDT

This section seeks to help Mila researchers understand the mission and role of the IDT team.

Support To reach the Mila infrastructure support, please file a ticket at https://it-support.mila.quebec/

Contribution If you find any error in the documentation, missing or unclear sections, or would simply like to contribute, please open an issue or make a pull request on the github page.