A Complete Guide for using Compute Canada with Julia!

> This is a Julia version of Compute Canada tutorial. Thanks to the author of Python version. Here is the markdown and pdf version.

1. Introduction

Compute Canada is Canada's national high-performance compute (HPC) system. The system gives users access to both storage and compute resources to make running large data analyses more efficient than on a standard desktop computer. Users can access Compute Canada from any computer with an active internet connection.

The following information help users get start with Compute Canada by walking through the Niagara cluster (a homogeneous cluster, owned by the University of Toronto and operated by SciNet). We provide detailed examples with Julia, if you are using Python, please refer to this Python version of tutorial.

1.1. Terminology

Throughout this Julia version of tutorial I will use the same terms with this Python version of tutorial. This section provides a brief explanation of these terms.

- **Local**: This refers to anything that you own or have in your personal computer or file structure that is not on Compute Canada or any other computer outside of your personal network (whatever is inside your own home). For instance, your local machine is your personal MacBook, ThinkPad, etc that you use for your personal work, entertainment, etc. A local file is a file stored on your personal computer.
- **Remote**: This refers to anything that is on a network or machine outside your personal network. In this tutorial this will mostly refer to the 'remote machine' or 'remote server', which refers to the specific node you are using on Compute Canada. In general it refers to the compute infrastructure that you are accessing using the internet.
- **Cluster**: Compute Canada offers several clusters. A cluster is a collection of computers that shares some properties, such as the location of the cluster. The computers within a cluster are usually interconnected and will have the same file structure so you can access all your files on any computer within a cluster. A node is another term for a computer within a cluster. In this tutorial we will be using the Niagara cluster on Compute Canada, as it has many nodes with high memory GPUs which are suited for deep learning.
- **Node**: A node refers to a computer within a cluster. You typically use one or more nodes when you submit or run a job on Compute Canada. You can specify how many CPUs and GPUs your job needs to access, and for how long, on each node.
- **Shell / Terminal**: This refers to the program and interface running on your local computer or the remote node which lets you control the computer using only text-based commands. Using the shell/terminal is fundamental for using Compute Canada as you cannot control the node on Compute Canada using graphical or cursor-based interfaces as you normally would on your personal computer. It is a good idea to get familiar with using the terminal in general, and I hope this tutorial helps you get started on this journey. If you want to learn these skills in more depth, this is a great website which provides tutorials covering things like the shell, command-line environments: https://missing.csail.mit.edu/

1.2. Useful Links

- Official Wiki
  - https://docs.alliancecan.ca/wiki/Technical_documentation
- Official Guickstart
  - Niagara

1.3. Notes

- All clusters of Compute Canada shares almost the same procedure except for small difference in batch job scripts. See Links.
- If there are any errors, please search the official wiki first, then refer to technical support.

2. Setting up your environment

2.1. Getting a Compute Canada account

Before you begin, you will need to register for a Compute Canada (CC) account, which will allow you to log in to CC clusters. You need to sign up on CCDB , and you will need a supervisor (like your PhD supervisor) to sponsor your application. This process can take 2-5 days so it's best to do this as soon as possible.

If your supervisor does not have a CC account, they will first need to register using the same website. Once they have been approved you can apply for your own account. You will need to provide your supervisor's CCDB ID, which is usually in the form `abc-123`.

Specially, while you can also compute on Niagara with your Default RAP, you need to request access to this cluster. Go to this page, and click on "Join" next to Niagara and Mist.

2.2. SSH keys

Secure Shell (SSH) is a widely used standard to connect to remote servers in a secure way. SSH is the normal way for users to connect in order to execute commands, submit jobs, follow the progress of these jobs and in some cases, transfer files.

There are three different connections for which you need to set up SSH keys:

1. Between your local machine and your Compute Canada account (used for login)
2. Between your local machine and GitHub/GitLab (used for Git synchronization only, refer to the github tutorials)
3. Between your Compute Canada account and GitHub/GitLab (used for Git synchronization only, refer to the github tutorials)

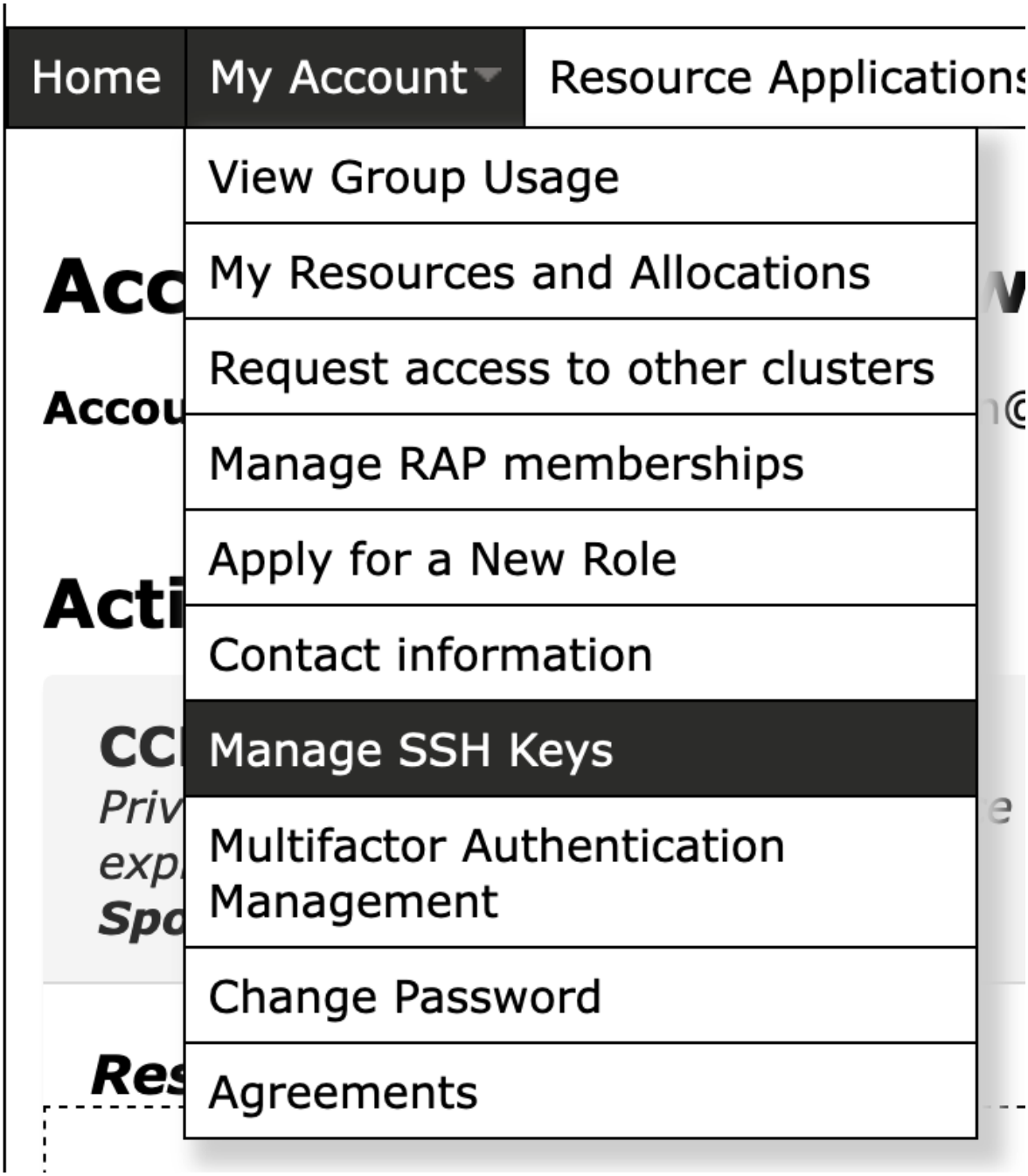
The steps for the three connections are the largely the same, and here I will outline them for the connection between your local machine and your Compute Canada account. More details can be found here.

1. Generate an SSH key on your local machine (you can skip this step if you have previously generated an SSH key)
    - Step 1: check if you have an existing SSH key on your local machine by looking at the following folder `~/.ssh (Linux or Macos)`, `C:\Users\username\.ssh (Windows)`. If you see two files which are named similarly to `id_ed25519` and `id_ed25519.pub` then you don't need to generate new keys (skip this step).
    - Step 2: If you don't have an existing key, generate one by doing (if `ssh-keygen` not work on Windows, please refer to Link):

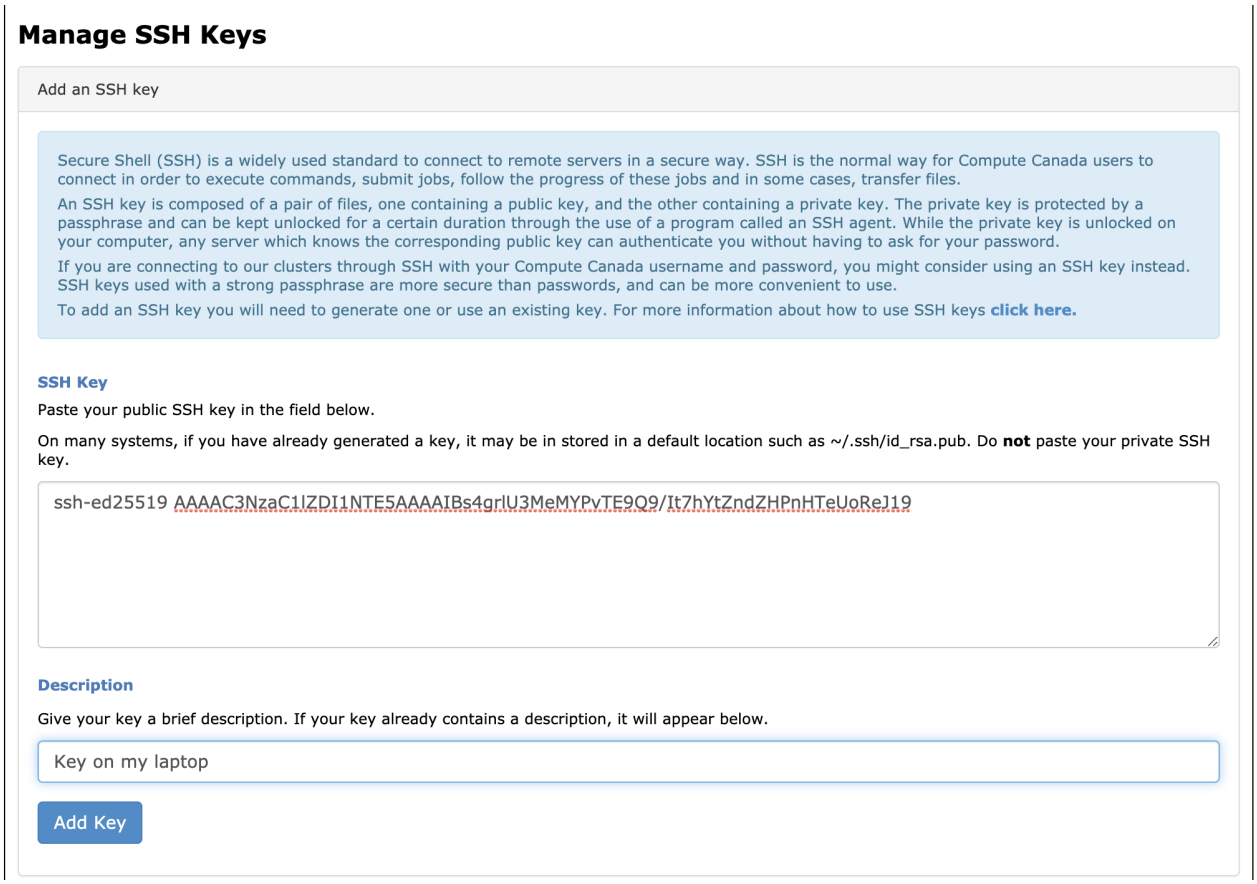
```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

2. Upload the SSH public key (e.g. `id_ed25519.pub`) to Compute Canada (Please keep your private key safely (e.g. `id_ed25519`)).

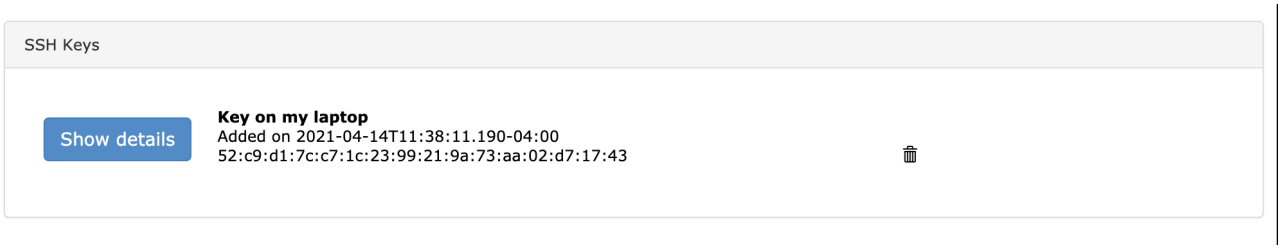
- STEP 1 - Go to the CCDB SSH page. Or via the CCDB menu:



- STEP 2 - Open your SSH public key with Notepad (e.g. `id_ed25519.pub`), copy the public key into CCDB form：



After clicking "Add Key" your SSH key will show up in the section below.



2.3. Logging into the systems

- Open a terminal window
- then SSH into the Niagara login nodes with your Compute Canada credentials:

```
ssh -i /path/to/ssh_private_key -Y username@niagara.scinet.utoronto.ca
```

or

```
ssh -i /path/to/ssh_private_key -Y username@niagara.computecanada.ca
```

- Example:

```
ssh -i C:\Users\username\.ssh\id_ed25519 -Y
username@niagara.computecanada.ca
```

3. File systems and transfer

3.1. File systems

You have four kinds of directory on the Compute Canada clusters. Each cluster has different sizes and rules for each kind of directory. Please refer to the wiki page of each cluster for more details. Generally speaking, you are recommended to use `Project` directory to develop your project.

Their paths on the Nigara cluster are:

```
$HOME=/home/g/groupname/username
$SCRATCH=/scratch/g/groupname/username
$PROJECT=/project/g/groupname/username
$ARCHIVE=/archive/g/groupname/username
```

3.2. File transfer

1. use scp to copy files directories

```
scp file username@niagara.computecanada.ca:/path
scp username@niagara.computecanada.ca:/path/file localPath
```

2. For windows, `WinSCP` can be used for file transfer

3.3. Git for codes

The optimal way to 'transfer' codes such as Julia files, Jupyter notebooks, and generally any text-based files, is to use Github. You should commit and push changes from your local machine and then pull them on Compute Canada to ensure you have the latest version of your code (and vice-versa if you are editing on Compute Canada and want your local machine to have the latest version).

Please refer to the tutorials from Compute Canada and Github.

4. Software modules

4.1. Loading software modules

You have two options for running code on Niagara: use existing software, or compile your own.

To use existing software, common module subcommands are:

- `module load <module-name>`: load the default version of a particular software.
- `module load <module-name>/<module-version>`: load a specific version of a particular software.
- `module purge`: unload all currently loaded modules.
- `module spider` (or `module spider <module-name>`): list available software packages.
- `module avail`: list loadable software packages.

- `module list`: list loaded modules.

To compile your own, please refer to the official wiki, such as CPLEX, Gurobi.

4.2. Setting up Julia enviromnent

Here is the typical steps to setup the Julia enviroment:

```
module load julia
julia
# In Julia:
julia> using Pkg
julia> Pkg.add('Julia module')
```

This is a one-time operation. Once you have added all the Julia modules, you can use then from now on by only typing `module load julia`.

- Specially, if you want to use your own Julia modules, you need to precompile your own Julia modules before submitting the jobs by:

```
julia pre_load.jl
```

in which the `pre_load.jl` should be like:

```
# This is the file for precompiling self-created packages in niagara
system

if !("src/" in LOAD_PATH)
    push!(LOAD_PATH, "src/") # path to your-own-module
end

using you-own-module1, you-own-module2, ...
```

5. Submitting jobs

5.1. Basic rules

As described at the very beginning, Niagara is a cluster, which is a collection of nodes (computers) which are connected to each other. When you log into Niagara, you are logging into a 'head' node or a 'log-in' node. This node itself does not have the resources (memory, CPU, GPU) to run computations that are even a little demanding. It is not appropriate to run any computations on the log-in node, and this is actively discouraged.

There are several hundreds of compute nodes on which you are allowed to run jobs, however to do this you need to submit your program/script as a 'job'. In addition to providing the command necessary to run your

job (e.g. `julia testing.jl`), you also need to provide details of what types of resources you need to use and for how long, so the scheduling system on Niagara can appropriately allocate resources for your script.

Niagara uses `SLURM` as its job scheduler. More-advanced details of how to interact with the scheduler can be found on the Slurm page.

5.2. Job scripts

Job scripts are terminal scripts (e.g. `xxx.sh`) to describe the resourses you need for your job and the real running command.

Frequently used SLURM options

```
#SBATCH --time=0-05:00 # walltime in d-hh:mm or hh:mm:ss format
#SBATCH --ntasks=X # request X tasks; with cpus-per-task=1 (the default)
this requests X cores #SBATCH --nodes=X # request a minimum of X nodes
#SBATCH --nodes=X-Y # request a minimum of X nodes and a maximum of Y
nodes
#SBATCH --cpus-per-task=X # request a minimum of X CPUs per task
#SBATCH --tasks-per-node=X # request a minimum of X tasks be allocated per
node
#SBATCH --output=name%j.out # standard output and error log
#SBATCH --error=name.err # standard error log
#SBATCH --mem=2000 # request 2000 MB of memory in total
#SBATCH --mem-per-cpu=2000 # request 2000 MB of memory per CPU #SBATCH --
gres=gpu:1 # request 1 GPU per node
```

Example of job scripts

```
#! /bin/bash
#SBATCH --nodes=1 # number of nodes request
#SBATCH --ntasks-per-node=40 # number of tasks per node
#SBATCH -t 0-06:00 # maximum running time request, d-hh:mm
#SBATCH --output=%j-%N.out # the print of xxx.jl will be logged in this
file, %N for node name, %j for job id

Module load …
julia xxx.jl > xxx.log # run xxx.jl and print the result in xxx.log
```

After creating the terminal script (e.g. `xxx.sh`), type the following command in the terminal:

```
sbatch xxx.sh # submit the job
```

5.3. Interactive jobs

Interactive jobs are also supported, and you can do data exploration, software development/debugging at the command line:

```
salloc --time=1:0:0 --nodes=1 --ntasks-per-node=15 --mem-per-cpu=4gb
```

5.4. Demo

Here is a demo of model predictive control problem in Julia, **please setup the Julia environment before running this demo**.

Here is the main Julia file `MPC.jl`:

```
#import Pkg
#Pkg.add("JuMP")
#Pkg.add("Ipopt")

using JuMP
using Ipopt

# user defined module
if !("." in LOAD_PATH)
    push!(LOAD_PATH, ".") # path to your-own-module
end
using process_model

# optimal control
function OptimalControl(x0)
    m=Model(Ipopt.Optimizer)
    @variable(m, x[i in 1:2, t in 0:(N)])
    @variable(m, u[i in 1:2, t in 0:(N-1)])

    @constraint(m, [i in 1:2], x[i,0] == x0[i])
    #x1_plus = 2*x[1] + 1*x[2] + 1*u[1]
    @constraint(m, [t in 0:N-1], x[1, t+1] == 2*x[1,t] + 1*x[2,t] +
1*u[1,t])
    #x2_plus = 0*x[1] + 2*x[2] + 1*u[2]
    @constraint(m, [t in 0:N-1], x[2, t+1] == 0*x[1,t] + 2*x[2,t] +
1*u[2,t])

    @constraint(m, [t in 0:N-1], -5<= x[1, t] <=5 )
    @constraint(m, [t in 0:N-1, i in 1:2], -1<= u[i, t] <=1 )

    @constraint(m, -.1<= x[1,N] <=.1 )
    @constraint(m, -.1<= x[2,N] <=.1 )

    @objective(m, Min, 0.5*sum(alpha*x[i,t]^2 + u[i,t]^2 for t in 0:(N-1),
i in 1:2))

    JuMP.optimize!(m)
    return [JuMP.value(u[1,0]),JuMP.value(u[2,0])]
```

```
end

# main
N=3
N_sim = 30
alpha=2

x = zeros(Float64,2,N_sim+1)
u = zeros(Float64,2,N_sim)
x[:,1] = [0.5, 0.5]
for t = 1:N_sim
    println("################################################")
    println("Simulation time: ", t, " / ", N_sim, "")
    u[:,t] = OptimalControl(x[:,t])
    x[:,t+1] = StepModel(x[:,t], u[:,t])
    println("x[:, ", t+1, "] = ", x[:,t+1], ", ", "u[:, ", t, "] = ",
u[:,t])
    if t == 15
        x[:,t+1] += [0.5, 0.1]
    end
    sleep(3)
end
println("#####################final result#####################")
println(u)

# plot result
using Plots
l = @layout [a; b]
p1 = plot(x[1,:], label="x 1")
plot!(p1, x[2,:], label="x 2")
p2 = plot(u[1,:], label="u 1")
plot!(p2, u[2,:], label="u 2")
plot(p1, p2, layout = l)
savefig("result.png")
```

Here is the module `process_model.jl`:

```
module process_model

export StepModel

function StepModel(x, u)
    x1_plus = 2*x[1] + 1*x[2] + 1*u[1]
    x2_plus = 0*x[1] + 2*x[2] + 1*u[2]
    return [x1_plus, x2_plus]
end

end
```

Here is the precompiling file `pre_load.jl`:

```
# This is the file for precompiling self-created packages in niagara
system

if !("." in LOAD_PATH)
    push!(LOAD_PATH, ".") # path to your-own-module
end

using process_model
```

Here is the job scripts `runjob.sh`:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH -t 0-01:00
#SBATCH --output=%N-%j.out

module load julia
julia MPC.jl > MPC.log
```

Here is the whole procedure to submit the job, **please setup the Julia environment before running this demo**:

```
module load julia
# if you have you-own-Julia-modules, julia pre_load.jl
cd /path-to-runjob.sh
sbatch runjob.sh
```

6. Monitoring jobs

The most basic way to monitor your jobs is to use the `squeue` command. This will tell you whether your CC job has started or is queued, as well as how much time is left if it has started:

```
squeue -u username # list all current jobs
jobperf jobid # more detailed job infomation
watch -d -n 15 'squeue -u username; jobperf jobid' # automated job
monitoring command
```