Snakemake on NeSI: A short tutorial on using Snakemake on NeSI HPC

Release 2019.07

Sebastian Schmeier (https://sschmeier.com)

Jul 11, 2019

CONTENTS

CHAPTER

ONE

A SHORT TUTORIAL ON USING THE WORKFLOW ENGINE SNAKEMAKE ON THE NESI HPC CLUSTER

This is an introductory tutorial for working with Snakemake[1] on NeSI high-performance computer cluster. This tutorial assumes that you have a basic knowledge of Snakemake[2] and Bioconda[3] to develop your bioinformatics workflows.

Note: Get a quick introduction to conda[4] and Snakemake[5] at https://reproducibility.sschmeier.com.

Here, you will learn how to configure the NeSI computing cluster and your workflow to run all jobs that your workflow needs to run with the Slurm[6] system.

Currently, Dr. Sebastian Schmeier[7] is teaching this material at Massey University in Auckland, New Zealand. More information about other bioinformatics material, tutorials, and our research can be found on the webpages of the Schmeier Group[8] (https://sschmeier.com).

Note: A online version of this tutorial can be accessed at https://snakemake-on-nesi.sschmeier.com.

1.1 Slurm primer

Slurm[9] is an open source cluster management and job scheduling system for large and small Linux clusters and the job scheduler of choice on the NeSI HPC cluster.

Whenever you want to run several jobs, you need to submit your jobs to a **job queue**. Jobs in the queue are executed on the cluster based on their **priority**. The job priority depends on the hardware resources your job needs. If you need a lot of resources, your job will get a low priority and likely has to wait a while until it gets executed.

To be able to get a good priority, you need to know some particulars about your individual jobs. The most important information you need to know for each job are:

- Time your job will likely run

[1] http://snakemake.readthedocs.io/en/latest/
[2] http://snakemake.readthedocs.io/en/latest/
[3] https://bioconda.github.io/
[4] http://conda.pydata.org/miniconda.html
[5] http://snakemake.readthedocs.io/en/latest/
[6] https://slurm.schedmd.com/man_index.html
[7] https://sschmeier.com
[8] https://sschmeier.com
[9] https://slurm.schedmd.com/man_index.html

- Number of CPU your job needs
- Amount of memory your job needs

To be as close as possible to the real requirements in your estimates is essential to get a good **priority** in the queue.

Note: Often you do not know how much time your jobs need to run. In this instance it is good to schedule only one or a few jobs and submit them with more resources they likely are using, e.g. 10 minutes instead of the real 2 minutes and see once the job is done, how long it took. Then you can submit all jobs with a better estimated time.

1.1.1 Slurm partitions

A partition in the Slurm[10] system is a set of computing resources that are bundled and can be accessed separately, ie by sending your jobs to a particular queue of the partition. Mahuika (the NeSI cluster you are likely going to use) has several different partitions configured, see here[11].

Depending on the answers to those requirements for your jobs above, you need to submit your jobs to the correct Slurm[12] partition queue.

For example, the main partition on Mahuika is called *"large"*. It has many computing nodes (226) with many CPUs each (72), but each node only has 108GB of memory available. As one of your jobs will like run on one node only, we could allocate a maximum of 108GB of memory to that job. However, on *large* we should not aim to use more than 1.5GB/core that our job uses. For example, if we would run a job with 10 cores, we should not allocate more than 15GB to that job. This is done so that the other unused cores on that node still have enough memory resources available to be used and are not "wasted". Another requirement for the *large* partition is that a job can run a maximum allowed time of 3 days. This means if you have many short jobs that do not need much memory, this is the partition to use. It also means though if you have a job that needs lets say 50GB of memory and uses few cores or would run more than 3 days, this is the wrong partition to use.

Check the different partitions at the link above.

We can see the jobs currently submitted to the queue of a particular partition (e.g. here "**large**") with the command `squeue -p large`. To only see the jobs of a particualr user you could type `squeue -p large -u username`.

1.1.2 Submitting a job with `sbatch`

To submit a job to the cluster one would generally write a job script and submit this script via the Slurm[13] program `sbatch` (see here[14]).

A job script is a relative simple affair, for example the following script (e.g. `test.sh`) would submit a python script (`test.py`) to the partition *"large"*:

Listing 1.1: : Simple bash script (`test.sh`) for slurm cluster submission.

```
#!/bin/bash
#SBATCH --account=yourProjectId       # The project id given to you by NeSI
#SBATCH --partition=large             # The partition we want to use
#SBATCH --ntasks=1                    # Run on a single CPU
```

(continues on next page)

[10] https://slurm.schedmd.com/man_index.html
[11] https://support.nesi.org.nz/hc/en-gb/articles/360000204076-Mahuika-Slurm-Partitions
[12] https://slurm.schedmd.com/man_index.html
[13] https://slurm.schedmd.com/man_index.html
[14] https://slurm.schedmd.com/sbatch.html

(continued from previous page)

```
#SBATCH --mem=1g                    # Job memory request
#SBATCH --time=00:05:00             # Time limit hrs:min:sec
#SBATCH --output=logs/%x-%j.out     # Standard output log
#SBATCH --error=logs/%x-%j.err      # Standard error log

module load Miniconda3/4.4.10
echo "Running python script on a single CPU"
python test.py
```

This script could be submitted to the queue of partition "*large*" with the command: `sbatch test.sh`.

The only particular about this bash-script are the lines that start with `#SBATCH`. These tell `sbatch` to use the specified `sbatch` parameters, without us using them on the command-line like so:

```
$ sbatch --account=yourProjectId
         --partition=large
         --ntasks=1
         --mem=1g
         --time=00:05:00
         --output=logs/%x-%j.out
         --error=logs/%x-%j.err
         test.sh
```

Note: We aim at using Snakemake[15] for job submission, thus we are not going to use any of these methods to submit our jobs. However, we need to understand the parameters of `sbatch` as well as the different partitions, as we need to specify them as well.

1.2 Configuring NeSI

Here we assume you created an NeSI account and you have a project allocation and an project code.

First, we need to configure NeSI in a way that we can run Snakemake[16] on Mahuika (the cluster we are going to use for our analyses). This involves two steps:

1. Load the conda[17] module
2. Install Snakemake[18] into an environment

Attention: As of June 2019 a Snakemake[19] module exists on NeSI that we can simply load with `module load snakemake/5.5.0-gimkl-2018b-Python-3.7.3`. Once loaded, the steps below are not needed anymore. They are still displayed here in case you need a particular Snakemake[20] version installed.

1.2.1 Loading a module

A module on the cluster includes a software package into our path so that we are able to make use of the software. To load a module on the command-line, you only need to type:

[15] http://snakemake.readthedocs.io/en/latest/
[16] http://snakemake.readthedocs.io/en/latest/
[17] http://conda.pydata.org/miniconda.html
[18] http://snakemake.readthedocs.io/en/latest/
[19] http://snakemake.readthedocs.io/en/latest/
[20] http://snakemake.readthedocs.io/en/latest/

```
$ module load Miniconda3/4.4.10
```

Once executed, the conda command should be readily available to you.

1.2.2 Loading conda[21] persistently

To have the conda[22] module loaded every time we open a new shell, you can add two lines to the end of your ~/.bashrc shell configuration file:

Listing 1.2: .bashrc

```
...
module load Miniconda3/4.4.10
. /opt/nesi/mahuika/Miniconda3/4.4.10/etc/profile.d/conda.sh
```

1.2.3 Installing Snakemake[23]

We will be using the workflow management software Snakemake[24] here. We will create a conda[25] environment for Snakemake[26]:

```
$ conda create -n my-base snakemake>5.4.0
```

Conda will create a .conda folder in your home directory and store environments in .conda/envs by default.

Install conda[27] channels to be able to search for bioinformatics software:

```
$ conda config --add channels defaults
$ conda config --add channels bioconda
$ conda config --add channels conda-forge
```

Note: On your own system you probably would install Snakemake[28] in the *base* conda environment and not create a new environment specifically for Snakemake[29]. However, on NeSI we are not allowed to write into the directory of the *base* environment, hence we create a new one.

1.2.4 Automatically loading the environment

We will append one more line to our .bashrc file to have the my-base environment loaded as a standard when we open a new shell. The end of the .bashrc file now looks like:

Listing 1.3: .bashrc

```
...
module load Miniconda3/4.4.10
. /opt/nesi/mahuika/Miniconda3/4.4.10/etc/profile.d/conda.sh
conda activate my-base
```

[21] http://conda.pydata.org/miniconda.html
[22] http://conda.pydata.org/miniconda.html
[23] http://snakemake.readthedocs.io/en/latest/
[24] http://snakemake.readthedocs.io/en/latest/
[25] http://conda.pydata.org/miniconda.html
[26] http://snakemake.readthedocs.io/en/latest/
[27] http://conda.pydata.org/miniconda.html
[28] http://snakemake.readthedocs.io/en/latest/
[29] http://snakemake.readthedocs.io/en/latest/

Now, whenever we log into NeSI or opening a new shell, the conda[30] module and our environment will be loaded. To make the changes available in your current shell session, type `source ~/.bashrc`.

1.3 Snakemake setup

The example below will make use of a workflow and dataset I prepared for teaching purposes. It can be accessed at https://gitlab.com/schmeierlab/reproduce-tutorial.

Now the `snakemake` command should be available.

1.3.1 Getting your workflow and data onto NeSI

You could of course just `scp` or `rsync` your workflow over to the Mahuika cluster. However, I highly recommend that you use Git[31] and a remote provider like GitLab[32] or GitHub[33] for this purpose. You should version control your workflow in any case, thus an additional step of using a remote is not much more of an effort.

Once you have your workflow securely stored on a remote location getting it downloaded to NeSI is as easy as typing the following command (adjusting the URL to your workflow):

```
$ git clone https://gitlab.com/schmeierlab/reproduce-tutorial.git

# or if ssh has been set up
$ git clone git@gitlab.com:schmeierlab/reproduce-tutorial.git
```

This workflow comes with some example data, so no need to up/download data for it separately.

However, for your own data you might want to use `scp` to copy it to NeSI from your local computer. If its a lot of data you might want to use the *not backup'ed* section of your project for this purpose. This can be done easily with `scp` from your local machine, e.g. a variation of:

```
$ scp -r data username@login.mahuika.nesi.org.nz:/nesi/nobackup/yourprojectcode/
```

Note: See here for more explanation of data transfer to/from NeSI via scp[34] or Globus[35].

1.3.2 An example workflow

Download the workflow

Here, we are going to use the example workflow mentioned in the last section to illustrate how to set up your workflow for running it on NeSI. You do not have to use this example workflow but can adjust your workflow based on the example below. Move to your project's persistent folder and download it, if you have not already, with:

```
# change to folder
$ cd /nesi/project/yourprojectcode/

$ git clone https://gitlab.com/schmeierlab/reproduce-tutorial.git
```

(continues on next page)

[30] http://conda.pydata.org/miniconda.html
[31] https://git-scm.com/
[32] https://gitlab.com/
[33] https://github.com/
[34] https://support.nesi.org.nz/hc/en-gb/articles/360000578455-File-Transfer-with-SCP
[35] https://support.nesi.org.nz/hc/en-gb/articles/360000576776-File-Transfer-with-Globus

(continued from previous page)

```
# or if ssh has been set up
$ git clone git@gitlab.com:schmeierlab/reproduce-tutorial.git
```

Note: If the cloning with Git[36] does not work, you can download a zipped archive of the whole repository here https://gitlab.com/schmeierlab/reproduce-tutorial/-/archive/master/reproduce-tutorial-master.zip. The locally on the command-line, you can type `unzip reproduce-tutorial-master.zip; mv reproduce-tutorial-master reproduce-tutorial`.

Inspect and setup the workflow

Let us inspect what we just downloaded:

```
# After successful download we can inspect it with
$ cd reproduce-tutorial
$ ls -1F
data/
envs/
examples/
fastq/
help/
LICENSE
logs/
README.md
Snakefile
```

This is data from an RNA-seq experiment. We want to map the single end fastq-files in the folder `fastq` to the yeast reference genome in the folder `data`. The `Snakefile` is empty, as its being developed throughout the Reproducibility tutorial[37]. We replace this file with the final version of workflow like this:

```
$ rm Snakefile
$ cp examples/Snakefile_v7 Snakefile
```

The workflow consists of four rules that will trim the data, build an genome index, map the fastq-files to the genome and count reads per feature in the genome.

The workflow makes use of a few programs that can be installed on the fly by Snakemake[38] when using the `--use-conda` parameter for Snakemake. However, I found that for large workflows that uses many environments, it will create many many files in the local `.snakemake` directory and you may soon reach your file number limit on the cluster. Thus, here, we are going to use one environment for the whole workflow that we are going to create upfront with:

```
# create env
$ conda env create -n tutorial -f data/nesi/cluster-nesi-condaenv.yaml

# activate env
$ conda activate tutorial
```

Slurm and NeSI specific setup

In order to have Snakemake[39] distribute our jobs to the cluster, Snakemake will create automatically one job script per job it needs to run and submit it to the cluster. However, in order to do so, Snakemake

[36] https://git-scm.com/
[37] https://reproducibilty.sschmeier.com
[38] http://snakemake.readthedocs.io/en/latest/
[39] http://snakemake.readthedocs.io/en/latest/

needs some information about the sbatch parameters it should use. We are going to do two things to run the workflow on the cluster:

- Setup a config-file that specifies sbatch parameters per rule
- Use special Snakemake parameters to make use of these sbatch parameters

Lets look at the second part first. The command structure for Snakemake will look like this:

```
$ snakemake --jobs 999
            --printshellcmds
            --rerun-incomplete
            --cluster-config data/nesi/cluster-nesi-mahuika.yaml
            --cluster "sbatch --account={cluster.account}
                              --partition={cluster.partition}
                              --mem={cluster.mem}
                              --ntasks={cluster.ntasks}
                              --cpus-per-task={cluster.cpus-per-task}
                              --time={cluster.time}
                              --hint={cluster.hint}
                              --output={cluster.output}
                              --error={cluster.error}"
```

Attention: If you would want a per rule environment run you can specify the parameter --use-conda. However, due to the above mentioned reason we are not using this here.

There are two parameters in the command that are cluster-specific, let's have a look:

- --cluster: This parameter specifies the sbatch submit command that Snakemake will use. It contains some wildcards (like the ones used within the Snakefile). These wildcards will be replaced during job submission with rule-specific values. The values per rule will be specified in the cluster config file: data/nesi/cluster-nesi-mahuika.yaml in the above example.
- --cluster-config: This file contains the parameters for sbatch on a per rule basis.

Let us have a look at the cluster config-file, first entry:

Listing 1.4: : A cluster file for the NeSI Mahuika cluster: data/nesi/cluster-nesi-mahuika.yaml

```
1  __default__:
2     account: yourprojectcode
3     time: 00:10:00
4     ntasks: 1
5     cpus-per-task: 1
6     mem: 1500m
7     partition: large
8     hint: nomultithread
9     output: logs/%x-%j.out
10    error: logs/%x-%j.err
```

The first entry (__default__) shown here is the default entry that Snakemake will use for all rules except if parameters are overwritten in rule-specific entries in this file.

For example, the third rule in the Snakefile, called map is mapping reads to the genome using bwa mem. Here, bwa mem can make us of multiple cores at the same time to speed up things, e.g. we could decide to use 8 cores. Thus, we could specify multithreading and on the *large* partition we could allocate up to 8*1.5GB = 12GB of memory to the job. If we want more memory we would need to move to another partition, e.g. *bigmem*.

For example purposes, I want that the job runs with 20GB on the *bigmem* partition. We specify in the cluster config-file an entry for the rule and overwrite the parameter specifying the multithreading option (line 16) and specifying the *bigmem* partition (line 14). The entry only needs to specify the parameters

we want to change from the `__default__` entry, e.g. we also specify that this rule is allowed to use more time, here 20 minutes (line 13), than the default of 10 minutes. Finally, we specify the memory (line 17) and number of cores the job should be using (line 15).

Listing 1.5: : A cluster file for for NeSI Mahuika cluster: `data/nesi/cluster-nesi-mahuika.yaml`

```
1  __default__:
2     account: yourprojectcode
3     time: 00:10:00
4     ntasks: 1
5     cpus-per-task: 1
6     mem: 1500m
7     partition: large
8     hint: nomultithread
9     output: logs/%x-%j.out
10    error: logs/%x-%j.err
11
12 map:
13    time: 00:20:00
14    partition: bigmem
15    cpus-per-task: 8
16    hint: multithread
17    mem: 20g
```

Similarly, we add two more entries for rules `makeidx` and `featurecount` (lines 19 and 22). The `featurecount` rule can make use of multiple cores as well, thus we add parameters here as well to change this. However, we want them to be past to the *large* partition, so we keep the default (which we do not have to specify explicitly again).

Listing 1.6: : A cluster file for the NeSI Mahuika cluster: `data/nesi/cluster-nesi-mahuika.yaml`

```
1  __default__:
2     account: yourprojectcode
3     time: 00:10:00
4     ntasks: 1
5     cpus-per-task: 1
6     mem: 1500m
7     partition: large
8     hint: nomultithread
9     output: logs/%x-%j.out
10    error: logs/%x-%j.err
11
12 map:
13    time: 00:20:00
14    partition: bigmem
15    cpus-per-task: 8
16    hint: multithread
17    mem: 20g
18
19 makeidx:
20    time: 00:20:00
21
22 featurecount:
23    time: 00:20:00
24    cpus-per-task: 4
25    hint: multithread
26    mem: 6g
```

The `__default__` specifies that we do not allow multi-threading (`hint: nomultithread`), we change this for e.g. rule `map` to `hint: multithread` and add the `cpus-per-task` parameter, hence allowing up to 8

CPUs being reserved on the "*bigmem*" partition by jobs that use this rule. Just remember that you still need to specify the `threads: 8` in the `Snakefile` for the rule `map`, and make the threads available to `bwa mem` in the shell command (see below lines 53 and 59).

Listing 1.7: : Excerpt from the Snakefile, showing the bwa multi-threading setup.

```
43 rule map:
44     input:
45         reads = "analyses/results/{sample}.trimmed.fastq.gz",
46         idxdone = "data/makeidx.done"
47     output:
48         "analyses/results/{sample}.bam"
49     log:
50         "analyses/logs/{sample}.map"
51     benchmark:
52         "analyses/benchmarks/{sample}.map"
53     threads: 8
54     conda:
55         "envs/map.yaml"
56     params:
57         idx = "data/Saccharomyces_cerevisiae.R64-1-1.dna_sm.toplevel.fa"
58     shell:
59         "bwa mem -t {threads} {params.idx} {input.reads} | "
60         "samtools view -Sbh > {output} 2> {log}"
```

Finally, when using the above command, Snakemake will create a job script for all jobs based on the rules and requested final targets in the `Snakefile` and submit them via `sbatch` with the correct configuration to the cluster.

1.4 Singularity (test phase)

On NeSI's Mahuika cluster you can invoke the Singularity[40] module with the command: `module load Singularity/.3.2.1`. Note the "." in front of the *3.2.0*, which means it is a hidden module, as it is still in its test phase. You could add this line to the end of your `.bashrc` file to have it always loaded when a shell is opened.

1.4.1 Creating a Singularity image

You can create a singularity image on a machine where Singularity[41] is installed and **you have root privileges**. This means that you will not be able to build any images on the NeSI cluster. You can install Singularity[42] on your own machine where you have root privileges.

An example Singularity[43] image configuration file is depicted below. It makes use of a conda[44] base image pulled from Docker[45] and installs some bioinformatics tools via the Bioconda[46] channel into the base image.

[40] http://singularity.lbl.gov/
[41] http://singularity.lbl.gov/
[42] http://singularity.lbl.gov/
[43] http://singularity.lbl.gov/
[44] http://conda.pydata.org/miniconda.html
[45] https://www.docker.com/
[46] https://bioconda.github.io/

Listing 1.8: : Singularity config-file for the biotools container used in the tutorial.

```
# Filename: Singularity
Bootstrap: docker
From: continuumio/miniconda3

%labels
   AUTHOR schmeier@...

# This sets global environment variables for anything run within the container
%environment
  export PATH="/opt/conda/bin:/usr/local/bin:/usr/bin:/bin:"
  unset CONDA_DEFAULT_ENV
  export ANACONDA_HOME=/opt/conda

%post
   export PATH=/opt/conda/bin:$PATH
   echo "We add conda channels."
   conda config --add channels defaults
   conda config --add channels bioconda
   conda config --add channels conda-forge
   echo "We install tools."
   conda install --yes bwa=0.7.15 sickle-trim=1.33 subread=1.6.1 samtools=1.8
   conda clean --index-cache --tarballs --packages --yes
```

Once you are confident the image builds correctly and the tools you need are indeed working, you can create a GitHub[47] repository and upload the container config-file like the one above. Then you can connect the repository to Singularity Hub[48] to have the container build on the Singularity Hub[49] systems, which is then ready for download, wherever you want to use it. The GitHub[50] repository associated to the Singularity Hub[51] image in this example is accessible at: https://github.com/sschmeier/biotools

1.4.2 Snakefile changes

We need to change the Snakefile to include a directive that Snakemake[52] knows which Singularity[53] image to use to run the workflow. Here, we specify the URL to the image location on Singularity Hub[54] (see https://www.singularity-hub.org/collections/1107). In addition, we need to supply the correct parameter to the snakemake command upon execution (see below, line 1).

Listing 1.9: Snakefile changes to include the singularity directive.

```
singularity: "shub://sschmeier/biotools:latest"

SAMPLES, = glob_wildcards("fastq/{sample}.fastq.gz")

rule all:
    input:
        "analyses/results/counts.txt"

rule trimse:
    input:
        "fastq/{sample}.fastq.gz"
    output:
```

(continues on next page)

[47] https://github.com/
[48] https://www.singularity-hub.org
[49] https://www.singularity-hub.org
[50] https://github.com/
[51] https://www.singularity-hub.org
[52] http://snakemake.readthedocs.io/en/latest/
[53] http://singularity.lbl.gov/
[54] https://www.singularity-hub.org

(continued from previous page)

```
        "analyses/results/{sample}.trimmed.fastq.gz"
    log:
        "analyses/logs/{sample}.trimse"
    benchmark:
        "analyses/benchmarks/{sample}.trimse"
    conda:
        "envs/sickle.yaml"
    params:
        qualtype="sanger"
    shell:
        "sickle se -g -t {params.qualtype} -f {input} -o {output}"
        " 2> {log}"
...
```

1.4.3 Execute Snakemake in Singularity-mode

Execute Snakemake[55] in Singularity[56]-only mode:

```
$ snakemake --use-singularity
            --singularity-args "--bind path/to/include"
            --jobs 999
            --printshellcmds
            --rerun-incomplete
            --cluster-config data/nesi/cluster-nesi-mahuika.yaml
            --cluster "sbatch --account={cluster.account}
                              --partition={cluster.partition}
                              --mem={cluster.mem}
                              --ntasks={cluster.ntasks}
                              --cpus-per-task={cluster.cpus-per-task}
                              --time={cluster.time}
                              --hint={cluster.hint}
                              --output={cluster.output}
                              --error={cluster.error}"
```

With the parameter `--use-singularity` we instruct Snakemake[57] to use Singularity[58]. None of the conda[59] directives in the rules will be used. The container image will be downloaded on first execution of Snakemake[60] and will be store in the workflow `.snakemake` subdirectory for future runs of the workflow. The big advantage using Singularity[61] is that all the tools and their versions as well as their dependencies are fixed upfront as is the operating system. In this way, we are not dependent on the cluster environment operating system, installed tools, etc. It is a great way to achieve better reproducibility of your work.

Attention: If you want to read or store data on a particular disk then the one you are executing Snakemake[62] from, you need to supply `--singularity-args "--bind path/to/include"` so that Singularity[63] can read and write from that location. I tend to include it anyways to be sure.

[55] http://snakemake.readthedocs.io/en/latest/
[56] http://singularity.lbl.gov/
[57] http://snakemake.readthedocs.io/en/latest/
[58] http://singularity.lbl.gov/
[59] http://conda.pydata.org/miniconda.html
[60] http://snakemake.readthedocs.io/en/latest/
[61] http://singularity.lbl.gov/
[62] http://snakemake.readthedocs.io/en/latest/
[63] http://singularity.lbl.gov/