TCML Cluster Documentation

Table of contents

1 Introduction

This document is a comprehensive guideline for using the GPU cluster of the BMBF funded project Training Center for Machine Learning (TCML) at the Eberhard Karls Universität Tübingen. The GPU cluster is administered by the Cognitive Systems group in the Department of Computer Science (Wilhelm-Schickard-Institute). The following research groups are project partners in the TCML project and their members and students have the right to use the cluster for their research:

1. PD Dr. Philipp **Berens** (Neural Data Science)
2. Prof. Dr. Martin **Butz** (Cognitive modelling)
3. Jun.-Prof. Dr. Enkelejda **Kasneci** (Perception Eng.)
4. Prof. Dr. Hendrik **Lensch** (Computergraphik)
5. Prof. Dr. Ulrike **von Luxburg** (Machine Learning)
6. Prof. Dr. Kay **Nieselt** (Integrative Transcriptomics)
7. Prof. Dr. Nico **Pfeifer** (Medical Informatics)
8. Prof. Dr. Wolfgang **Rosenstiel** (Computer Eng. , BCI)
9. Prof. Dr. Andreas **Schilling** (Media Inf., Visual Comp.)
10. Prof. Dr. Felix **Wichmann** (Neural Inform. Processing)
11. Prof. Dr. Andreas **Zell** (Cognitive Systems) (***administrators***)

This cluster is intended to be used by various groups of people. The master and bachelor students working in the field of machine learning can benefit from the powerful computation capabilities in their practical assignments in courses and their thesis/projects research. It is also intended to be used for training courses on machine learning for participants from industry.

1.1 Structural Overview

The cluster consists of various nodes having different functionalities. Figure 1 below illustrates an overview of all the nodes and their communication among each other.

The head node: The head (master) node controls all the functionalities of the cluster in a centralized manner. This makes sure that the jobs (tasks) on the cluster are scheduled and monitored properly. The users are only allowed to access this node and submit their jobs here.
The compute nodes: There are 40 compute nodes on which all the computations are performed. Each compute node has the following hardware specifications:

- 2 TB SSD disk space
- 256 GB memory
- Intel XEON CPU E5-2650 v4
    - 2 sockets with12 cores each

    - 2.2 GHz
- 38 nodes have 4x GeForce GTX 1080 Ti GPUs
    - 11 GByte GDDR5X memory
    - 3584 cuda cores
    - Use `#--gres=gpu:1080ti:1` in the sbatch file to explicitly use this kind of gpu.
- 2 nodes have 4x RTX A4000 GPUs
    - 16 GByte GDDR6 memory
    - 6144 cuda cores
    - Use `#--gres=gpu:A4000:1` in the sbatch file to explicitly use this kind of gpu.

A workload manager (**Slurm**) schedules the jobs based on the requested resources, availability of resources and priority of tasks in an optimal way. Slurm allocates different processing nodes to different jobs in an intelligent manner.

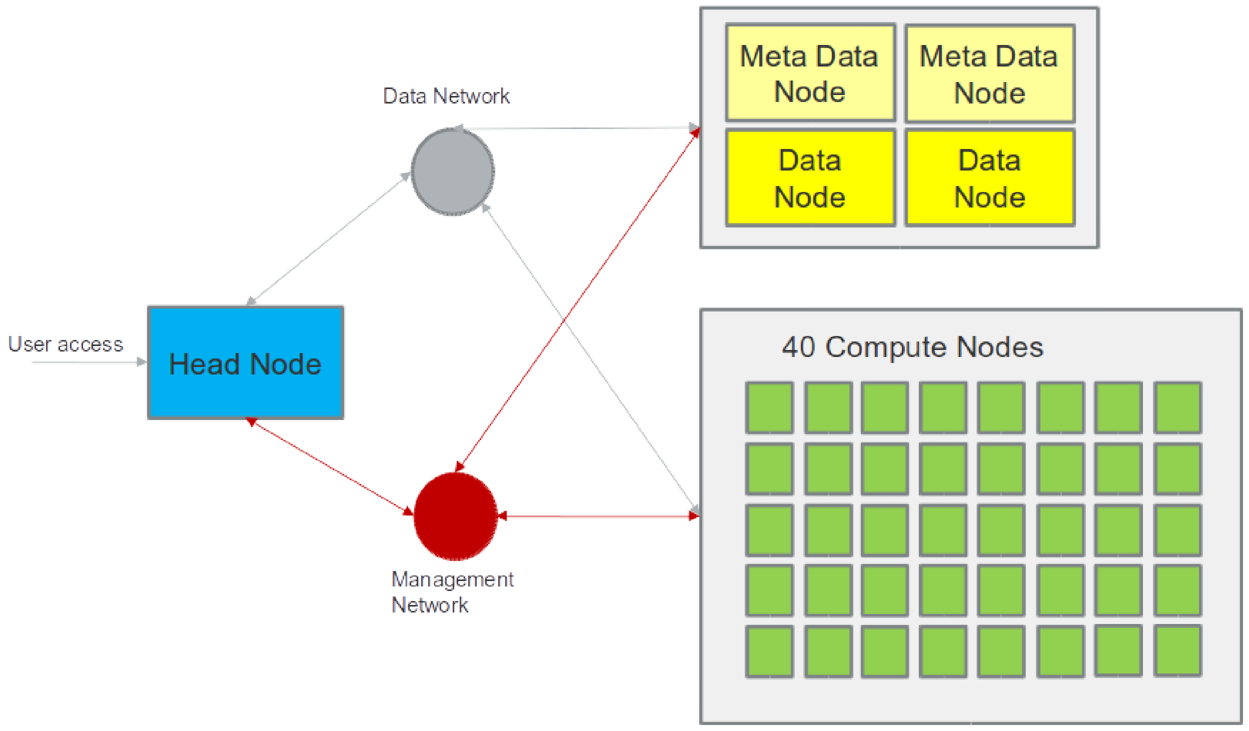


Figure 1: The structure of the TCML Cluster

The data nodes: In addition there are two data nodes which hold all data on the cluster. The hardware specifications of each data node is as follows:

- 73 TB disk space
- 135 GB memory
- Intel XEON CPU E5-2620 v4
    - 2 sockets with 8 cores each

    - 2.61 GHz

The cluster uses **BeeGFS** (a parallel cluster file system) to spread data across multiple nodes in an efficient way. It provides a virtualized file system over the two data nodes which can be accessed from the compute and the head nodes. The most important directories for the users are explained below.

1.2 Accessing the cluster

This section describes how to access the cluster and then gives a brief introduction to the most important directories accessible to everyone.

1.2.1 Logging in to the Head Node

To use the server, the users must login to the head node and execute their tasks from there.
The head node has the following IP address or alias:

IP address: *134.2.17.101*
Alias: *tcml-master01.uni-tuebingen.de*

To access the head node, use ssh in the following way:

```
$ ssh username@tcml-master01.uni-tuebingen.de
```

1.2.2 Important directories

'/home' : Home directories for all users. It is located on the storage nodes. The same home directory is used on all nodes. Users can keep their codes and data in these directories.

'/common'

'/datasets': Contains the most used machine learning data sets (explained below)

'/scripts': Contains useful scripts

'/share': Location to share data with other users

'/singularityImages': Contains various singularity images

'/userGuides': Contains a copy of this guide and additional information

'/scratch/*jobID*': Directory for temporary computing files. Located on the compute nodes.

1.2.3 Important notes on using the directories

- The same home directory is mounted on all nodes. In that way you can access your data on each node easily. Due to massively high network traffic, temporary computing files must be saved in the scratch folder of the node (/scratch/"jobID") and must NOT be saved in the home directory.
- Keep your home directory small. Delete all data you don't need anymore!

- **The home directory is not backuped!** So keep your important data safe manually.

1.2.4 Provided Datasets

In the ‘/common/datasets’ directory, we have the following machine learning (Computer vision) datasets already available:

- CIFAR-10
- City-scapes
- COCO_2017
- ImageNet_ILSVRC2012 (also in TFRecords format)
- Kitti
- MNIST
- Pascal VOC 2012

These datasets are in standard format. To read and use them in code, users have to implement the data reading pipeline themselves. It is also highly recommended to copy the datasets to the local scratch folder while training the models because the network traffic is very high if the data is read from the storage nodes constantly during training. Please remember to remove unnecessary data from ‘/home’ after your job has finished.

1.2.4 How to mount your cluster home directory on your local machine with sshfs:

```
sudo apt-get update
sudo apt-get install fuse
sudo apt-get install sshfs
```

Create local mount point:

```
e.g.mkdir ~/mnt/tcml
```

Mount the directory:

```
sshfs *your username*@tcml-master01.uni-tuebingen.de:/home/*your username* ~/mnt/tcml
```

Trust the key and enter your password

To unmount the directory use:

```
umount ~/mnt/tcml
```

Or use:

```
fusermount -u ~/mnt/tcml-cluster
```

You can create aliases for these commands in the `~/.bashrc` file on your local machine:

To do so, append the following lines:

```
alias mounttcmlcluster='sshfs
*yourusername*@tcml-master01.uni-tuebingen.de:/home/mutschler ~/mnt/tcml-cluster'
alias umounttcmlcluster='fusermount -u ~/mnt/tcml-cluster'
```

1.3 Backups

Backups of any directory are **not** done automatically. Use *rsync* manually and regularly to save your data.

1.4 How to apply for an User Account on the Cluster

If you need an account on the cluster, write an email with the following content to tcml-contact@listserv.uni-tuebingen.de. With this email you accept our privacy policy.:

*Subject: Account application *your name**

Body:
*First Name: *your first name**
*Last Name: *your last name**
*Department: *your department**
*Phone number: *your university phone number if available**
*Student ID: *your matriculation number if available**
*Research Group/ Chair: *your research group (aka chair)**
*Position: *Professor, Ph.D. student, university employee, student, ... **
*Linux User ID: *Optional. For compatible data transfer from other WSI,ZDV clusters it might be useful to provide your ZDV linux user ID here.* You can find the latter by using the 'id' command when logged in to your ZDV account.*
*Reason: *For what reason do you want to use the cluster? What exactly do you want to compute on it?**
*Comments: *your further comments, questions ...**

After applying successfully, you will get an email with your username and password.

Important notes:

- After receiving your password, log in to the cluster and change your password with the *passwd* command
- It is not allowed to use the account for anything else as specified in your given reason. If you use it for any other purpose your account will be removed.
- If any illegal actions or actions prohibited by the university were done, the university can take disciplinary action against the individual which might lead to exmatriculation in extreme cases.
- Members of research groups that are project partners in TCML project will get precedence. If you are collaborating with a TCML project partner, you can still apply for an account. However, we cannot guarantee an account for external users since it depends upon the current load of the cluster.

1.5 Privacy Policy

We use the data provided to us only to create a user account on the cluster and to contact you with issues concerning the cluster. We won’t share given data with anyone.
Administrators are allowed to delete your data after warning you beforehand. As long as it does not contradict to the former three sentences, the privacy policy of the Eberhard Karls Universität Tübingen holds.

2 The Software

2.1 The Workload Manager Slurm

Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. It allocates access of compute nodes to users for a particular duration of time. A user has to describe the job they want to run as well as the resources they need in a .sbatch file. To allocate the resources for a job and run it, the job has to be committed to a queue (called a partition in slurm) using the *sbatch* command. A job will be queued depending on its priority. The priority is determined by the amount of resources the job needs, the time it is waiting as well as the amount of resources the user and his group have used before. Further information about slurm can be found at:
https://slurm.schedmd.com/documentation.html

Note:

- We are running slurm-20.11
- A german tutorial of different slurm job types and how to use them can be found on the server at: /common/userGuides/Slurm-Jobguide-sysGen.pdf

2.1.1 Important Slurm commands

sbatch: Queues a job described by a .sbatch file
scancel <jobID>: cancels the job and removes it from the queue if you are the owner of the job
squeue: Provides an overview over all queued jobs. --start provides the estimates of job start times, if available.
sinfo: Provides an overview over the allocated nodes by each partition. Use -s for more detailed information about the nodes. Use -o %P,%D,%c,%X,%m,%f for information of how many resources are used in detail.
smap: Graphical combination of squeue and sinfo
sacct: Overview of jobs a user has run in on this day. Use -S 010118 to see all jobs of a user..
sshare: Provides a.o. the users fairshare factor. Which is mainly relevant for the users' jobs priority. Use -l to also get an overview of your used resources.
sprio <jobID>: Shows the priority of a job.

2.1.2 Parameters of the .sbatch file:

The following provides an overview over the most important .sbatch file parameters which are important for our cluster. A more detailed overview can be found at:
https://slurm.schedmd.com/archive/slurm-20.11-latest/sbatch.html

--job-name=_*the job name*_

--partition*=*partition name**
The slurm partition the job will be queued to.

--cpus-per-task*=*amount of cpus the job needs**
One node has a maximum of 24 CPUs available.

--mem-per-cpu*=*amount of memory each cpu needs e.g. 2G**
251GB are available in total on a node. The default value is 10G.

--gres*=gpu:*amount of graphic cards**
A maximum of 4 graphic cards are available per node.
Use --gres=gpu:1080ti:1 if you explicitly want to have a GeForce GTX 1080Ti GPU. And --gres=gpu:A4000:1 if you explicitly want to have a RTX A4000 GPU. If it does not care, just use --gres=gpu:1

--time*=*time in minutes**
The time the scripts needs to run. If it takes longer, it will be cancelled.

--error*=*filepath**
Filepath of the file that will contain the stderr output of the job. Will be created by slurm.

--output*=*filepath**
Filepath of the file that will contain the stdout output of the job. Will be created by slurm.

--mail-type*=*type**
Get a mail notification if an event appears.
Valid type values are NONE, BEGIN, END, FAIL, REQUEUE, ALL (includes BEGIN, END, FAIL).

--mail-user*=*mail address**
The mail address a notification will be send to

2.2 Our Slurm Configuration

2.2.1 Scheduler:

A backflil scheduler is used. It will start a lower priority job, if doing so does not delay the expected start time of any higher priority jobs.

2.2.2 Partitions (Queues):

4 partitions with different time limits are provided:

Partition name	**TimelLimit**	**Always accessible nodes**	**Additional nodes accessible if unused**
test (default)	15 min	3	37
day	1 day	10	25
week	7 days	17	10
month	30 days	10	0

Note:

- *Test* is the default partition.
- A job that exceeds the time limit of its partition will get canceled.
- RTX A4000 gpus are only available in the day, week, and test partition.

2.2.3 Priority Determination

The priority of a job is determined by a combination of 2 factors: The age of the job and the fair share factor.

Job age: The jobs which have waited longer, will get a higher priority. The maximum value is reached after 7 days. The contribution of Job age factor is 20% (2/10) to the total priority determination.

Fairshare: This factor is calculated by slurm depending on how many resources a user and its user group (department) used in the past. For each job a cost value is calculated. The costs of various resources are as follows: 1 CPU costs 1/s, 1 GPU costs 7/s, 1 GB memory costs 0.11/s. The job cost value decreases over time and is halved after 7 days. Computations on the test partition are for free and don't count in the fairshare factor. The contribution of fairshare factor is 80% (8/10) to the total priority determination.

Bonus: Members of the Zell-lab get a 50% priority bonus since the cluster is now fully financed and maintenanced by this group.

2.2.4 The scratch folder

Slurm creates a scratch folder for each job at “/scratch/jobid” at the job’s compute node. All temporary calculations have to be saved here and not in the home directory. The reason is that the home directory on the nodes is mounted on the storage node. Storing temporary computation data there will lead to an unacceptably high network load. After the job execution finished the job’s scratch folder will be automatically deleted.

2.2.5 Cgroups

Slurm is using Linux cgroups, that means your software will only be able to see and use the amount of CPUs and GPUs that have been allocated by slurm for this job. Memory is not restricted.

2.2.6 Resource granularity:

The resources within a node are individually allocated as consumable resources. In our case CPUs and memory are consumable resources. This means, that multiple jobs can run on one node as long as there are sufficient amounts of CPUs and memory available. In practice, you should specify the amount of CPUs and memory a job needs, so that a high efficiency is achieved on the cluster. Please, don’t use a full node if you do not need it!

2.2.7 User restrictions and Quality of services (QOS)

Quality of services describe requirements a job, partition or user must fulfill so that a job can be accepted by sbatch.

Currently configured QOS:

- All student accounts are only allowed to have 1 job in the queue and to run 1 job. If they have 1 job running no second one is allowed in the queue.
- One user is not allowed to use more than 80% of the graphic cards excluding the graphic cards on the test nodes (=116 graphic cards). This is implemented so that a single user doing excessive grid searches does not fill out the whole cluster.

2.2.8 Important notes on using slurm

- None at the moment

2.3 The Container virtualization software - Singularity:

Using containers provides the user the freedom to use any kind of linux working environment on the cluster. For each use case, a different operating system with different libraries installed can be used. In contrast to a virtual machine, a singularity container does not need any emulated kernel but can run directly on the host kernel. This makes the container execution as fast as running an ordinary application. Singularity is able to import and run docker images. Predefined docker images can be found at https://hub.docker.com/ and similarly predefined singularity images can be found at https://singularity-hub.org/ . A singularity image is defined by and created from a recipe file. More information is provided at http://singularity.lbl.gov/quickstart The current installed version is 3.1.1.1.

2.3.1 Most important commands:

$ singularity run-help *imagefile*: get summarized information about the container
$ singularity inspect *imagefile*: get detailed information about the container
$ singularity shell *imagefile*: Invoke a shell within the container. Changes to the singularity image will not be saved.
$ singularity exec *imagefile* *code*: execute code inside the container. The --nv argument for nvidia cuda support is used by default.
$ sudo singularity build *container name* *recipe file* : build a new image from a recipe file, do this on your own machine, then copy the image to the head node.

2.3.2 How to create your own singularity container:

A detailed description of how to build your own singularity container and how to get it on a cluster is provided by the ZDF for the BinAC cluster. You can use this guide also for our cluster.
http://www.zdv.uni-tuebingen.de/dienstleistungen/computing/container/singularity.html
Please have a look at important notes about singularity for more details.
Further on, the original documentation of Singularity might be useful:
https://sylabs.io/guides/3.7/user-guide/
It might also be useful to look at the recipe files provided on the cluster at:
/common/singularityImages

2.3.3 How to use a docker container:

Just link to its docker hub link in your recipe file exactly as it was done in the provided recipe files on the cluster at /common/singularityImages.

2.3.4 Our Singularity Configuration

The directories which are directly mounted from the node into the container file system are:

- /home

- /tmp
- /common
- /scratch

Less important mounted directories are:

- /proc
- /sys
- /dev
- /devpts
- /etc/localtime
- /etc/hosts

These are mounted at the same location as in the node system. Note that '/home' is mounted from the storage server and is the same as your '/home' at the head node. Additional bindings are prohibited

2.3.5 Provide your Singularity Container (existing on hubs):

If you created or found a docker or singularity container that might be useful for multiple users on the system, just send an email to: tcml-contact@listserv.uni-tuebingen.de
We will provide it for all users in the directory /common/singularityImages.

2.3.6 How to make additions to an existing container:

If you just want to make small changes to an existing container, you can work with overlay images applied to this container. For further details, please see the documentation at:
https://sylabs.io/guides/3.7/user-guide/persistent_overlays.html

2.3.7 Provided Singularity Containers:

Currently, there is a default singularity image provided in the directory /common/singularityImages:
Former images can be found in /common/singularityImages/legacy.

1. TCML-Cuda11_0_TF2_4_1_PT1_7_1.simg
   - OS: Ubuntu 18.04.5 LTS
   - Cuda: 11.0.3
   - Pytorch 1.7.1
   - Tensorflow-gpu: 2.4.1
   - Keras: 2.4.3
   - Python: 3.6.9
   - OpenCV 4.5.1
   - CuDNN 8.0.4

This information can also be seen by using the command: *singularity help *imagename**.
For detailed information, *singularity inspect *imagename** command can also be used.

2.3.8 Important notes on using Singularity

- none

3 How to run your code on the cluster:

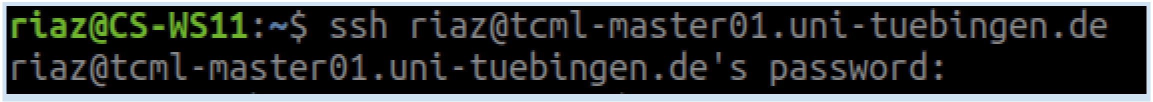
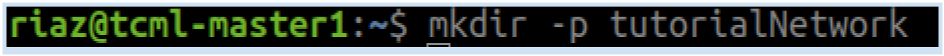
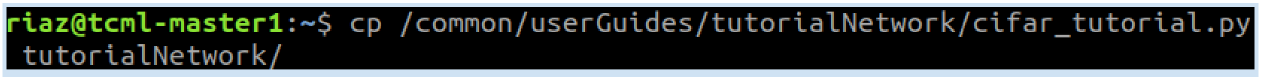
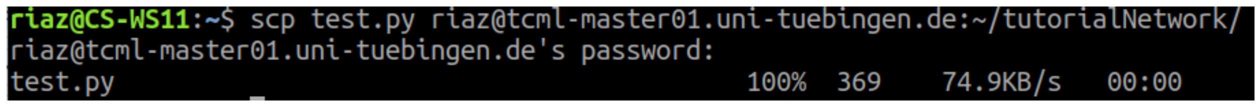
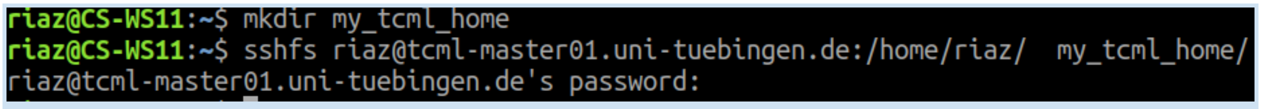
This section gives a tutorial on how to run your code on the cluster. We highly recommend you to follow this **Best Practice Method** to run your scripts because it will make sure you won't face any library conflicts or resource allocation issues while running your code.

1) Copy all needed data to your home directory (via scp, rsync or sshfs (see here)). (Check before, if the dataset you need is already available at /common/datasets)
2) Check whether the default Singularity Container fits your task.
    a) Otherwise download an image from singularity hub
    b) Or download a container from Docker hub and convert it to a singularity container
3) Create a .sbatch file to submit your task to a slurm partition
    a) Define slurm job parameters
    b) Copy all needed data to the job's scratch folder on the node
    c) Execute your code in a specific singularity container
    d) Write your checkpoints to your home directory, so that you still have them if your job fails.
    e) Use a virtual python environment if you need a other python version or other python modules
4) Submit the .sbatch file to a slurm partition
5) Wait until your job will be finished
6) Remove all data you won't need in near future from your home directory

3.1 Example - Best Practice Method:

We are going to use the cluster to train a small convolutional neural network on the CIFAR-10 dataset (consists of 60000 32x32 colour images in 10 classes). The network is a slightly modified version from the following pytorch tutorial: Training a Classifier
For your understanding and testing, you can also run the following google colab notebook: cifar10_tutorial.ipynb
The script and the used .sbatch file can be found at /common/userGuides/tutorialNetwork.

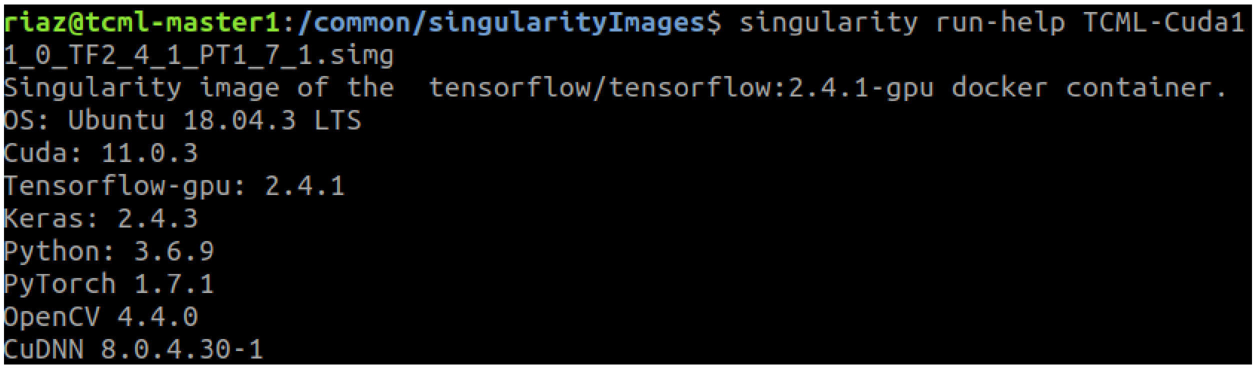
3.1.1 Copying the data to your home directory:

1. First, we will login to the cluster and make a folder *'tutorialNetwork'* in our home directory. Note that you should replace mutschler with your own username at all instances in this example.
   $ ssh mutschler@tcml-master01.uni-tuebingen.de
   $ mkdir -p tutorialNetwork
   
   ```
   riaz@CS-WS11:~$ ssh riaz@tcml-master01.uni-tuebingen.de
   riaz@tcml-master01.uni-tuebingen.de's password:
   ```
   
   ```
   riaz@tcml-master1:~$ mkdir -p tutorialNetwork
   ```
   Note that you might be asked for a password.
2. This folder will contain our python script and the slurm sbatch file. Therefore, we will first copy the python script from /common/userGuides/tutorialNetwork/cifar_tutorial.py to our folder.
   $ cp /common/userGuides/tutorialNetwork/cifar_tutorial.py tutorialNetwork/
   
   ```
   riaz@tcml-master1:~$ cp /common/userGuides/tutorialNetwork/cifar_tutorial.py
    tutorialNetwork/
   ```
3. (Optional) Note that you can also copy data from your local PC to your home directory in the cluster using scp (secure copy). For example, to copy a python script 'test.py', open a new terminal on your local PC and run the following commands:
   $ scp test.py mutschler@tcml-master01.uni-tuebingen.de:~/tutorialNetwork
   
   ```
   riaz@CS-WS11:~$ scp test.py riaz@tcml-master01.uni-tuebingen.de:~/tutorialNetwork/
   riaz@tcml-master01.uni-tuebingen.de's password:
   test.py                                         100%  369    74.9KB/s   00:00
   ```
4. (Optional) You can also mount your home directory in the cluster to your local PC by using sshfs command. This is more user friendly for modifying your code and reading the output, since you can access your files from your local PC. First you must create a directory in your local PC and then mount using the following commands.
   $ mkdir -p my_tcml_home
   $ sshfs mutschler@tcml-master01.uni-tuebingen.de:/home/mutschler my_tcml_home/
   
   ```
   riaz@CS-WS11:~$ mkdir my_tcml_home
   riaz@CS-WS11:~$ sshfs riaz@tcml-master01.uni-tuebingen.de:/home/riaz/  my_tcml_home/
   riaz@tcml-master01.uni-tuebingen.de's password:
   ```
   If you want to unmount the system, you can run with following command:
   $ fusermount -u my_tcml_home

3.1.2 Do we already have a Singularity Container which fits our task?

Our network needs python 3.6 with the modules pytorch, matplotlib and numpy installed.
The *TCML-Cuda11_0_TF2_4_1_PT1_7_1.simg* image is sufficient for us as it contains all the libraries we need. We can check this by looking what exactly is installed in the container by executing the command:

```
$ cd /common/singularityImages/
$ singularity run-help TCML-Cuda11_0_TF2_4_1_PT1_7_1.simg
```



```
riaz@tcml-master1:/common/singularityImages$ singularity run-help TCML-Cuda1
1_0_TF2_4_1_PT1_7_1.simg
Singularity image of the  tensorflow/tensorflow:2.4.1-gpu docker container.
OS: Ubuntu 18.04.3 LTS
Cuda: 11.0.3
Tensorflow-gpu: 2.4.1
Keras: 2.4.3
Python: 3.6.9
PyTorch 1.7.1
OpenCV 4.4.0
CuDNN 8.0.4.30-1
```

Note: You can get more information by using *singularity inspect* or by shelling into the container with *singularity shell.* The warnings are arise since singularity is run on the head node. On the compute nodes they do not appear.

3.1.3 Assuming that no suitable container is available:

In this case, we will search for a suitable container at docker hub (https://hub.docker.com/) or singularity hub (https://singularity-hub.org/). If a container is found, we have to create a singularity *recipe* file (having a file extension *.recipe). The *recipe* file holds the configuration of the container and it is used to build a new container.
To get a container from docker hub, our recipe file must at least contain the following lines of code:

```
#header
Bootstrap: docker
From: <path to container e.g: tensorflow/tensorflow:1.6.0-gpu-py3>
```

To get a container from shub, the following code is needed:

```
Bootstrap: shub
From: <path to container>
```

After saving the recipe file, we can build the container by using the following command.

```
sudo singularity build *container name*.simg *recipifile*
```

Since we need sudo rights for this, this has to be done on your local machine. (more information: http://singularity.lbl.gov/docs-build-container.)

Note: You can add additional software, data, environment variables etc. to this container by means of the recipe file if necessary (http://singularity.lbl.gov/docs-recipes). The recipe files which were used to build the container images are also available at /common/singularityImages.

3.1.4 Create a .sbatch file to submit your task to a slurm partition

To submit a job to a slurm partition, we need to create a .sbatch file. This file defines the configuration parameters and required resources by the job. Take a look at the example .sbatch file cifar_tutorial.sbatch in the directory /common/userGuides/tutorialNetwork/.
The contents of the file are shown and explained below.
---

```
#!/bin/bash

####
#a) Define slurm job parameters
####

#SBATCH --job-name=CifarTutorial

#resources:

#SBATCH --cpus-per-task=4
# the job can use and see 4 CPUs (from max 24).

#SBATCH --partition=test
# the slurm partition the job is queued to.

#SBATCH --mem-per-cpu=3G
# the job will need 12GB of memory equally distributed on 4 cpus.
(251GB are available in total on one node)

#SBATCH --gres=gpu:1
#the job can use and see 1 GPUs (4 GPUs are available in total on one
node) use SBATCH --gres=gpu:1080ti:1 to explicitly demand a Geforce
1080 Ti GPU. Use SBATCH --gres=gpu:A4000:1 to explicitly demand a RTX
A4000 GPU

#SBATCH --time=10:00
# the maximum time the scripts needs to run
# "minutes:seconds", "hours:minutes:seconds", "days-hours",
"days-hours:minutes" and "days-hours:minutes:seconds"

#SBATCH --error=job.%J.err
# write the error output to job.*jobID*.err

#SBATCH --output=job.%J.out
```

```
# write the standard output to job.*jobID*.out

#SBATCH --mail-type=ALL
#write a mail if a job begins, ends, fails, gets requeued or stages
out

#SBATCH --mail-user=**@uni-tuebingen.de
# your mail address

####
#b) copy all needed data to the jobs scratch folder
# We copy the cifar10 datasets which is already available in common
datasets folder to our job's scratch folder.
# Note: For this script, cifar-10 would be downloaded directly from
internet, if you didn't copy it yourself.
####

cp -R /common/datasets/cifar_tutorial/ /scratch/$SLURM_JOB_ID/

####
#c) Execute your code in a specific singularity container
#d) Write your checkpoints to your home directory, so that you still
have them if your job fails
####

singularity exec --nv
/common/singularityImages/TCML-Cuda11_0_TF2_4_1_PT1_7_1.simg python
cifar_tutorial.py /scratch/$SLURM_JOB_ID/cifar_tutorial/

echo DONE!
```

You must first copy this sbatch file to your home directory by using the following command:

```
$ cp /common/userGuides/tutorialNetwork/cifar_tutorial.sbatch ~/tutorialNetwork/
```

Then, you must modify the necessary SLURM configurations according to your own requirements. For this tutorial, kindly fill out your email address to get updates on job status.

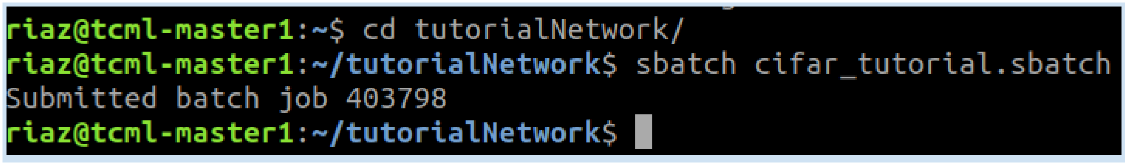
Note: The resource definitions must be as exact as possible otherwise the priority factor of our future jobs will be decreased. If the time value is exceeded, the job will be canceled!

Further informations about slurm job parameters can be found here: https://slurm.schedmd.com/sbatch.html.

3.1.5 Submit the sbatch file to a slurm partition

To run the job, first change directory to tutorial network and use the sbatch command in the following way:

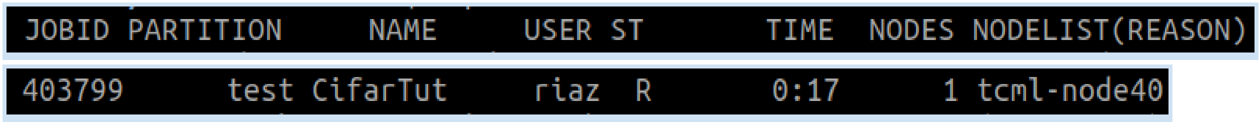
```
$ cd ~/tutorialNetwork/
$ sbatch cifar_tutorial.sbatch
```



```
riaz@tcml-master1:~$ cd tutorialNetwork/
riaz@tcml-master1:~/tutorialNetwork$ sbatch cifar_tutorial.sbatch
Submitted batch job 403798
riaz@tcml-master1:~/tutorialNetwork$
```

We will get an email when our job starts and stops computing. Therefore, it is highly recommended to provide the correct email address in the .sbatch file. After the job is submitted, a job.*ID*.err and job.*ID*.out file is created in the ~/tutorialNetwork folder. These files are the output of *stderr* and *stdout* of the job, respectively. You can read the content of these files using a text editor. Apart from that, a checkpoint file *cifar_net.pth* is also created at the end of training. We can check if our job has already started by executing:

```
$ squeue
```



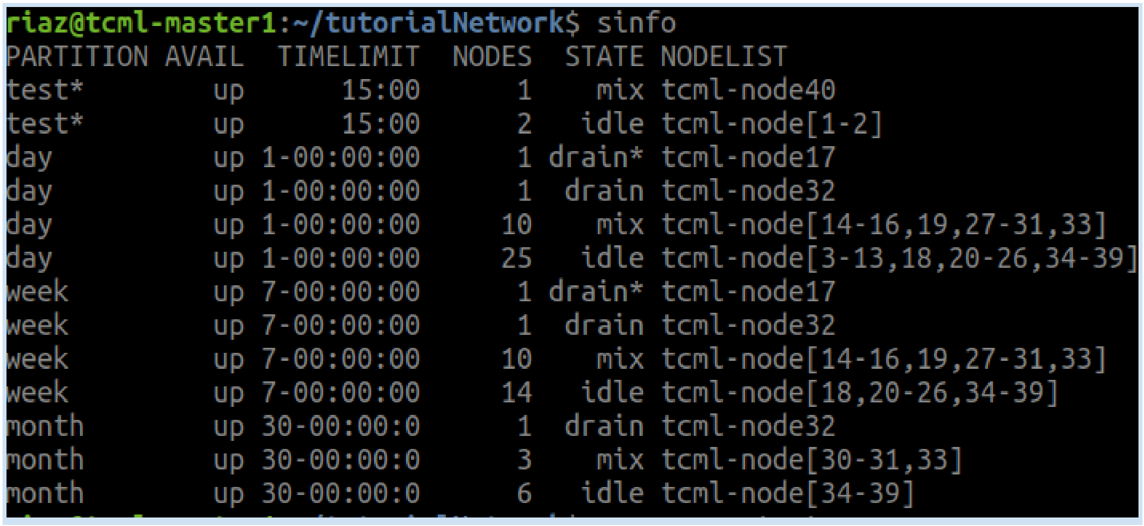
```
 JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
403799       test CifarTut     riaz  R       0:17      1 tcml-node40
```

You can use the following command to see if your job is lined up in the queue for being executed later:

```
$ squeue --start
```

To see the status of partitions and which nodes are occupied, we can use *sinfo* as follows:

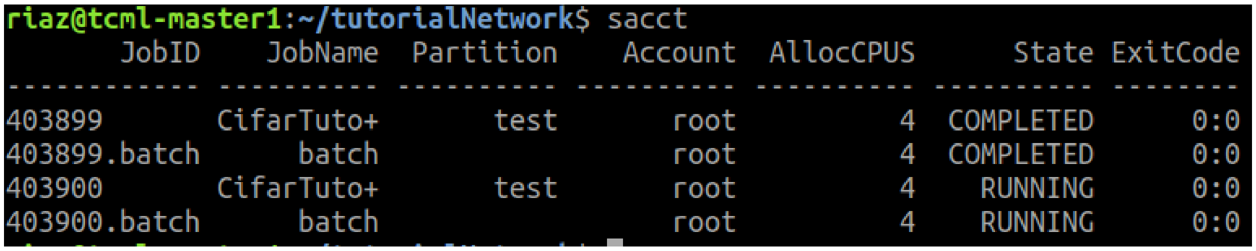
```
$ sinfo
```



```
riaz@tcml-master1:~/tutorialNetwork$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
test*        up      15:00      1    mix tcml-node40
test*        up      15:00      2   idle tcml-node[1-2]
day          up 1-00:00:00      1 drain* tcml-node17
day          up 1-00:00:00      1  drain tcml-node32
day          up 1-00:00:00     10    mix tcml-node[14-16,19,27-31,33]
day          up 1-00:00:00     25   idle tcml-node[3-13,18,20-26,34-39]
week         up 7-00:00:00      1 drain* tcml-node17
week         up 7-00:00:00      1  drain tcml-node32
week         up 7-00:00:00     10    mix tcml-node[14-16,19,27-31,33]
week         up 7-00:00:00     14   idle tcml-node[18,20-26,34-39]
month        up 30-00:00:0      1  drain tcml-node32
month        up 30-00:00:0      3    mix tcml-node[30-31,33]
month        up 30-00:00:0      6   idle tcml-node[34-39]
```

With sacct the history of all run jobs can be seen

```
$ sacct
```



```
riaz@tcml-master1:~/tutorialNetwork$ sacct
       JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
403899       CifarTuto+       test       root          4  COMPLETED      0:0
403899.batch      batch                  root          4  COMPLETED      0:0
403900       CifarTuto+       test       root          4    RUNNING      0:0
403900.batch      batch                  root          4    RUNNING      0:0
```

3.1.6 Useful slurm commands

Besides squeue, sacct, and sinfo there are some more useful commands; for example, if you want to see your fairshare-score, use sshare -l. This yields your fairshare score as a number in between 0 and 1 where higher is better.
If your job does not get assigned to a compute node right away because no resources are available you can check the job's fairshare score with the command sprio which looks like this:

```
JOBID PARTITION   PRIORITY       SITE          AGE  FAIRSHARE     QOS
403860 test          5213        0        0     213     5000
403861 test          5213        0        0     213     5000
403862 test          5213        0        0     213     5000
403863 test          5213        0        0     213     5000
403864 test          5213        0        0     213     5000
403865 test          1173        0        0     1173          0
403866 test          6600        0        0     1600    5000
403867 test          6600        0        0     1600    5000
403868 test          6600        0        0     1600    5000
403869 test          6600        0        0     1600    5000
403870 test          6600        0        0     1600    5000
403871 test          6600        0        0     1600    5000
403872 test          6600        0        0     1600    5000
403873 test          6600        0        0     1600    5000
```

Additionally, queue --start, prints the estimated date when your job will start. For example:

```
JOBID PARTITION     NAME  USER ST      START_TIME  NODES SCHEDNODES
403866          test     test     laube PD 2021-04-19T10:33:09        1 tcml-node40
403867          test     test     laube PD 2021-04-19T10:33:09        1 tcml-node1
403868          test     test     laube PD 2021-04-19T10:33:09        1 tcml-node2
403869          test     test     laube PD 2021-04-19T10:38:00        1 tcml-node40
403870          test     test     laube PD 2021-04-19T10:38:00        1 tcml-node1
403871          test     test     laube PD 2021-04-19T10:38:00        1 tcml-node2
403860          test dynamic_  messmer PD 2021-04-19T10:43:00  1 tcml-node2
403872          test     test     laube PD 2021-04-19T10:43:00        1 tcml-node40
403873          test     test     laube PD 2021-04-19T10:43:00        1 tcml-node1
```

```
403861        test dynamic_  messmer PD 2021-04-19T10:48:00  1 tcml-node1
403862        test dynamic_  messmer PD 2021-04-19T10:48:00  1 tcml-node40
403863        test dynamic_  messmer PD 2021-04-19T10:58:00  1 tcml-node2
403864        test dynamic_  messmer PD 2021-04-19T11:03:00  1 tcml-node1
403865        test Cardiac2  frueh PD 2021-04-19T11:03:00    1 tcml-node40
```

3.1.7 Remove unnecessary data from your home directory

To make sure that the storage in the cluster is not filled up unnecessarily, we will clear our home directory from any temporary data after finishing your jobs. Moreover, we make sure we don't keep datasets in your home directory permanently. We only keep data that we need in near future in your home directory. As a result, we use the following command to remove the tutorialNetwork folder:

```
$ rm -rf tutorialNetwork
```

4 Contact

If you have any question, complaint, suggestion of improvement or anything else feel free to contact us via tcml-contact@listserv.uni-tuebingen.de.