

# TCML Cluster Documentation

December 19, 2024

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Overview	2
1.2	Login Nodes	2
1.3	Slurm	2
1.4	Singularity	2
<b>2</b>	<b>Step-by-step guide</b>	<b>3</b>
2.1	Upload your Data and Script	3
2.2	Log in	3
2.3	Configure the Singularity Container	3
2.4	Create the .sbatch file	3
2.5	Run the project with the sbatch command	4
2.6	Wait until the job is done	4
2.7	Check the output or errors of the job	5
<b>3</b>	<b>Example</b>	<b>5</b>
<b>4</b>	<b>Frequently Asked Questions</b>	<b>7</b>
4.1	Why does my job not start?	7
4.2	Why is my job taking so long?	7
4.3	Why is the login node slow?	7
<b>5</b>	<b>Components</b>	<b>8</b>
5.1	Computing Nodes	8
5.2	Login Nodes	8
5.3	Directories	8
5.4	Slurm	8
5.4.1	Important Slurm Commands	9
5.4.2	Partitions	9
5.4.3	Priority Determination	9
5.5	Singularity	9
5.5.1	How to build an image	9

# 1 Introduction

This Document is a guideline for using the Training Center for Machine Learning (TCML) GPU Cluster.

## 1.1 Overview

The cluster contains:

- 3 login Nodes WITHOUT GPU (2 virtual machines, 1 physical)
- 34 nodes with 4x 1080ti [1-17,20-36]
- 2 nodes with 4x A4000 [18-19]
- 4 nodes with 8x 2080ti [36-40]
- storage space for the datasets

## 1.2 Login Nodes

Users will connect to one of the 3 [login nodes](#) where, using the [sbatch](#) command, they will queue a job to the cluster. This job is any machine learning script you wish to train.

## 1.3 Slurm

The jobs are scheduled by [Slurm](#), a job management system. The greater the required processing power, the less priority the job will have. If all nodes are occupied, the job will be queued and will not start. Check the queue with the `squeue` command.

## 1.4 Singularity

In order to prepare the environments you need for the script, [Singularity](#) is used. This container manager takes a recipe with all the required packages and builds an image, which you will then use in the `.sbatch` file to run your script.

## 2 Step-by-step guide

### IMPORTANT

Change your password as is described in the message of the day, which is displayed in the terminal once you log in!

### 2.1 Upload your Data and Script

With the command:

```
scp SOURCE USERNAME@login1.tcml.uni-tuebingen.de:~/
```

This command will copy the SOURCE file from your computer to your home folder in the cluster. Use the -r flag if you wish to copy entire directories.

### IMPORTANT

Do NOT store more than 5 million files or more than 6 TB of data. If the storage is full, the system will crash for everybody and other users will be affected.

### 2.2 Log in

From the university network, including eduroam or VPN<sup>1</sup>, you can use the following command to log in to the cluster:

```
ssh USERNAME@login1.tcml.uni-tuebingen.de
```

This command will connect you to login node 1. For another login node simply swap login1 with login2 or login3 (all have the same functionality, more info in section 5.2). If logging in to a node does not work, simply try another one.

### IMPORTANT

Reminder! The login nodes DO NOT HAVE A GPU. Scripts executed directly on the node will take longer than usually. Do not forget to submit your job in the queue with the sbatch command, as described in the next steps.

### 2.3 Configure the Singularity Container

Easiest way to do this is to check in `/common/singularityImages/recipes/` whether there is already one container available that fits your needs.

If not, then one needs to either be created from scratch or an existing one modified.

### 2.4 Create the .sbatch file

Create a file with the extension .sbatch (for example project1.sbatch). There is an example file in `/common/userGuides/tutorialNetwork/`. It will then need to have the following content:

```
#!/bin/bash

#SBATCH --job-name=JobName
# give it any name you want

#SBATCH --cpus-per-task=4
# max 24 per node
```

---

<sup>1</sup><https://uni-tuebingen.de/einrichtungen/zentrum-fuer-datenverarbeitung/dienstleistungen/netze/netzzugang/remote-zugang-vpn/>

```

#SBATCH --partition=day
# choose out of day, week, month depending on job duration

#SBATCH --mem-per-cpu=3G
# max 251GB per node

#SBATCH --gres=gpu:1
# how many gpus to use
# each node has 4 gpus

#SBATCH --time=10:00
# job length: the job will run either until completion or until this timer runs out

#SBATCH --error=job.%J.err
# %J is the job ID, errors will be written to this file

#SBATCH --output=job.%J.out
# the output will be written in this file

#SBATCH --mail-type=ALL
# write a mail if a job begins, ends, fails, gets requeued or stages out
# options: NONE, BEGIN, END, FAIL, REQUEUE, ALL

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

# here will be your commands for running the script

```

#### Additional Information

Any `.SBATCH` file works in two ways: the lines starting with `#SBATCH` contain arguments for the slurm workload manager. It will use these arguments for the training job it is starting. From a bash point of view, these lines are arguments since they start with a `'#'`. The lower part of the file – without the `'#'`-signs – are bash commands, which will be executed on the compute node once slurm started the job.

## 2.5 Run the project with the sbatch command

```
sbatch project1.sbatch
```

This command will `QUEUE` the job defined in the `project1.sbatch` file (using slurm, see section 5.4). It will only start computing right away if there are enough resources available.

#### IMPORTANT

If there are not enough nodes available to start your job, it will have to wait for others to finish. Run the `squeue` command to see an overview of the queue.

The place of your job in the queue is determined by Slurm, see section 5.4.3 for more details.

## 2.6 Wait until the job is done

If you have enabled email notifications in the `.SBATCH` file, you will get an email when it is done, otherwise you will have to check manually.

#### VERY IMPORTANT

If you encounter problems or have questions, please consult the [Frequently Asked Questions](#) before writing an email to the system Admin.

## 2.7 Check the output or errors of the job

The output can be found in `job.JOBNUMBER.out` and the errors in `job.JOBNUMBER.err` which will be created in the same directory as the `.sbatch` file.

## 3 Example

```
moldovan@login3:~$ cp -R /common/userGuides/tutorialNetwork ~/
```

We copy the tutorial training algorithm, which will train a model on the [CIFAR](#) dataset, to the home directory.

Next step is to configure the already existing sbatch file (everything is already prepared, you just need to change the email):

Everything until the mail can be left as is.

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

Here is the command to copy the dataset from the datasets folder to the scratch folder. This is done so it will be deleted from the scratch folder after the job is done so it won't take up space.

```
#####  
#b) copy all needed data to the jobs scratch folder  
#####
```

```
cp -R /common/datasets/MNIST/ /scratch/$SLURM_JOB_ID/
```

To execute code in a singularity container, use this command. This example is for python. The whole command needs to be in one line (the way it is in the provided sbatch file).

```
singularity exec --nv /common/singularityImages/TCML-CUDA12_4_TF2_17_PT_2_4.simg  
python3 ~/tutorialNetwork/cifar_tutorial.py
```

Next, with the sbatch command we queue the job. Because there is a free node for me, it will start immediately, and we can see it running.

In these files, you can see the output and the errors of the job.

```
moldovan@login3:~/tutorialNetwork$ sbatch cifar_tutorial.sbatch  
Submitted batch job 1183070  
moldovan@login3:~/tutorialNetwork$ squeue  
      JOBID PARTITION    NAME     USER ST       TIME  NODES NODELIST(REASON)  
    1183062      day exercise  stud123 R    10:30:09      1 tcml-node20  
    1183070      test Tutorial  moldovan R         0:21      1 tcml-node40  
    1182990      week pd+_lat_   raible  R    19:52:52      1 tcml-node17
```

Figure 1: Job 1183070 is my job, and once it is started, two files will be created: `job.1183070.err` and `job.1183070.out`.

```
messmer@tcml-master1:~/test_job$ squeue
  JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
 1207518      day dynamic_  messmer PD        0:00      1 (Priority)
 1207519      day dynamic_  messmer PD        0:00      1 (Priority)
 1207520      day dynamic_  messmer PD        0:00      1 (Priority)
 1207521      day dynamic_  messmer PD        0:00      1 (Priority)
 1207522      day dynamic_  messmer PD        0:00      1 (Priority)
 1207523      day dynamic_  messmer PD        0:00      1 (Priority)
 1207524      day dynamic_  messmer PD        0:00      1 (Priority)
 1207525      day dynamic_  messmer PD        0:00      1 (Priority)
 1207526      day dynamic_  messmer PD        0:00      1 (Priority)
 1207527      day dynamic_  messmer PD        0:00      1 (Priority)
 1207517      day dynamic_  messmer PD        0:00      1 (Priority)
 1207516      day dynamic_  messmer PD        0:00      1 (Resources)
 1207488      day dynamic_  messmer R         0:03      1 tcml-node5
 1207489      day dynamic_  messmer R         0:03      1 tcml-node6
 1207490      day dynamic_  messmer R         0:03      1 tcml-node7
 1207491      day dynamic_  messmer R         0:03      1 tcml-node8
```

Figure 2: Another example of the output of `squeue` where we can see jobs waiting in the queue. The jobs with '(Priority)' and '(Resources)' are submitted, but have not been started yet.

## 4 Frequently Asked Questions

### 4.1 Why does my job not start?

At first, check `squeue`. There are two possibilities:

1. Your job does not appear in the output of `squeue`. This means, there probably was an error in the `.sbatch` file. Read the `job.JOBNUMBER.err` file for more information. Another reason may be, that the directory to which slurm should write its output files does not exist. In this case, the job does not get submitted, without an error output.
2. Your job does appear in the output of `squeue` and has not started yet. This means, that the cluster is full and all nodes are already occupied. In this scenario, your job will be in the queue, waiting for nodes to be free. The place of the job in the queue is easy to check with the `squeue` command that gives you an overview of all nodes and the queue. More information about priority in the queue can be found in section 5.4.3. You can see an example of how this looks in Figure 2.

### 4.2 Why is my job taking so long?

A reason for this may be, that you didn't properly submit your job to slurm (with the command `sbatch FILE.sbatch`) but just executed the `sbatch` file directly (with `sh FILE.sbatch` or similar). See also the info box in section 2.4.

The login nodes DO NOT have a GPU! If you run capacity intensive scripts (like model training) directly on the node, it will take a long time for these to be completed. The login node is only an interface between you and the cluster. All jobs need to be submitted to the cluster with the `sbatch` function, as described in section 2.

### 4.3 Why is the login node slow?

Training algorithms usually need big datasets that take up a lot of space. For this reason, users should copy the dataset to the scratch directory:

```
/scratch/$SLURM_JOB_ID/
```

If this is not done, the storage will be overworked and the node will slow down for everyone.

## 5 Components

### 5.1 Computing Nodes

As mentioned in the [Introduction](#), there are 40 nodes. If you want to choose a specific Node and GPU for your job, this is how:

When writing the `.sbatch` file, the relevant field will need to be changed:

```
#SBATCH --gres=gpu:1080ti:4
# to use 4 (the maximum amount!) 1080ti GPUs

#SBATCH --gres=gpu:2080ti:8
# to use 8 (the maximum amount!) 2080ti GPU
Or
#SBATCH --gres=gpu:A4000:1
# to use 1 (the maximum amount is 4) A4000 GPUs
```

The nodes with A4000 or 1080ti GPUs have 4 GPUs per node, while the nodes with 2080ti have 8 GPUs per node. If you try to acquire more GPUs on a single node, slurm will complain and not start your job.

### 5.2 Login Nodes

All login nodes have the directory `/home/` mounted on the shared cluster file system, therefore it appears as they have the same files on them. The first two (`login1` and `login2`) are virtual machines with identical hardware resources; they have 4 cores, 16 GB of RAM and NO GPU. `login3` is a physical machine and more powerful, with 12 cores and 256 GB of RAM, but still NO GPU.

The intention here is for `login1` and `login2` to be used most of the time, since users only need to upload their files, set up the singularity container and schedule the `sbatch` job. Thus, the nodes don't need much computing power. The reason for having multiple login nodes is that if one is down for maintenance, users can still use the other. Additionally, `login3` is much more powerful and is intended for remote development.

### 5.3 Directories

The main directory is the `/home` directory. Here, every user has their own folder where they can keep their files.

A few useful scripts, datasets and singularity recipies can be found in `/common`:

1. `/datasets`: some of the most well-known machine learning datasets. The user must implement the data reading pipeline themselves.
2. `/scripts`: some of the most well-known training algorithms
3. `/share`: here you can share files with other users
4. `/singularityImages`: helpful singularity images and recipes
5. `/userGuides`: more guides and a tutorial script

#### IMPORTANT

Keep your home directory small and delete all files you don't need. Storage space does not grow on trees.

Also, the home directory does not have a backup, so keep your data safe on your own computer.

### 5.4 Slurm

For more detailed information visit <https://slurm.schedmd.com/quickstart.html>

Slurm is a job scheduling system. This is the program that decides when your script will be run. The priority is determined by the amount of resources the job needs and the time it waited for



other jobs to complete.

Please note, that we are not running the latest version of slurm on the cluster. Check the exact version with `slurmctld -V` before reading slurm's documentation.

### 5.4.1 Important Slurm Commands

**sbatch** <projectName.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. With -U, it will cancel all your jobs.

**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.

### 5.4.2 Partitions

Partition	Time limit	Number of nodes
test	15 min	3
day	24 h	38
week	7 days	27
month	30 days	10

Notice: Any job that exceeds the time limit of its partition will get canceled. Check the details for which node belongs to which queue with `sinfo`.

### 5.4.3 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.

1. Job age: The longer a job waits in the queue, the higher priority it will get. The maximum value is reached after 7 days. The contribution of Job age factor is 20% (2/10) to the total priority determination.
2. Fairshare: This factor is calculated by Slurm depending on how many resources a user and their 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 dont count in the fairshare factor. The contribution of fairshare factor is 80% (8/10) to the total priority determination.

## 5.5 Singularity

Singularity is a platform for managing containers, in order to have any environment you want on the cluster.

### 5.5.1 How to build an image

The images are the environments you use. They are built from recipes.

In `/common/singularityImages` you will find one base image with some basic python libraries:

- python version 3.11
- pytorch
- tensorflow
- keras
- opencv

You can check a singularity image by using the following command:

```
singularity inspect IMAGENAME
```

If you need something else for your project, in `/common/singularityImages/recipes` there are many singularity recipes for building images.

In order to build an image from a provided recipe, you should first copy the recipe locally, preferably in a directory:

```
mkdir singularity_build
cd singularity_build
scp /common/singularityImages/recipes/TCML-CUDA12_4_TF2_17_PT_2_4.recipe ./my.recipe
```

Then we build the image with the following command:

```
singularity build --fakeroot new_image.simg my.recipe
```

There is a second way of building an image: you take an existing image or recipe and make a sandbox environment. These are more flexible but require a little more work. First, create a new directory and create the image:

```
mkdir singularity_build
cd singularity_build
singularity build --fakeroot --sandbox sandbox_image my.recipe
```

Then you open a shell for this image and use pip install as you would normally:

```
singularity shell --writable sandbox_image
pip install packageName
exit
singularity build --fakeroot new_image.simg sandbox_image
```

The image can then be used in the `.sbatch` file, as described in section [3](#).