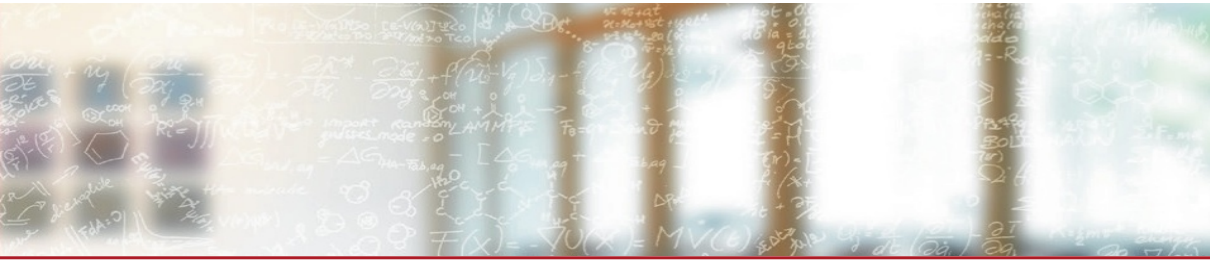
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



User Lab: Getting started at CSCS

A brief Intro to the User Lab
Webinar for the CSCS User Comminity

Victor Holanda, Software Engineer, CSCS

March 11th, 2022

DISCLAIMER

- The seminar is based on publicly available info CSCS provided by CSCS
  - The main CSCS website – www.cscs.ch
  - The user portal – https://user.cscs.ch
  - The User management portal – https://account.cscs.ch
  - CSCS's GitHub public repos – https://github.com/eth-cscs
  - CSCS's advertised products – http://products.cscs.ch
- Slides will be available at https://www.cscs.ch/publications/tutorials/
- Please write on the chat if you have any questions
- **Give us feedback**, please help us improve our documentation and presentations

Interacting with CSCS staff

How to interact with us?

Open a support ticket
support.cscs.ch

Among other things

- This increases the visibility of your request/question

but before that, please **read the info** at

- The user portal – https://user.cscs.ch
- The main CSCS website – www.cscs.ch
- **Give us feedback**, please help us improve our documentation

Accessing CSCS User Lab resources

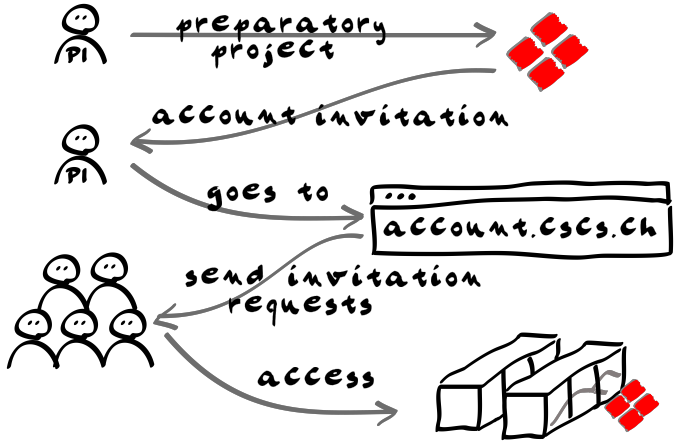
CSCS User Lab resources

- CSCS User Lab **resources are bound to projects**
- There are three types of projects
  - **Production projects** - aimed at the production work for a specific scientific investigation;
  - **Development projects** aimed to develop codes and algorithms; and
  - **Preparatory projects** intended to allow **new users to CSCS** to port and test their codes and **acquire technical data** to apply for a Production Project
- **User accounts are associated to projects**
- Projects must have a **Principal Investigator (PI)**

More information at https://www.cscs.ch/user-lab/allocation-schemes/

Account creation

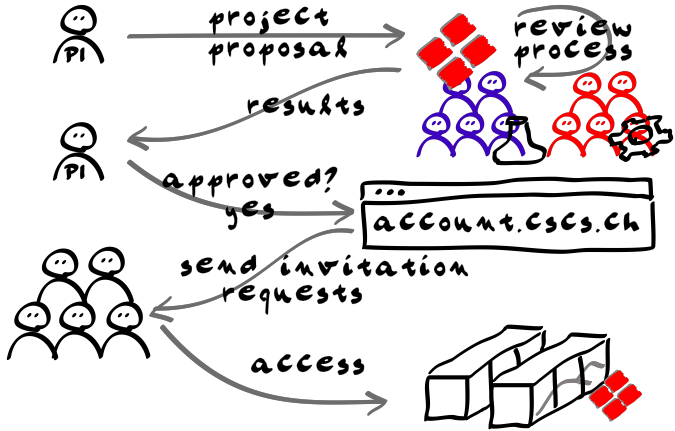
- To get an account, **one needs to be invited** either by CSCS admin staff or by a PI
- PIs **must apply for a preparatory project** to receive the invitation to an account at CSCS
- PIs **invite users to their projects** using account.cscs.ch



More information at https://www.cscs.ch/user-lab/applying-for-accounts/

Project allocation and user account creation

- When ready, principal investigators **apply to production projects**
- Production projects undergo scientific and technical reviews
- If project is approved, PIs **invite users to their projects** using account.cscs.ch

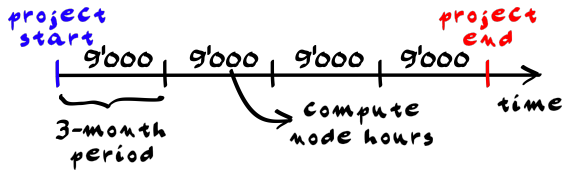


More information at https://www.cscs.ch/user-lab/allocation-schemes/

Project resource allocations

- **Resource quotas are allocated in natural quarters**, starting on April 1st, July 1st, October 1st, and January 1st
- Make sure to fully use your quarterly compute budget within the corresponding time frame because **unused resources are not transferred between quarters**
- **Monitor project resource utilisation** using https://account.cscs.ch/
- Resource **utilisation are measured in compute node hours**

Example for a 36'000 node-hours allocation



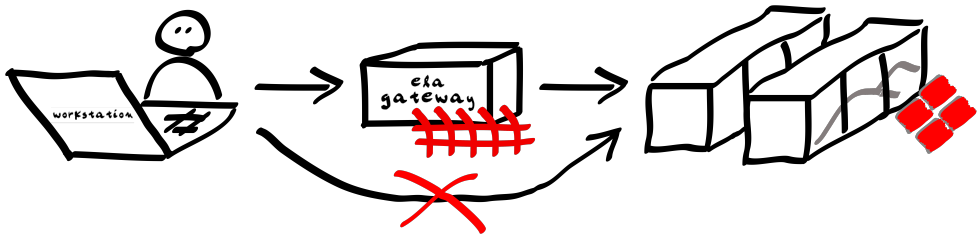
More information at https://user.cscs.ch/access/accounting/

Accessing CSCS User Lab resources

There are four ways to access CSCS resources

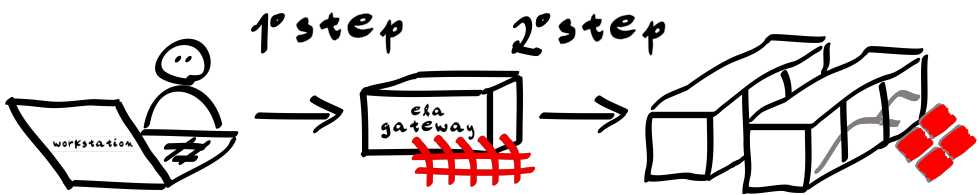
- **via SSH - discussed in this presentation**
- Interactive SuperComputing (JupyterLab) - more information at https://user.cscs.ch/tools/interactive/
- Continuous Integration Service - more information at https://user.cscs.ch/tools/continuous/
- FirecREST API - more information at https://products.cscs.ch/firecrest/

Accessing CSCS resources via SSH



- You need to connect to ela.cscs.ch first in order to connect to internal systems.
- SSH proxyjump can be used to avoid having to type ssh twice
- We **advise to use SSH keys with strong passphrase** to access our systems

Accessing CSCS resources via SSH - 2 step process



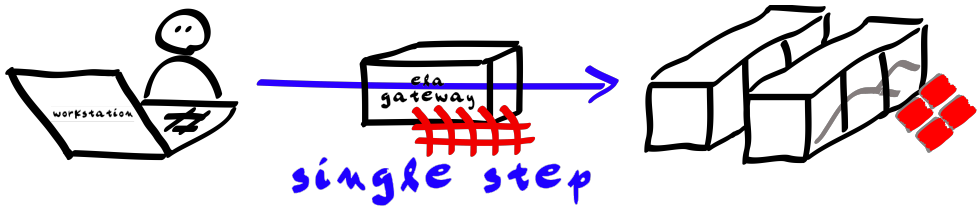
1° step - connect to ela

```
me@myworkstation:~$ ssh my_cscs_user@ela.cscs.ch
my_cscs_user@ela.cscs.ch's password:
my_cscs_user@ela:~$
```

2° step - connect to daint from ela

```
my_cscs_user@ela:~$ ssh daint
my_cscs_user@daint.cscs.ch's password:
my_cscs_user@daint:~$
```

SSH proxy jump



- You can automate the SSH hoping by using Proxy Jump
- We are showing how to do it using SSH keys and strong passphrase

More information at https://user.cscs.ch/access/auth/#generating-ssh-keys and https://tinyurl.com/2p92ueme

SSH proxy jump to daint using ssh key and passphrase

1. workstation configuration

```
me@myworkstation:~$ ssh-keygen -t ed25519
me@myworkstation:~$ ssh-copy-id -i ~/.ssh/id_ed25519.pub my_cscs_user@ela.cscs.ch
me@myworkstation:~$ eval "$(ssh-agent)"
me@myworkstation:~$ ssh-add ~/.ssh/id_ed25519
Enter passphrase for /Users/me/.ssh/id_ed25519:

me@myworkstation:~$ cat ~/.ssh/config
Host ela
  Hostname ela.cscs.ch
  User my_cscs_user

Host daint
  Hostname daint.cscs.ch
  User my_cscs_user
  ProxyJump ela
  ForwardAgent yes
```

2. connect from workstation to daint

```
me@myworkstation:~$ ssh daint
...
[my_cscs_user@daint:~]$
```

Why would you proxy jump?

- Access the CSCS systems directly
- Copy *small* files directly to file systems that are not mounted on ela

Example: Copy GROMACS input file to `$SCRATCH` on daint

```
me@myworkstation:~$ scp gromacs_input.tpr daint:\$SCRATCH
Enter passphrase for key '/absolute/path/to/.ssh/id_ed25519':
...
me@myworkstation:~$ ssh daint
[my_cscs_user@daint:~]$ cd $SCRATCH
[my_cscs_user@daint:/scratch/snx3000/my_cscs_user]$ ls
gromacs_input.tpr
[my_cscs_user@daint:/scratch/snx3000/my_cscs_user]$
```

Why would you proxy jump?

- Access the CSCS systems directly
- Copy *small* files directly to file systems that are not mounted on ela

Example: Copy GROMACS input file to `$SCRATCH` on daint

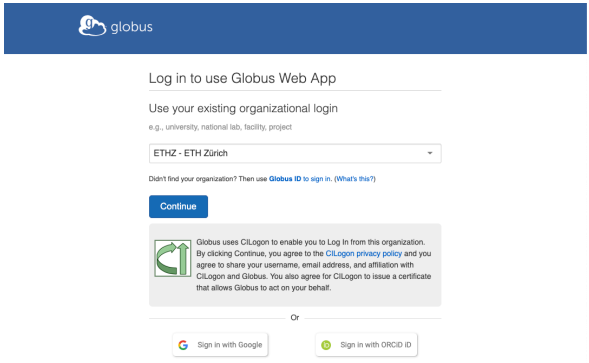
```
me@myworkstation:~$ scp gromacs_input.tpr daint:\$SCRATCH
Enter passphrase for key '/absolute/path/to/.ssh/id_ed25519':
...
me@myworkstation:~$ ssh daint
[my_cscs_user@daint:~]$ cd $SCRATCH
[my_cscs_user@daint:/scratch/snx3000/my_cscs_user]$ ls
gromacs_input.tpr
[my_cscs_user@daint:/scratch/snx3000/my_cscs_user]$
```

How do we copy large files?

What file systems are mounted for the User Lab systems?

How do we copy large files to CSCS systems?

- Globus Online EndPoint (recommended way)
- Globus-Url-Copy (deprecated)
- One can perform parallel copy using Globus
- Avoid `rsync` and `scp`



More information at https://user.cscs.ch/storage/transfer/external/

Mounted file systems available to the User Lab

	/scratch (Piz Daint)	/scratch (Alps)	/users	/project	/store
Type	Lustre	Lustre	GPFS	GPFS	GPFS
Quota	1M files	1M files	50GB/user 500K files	Maximum 50K files/TB	Maximum 50K files/TB
Expiration	**30 days**	**30 days**	Account closure	End of Project	End of contract
Data Backup	None	None	90 days	90 days	90 days
Access Speed	**Fast**	**Fast**	Slow	Medium	Slow
Capacity	8.8 PB	8.7 PB	86 TB	4.7 PB	3.6 PB
Environment variable	`$SCRATCH`	`$SCRATCH`	`$HOME`	`$PROJECT`	—

- **/scratch is the only adequate file system to run simulations**
- **/scratch quota only applies for submitting new jobs**
- **compute nodes mount /project and /store as read-only**
- ela.cscs.ch nodes only mount /users
- /scratch inodes quota is to prevent excessive loads on the Lustre file systems

More information at https://user.cscs.ch/storage/file_systems/

How do we run simulations?

We use the installed workload manager, Slurm

- We strive to achieve a fair share of resources, so that every user can consume their allocated resources
- Slurm is installed in all CSCS User Lab systems
- We implement a Fair Usage of Shared Resources policy
  - **It is not allowed to run applications on the login nodes**
  - Users are not supposed to submit arbitrary amounts of Slurm jobs and commands at the same time
  - **Applications must be executed on compute nodes** managed by Slurm
  - Jobs are scheduled based on multifactor priorities with well defined weights

More information at https://user.cscs.ch/access/accounting/

Slurm at CSCS User Lab

- Slurm runs jobs. **A job can be a script, a program or an interactive session**
- Slurm **allocates exclusive access to compute nodes** to users for some duration of time
- One has to **explicitly select the project** they want to consume computing resources from
- **We provide a Slurm jobscript generator** https://user.cscs.ch/access/running/jobscript_generator/ to help select Slurm options compatible with Piz Daint

Slurm at CSCS User Lab

- **Resources are selected using different Slurm options.** e.g. `constraints`, `partitions`, and `mem`
  - **`constraints` are used to select different hardware** - compute nodes with (`--constraint=gpu`) or without GPUs (`--constraint=mc`)
  - **`partitions` or queues are used to select different workflows.** e.g. running simulation, debugging code, and perform pre- and post-analyses. They are set using the `--partition` option.
  - **`mem` is used to select compute nodes with larger memory**. e.g. compute nodes on Piz Daint have 64GB but a selected few have 120GB

More information at https://user.cscs.ch/access/running/

Relevant Slurm queues at CSCS

Queue name	Max duration	Max number of nodes	Description
debug	30 minutes	4	Quick turnaround for test jobs
long	72 hours	4	Maximum 5 long jobs in total
normal	24 hours	2400(gpu)/512(mc)	**Standard queue for production work**
prepost	30 minutes	1	High priority pre/post processing
xfer	24 hours	1	Internal data transfer queue

- For especial requests, contact us

More information at
https://user.cscs.ch/access/running/piz_daint/#slurm-batch-queues

Submitting a Slurm job

Example Slurm jobscript.sh file

```
#!/bin/bash -l
#SBATCH --job-name="job_name" # or -J "job_name"
#SBATCH --account="project" # or -A "project"
#SBATCH --time=01:00:00 # or -t 01:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=12
#SBATCH --constraint=gpu # or -C gpu
#SBATCH --hint=nomultithread

export CRAY_CUDA_MPS=1

srun ./executable.x
```

submit script using `sbatch`

```
[my_cscs_user@daint:~]$ sbatch jobscript.sh
```

add options when submitting scripts

```
[my_cscs_user@daint:~]$ sbatch -plong jobscript.sh
```

modify options when submitting scripts

```
[my_cscs_user@daint:~]$ sbatch -C mc jobscript.sh
```

submit directly using `srun`

```
[my_cscs_user@daint:~]$ export CRAY_CUDA_MPS=1
[my_cscs_user@daint:~]$ srun -J"name" \
 -A"project" -t 01:00:00 --nodes=1 \
 --ntasks-per-node=12  -Cgpu \
 --hint=nomultithread ./executable.x
```

Check job status

- Check job status using `squeue`
- Customise `squeue` output either in the command line or using environment variables

standard CSCS squeue config

```
[my_cscs_user@daint:~]$ squeue -u my_cscs_user
  JOBID     USER ACCOUNT           NAME  ST REASON     START_TIME                TIME  TIME_LEFT NODES    CPUS
99999981 my_cscs    s9999       job_name   R None       05:35:26               10:09:57   13:50:03     2      48
99999982 my_cscs    s9999       job_name   R None       06:36:32                9:08:51   14:51:09     2      48
99999983 my_cscs    s9999       job_name   R None       06:49:09                8:56:14   15:03:46     2      48
99999984 my_cscs    s9999       job_name  PD Priority   Tomorr 06:                 0:00 1-00:00:00    36     432
```

customised squeue output

```
[my_cscs_user@daint:~]$ export SQUEUE_FORMAT="%.8A %.8j %.3t %9r %.10S  %.5D %.4C %Z %N"
[my_cscs_user@daint:~]$ unset SQUEUE_SORT
[my_cscs_user@daint:~]$ squeue -u $USER
   JOBID     NAME  ST REASON    START_TIME  NODES CPUS WORK_DIR NODELIST
99999994 job_name  PD Priority  Tomorr 04:     36  432 /scratch/snx3000/my_cscs_user/dir4
99999991 job_name   R None         06:57:25     16  384 /scratch/snx3000/my_cscs_user/dir1 nid0[2007,2015-2029]
99999992 job_name   R None         06:49:09      3   72 /scratch/snx3000/my_cscs_user/dir2 nid0[6307,6424,6490]
99999993 job_name   R None         06:49:09      2   48 /scratch/snx3000/my_cscs_user/dir3 nid0[4006,4080]
```

Good practices when submitting jobs

Specify accurate wall times

```
#!/bin/bash -l
#SBATCH --time=00:30:00
#SBATCH --nodes=120
#SBATCH --constraint=gpu
[...]
```

Run off `$SCRATCH`

```
#!/bin/bash -l
#SBATCH --nodes=120
[...]
cd $SCRATCH
srun $SCRATCH/executable
```

For jobs with many tasks, use GREASY

```
#!/bin/bash -l
#SBATCH --nodes=120
#SBATCH --constraint=gpu
#SBATCH --gres=gpu:0,craynetwork:4
[...]
module load daint-gpu
module load GREASY

export CRAY_CUDA_MPS=1
export CUDA_VISIBLE_DEVICES=0
export GPU_DEVICE_ORDINAL=0
greasy tasks.txt
```

If you cannot use GREASY, wait between `srun` calls

```
#!/bin/bash -l
#SBATCH --nodes=120
[...]
function waitabit() {
  rt=$?;
  if [[ ${rt} -ne 0 ]]; then
    sleep 2
  fi
  return ${rt}
}
srun mytask1 ; waitabit
srun mytask2 ; waitabit
```

Good practices when submitting jobs

Use Slurm e-mail notification for updates on job status

```
#!/bin/bash -l
#SBATCH --nodes=120
#SBATCH --constraint=gpu
#SBATCH --mail-type=ALL
#SBATCH --mail-user=<your_email>
[...]
```

Explicitly select the in-core multi-threading

```
#!/bin/bash -l
#SBATCH --nodes=120
#SBATCH --ntasks-per-core=1
#SBATCH --hint=nomultithread
[...]
```

Write self-contained and reproducible jobscripts

```
#!/bin/bash -l
#SBATCH --job-name="job_name"
#SBATCH --account="project"
#SBATCH --time=01:00:00
#SBATCH --nodes=120
#SBATCH --ntasks-per-core=1
#SBATCH --ntasks-per-node=12
#SBATCH --cpus-per-task=1
#SBATCH --partition=normal
#SBATCH --constraint=gpu
#SBATCH --hint=nomultithread

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export CRAY_CUDA_MPS=1

module load daint-gpu
module load GROMACS

srun gmx_mpi mdrun ...
```

What NOT to do when submitting jobs

Jobs that submit other jobs in loops

```
#!/bin/bash
#SBATCH ...
[...]
while :
do
srun sbatch job_script.sh
sleep 1
done
```

Job with thousands of tasks

```
$ sacct -j 123456789 | wc -l
25337
```

Jobs with hundreds of tasks in parallel

```
#!/bin/bash -l
#SBATCH --nodes=120
[...]
srun mytask1 &
srun mytask2 &
[...]
srun mytask2000
```

Jobs that run off `$HOME`

```
#!/bin/bash -l
#SBATCH --nodes=120
[...]
srun ~/executable ~/input_file
```

What NOT to do on the login nodes

Run squeue without filtering

```
[my_cscs_user@daint:~]$ squeue | grep ${USER} # bad!
[my_cscs_user@daint:~]$ squeue -u ${USER} # good!
```

Monitor Slurm with `watch`

```
[my_cscs_user@daint:~]$ watch squeue -u ${USER} # bad!
```

Run unbounded GNU make

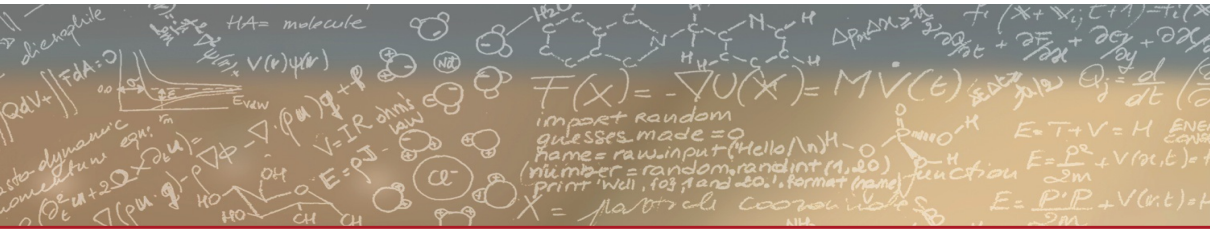
```
[my_cscs_user@daint:~]$ make -j # bad!
[my_cscs_user@daint:~]$ make -j6 # good!
```

Run servers

```
[my_cscs_user@daint:~]$  redis-server # bad!
```

Run pre-post analyses

```
[my_cscs_user@daint:~]$  ./my_script.sh # bad!
```



Thank you!
Have fun!