ICT Division
IT Infrastructures Department

אגף התקשוב
מחלקת תשתיות תקשוב

# BIU HPC Cluster

# Slurm User Guide

# Contents

1. Introduction to Slurm and the HPC Cluster
   - 1.1 What is Slurm?
   - 1.2 Cluster Structure
   - 1.3 Login Nodes
   - 1.4 What are slurm Partition?
   - 1.5 Compute Nodes
2. Cluster Partitions, Accounts and Resources
   - 2.1 Choosing a Partition
   - 2.2 General Partitions Summary
   - 2.3 Private Partitions Summary
   - 2.4 Private Accounts and Access – How to submit jobs to private partition
3. Getting Started: your Slurm Job
   - 3.1 Connect to a Login Node
   - 3.2 Create a Batch Script
   - 3.3 Submit the Job
   - 3.4 Monitor the Job
   - 3.5 Output and Error Files
   - 3.6 Cancel a Job
4. Requesting Resources
   - 4.1 CPUs
   - 4.2 Memory
   - 4.3 GPUs
   - 4.4 Maximum Runtime
   - 4.5 Email Notifications
5. Batch Job Examples
   - 5.1 Basic CPU Job
   - 5.2 Basic GPU Job
   - 5.3 Conda Job
   - 5.4 Using Docker
   - 5.5 Using Job Arrays
   - 5.6 Matlab Job
   - 5.7 Using singularity
6. Interactive Jobs and using Jupyter
   - 6.1 Interactive session using srun
   - 6.2 Using Jupyter on slurm - running on a Compute Node
   - 6.3 Using Jupyter notebook in VS Code
7. Time Limits, Requeue and Checkpointing
   - 7.1 Partition Time Limits and Reque policy
   - 7.2 Checkpointing
   - 7.3 Interactive Jobs and Requeue
8. Cluster Usage Guidelines
   - 8.1 Login Node Usage
   - 8.2 Development and Debugging
   - 8.3 Request Only What You Need
   - 8.4 Long-Running Jobs
   - 8.5 Clean up unnecessary jobs and processes

# 1. Introduction to Slurm and BIU HPC Cluster

## 1.1 What is Slurm?

Slurm is a Linux open-source resource manager and job scheduler designed to utilize a fair share of the computing resources. Slurm allocates resources, starts and monitors jobs, and schedules pending work when resources are shared by many users.

## 1.2 Cluster Structure

Your computer -> Login node -> Slurm partition -> Compute node

You connect to a login node and submit your job to Slurm. The selected partition defines the group of eligible compute resources and its usage rules. Slurm selects the appropriate compute node on which the job will run.

## 1.3 Login Nodes

Login nodes are the entry point to the HPC environment. Use them to prepare and manage work, not to perform heavy computation.

- Edit and prepare job scripts.
- Manage files and software environments.
- Submit jobs with sbatch or start short interactive allocations.
- Check job status and output.
- Cancel or inspect jobs.

> Our login nodes:
>
> slurm-login1.lnx.biu.ac.il, slurm-login2.lnx.biu.ac.il, slurm-login3.lnx.biu.ac.il

## 1.4 What are Slurm Partitions?

A partition in Slurm is a logical group of compute nodes (servers) used to run jobs. Partitions are similar to queues in other workload managers.

Each partition may have different hardware resources (CPUs/GPUs), maximum job runtime, resource limits, and access permissions.

Users should select a partition that matches their job requirements. If no partition is specified, Slurm uses the default partition.

## 1.5 Compute Nodes

Compute nodes are the servers where Slurm jobs run. They may provide different CPUs, memory, GPUs, or other resources. Users request the resources they need, Slurm selects a suitable compute node from the chosen partition.

Back to Contents

# 2. BIU Cluster Partitions, Accounts and Resources

## 2.1 Choosing a Partition

Users should select a partition that matches their job requirements and access permissions.

Selecting a partition does not select a specific server. Slurm automatically assigns an appropriate compute node based on the requested resources and their availability.

Specify the partition in your batch script:

```
#SBATCH --partition=<partition-name>
```

If not specified, the default partition (generic) is used.

To view available partitions and their status, run the command: `sinfo`

The cost of usage of the computational resources is detailed in the following link: usage-costs

## 2.2 General Partitions Summary

| Partition | GPU Type / CPU | MaxTime | Max Jobs Per User | Max GPU/CPU Per User | Max GPU Per Group | Node Mem | Allowed Accounts |
|---|---|---|---|---|---|---|---|
| A100-4h | GPU-A100 | 4h | 4 | 4 | NoLimit | 512G | ALL |
| B200-4h | GPU-B200 | 4h | 2 | 2 | NoLimit | 2TB | ALL |
| B200-8h | GPU-B200 | 8h | 2 | 2 | NoLimit | 2TB | ALL |
| generic | GPU-1080,2080,3070 | 4h | 26 | 26 | NoLimit | 192G | ALL |
| generic-48G | GPU-RTX6000-Ada | 4h | 2 | 2 | NoLimit | 128G | ALL |
| H200-12h | GPU-H200 | 12h | 3 | 3 | NoLimit | 2TB | ALL |
| H200-4h | GPU-H200 | 4h | 3 | 3 | NoLimit | 2TB | ALL |
| L40s-4h | GPU-L40S | 4h | 2 | 2 | NoLimit | 512G | ALL |
| L4-12h | GPU-L4 | 12h | 6 | 6 | NoLimit | 512G | ALL |
| L4-4h | GPU-L4 | 4h | 6 | 6 | NoLimit | 512G | ALL |
| RTX6000-4h | GPU-RTX-PRO-6000 | 4h | 6 | 6 | NoLimit | 2TB | ALL |
| cpu192G-48h | CPU | 48h | 56 | 840 | NoLimit | 192G | ALL |
| cpu1T-24h | CPU | 24h | 16 | 92 | NoLimit | 1TB | ALL |
| cpu512G-48h | CPU | 48h | 36 | 560 | NoLimit | 512G | ALL |

Back to Contents

## 2.3 Private Partitions Summary

These partitions are restricted to users associated with the corresponding Slurm accounts.
Jobs submitted to private partitions receive **higher scheduling priority** on their dedicated resources.
For more information: see Section 2.4 – Private Partitions and Accounts: How to Submit Jobs to Private Partitions.

| Partition | GPU Type / CPU | Max Time | Max Jobs Per User | Max GPU/CPU Per User | Max GPU Per Group | Allowed Groups | Allowed Accounts |
|---|---|---|---|---|---|---|---|
| p_b200_chechik | GPU-B200 | 4h | Unlimited | 1 | 1 | Prof. Chechik | ug_chechik |
| p_b200_dagan | GPU-B200 | 4h | Unlimited | 1 | 1 | Prof. Dagan | ug_dagan |
| p_b200_eng | GPU-B200 | 8h | Unlimited | 1 | 1 | Prof. Gannot, Prof. Fetaya, Prof. Lindenbaum, Prof Tirer, Prof. Shavit | ug_gannot, ug_fetaya, ug_lindenbaum, ug_tirer, ug_shavit |
| p_b200_goldberg | GPU-B200 | 4h | Unlimited | 1 | 1 | Prof. Goldberg | ug_goldberg |
| p_b200_kraus | GPU-B200 | 8h | Unlimited | 1 | 1 | Prof. Kraus | ug_kraus |
| p_b200_major | GPU-B200 | 4h | Unlimited | Unlimited | 1 | Prof. Major | ug_major |
| p_b200_nlp | GPU-B200 | 4h | Unlimited | Unlimited | 4 | Prof. Tsarfaty, Dagan, Goldberg | ug_tsarfaty, ug_dagan, g_goldberg |
| p_b200_schwartz | GPU-B200 | Unlimited | Unlimited | Unlimited | Unlimited | Prof. Schwartz | ug_schwartz |
| p_b200_tsarfaty | GPU-B200 | 4h | Unlimited | 1 | 1 | Prof. Tsarfaty | ug_tsarfaty |
| p_glickman | GPU-1080 | Unlimited | Unlimited | Unlimited | Unlimited | Prof. Glickman | ug_cs_dsi |
| p_kraus | GPU-7080,3070 | Unlimited | Unlimited | Unlimited | Unlimited | Prof. Kraus | ug_kraus |
| p_kugler | GPU-A100 | Unlimited | Unlimited | Unlimited | Unlimited | Prof. Kugler | ug_kugler |
| p_l40s_dallatorre | GPU-L40S | Unlimited | Unlimited | Unlimited | Unlimited | Prof. Dallatorre | ug_dallatorre |
| p_L4_senderowitz | GPU-L4 | Unlimited | Unlimited | Unlimited | Unlimited | Prof. Senderowitz | ug_senderowitz |
| p_peer | GPU-A100 | Unlimited | Unlimited | Unlimited | Unlimited | Prof. Peer | ug_peer |
| p_rtx_schwartz | GPU-RTX-PRO-6000 | Unlimited | Unlimited | Unlimited | Unlimited | Prof. Schwartz | ug_schwartz |
| p_rtx_elazar | GPU-RTX-PRO-6000 | Unlimited | Unlimited | Unlimited | Unlimited | Prof. Elazar | ug_elazar |
| p_uriofir | GPU-A100 | Unlimited | Unlimited | Unlimited | Unlimited | Prof. Shaham, Prof. Lindenbaum | ug_uri_ofir |
| p_weiss | GPU-RTX6000-Ada | Unlimited | Unlimited | Unlimited | Unlimited | Prof. Weiss | ug_weiss |
| p_amsterdamer | CPU | Unlimited | Unlimited | Unlimited | Unlimited | Prof. Amsterdamer | ug_amsterdamer |
| p_major | CPU | Unlimited | Unlimited | Unlimited | Unlimited | Prof. Major | ug_major |
| p_zohar | CPU | Unlimited | Unlimited | Unlimited | Unlimited | Prof. Zohar | ug_zohar |

Back to Contents

Partition configurations may change over time. To view the most up-to-date partition information, run the following command from any **slurm login node**: `Partition-Info`

## 2.4 Private Accounts and Access – How to submit jobs to private partition

Private partitions require both a **partition** and an **account**. Add the matching values to your batch script:

```
#SBATCH --partition=<partition-name>
#SBATCH --account=<Allowed_Account>
```

Refer to the **Private Partitions** summary for the allowed account associated with each partition.

To check which Slurm accounts (groups) you belong to, run:

```
sacctmgr list assoc where user=$USER format=User,Account -nP
```

Replace $USER with your actual username.
Users who are not part of the appropriate group **cannot** use these private partitions.

For the exact SBATCH commands for using Private partitions refer to the “SBATCH commands” column in the table in the next page: SBATCH commands for using Private partitions.

Back to Contents

## SBATCH commands for using Private partitions:

| Partition | Group | Account | SBATCH command |
|---|---|---|---|
| p_b200_chechik | Prof. Chechik | ug_chechik | #SBATCH --partition=p_b200_chechik<br>#SBATCH --account=ug_chechik |
| p_b200_dagan | Prof. Dagan | ug_dagan | #SBATCH --partition=p_b200_nlp<br>#SBATCH --account=ug_dagan |
| p_b200_eng | Prof. Gannot,<br>Prof. Fetaya,<br>Prof. Lindenbaum,<br>Prof. Tirer,<br>Prof. Shavit | ug_gannot<br>ug_fetaya<br>ug_ Tirer<br>ug_shavit<br>ug_ Lindenbaum | #SBATCH --partition=p_b200_eng<br>#SBATCH --account=ug_gannot<br>#SBATCH --account=ug_fetaya<br>#SBATCH --account=ug_Tirer<br>#SBATCH --account=ug_shavit<br>#SBATCH --account=ug_ lindenbaum |
| p_b200_goldberg | Prof. Goldberg | ug_goldberg | #SBATCH --partition=p_b200_nlp<br>#SBATCH --account=ug_goldberg |
| p_b200_kraus | Prof. Kraus | ug_kraus | #SBATCH --partition=p_b200_kraus<br>#SBATCH --account=ug_kraus |
| p_b200_major | Prof. Major | ug_major | #SBATCH --partition=p_b200_major<br>#SBATCH --account=ug_major |
| p_b200_nlp | Prof. Tsarfaty,<br>Prof. Dagan,<br>Prof. Goldberg | ug_tsarfaty<br>ug_dagan<br>ug_goldberg | #SBATCH --partition=p_b200_nlp<br>#SBATCH --account=ug_tsarfaty<br>#SBATCH --account=ug_dagan<br>#SBATCH --account=ug_goldberg |
| p_b200_schwartz | Prof. Schwartz | ug_schwartz | #SBATCH --partition=p_b200_schwartz<br>#SBATCH --account=ug_schwartz |
| p_b200_tsarfaty | Prof. Tsarfaty | ug_ tsarfaty | #SBATCH --partition=p_b200_tsarfaty<br>#SBATCH --account=ug_tsarfaty |
| p_glickman | Prof. Glickman | ug_glickman | #SBATCH --partition=p_glickman<br>#SBATCH --account=ug_glickman |
| p_kraus | Prof. Kraus | ug_kraus | #SBATCH --partition=p_ kraus<br>#SBATCH --account=ug_kraus |
| p_kugler | Prof. Kugler | ug_kugler | #SBATCH --partition=p_kugler<br>#SBATCH --account=ug_kugler |
| p_l40s_dallatorre | Prof. Dallatorre | ug_dallatorre | #SBATCH --partition= p_l40s_dallatorre<br>#SBATCH --account= ug_dallatorre |
| p_L4_senderowitz | Prof. Senderowitz | ug_ senderowitz | #SBATCH --partition=p_l4_senderowitz<br>#SBATCH --account=ug_senderowitz |
| p_peer | Prof. Peer | ug_peer | #SBATCH --partition=p_peer<br>#SBATCH --account=ug_peer |
| p_rtx_schwartz | Prof. Schwartz | ug_schwartz | #SBATCH --partition=p_rtx_schwartz<br>#SBATCH --account=ug_ schwartz |
| p_rtx_elazar | Prof. Elazar's group | ug_elazar | #SBATCH --partition=p_rtx_elazar<br>#SBATCH --account=ug_ schwartz |
| p_uriofir | Prof. Shaham,<br>Prof. Lindenbaum | ug_uri_ofir | #SBATCH-- partition= p_uriofir<br>#SBATCH --account= ug_uri_ofir |
| p_weiss | Prof. Weiss | ug_weiss | #SBATCH --partition=p_weiss<br>#SBATCH --account=ug_weiss |
| p_amsterdamer | Prof. Amsterdamer | ug_amsterdamer | #SBATCH --partition=p_amsterdamer<br>#SBATCH --account=ug_amsterdamer |
| p_major | Prof. Major | ug_major | #SBATCH --partition=p_major<br>#SBATCH --account=ug_major |
| p_zohar | Prof. Zohar | ug_zohar | #SBATCH --partition=p_zohar<br>#SBATCH --account=ug_zohar |

Back to Contents

# 3. Getting Started: your Slurm Job

## 3.1 Connect to a Login Node

Connect using SSH to one of the supported Slurm login nodes:

```
ssh slurm-login1.lnx.biu.ac.il
# or
ssh slurm-login2.lnx.biu.ac.il
# or
ssh slurm-login3.lnx.biu.ac.il
```

In case it is not recognize your userid please use this: ssh **$USERID**@slurm-login[1-3].lnx.biu.ac.il

## 3.2 Create a Batch Script

A Slurm batch script is normally a shell script containing #SBATCH directives followed by the commands you want to run.

```
#!/bin/bash
#SBATCH --job-name=hello_slurm
#SBATCH --output=hello_slurm_%j.out
#SBATCH --error=hello_slurm_%j.err
#SBATCH --partition=<partition-name>
#SBATCH --time=00:10:00

hostname
date
```

The #SBATCH lines are interpreted by Slurm at submission time. The remaining commands are executed when the job starts.

## 3.3 Submit the Job

```
sbatch myJob.sh
```

Slurm returns a job ID. Keep this ID; it is used by monitoring, accounting and cancellation commands.

## 3.4 Monitor your jobs

```
squeue -u $USER
```

Common states include PD (Pending), R (Running) and CG (Completing).

## 3.5 Output and Error Files

Use --output and --error in the batch script to control log filenames. %j is replaced with the Slurm job ID:

```
#SBATCH --output=matlab_job_%j.out
#SBATCH --error=matlab_job_%j.err
```

Good practice: Check the output and error files when a job is running, finishes unexpectedly, or appears to make no progress.

## 3.6 Cancel a Job

```
scancel <job_id>  # if you need to stop it
```

Back to Contents

# 4. Requesting Resources

## 4.1 CPUs

Request the number of CPUs your application can actually use:

```
#SBATCH --cpus-per-task=<N>
```

By default, 1 CPU is allocated to a job

## 4.2 Memory

Request memory using the site-approved Slurm option. A common form is:

```
#SBATCH --mem=<size>G
```

By default, 16 GB of memory is allocated for each CPU assigned to the job.

## 4.3 GPUs

GPU jobs must request GPU resources. By default, **no** GPUs are allocated to a job. To request GPUs use:

```
#SBATCH --gres=gpu:<N>   where N is the number of GPUs you need.
```

Ensure that *N* does not exceed the GPU limit of the partition.
In case the partition includes several types of GPU's and you want a specific GPU, use

```
#SBATCH --gres=gpu:<N>   where N is the number of GPUs you need.

To Request a specific GPU type, if supported:
#SBATCH --gres=gpu:<type>:<count>
```

## 4.4 Maximum Runtime

Use --time to set the maximum runtime you are requesting:

```
#SBATCH --time=HH:MM:SS
```

If --time is omitted, the effective time limit is the partition Max Time.

Do not request a runtime longer than the partition maximum Time.

## 4.5 Email Notifications

Slurm can send job notifications when mail support is configured:

```
#SBATCH --mail-user=your.email@example.com
#SBATCH --mail-type=BEGIN,END,FAIL
```

Back to Contents

# 5. Batch Job Examples

## 5.1 Basic CPU Job

```
#!/bin/bash
#SBATCH --job-name=cpu_test
#SBATCH --output=cpu_test_%j.out
#SBATCH --error=cpu_test_%j.err
#SBATCH --partition=<partition-name>
#SBATCH --cpus-per-task=4
#SBATCH --mem=<size>G
#SBATCH --time=01:00:00

python script.py
```

## 5.2 Basic GPU Job

```
#!/bin/bash
#SBATCH --job-name=gpu_test
#SBATCH --output=gpu_test_%j.out
#SBATCH --error=gpu_test_%j.err
#SBATCH --partition=<gpu-partition>
#SBATCH --gres=gpu:1
#SBATCH --mem=<size>G

python gpu_script.py
```

## 5.3 Conda Job

```
#!/bin/bash
#SBATCH --job-name=conda_job
#SBATCH --output=conda_job_%j.out
#SBATCH --error=conda_job_%j.err
#SBATCH --partition=<partition-name>
#SBATCH --mem=<size>G

source ~/miniconda3/etc/profile.d/conda.sh
conda activate myenv
python script.py
```

Adjust the environment initialization to match the Conda/Miniconda installation used at your site.

Back to Contents

## 5.4 Using Docker

The existing guide requires Docker containers launched by Slurm jobs to run in the foreground and to use a job-based container name for traceability.

If you are using docker:

- Run the container in the foreground (**without using -d**) so container will remain attached to the Slurm job.
- Docker container name should be **slurm-$SLURM_JOB_ID** for traceability.
  Use this in your batch script:

```
DockerName=slurm-job-$SLURM_JOB_ID
docker run --name "$DockerName" --rm my-image:latest python script.py
```

- If your Slurm job requests **GPUs**, use the list that Slurm provides in **SLURM_JOB_GPUS** variable to restrict Docker to the GPUs allocated to you.
  **In your sbatch job script:**

```
docker run --name "$DockerName" --rm --gpus "device=${SLURM_JOB_GPUS}"
my-image:latest python script.py
```

  **SLURM_JOB_GPUS** is typically a comma-separated list of device indices (e.g., 0,2).

  If your workflow relies on **CUDA_VISIBLE_DEVICES**, pass it into the container:

```
CUDA_VISIBLE_DEVICES="${SLURM_JOB_GPUS}"
```

**Example of a Docker batch script:**

```
#!/bin/bash
#SBATCH --job-name=docker_job
#SBATCH --output=docker_job_%j.out
#SBATCH --error=docker_job_%j.err
#SBATCH --partition=<partition-name>
#SBATCH --gres=gpu:1

DockerName="slurm-job-${SLURM_JOB_ID}"
docker run --name "$DockerName" --rm my-image:latest python script.py

docker run --name "$DockerName" --rm --gpus "device=${SLURM_JOB_GPUS}"  my-image:latest python script.py
```

In case you need to use **more GPUs** in the docker use the following:

```
#SBATCH --gres=gpu:2
docker run --name "$DockerName" --rm  --gpus '"device="${SLURM_JOB_GPUS}"""' rockylinux:9 sleep 360
```

In case you are using **srun** use: ${SLURM_STEP_GPUS}

```
docker run --name "$DockerName" --rm --gpus "device=${SLURM_STEP_GPUS}"  my-image:latest python script.py
```

Back to Contents

## 5.5 Using Job Arrays

A **Slurm job array** is useful when you need to run the same job multiple times with different inputs.

Example:

```
#!/bin/bash
#SBATCH --job-name=array_test
#SBATCH --partition=<partition-name>
#SBATCH --array=1-10%4
#SBATCH --output=array_%A_%a.out
#SBATCH --error=array_%A_%a.err
#SBATCH --time=01:00:00

echo "Running task ${SLURM_ARRAY_TASK_ID}"

python script.py ${SLURM_ARRAY_TASK_ID}
```

**`#SBATCH --array=1-10%4`** - creates **10 tasks**, numbered 1 through 10.
**%4** means that no more than 4 array tasks may run at the same time.
Each task receives its own task number in: $SLURM_ARRAY_TASK_ID

- Task 1 runs: python script.py 1
- Task 2 runs: python script.py 2
- ...
- Task 10 runs: python script.py 10

For the output files: %A is the main array job ID, %a is the individual array task ID.

Submit the array normally:

```
sbatch array_job.sh
```

You can monitor it using:

```
squeue -u $USER
```

Back to Contents

## 5.6 MATLAB Job

```
#!/bin/bash
#SBATCH --job-name=matlab_job
#SBATCH --output=matlab_job_%j.out
#SBATCH --error=matlab_job_%j.err
#SBATCH --partition=<partition-name>
#SBATCH --cpus-per-task=<N>
#SBATCH --mem=<size>G

matlab -batch "run('mymatlab.m')"
```

### Parallel MATLAB:

If the MATLAB code uses parallel workers, request a CPU count appropriate for the workers you intend to run When Running MATLAB with parallel workers and submitting a MATLAB job with Slurm using:

```
#SBATCH --cpus-per-task=48       <-- 48 is just an example
```

In order to ensure MATLAB uses all the CPU cores allocated by Slurm, add the following to your MATLAB job script (e.g. mymatlab.m):

```
num_workers = str2double(getenv('SLURM_CPUS_PER_TASK'));

c = parcluster('local');

c.NumWorkers = num_workers;    % or  exact number of cores you want to use (e.g. 48)

saveProfile(c);
```

**Matlab is installed on all the nodes in cpu partitions, you can see the version by matlab -v**

## 5.7 Using Singularity

Singularity can be used to run containerized applications inside a Slurm job.

Example:

```
#!/bin/bash

#SBATCH --job-name=singularity_test
#SBATCH --output=singularity_test_%j.out
#SBATCH --error=singularity_test_%j.err
#SBATCH --cpus-per-task=6
#SBATCH --partition=<partition-name>

singularity exec    docker://nvidia/cuda:12.3.0-base-ubuntu22.04    sleep 120
```

In this example, singularity exec runs the command inside the specified container image. Replace sleep 120 with the command you want to run.

For GPU containers, request a GPU from Slurm and use:

singularity exec --nv <image> <command>

The --nv option makes the NVIDIA GPU available inside the container.

Back to Contents

# 6. Interactive Jobs and using Jupyther

## 6.1 Interactive Session using srun

srun command used to run immediate/interactive commands directly on compute nodes managed by Slurm.

srun can both request resources and immediately run a command. Use srun for short tests, debugging and interactive development on compute resources.

To run a single command on a compute node in the specified partition:

```
srun --partition=<partition-name>  <command>
```

To run a single command on a compute node in the specified partition:

```
srun --partition=<partition-name> --pty bash
```

Open an interactive shell with 1 GPU for certain time.

```
srun --partition=<partition-name> --gres=gpu:1 --time=HH:MM:SS --pty bash
```

Exit the shell when finished so the allocation is released.

**Important Notes:**

- Interactive jobs launched this way will not be automatically requeued.
- Use srun only for quick tests, debugging, or interactive development on compute resources. For longer jobs or those requiring resilience to timeouts, use sbatch with checkpointing

Back to Contents

## 6.2 Using Jupyter on slurm - running on a Compute Node

Jupyter should run on a Slurm compute allocation rather than directly on a login node. A recommended procedure is:

1. Install Jupyter Notebook: Connect to one of the a slurm-login nodes and install Jupyther Notebook in your user environment, if it is not already installed:

```
pip3 install --user notebook
```

2. Request a compute allocation with srun or submit a dedicated Jupyter batch job.
   Start Jupyter Notebook on the allocated compute node with --no-browser, run:

```
srun --partition=<partition> --gres=gpu:1 jupyter-notebook --time=02:00:00 --no-browser --ip=0.0.0.0 &
```

   Jupyter will print URLs similar to:

```
http://nodename:8888/tree?token=e34d73f70fa48254500e0556663db7b859985448f9d9829d
http://127.0.0.1:8888/tree?token=e34d73f70fa48254500e0556663db7b859985448f9d9829d
```

   Note:
   - The compute node name assigned by Slurm. In this example: Compute node: nodename.
   - The Jupyter port number (for example, 8888).
   - Copy and save the URL that Jupyter prints (**second line http://127.0.0.1:8888 -** you'll need it later)

3. Create an SSH tunnel from the local computer through the login environment to the compute node On your local PC, open a terminal and run:

```
ssh -L <local-port>:localhost:<jupyter-port> -J <username>@<slurm-login_server> <username>@<compute-node>
```

   For example:

```
ssh -L  8888:localhost:8888 -J myuser@slurm-login1 myuser@nodename
```

   Replace:
   - <username> with your username.
   - <slurm-login-server> with the Slurm login server you are working with.
   - <compute-node> with the node assigned to your job in Step 2.
   - <jupyter-port> with the port shown by Jupyter.

   Keep this SSH session running while using Jupyter.

4. **Open Jupyter in Your Browser**
   On your local PC, open a web browser and use the 127.0.0.1 Jupyter URL from Step 2 (the second line that you copied and saved), for example:

```
http://127.0.0.1:8888/tree?token=e34d73f70fa48254500e0556663db7b859985448f9d9829d
```

   You should now see the Jupyter Notebook interface running on the remote Slurm compute node.

5. Stop Jupyter and release the Slurm allocation when finished.

Back to Contents

## 6.3 Using Jupyter notebook in VS Code

1. Prerequisites:
   - Install these VS Code extensions on your local PC (if it is not already installed): Python (Microsoft), Jupyter, Remote – SSH
   - Install Jupyther Notebook in your user environment, if it is not already installed:

```
pip3 install --user notebook
```

2. Start Jupyter Notebook on the allocated compute node with --no-browser, run:

```
srun --partition=<partition> --gres=gpu:1 jupyter-notebook --time=02:00:00 --no-browser --ip=0.0.0.0 &
```

Jupyter will print URLs similar to:

```
http://nodename:8888/tree?token=e34d73f70fa48254500e0556663db7b859985448f9d9829d
http://127.0.0.1:8888/tree?token=e34d73f70fa48254500e0556663db7b859985448f9d9829d
```

Note:
- The compute node name assigned by Slurm. In this example: Compute node: nodename.
- The Jupyter port number (for example, 8888).
- Copy and save the URL that Jupyter prints (**first line http://nodename:8888**... - you'll need it later)

3. Create an SSH tunnel from the local computer through the login environment to the compute node On your local PC, open a terminal and run:

```
ssh -L <local-port>:localhost:<jupyter-port> -J <username>@<slurm-login_server> <username>@<compute-node>
```

For example:

```
ssh -L  8888:localhost:8888 -J myuser@slurm-login1 myuser@nodename
```

Replace:
- <username> with your username.
- <slurm-login-server> with the Slurm login server you are working with.
- <compute-node> with the node assigned to your job in Step 2.
- <jupyter-port> with the port shown by Jupyter.

Keep this SSH session running while using Jupyter.

**3. Connect VS Code**

1. Open VS Code.
2. Press Ctrl+Shift+P → **Remote-SSH: Connect to Host**. – connect to your slurm-loginX server.
3. Open an existing .ipynb file or create a new Jupyter Notebook.
4. Click **Select Kernel** → **Existing Jupyter Server**.
5. Paste the line from step 2 that you saved earlier: http://<NodeName>...
6. Press Enter to confirm.
7. Select **Python 3 (ipykernel)**.

VS Code is now using the Jupyter server running on the Slurm compute node.

Back to Contents

# 7. Time Limits, Requeue and Checkpointing

## 7.1 Partition Time Limits and Reque Policy

Each partition may define a maximum job runtime.

Batch jobs reaching a partition time limit are **suspended and requeued** by the system when they reach the time limit of the partition they were submitted to.

## 7.2 Checkpointing

If jobs may be requeued or interrupted, applications should save progress periodically and resume from the latest checkpoint.

- Save model/application state periodically.
- Store checkpoints on persistent storage.
- On restart, detect and load the most recent valid checkpoint.
- Test restart behavior before relying on it for long jobs.

Without checkpoint/restart support, a requeued job may restart its application from the beginning even if Slurm preserves the job submission.

## 7.3 Interactive Jobs and Requeue

Interactive allocations are primarily for short tests and development. Interactive srun session will NOT be automatically requeued or restored after a timeout or interruption.

Back to Contents

# 8. Cluster Usage Guidelines

## 8.1 Login Node Usage

- Do not run long CPU-intensive workloads directly or large-memory processing  on login nodes.
- Keep login-node tasks limited to editing, file management, compilation/preparation and job management.

## 8.2 Development and Debugging

Use short interactive jobs or lower-demand partitions for testing when appropriate. Move stable workloads to sbatch scripts for repeatability and fair cluster usage.

## 8.3 Request Only What You Need

- Avoid requesting unused CPUs or GPUs.
- Choose realistic memory and runtime requests.

## 8.4 Long-Running Jobs

For long-running workloads, use checkpointing and choose a partition that matches the job design.

## 8.5 Clean up unnecessary jobs and processes

If needed, stop your slurm job with the command scancel:

```
scancel -u <username>
```

Check for and terminate unnecessary processes running on the Slurm login servers.
Use pkill when appropriate to stop processes by name.

For example, to terminate all VS Code-related processes owned by your user:

```
pkill -u $USER -f vscode
pkill -u <username> -f vscode
```

Back to Contents