Submitting a Job

every field in the submit form explained (partition, account, QOS, resources, script, arrays, MPI)

The Job Submission Form

The submit form is the main way to run jobs on the cluster. Open it from **Submit Job** in the navbar. Every field maps directly to an sbatch option — PowerPortal builds the submission for you.

Required fields

Field	What it sets
Job name	A label for your job. Appears in the Jobs page and in Slurm output.
Partition	The queue to submit to. Determines available nodes, time limits, and which accounts and QOS you can use. Select this first — other fields update based on your choice.
CPUs per task	Number of CPU cores per task. For single-task jobs this is your total core count.
Memory (GB)	RAM per node in gigabytes.
Time limit	Maximum wall-clock time in `HH:MM:SS` or `D-HH:MM:SS` format. The job is killed when this limit is reached.
Job script	The shell commands to run. You do not need a shebang — PowerPortal adds `#!/bin/bash` and `source ~/.bashrc` automatically so `module load` works.

Optional fields

Field	What it sets
Account	Slurm account to charge. Defaults to your primary account if left blank.
QOS	Quality of service — controls priority and limits. Only QOS allowed for the selected partition are shown.
Nodes	Number of nodes to allocate.
Node list	Request specific nodes by name (comma-separated). Leave blank to let Slurm choose.
Exclude nodes	Nodes to avoid. Useful if a node is known to be problematic.

Field	What it sets
Constraints	Node feature requirements (e.g. a specific CPU generation). Multiple constraints are ANDed.
Tasks	Total number of MPI tasks. Leave blank for non-MPI jobs.
Tasks per node	MPI tasks per node. Leave blank for non-MPI jobs.
Array	Job array specification, e.g. `1-100` or `0-9%5`. See the Job Arrays page.
Output file	Path for stdout. Defaults to `slurm-%j.out` in your working directory.
Error file	Path for stderr. Defaults to `slurm-%j.err` in your working directory (a separate file from stdout, not the same file).
Working directory	Directory the job runs from. Defaults to your home directory. Use absolute paths in your script if your input files are elsewhere.
Dependency	Wait for other jobs before starting, e.g. `afterany:12345`. Separate multiple jobs with `:`, e.g. `afterany:101:102:103`.
Notify email	Email address to notify about job events. Defaults to none.
Notify on	Which events trigger a notification email — Begin, End, Fail, Requeue, Time limit / 90% / 80% / 50% reached, Stage out, Invalid dependency, Array tasks, or All. Requires Notify email to be set.
GPU fields	GPU count and type appear only when you select a GPU-enabled partition.

Submitting

Click **Submit Job**. On success, PowerPortal redirects you to the Jobs page after 2 seconds where your new job appears at the top of the list.

Saving as a template

Click **Save as template** before submitting to store the current configuration for reuse. See the Templates chapter for details.

Writing Your Job Script

The job script is the shell code your job runs on the compute node. Paste it into the **Script / Command** field on the submission form — commands only, nothing else.

What PowerPortal adds automatically

The submit form shows two read-only lines above the editor:

```
#!/bin/bash
source ~/.bashrc
```

These are prepended to every job before submission. You do not need to type them. PowerPortal also sources the module system on the backend, so `module load` works inside your job without any extra setup.

Do not use #SBATCH directives in the script

All resource settings (partition, CPUs, memory, time limit, etc.) are set via the form fields — not via `#SBATCH` lines in the script. If you paste a script that contains `#SBATCH` directives, PowerPortal will strip them automatically and show a warning. Use the form fields instead.

A minimal example

```
module load python/3.11
python my_analysis.py --input data.csv --output results/
```

A more complete example

```
module load openmpi/openmpi-5.0.7-rocky9-slurm
cd $HOME/myproject
python preprocess.py

mpirun -n 16 ./simulate --config config.yaml
```

Working directory

Slurm sets the working directory to the value you enter in the **Working Directory** field. Use absolute paths or `cd` explicitly if your input files are in a different location.

Output and error files

By default, stdout and stderr are written to `slurm-<jobid>.out` in the working directory. Set custom paths in the **Output file** and **Error file** fields. Use `%j` for job ID and `%a` for array task ID in file names.

Tips

- Add `set -e` at the top to stop the script immediately if any command fails.
- Print key variables at the start (`echo "Running on $(hostname)"`) to make debugging easier.
- Test your script interactively with `salloc` before submitting a long job.

Job Arrays

A job array runs the same script multiple times in parallel, each with a different index. Use it to process many inputs with identical resource requirements — for example, running the same analysis on 100 different samples.

Setting up an array

Enter the array specification in the **Array** field on the submission form. Examples:

Value	What it does
`1-100`	100 tasks, indexed 1 to 100.
`0-9`	10 tasks, indexed 0 to 9.
`1-100%10`	100 tasks, at most 10 running simultaneously.
`1,3,7,42`	4 tasks with specific indices.

Using the index in your script

Slurm sets the environment variable `$SLURM_ARRAY_TASK_ID` to the current task's index. Use it to select the right input for each task:

```
module load python/3.11

SAMPLES=(sample_01 sample_02 sample_03 ...)
SAMPLE=${SAMPLES[$SLURM_ARRAY_TASK_ID]}

python analyse.py --input data/${SAMPLE}.csv --output results/${SAMPLE}/
```

Or if your inputs are numbered files:

```
python analyse.py --input data/sample_${SLURM_ARRAY_TASK_ID}.csv
```

Output files

Use `%a` in the output file path to get a separate file per task. For example, setting the output field to `logs/job_%j_%a.out` produces one log file per task.

Monitoring array jobs

Array jobs appear in the Jobs page as a single entry. Each individual task is listed with its own task ID and state. A parent job shows as running until all tasks complete.

Limiting concurrency

Use the `%N` suffix (e.g. `1-500%20`) to cap how many tasks run at once. This prevents a large array from monopolising the partition and avoids hitting per-user job limits.

MPI Jobs

MPI jobs run a single program across multiple processes, potentially spanning several nodes. Use the **Tasks** and **Tasks per node** fields to configure the MPI layout.

Fields to set

Field	sbatch equivalent	When to use
Nodes	`--nodes`	Number of nodes to allocate.
Tasks	`--ntasks`	Total MPI process count across all nodes.
Tasks per node	`--ntasks-per-node`	MPI processes per node. Use instead of Tasks when you want even distribution.
CPUs per task	`--cpus-per-task`	Threads per MPI process. Set to 1 for pure MPI; higher for hybrid MPI+OpenMP.

Example — pure MPI, 4 nodes × 32 tasks

```
module load openmpi/openmpi-5.0.7-rocky9-slurm
cd $HOME/myproject


mpirun ./simulate --config config.yaml
```

Set **Nodes** to `4`, **Tasks per node** to `32`, **CPUs per task** to `1`. Do not pass `-n` to `mpirun` — Slurm passes the process count automatically via PMI.

Example — hybrid MPI + OpenMP

```
module load openmpi/openmpi-5.0.7-rocky9-slurm
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
cd $HOME/myproject

mpirun ./simulate --config config.yaml
```

Set **Tasks per node** to the number of MPI ranks per node, and **CPUs per task** to the number of OpenMP threads. The product of the two should not exceed the node's core count.

Choosing a partition

MPI jobs that span nodes require a partition with inter-node connectivity. Check with HPC support if you are unsure which partition to use for multi-node jobs.