UG2.6: Production Environment

In this page:

- SLURM Scheduler
- The Software Catalog
- The "module" command
- How to install your software with Spack
- Python and additional software

Additional page:

- UG2.6.1: How to submit the job - Batch Scheduler SLURM

SLURM Scheduler

Since our HPC systems are shared among many users, long production jobs should be submitted using a scheduler. This guarantees that access to our resources is as fair as possible.

Roughly, there are two different modes to use an HPC system:

- **interactive**, for data movement, archiving, code development, compilations, basic debugger usage: also for very short test runs and general interactive operations. A task in this class should not exceed 10 minutes CPU-time and is free of charge on HPC systems with the current billing policy.
- **batch**, for production runs. Users must prepare a shell script containing all the operations to be executed in batch mode, once the requested resources are available and assigned to the job. The job then starts and executes on compute nodes of the cluster. Remember to put all your data, programs and scripts in the $WORK or $CINECA_SCRATCH filesystem, which are the best storage areas accessible by compute nodes.

You must have valid active projects on the system in order to run batch jobs. Moreover, remember that on our systems there may be specific policies for the use of project budgets.

On all of our current HPC systems, the queuing system or scheduler is SLURM. SLURM Workload Manager (or simply SLURM, which stands for "Simple Linux Utility for Resource Management") is an open-source and highly scalable job scheduling system.

SLURM has three key functions. Firstly, it allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time, so they can perform their work. Secondly, it provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes. Finally, it arbitrates contention for resources by managing the queue of pending jobs.

Comprehensive documentation of SLURM and some examples on how to submit your job is described in a separate section under this chapter, as well as on the original SchedMD site.

The Software Catalog

CINECA offers a variety of third-party applications and community codes that are installed on its HPC systems. Most of the third-party software is installed using software modules mechanism (see “The module command” later in this section).

Information on the available packages and their detailed descriptions are organized in a full catalog optionally divided by discipline in our web site by selecting "software" and "Application Software for Science". It can be also obtained working interactively on the HPC systems, using the module or modmap commands (see later in this chapter).

If you do not find an application you are interested in on our web site or on the specific system or if you have a question about software that is currently available, please contact our specialists (superc@cineca.it).

The "module" command

All software programs installed on the CINECA machines are available as modules.

A basic default modules environment is already set up by the system login configuration files.

In order to have a list of available modules and select a specific one, you have to use the module command. The following table contains its basic options:
<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>module avail</td>
<td>show the available modules on the machine</td>
</tr>
<tr>
<td>module load &lt;appl&gt;</td>
<td>load the module &lt;appl&gt; in the current shell session, preparing the environment for the application.</td>
</tr>
<tr>
<td>module load autoload &lt;appl&gt;</td>
<td>load the module &lt;appl&gt; and all dependencies in the current shell session</td>
</tr>
<tr>
<td>module help &lt;appl&gt;</td>
<td>show specific information and basic help on the application</td>
</tr>
<tr>
<td>module list</td>
<td>show the modules currently loaded on the shell session</td>
</tr>
<tr>
<td>module purge</td>
<td>unload all the loaded modules</td>
</tr>
<tr>
<td>module unload &lt;app&gt;</td>
<td>unload a specific module</td>
</tr>
</tbody>
</table>

As you will see by typing "module avail", the software modules are collected in different profiles (base, advanced...) and organized by functional categories (compilers, libraries, tools, applications,..).

In order to detect all profiles, categories, and modules available on our systems the command "modmap" is available.

```bash
> modmap -m <namesoftware>
```

It shows information about all available versions installed of the software of interest, and in which profile you can find them.

ATTENTION: Remember to load the needed modules in batch scripts too, before using the related applications.

### How to install your software with Spack

In case you don't find a software you can choose to install it by yourself. In this case, on Marconi100 we also offer the possibility to use the "spack" environment by loading the corresponding module (it will be soon available on other systems):

```bash
$ module load spack/<vers>
```

By loading this spack module, setup-env.sh file is sourced. Then $SPACK_ROOT is initialized to /cineca/prod/opt/tools/spack/<vers>/none, spack command is added to your PATH, and some nice command line integration tools too.

A folder is created into your default $WORK space ($USER/spack-<vers>) in order to contain some subfolders created and used by spack during the phase of a package installation:

- sources cache: $WORK/$USER/spack-<vers>/cache
- software installation root: $WORK/$USER/spack-<vers>/install
- module files location: $WORK/$USER/spack-<vers>/modulefiles

$WORK space will be removed at the end of the corresponding project. If you want define different paths for cache, installation and modules directories you can consult the spack guide in which you can find how to customize these paths.

The software we have installed through Spack is available as module by typing the following commands:

```bash
$ module load spack
$ module av
```
$ module av <module_name>

or by spack commands:
$ module load spack
$ spack find
$ spack find <pack_name>

You can show the dependencies, variants, and flags used for the installation of a specific package and the path where are located its binaries typing the following command:
$ spack find -ldvrp <name>

In order to find all the compilers available you can type the following command:
$ spack compiler list

In order to install a software through this spack module you can 1) install the needed compilers and libraries on which it depends on spack too or 2) use the corresponding modules already available for the cluster users.

In the first case, after installing the needed compiler through spack, remember to load the corresponding module and add it to compilers.yaml file by typing the following commands:
$ module load <compiler>
$ spack compiler find

The file compilers.yaml is created by default into $HOME/.spack/<platform> path.

In the second case, you use the compiler module already installed on the cluster, you have to specify it simply:

#e.g. gcc 8.3.0:
$ spack install <pack> %gcc@8.3.

If you want to use a library already available on the cluster in order to install your application through spack module you have to specify it through ^ type:

#e.g. zlib@1.2.11
$ spack <pack> ^zlib@1.2.11

In order to use the software just installed with spack you can load the corresponding module:
$ module load/unload <module>

or by spack commands:
$ spack load/unload <software name> or /<hash>

You can see the complete software name or the hash (e.g. icynozk ) of the software by tiping "spack find -l <name>" command

If the module of your installation has not been created, "module av", type this command
$ spack module tcl refresh <name>

**Python and additional software**

In case you need a particular python package that is not already installed on our systems you can install it by yourself using the virtualenv tool.
You can follow these instructions:

- load python interpreter from the module
  
  $ module load python/3.6.4

- create a virtualenv, basically, it is just a new directory (my_venv) containing all you need
  
  $ virtualenv my_venv

- activate the new virtualenv
  
  $ source my_venv/bin/activate

- install whatever you need (e.g matplotlib)
  
  (my_venv) $ pip install matplotlib

- when you have finished your business you can deactivate the virtualenv by
  
  (my_venv) $ deactivate

Some packages (mpi4py, numpy, scipy, ...) could be already available as modules, check with

$ modmap –m <package_name>