Using software on the SCITAS clusters

The clusters come with a common programming environment. Here is how to use the software included within.

- Software policy and release cycles
- Supported software stacks
- Modules and LMOD
- Searching for software
- The slightly ugly reality
- Saving your environment
- GPU environment and heterogeneous clusters
- Reverting to the old environment
- Related articles

Software policy and release cycles

SCITAS provides up to three software releases on clusters:

<table>
<thead>
<tr>
<th>Release name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deprecated</td>
<td>This is the old production environment and is no longer supported, but will be retained for one year.</td>
</tr>
<tr>
<td>stable</td>
<td>This is the current stable release for the year in progress. We guarantee that the modules here will work and that modules will not be removed.</td>
</tr>
<tr>
<td>future</td>
<td>This is the testing area for what will become the next production environment and it is not guaranteed to exist! Modules in here are not guaranteed to work and may be removed without warning.</td>
</tr>
</tbody>
</table>

When you connect to the clusters you will see stable, i.e. the current production release. To switch to a different release see the section Reverting to the old environment below.

Every summer (approximately around mid July) SCITAS will rotate the environments to update them. Future will become stable, stable will become deprecated and the current deprecated won't be accessible anymore. Users will be notified of the change well in advance, via e-mail or other communication means - including MOTD on clusters, so keep an eye on them!

Supported software stacks

SCITAS fully supports the following software stacks - a stack being composed of a compiler, an MPI library and a LAPACK library - on its clusters:

<table>
<thead>
<tr>
<th></th>
<th>Compiler</th>
<th>MPI</th>
<th>LAPACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU</td>
<td>GCC 7.4.0</td>
<td>MVAPICH 2.3.1</td>
<td>Openblas 0.3.6</td>
</tr>
<tr>
<td>Intel</td>
<td>Intel 2018 Update 5</td>
<td>Intel MPI 2018.4.274</td>
<td>Intel MKL 2018.5.274</td>
</tr>
</tbody>
</table>

The full support implies a commitment to act as promptly as possible to fix any issue in the software that has already been installed as part of either stack.

For the convenience of its users, SCITAS also supports on a best effort basis the following:

<table>
<thead>
<tr>
<th></th>
<th>Compiler</th>
<th>MPI</th>
<th>LAPACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU 8.3.0</td>
<td>GCC 8.3.0</td>
<td>MVAPICH 2.3.1</td>
<td>OpenBLAS 0.3.6</td>
</tr>
</tbody>
</table>

Software libraries (e.g. FFTW, HDF5, etc.) will be installed for all the aforementioned combinations. These libraries may have multiple versions due to the implementation and functionality required. For instance, the FFTW library is installed with different levels of support for multi-threading and MPI.

End-user applications (e.g. Quantum-ESPRESSO, Gromacs, etc.) will be installed only as part of the two fully supported stacks and only in one version (with the configuration decided by the SCITAS application experts). If users require different options then SCITAS will provide assistance for the user to compile their own version.

Modules and LMOD

The SCITAS managed clusters use the LMOD tool to manage scientific software. This is compatible with the classical Modules tool but brings with it a large number of improvements.
The official LMOD documentation can be consulted at: http://lmod.readthedocs.io/en/latest/

A slightly simplified example of using LMOD is:

1. Connect to a cluster and see what base modules are available. These are either compilers or stand alone packages such as MATLAB.

   ```bash
   $ module avail
   --------------- /path/to/base/modules ---------------
cmake  gcc  intel  matlab
   ```

2. Load a compiler to see the modules built with the chosen compiler. These may be scientific libraries, serial (non-MPI) codes or MPI libraries.

   ```bash
   $ module load gcc
   $ module avail
   --------------- /path/to/gcc/modules ---------------
gdb  fftw  hdf5  mvapich2  openmpi  python  R
   --------------- /path/to/base/modules ---------------
cmake  gcc  intel  matlab
   ```

3. Load a MPI library to see the modules that use this MPI flavour

   ```bash
   $ module load mvapich2
   $ module avail
   --------------- /path/to/mvapich2/modules ---------------
boost  fftw  hdf5  gromacs  espresso  parmentis
   --------------- /path/to/gcc/modules ---------------
cmake  gdb  fftw  hdf5  mvapich2  openmpi  python
   --------------- /path/to/base/modules ---------------
cmake  gcc  intel  matlab
   ```

   LMOD knows which modules are incompatible and will take the neccessary steps to ensure a consistent environment:

   ```bash
   $ module load openmpi
   Lmod is automatically replacing "mvapich2" with "openmpi"
   ```

**Searching for software**

Currently if you want to search for software you need to load one software stack. For example
The slightly ugly reality

In reality running "module avail fftw" returns (after having loaded gcc and mvapich2)

$ module load gcc mvapich2
$ module av fftw

/sssoft/spack/cornalin/v1/share/spack/lmod/linux-rhel7-x86_E5v2_IntelIB/mvapich2/2.2-pppg4iu/gcc/5.4.0
fftw/3.3.6-pl2-mpi-openmp fftw/3.3.6-pl2-mpi

/sssoft/spack/cornalin/v1/share/spack/lmod/linux-rhel7-x86_E5v2_IntelIB/gcc/5.4.0
fftw/3.3.6-pl2-openmp fftw/3.3.6-pl2 (D)

Where:
D: Default Module

The names are <module name> / <version - options> with the options being the "key" configuration options such as MPI or OpenMP activation.
The (D) after a module name indicates that, if there are two or more version of the same package available then this is the version that would be loaded by default.

If you need a specific version due to the options with which it is built then you have to specify the full name:
$ module purge
$ module load gcc
$ module load mvapich2
$ module load fftw/3.3.6-pl2-mpi-openmp

If you really want to know how a module was built then you need to run "module whatis <modulename>" which will contain, wherever possible, the list of options used at configure time:

$ module load intel intel-mpi
$ module whatis hdf5/1.10.1-mpi

hdf5/1.10.1-mpi     : Name : hdf5
hdf5/1.10.1-mpi     : Version : 1.10.1
hdf5/1.10.1-mpi     : Short description : HDF5 is a data model, library, and file format for storing and managing data. It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O and for high volume and complex data.

Saving your environment

If you have a few sets of modules that you use regularly then a nice feature of LMOD is the ability to save sets of modules and reload them with a single command

[user@system ~]$ module list
Currently Loaded Modules:
  1) gcc/5.4.0   2) mvapich2/2.2   3) gdb/7.12.1

[user@system ~]$ module save dev_env
Saved current collection of modules to: dev_env, for system: "x86_E5v2_IntelIB"

[user@system ~]$ module purge

[user@system ~]$ module list
No modules loaded

[user@system ~]$ module restore dev_env
Restoring modules to user's dev_env, for system: "x86_E5v2_IntelIB"

[user@system ~]$ module list
Currently Loaded Modules:
  1) gcc/5.4.0   2) mvapich2/2.2   3) gdb/7.12.1

Because each cluster has a different base module path then the saved set is only valid for one architecture (the system type given when you save).

If you try and load a module collection on a different system type you will see:

[user@other_system ~]$ module restore dev_env
Lmod has detected the following error:   User module collection: "dev_env" does not exist.
Try "module savelist" for possible choices.

For this reason you should never use module restore in job scripts. You can, of course, save the same set of modules with the same name on multiple clusters so as to have the same environment everywhere.

GPU environment and heterogeneous clusters
For homogeneous clusters such as Fidis and Castor the environment on the compute nodes is the same as that on the front-end machines. Deneb is slightly different as it has a partition containing machines with GPUs as well as a slightly different Infiniband configuration. If you wish to have access to the GPU node environment (i.e. the CUDA runtime and correct MPI) on the login machines then run:

```
[user@system ~]$ slmodules -s x86_E5v2_Mellanox_GPU -v
[INFO] S+L release: stable
[INFO] S+L systype: x86_E5v2_Mellanox_GPU
[INFO] S+L engaged!
```

To switch back to the architecture of the machine on which you are run the command without options:

```
$ slmodules -v
[INFO] S+L release: stable
[INFO] S+L systype: x86_E5v4_Mellanox
[INFO] S+L engaged!
```

The list of architectures you can currently switch to is:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Cluster(s) with nodes of this kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86_E5v2_IntelIB</td>
<td>Deneb (deprecated only)</td>
</tr>
<tr>
<td>x86_E5v2_Mellanox_GPU</td>
<td>Deneb (deprecated only)</td>
</tr>
<tr>
<td>x86_E5v3_IntelIB</td>
<td>Deneb (deprecated only)</td>
</tr>
<tr>
<td>x86_E5v4_Mellanox</td>
<td>Fidis</td>
</tr>
<tr>
<td>x86_S6g1_Mellanox</td>
<td>Fidis, Helvetios</td>
</tr>
</tbody>
</table>

Reverting to the old environment

In order to revert to the old environment then run "slmodules -r deprecated"

```
[user@system ~]$ slmodules -r deprecated -v
[INFO] S+L release: lafnetscha.v2
[INFO] S+L systype: x86_E5v4_Mellanox
[INFO] S+L engaged!
```

In a job script you need to source the full command "/ssoft/spack/bin/slmodules.sh”:

```
source /ssoft/spack/bin/slmodules.sh -r deprecated -v
module purge
module load foo
```

Behind the scenes

The software environment on the clusters is managed using SPACK. EPFL is a major contributor to the tool: [http://software.llnl.gov/spack/](http://software.llnl.gov/spack/)

This allows us to deploy software for multiple architectures and compiler/mpi variants in a consistent and automated manner.

Related articles

- First steps on the clusters
- Using the clusters the reproducible way
- Compiling codes on different systems
- Using software on the SCITAS clusters
- Requesting the installation of a software