Automating Software Stack Deployment on an HPE Cray EX Supercomputer

Dr. Pascal Jahan Elahi, Cristian Di Pietrantonio, Dr. Marco De La Pierre, Dr. Deva Kumar Deeptimahanti

2023 Cray User Group Conference

Pawsey Supercomputing Research Centre

Headquarted in Perth, Western Australia, Pawsey has a 20-year long history. Offers critical support to radioastronomy research around the Square Kilometre Array (SKA). The centre underwent a 70m capital refresh financed by the Australian government. Currently employs 60+ staff.

The Setonix supercomputer

- Australia's most powerful research supercomputer.
- HPE Cray EX system with 200'000 AMD Zen3 CPU cores and 750+ MI250X GPUs.
- 50 PFLOPS, 90% coming from AMD GPUs.
- 15PB `/scratch` storage.
- 15^{th} in TOP500, 4^{th} in Green500.
- Artwork by aboriginal artist Margaret Whitehurst.

Outline

HPC Software
Introduction and motivation

Supercomputing software stacks are becoming increasingly complex..

- Increasing number of applications and libraries from a variety of fields of science.
- Multiple CPU architectures, compilers and supporting libraries.
- Multiple configurations for the same software based on user requirements.
- Constant evolution of the software provided by HPE Cray.

Need automation of the deployment process to improve deployment times, reduce human errors and *reduce the workload on staff*.

Pawsey's Approach
Introduction and motivation

- Built around Spack and Singularity HPC (SHPC).
- Integrates with the HPE Cray environment to provide multiple builds.
- Allows the same Spack instance to be used for system-wide and user installations.
- Uses the Lmod module system to expose software deployed with Spack, SHPC and custom build methods.
- This process can deploy the current software stack in *less than a day*. We have deployed several software stacks on Setonix, an HPE Cray EX system.

The automation package can be downloaded from `github.com/PawseySC/pawsey-spack-config.git`.

Design

Software stack components and organization

- Software stack has three levels of deployment (and associated support):
  - System-wide: accessible by all and maintained by Pawsey.
  - Project-wide: maintained by the project and accessible by project members.
  - User-private: individual Pawsey users installations.
- It consists of
  - Optimized bare-metal installations accessed through modules.
  - Containers accessed through modules.
- The module files providing software are organized in informative, user-friendly categories (e.g., applications, libraries, containers, etc.).
- Supported versions follow a general $3 + 1$ rule (legacy, stable, latest).

Software Stack Organization

Software stack components and organization

The system-wide stack consists of five components, each having a top-level directory in the installation prefix <system>/DATE_TAG.

1. `spack/`: Spack package manager installation directory.
2. `software/`: Spack-built software packages installation directory.
3. `containers/`: SHPC deployed containers.
4. `modules/`: Module files for Spack-built software packages.
5. `pawsey/`: Contains modules and wrappers to properly expose three levels of installation.
6. `custom/`: Contains binaries built by means other than Spack.

Spack Instance

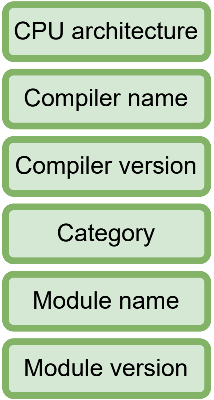
Software stack components and organization

- Currently using Spack 0.17.0 in production, moving to 0.19.x.
- A `spack` Linux user is employed for system-wide software installations.
- Additional scripts implement new features like multi-level installation paths (project-wide and user-level).
- Spack is patched to allow the use of configuration defined `.spack/` directory.
- New recipes and fixes to existing recipes (dependencies, specs, patching source code, integration with CPE). Examples are:
  - Existing - Amber, Charmpp, astropy, casacore
  - Quantum Computing (new) - Qutip, Xanadu Pennylane, Xanadu Strawberry Fields
  - Bioinformatics (new) - tower-agent, tower-cli, nf-core tools
  - Radioastronomy (new) - wsclean, vcstools

Module Hierarchies

Software stack components and organization

- HPC software stack must deal with multiple compilers & CPU architectures.
- This motivates use of software hierarchies, where paths reflect currently loaded modules and underlying CPU architecture.
- We use Lmod to handle software hierarchies.



Module Hierarchies for CPE

Software stack components and organization

- Cray's customized Lmod installation implements an undocumented custom way of handling hierarchies in substitution for the standard Lmod way.
  - Compiler, CPU architecture and MPI library modules scan the shell environment for a specific variable that sets additional module paths.
  - Paths are (un)set when the relevant module is (un)loaded (unloaded). For example, modules for a GCC compiler will look for `LMOD_CUSTOM_COMPILER_GNU_8_0_PREFIX`.
  - On Setonix, code is in `/opt/cray/pe/admin-pe/lmod_scripts/lmodHierarchy.lua`.
- We implement this process for Spack-built software packages through a auto-loaded module, `pawseyenv.lua`.

Module Hierarchies for CPE

Software stack components and organization

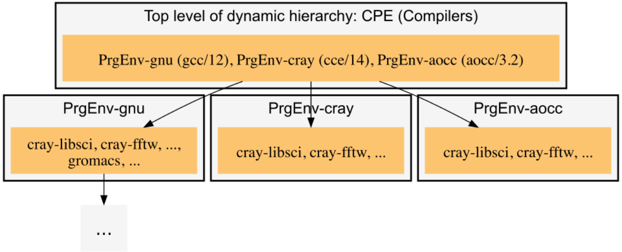


Figure: Example of CPE software hierarchy

Module Hierarchies for CPU Architectures
Software stack components and organization

- Setonix hosts a variety of AMD CPU architectures (Zen-2, Zen-3)
- Module hierarchies are based on the CPU architecture using `lscpu` at user login time to the node.
  - CPU architecture is saved in an environment variable for later use in forming search paths for module files.

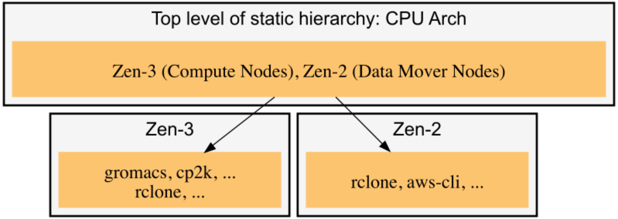


Figure: Example of CPU architecture software hierarchy

Containers-as-modules

Software stack components and organization

- Certain software packages are provided system-wide as containers using SHPC, which can be executed with the Singularity container engine. These include
  - Bioinformatics packages - complex dependency trees. Containers provided time-effective deployment.
  - OpenFoam - CFD package where older releases are required and poor IO performance resolved using overlay filesystems mounted to the running containers.
  - Bespoke Python stack for HPC - comprises all Pawsey-supported Python scientific packages.
- Containers are outside the software hierarchy of bare-metal builds through container-as-modules produced by SHPC.

User's View of Software Stack

Software stack components and organization

- Users access all Pawsey-supported software through Lmod modules.
- The resulting Lmod software hierarchy for Spack-built software is one where
  - An architecture hierarchy is static and set at shell login time.
  - A compiler hierarchy is dynamic and dependent on the current active CPE.
- Containers as modules do not have a hierarchy since the underlying container does not depend on the CPU architecture nor the active CPE.
- We also force module load commands to request both module name and module version to enforce good practice and improve reproducibility.

```
-------------------------------------------------- /software/setonix/current/containers/views/modules ---------------------------
   alphafold/2.2.3              gatk4/4.2.5.0--hdfd78af_0              openfoam-container/v2212                    (D)
   bamtools/2.5.1--hd03093a_10  gromacs-amd-gfx90a/2022.3.amd1_174     openfoam-org-container/7

---------------------------------------------- /software/setonix/current/modules/zen3/gcc/12.1.0/libraries ----------------------
   adios2/2.7.1-hdf5        hdf5/1.10.7-api-v18         hdf5/1.12.1-parallel-api-v112 (D)    netcdf-cxx/4.2
   blaspp/2021.04.01        hdf5/1.10.7-api-v110        hpx/1.6.0                            netcdf-cxx4/4.3.1

-------------------------------------------- /software/setonix/current/modules/zen3/gcc/12.1.0/applications --------------------
   cp2k/8.2   gromacs/2020.4   gromacs/2021.4 (D)   lammps/20210929.3   namd/2.14   nektar/5.0.2   nwchem/7.0.2   quantum-espresso/6.8
```

Figure: Example of modules on Setonix

Deployment system organization
Automated deployment

The deployment system is made of BASH scripts and Spack configuration files located in the following directories:

- `fixes/`: patches to adapt Spack for Pawsey-specific use cases.
- `repo/`: Pawsey-written Spack package recipes.
- `shpc_registry/`: custom Singularity-HPC (SHPC) recipes.
- `scripts/`: BASH scripts used to automate the deployment process.
- `systems/<system>`: a directory containing configuration files specific to a system.

The `scripts/install_software_stack.sh` is the top-level script that executes the installation from start to finish except licensed software, that need some manual work.

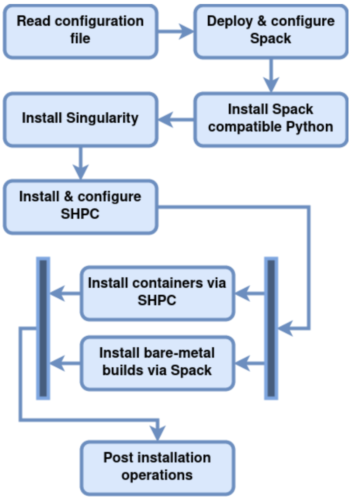
The ⟨system⟩ directory
Automated deployment

```
.
├── configs
│   ├── project
│   │   ├── config.yaml
│   │   ├── modules.yaml
│   │   ├── packages.yaml
│   ├── site
│   │   ├── compilers.yaml
│   │   ├── concretizer.yaml
│   │   ├── config.yaml
│   │   ├── modules.yaml
│   │   ├── packages.yaml
│   │   ├── repos.yaml
│   └── spackuser
│       ├── config.yaml
│       ├── modules.yaml
├── environments
│   ├── env_apps
│   │   └── spack.yaml
│   ├── env_num_libs
│   │   └── spack.yaml
│   :
├── settings.sh
└── templates
    └── modules
        └── modulefile.lua
```

1. Spack config files for each deployment level.
   - `site` configs apply to all users, and provide general settings.
   - `project` configs overrides paths for project-wide installations.
   - `spackuser` configs are used by Pawsey staff to deploy system-wide software.
2. `environments` specify the software to be installed system-wide, grouped in categories.
3. `settings.sh` sets a number of variables to customise the deployment process (system-wide installation path, build directories, containers to install).
4. Module file template for Spack to generate module files.

Deployment execution

Automated deployment



1. Deploy Spack with customised configuration for the particular system.
2. Need to install a newer Python version, with Spack, for Spack to work correctly.
3. Singularity is needed to install Singularity-HPC for container deployment.
4. Spack environments can be installed in parallel, and at the same time containers are deployed with SHPC.
5. Finally, we deal with licensed software, customise module files where necessary, and create directory trees in the system-wide and user-private deployment levels.

Rebuilds, upgrades, and maintenance

Automated deployment

- The same deployment can be updated by interactively adding new software.
- Re-deploy stack regularly (every 6 months) to take advantage of newer compilers and libraries.
- Stack has to be re-deployed when the Cray environment changes.
- Multiple deployments can be live on the system at the same time. This will ensure year-long support.

Evaluation

- 1st deployment: May 2022
  - Minimal automation, each supporting script executed manually.
  - Instrumental to validate each step of the process.
  - Troubleshooting of installing some packages with latest compilers, generation of module files, interplay between CPU architectures and Cray environment.
  - Deployment took two weeks.
- 2nd deployment: November 2022
  - Very first iteration of the automated process.
  - New person responsible for deployment, good test for usability and reproducibility.
  - Minimise required human interaction by introducing the "driver" script and collecting all the settings in a `settings.sh` file.
  - After several test iterations, we managed to deploy the software stack in a day.

Future work

- Minimise external dependencies, build everything with Spack.
- Better resolution of conflicts when creating module files.
- Integration of ROCm to build GPU-enabled packages.
- Reframe testing triggered by the deployment scripts.

Thanks for listening. Any questions?