Spack

A quick intro

Thomas Madlener

Nov 25, 2021

What is spack?

- Spack is a package management tool
  - Similar to e.g. apt, yum, pacman, conda, ...
- Designed to support multiple versions and configurations of software packages
- Builds software from scratch independently of language
- Originally developed in the HPC community
- Resources:
  - spack.io
  - spack.readthedocs.io
- Today: Basics of spack and environments
- Future: How spack is used in Key4hep, advanced usage

Some spack lingo

`package`	A python module in spack that describes how to build a software package or library
`variant`	Different configurations or features of a `package`
`spec`	An expression describing a particular build of a `package` (i.e. versions, `variant`s, etc. fixed)
`concretize`	The process of figuring out all the dependencies and their exact versions to arrive at a *concretized* `spec` from a `package`

Basic package

```
9  class Sio(CMakePackage):
10     """SIO is a persistency solution for reading and writing binary data in SIO
11     structures called record and block. SIO has originally been implemented as
12     persistency layer for LCIO.
13     """
14
15     url      = "https://github.com/iLCSoft/SIO/archive/v00-00-02.tar.gz"
16     homepage = "https://github.com/iLCSoft/SIO"
17     git      = "https://github.com/iLCSoft/SIO.git"
18
19     maintainers = ['vvolkl', 'tmadlener']
20
21     version('master', branch='master')
22     version('0.1',    sha256='0407c0daeae53660c0562f9302a220f72ab51547050cd9fe9113b995804ab4b4')
23     version('0.0.4', sha256='72e96e6a1cc8dd3641d3e2bb9876e75bf6af8074e1617220da9e52df522ef5c0')
24     version('0.0.3', sha256='4c8b9c08480fb53cd10abb0e1260071a8c3f68d06a8acfd373f6560a916155cc')
25     version('0.0.2', sha256='e4cd2aeaeaa23c1da2c20c5c08a9b72a31b16b7a8f5aa6d480dcd561ef667657')
26
27     variant('builtin_zlib', default=True,
28             description='Use and statically link against a builtin version of zlib')
29     variant('cxxstd', default='17',
30             values=('11', '14', '17', '20'),
31             multi=False,
32             description='Use the specified C++ standard when building.')
33
34     depends_on('zlib', when='~builtin_zlib')
```

- Where to get sources
- Existing versions
- Available variants
- Dependencies

Demo time

Getting Started (readthedocs)

Basic usage (readthedocs)

Setting up spack for the first time

- Get spack
  `git clone https://github.com/spack/spack`
- Setup spack (**also necessary if you want to use spack after initial setup**)
  `. spack/share/spack/setup-env.sh`
- Detect available compilers from the underlying system
  `spack compiler find`
- Detect packages from the underlying system (needs to be enabled by the package)
  `spack external find cmake python bash`
- Detect packages from the underlying system which should always be used
  `spack external find --not-buildable openssl`
- Spack will bootstrap the `clingo` concretizer the first time it is used

Installing a package

- Initialize spack (if not already happened)
  ```
  . spack/share/spack/setup-env.sh
  ```
- Install a package
  ```
  spack install sio
  ```
- Finding out what will be built before installing
  ```
  spack spec -Il sio
  ```
- Get available build options for a package
  ```
  spack info sio
  ```
- Specify non-default settings for a package
  ```
  spack spec -Il sio~builtin_zlib ^zlib@1.2.8 %gcc@9.3.0
  spack install sio~builtin_zlib ^zlib@1.2.8 %gcc@9.3.0
  ```
- Checking available installations
  ```
  spack find -lv sio
  ```

Using spack installed packages

- An installed package needs to be loaded in order to use it (potentially have to be specific of which one you want if there are multiple versions available)

```
spack load sio/gkhr556
spack load sio %gcc@10.3.0
```

- Removing them from the environment again via

```
spack unload sio
```

- You can get the commands for setting up the same environment via (sh in this case, other shells are available as well)

```
spack load --sh sio/gkhr
```

- You can load the same package multiple times (even in different versions)
  - Best way to end up with a hard to reproduce environment ;-)
- Spack takes care of a clean build environment when installing packages

Spack environments (readthedocs)

- Setting up a consistent runtime environment with several packages can be a bit cumbersome with spack
  - Spack does not enforce that `spack load <package>` yields a consistent environment
  - Shared dependencies might be present with different `spec`s
- Spack has its own notion of *environment*s
  - Can be used to make sure that different packages are built in a consistent way
  - Splits apart the steps of what to install, concretizing and installing
  - Can be used to install the **exact same** software stack again
- Comparable to *Conda environments* or *Python Virtual Environments*
- An alternative way to get a consistent software setup is the use of `BundlePackage`s
  - Slightly less flexible then environments

Creating a spack environment

- Create an empty environment called `tutorial-env`
  ```
  spack env create tutorial-env
  ```
- Activate the environment (the `-p` is optional and only decorates the prompt with the environment name)
  ```
  spack env activate -p tutorial-env
  ```
- Add packages to the environment
  ```
  spack add <packages>
  ```
- Check if you are in an activated environment, and if so in which
  ```
  spack env status
  ```
- Deactivate an environment (alias for `spack env deactivate`)
  ```
  despacktivate
  ```

A few things to know about spack environments

- Some spack commands behave (slightly) differently in an activated environment than they do in general
  - E.g. `spack find` will only show `spec`s that are part of the environment
- **Note that `spec`s installed in an environment will also be available outside the environment.**
  - There is fundamentally no difference between a `spec` in an environment to other `spec`s
  - Environments are just a way to collect and refer to different `spec`s

Spack environment concretization

- Environments settings and packages are recorded in a `spack.yaml` config file that lives at
  `$SPACK_ROOT/var/spack/environments/<env-name>/spack.yaml`
- Environments have to be concretized before they can be installed (if not created from a lockfile)
  `spack concretize`
- This will generate a `spack.lock` lockfile where all the concrete `spec`s to install for this environment are recorded
  - Adding new `spec`s will not change already concretized `spec`s in an environment unless you do
    `spack concretize -f`
- Can also create environments from a lockfile or a yaml config file

```
spack env create tutorial-env /spack_resources/tutorial-env.yaml
spack env create locked-env /spack_resources/spack.lock
```

Spack environment configuration and installation

- To edit an existing spack environment, either use environment sensitive commands (`spack add`, ...)
- Or edit the `spack.yaml` file
- Do not forget concretize again (depending on how you want to propagate your changes to the environment)
  `spack concretize -f`
- Install an environment
  `spack install`

Some observations (the good, the bad and the ugly)

- Spack commands are very discoverable from the command line (e.g.)

```
spack --help
spack install --help
spack env activate --help
```

- `spec`s are very specific and include all their dependencies
  - If one dependency changes many packages will be installed again even if the exact `spec` of the depenency does not matter too much
- Spack releases and package updates are not separated
  - Concretizer usually prefers newer versions so updating spack potentially means rebuilding a lot of your software
  - Has become a bit better recently with a `--reuse` flag which makes the concretizer prefer existing packages
- Spacks main purpose is to install software not so much develop software

Some more useful resources

- spack/spack - The spack github repository
- Spack configuration (readthedocs)
- Spack environments (readthedocs)