NHR FAU

Friedrich-Alexander-Universität Erlangen-Nürnberg

HPC Café

Software Installation for Users – Conda, Containers, SPACK

Markus Wittmann, Katrin Nusser, Thomas Gruber

September 12, 2023

Managing environments and packages with conda

HPC Services, NHR@FAU

hpc-support@fau.de

conda

- conda is a package and environment manager
- Not limited to Python packages, unlike pip

- conda is included in Anaconda and miniconda distribution
  - By default uses repo.anaconda.com, the Anaconda repository
  - Can be configured to use additional/other repos/channels

- Documentation: https://docs.conda.io/projects/conda/en/stable/
- Cheat sheet: https://docs.conda.io/projects/conda/en/stable/user-guide/cheatsheet.html

conda at NHR@FAU

- **`python/x.y-anaconda`** modules provide
  - Python **`x.y`**
  - An Anaconda installation with a base environment
  - Our base environment already comes with several packages installed
- No need to maintain an own Anaconda or miniconda installation

For example on Alex cluster:

```
module add python/3.9-anaconda
conda --version
# conda 23.1.0
conda list | wc -l
# 158
```

Configuration at NHR@FAU

Only required the first time you use any Python module:

Ensure environments are stored under **$WORK** and not **$HOME**

```
module add python/x.y-anaconda

# only required the first time you use a Python module
# change conda's config to store environments and packages under $WORK
conda config --add pkgs_dirs "$WORK/software/privat/conda/pkgs"
conda config --add envs_dirs "$WORK/software/privat/conda/envs"

conda info
# newly configured directories should show up
```

conda environments

- Named collection of installed conda packages
- You can activate, deactivate, and switch between environments
- Comparable to Python virtual environments (not discussed here)

- Typically you have different environments for different purposes that contain different packages

- Works to some degree also with pip

Creating and removing environments

- Create an environment

```
conda create -n <env name> [package[=version]]*
```

If you want to use **`pip`** inside the env. later:
always explicitly install a Python or pip package

```
# e.g. create environment with Python 3.10
conda create -n <env name> python=3.10
```

- Remove an environment

```
conda remove --name <env name> --all
# or
conda env remove --name <env name>
```

https://docs.conda.io/projects/conda/en/stable/user-guide/tasks/manage-environments.html
https://docs.conda.io/projects/conda/en/stable/user-guide/tasks/manage-python.html
https://docs.conda.io/projects/conda/en/stable/user-guide/configuration/pip-interoperability.html

Activating and deactivating environments

- Activate an environment

```
conda activate <env name>
```

Prefixes your prompt with `(<env name>)`:

```
$ conda activate torch
(torch) $
```

- Deactivate current environment

```
conda deactivate
```

- List available environments

```
conda info --envs
# or
conda env list
```

The `*` prefixed one is active

```
(test) $ conda info --envs
# conda environments:
#
base                      /apps/python/3.9-anaconda
pytorch-1.10              /.../envs/pytorch-1.10
tensorflow-gpu-2.7.0      /.../envs/tensorflow-gpu-2.7.0
...
test                   *  /home/.../conda/envs/test
test2                     /home/.../conda/envs/test2
```

Working with packages

- List installed packages inside current env.:

```
conda list
```

- Searching a package

```
conda search <pkg>
conda search --info <pkg>
```

<pkg> can contain wildcards, quote them: `'pytorch*'`

- Installing a package

```
conda install <pkg>[=version]
```

- Updating a package

```
conda update <pkg>
```

- Updating all packages

```
conda update --all
```

- Removing a package

```
conda remove <pkg>
```

https://docs.conda.io/projects/conda/en/stable/user-guide/tasks/manage-pkgs.html

conda and pip

You can use conda and pip together, up to a certain degree:

1. Always create an environment with `python` or `pip` package installed
2. Optionally: install packages with conda first
3. Afterwards, install packages with pip, **do not use** the `--user` flag
4. If you need to install additional conda packages
   - It might work, if not
   - Recreate the environment

```
# load Python module
module load python/x.y-anaconda

# create environment with Python installed
conda create -n <env> python=A.B

# activate environment
conda activate <env>

# optionally: install conda packages first
conda install <conda packages>

# install pip packages
pip install <pip packages>
```

- More details
  - https://www.anaconda.com/blog/using-pip-in-a-conda-environment
  - https://www.anaconda.com/blog/understanding-conda-and-pip
  - https://docs.conda.io/projects/conda/en/latest/user-guide/configuration/pip-interoperability.html

Using other channels

- Channels are locations where conda installs packages from
  - By default the default channel is used
- Other popular channels
  - conda-forge: https://conda-forge.org
  - bioconda: https://bioconda.github.io/

- Add a channel, e.g. conda-forge
  - `conda config --add channels conda-forge`
  - Alternatively: specify channel with `-c <channel>` for some commands
- Be careful about package collisions when multiple channels are used

- Further information:
  - Managing channels: https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-channels.html
  - Conda channels: https://docs.conda.io/projects/conda/en/latest/user-guide/concepts/channels.html
  - Using conda-forge: https://conda-forge.org/docs/user/introduction.html#how-can-i-install-packages-from-conda-forge

Issues you might encounter…

- Not all compute nodes can connect to the internet, configure a proxy before executing your application or install software via conda, pip, …

```
export http_proxy=http://proxy:80
export https_proxy=http://proxy:80
```

- If your **`~/.bashrc`** is not sourced
  - Is bash invoked with **`-l`** flag?
  - Do you have a **`~/.bash_profile`**? If not, create one with this content:

```
if [ -f ~/.bashrc ]; then . ~/.bashrc; fi
```

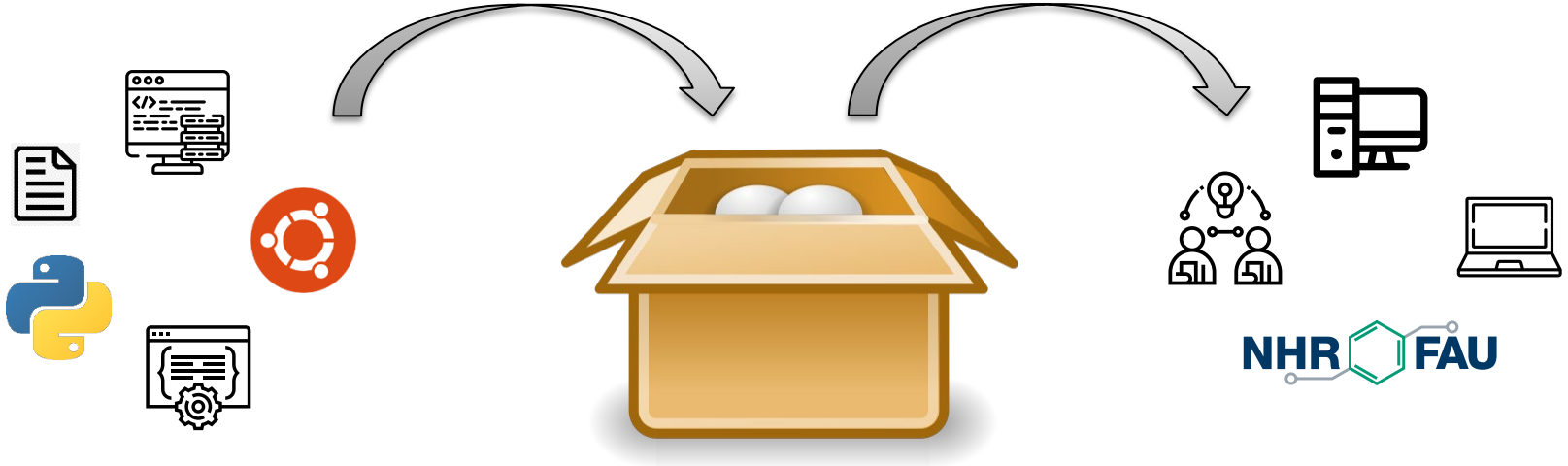
  - https://www.gnu.org/software/bash/manual/html_node/Bash-Startup-Files.html

Software installation for users

Containers

What is a Software Container?

A container allows you to stick your application and ALL of its dependencies into a single package. This makes the application portable, shareable and reproducible across different computing platforms and environments.

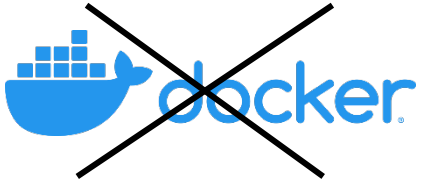


What are the most common use cases?

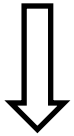
- Your software already comes pre-packaged inside a (Docker) container.

- You want to package your software stack yourself, because
  - dependencies are difficult to satisfy otherwise (OS version, system library versions, graphics libraries, …).
  - it is too complex to set it up from scratch on different systems.
  - you want exact reproducibility.
  - …

Container frameworks



Most popular container framework, but not suitable for HPC platforms due to security concerns.



(formerly known as Singularity)

- Specifically designed for HPC, no performance penalties
- No root access on production system necessary
- Can convert Docker containers to Apptainer format
- All file systems are automatically mounted in container (/apps, /home, /scratch,…)
- Supports GPU-dependent applications
- Caveat: problematic for MPI applications on multiple nodes

Basic usage

```
$ apptainer
Usage:
  apptainer [global options...] <command>

Available Commands:
  build       Build an Apptainer image
  cache       Manage the local cache
  exec        Run a command within a container
  help        Help about any command
  pull        Pull an image from a URI
  push        Upload image to the provided URI
  run         Run the user-defined default command within a container
  run-help    Show the user-defined help for an image
  search      Search a Container Library for images
  shell       Run a shell within a container
  sif         siftool is a program for Singularity Image Format (SIF) file manipulation
  sign        Attach a cryptographic signature to an image
  test        Run the user-defined tests within a container
  verify      Verify cryptographic signatures attached to an image
  version     Show the version for Apptainer
  […]
```

Apptainer Quick Start Guide (with examples for all basic commands and workflows):
https://apptainer.org/docs/user/latest/quick_start.html

Using existing containers – Example

- Download/pull container, e.g.:

```
apptainer pull docker://ubuntu
```

- Enter container with shell:

```
apptainer shell <container_name>
```

- Execute commands inside container:

```
apptainer exec <container_name> <command>
```

- Run pre-defined runscript of container:

```
apptainer run <container_name>
```
or
```
./<container name>
```

- Check container metadata:

```
apptainer inspect  --runscript <container_name>
```

Build your own container - interactively

- Containers can be build on the cluster frontend nodes!

- Create sandbox:
```
apptainer build --sandbox <sandbox_name> docker://ubuntu
```

- Enter (writable) container with shell:
```
apptainer shell --writable <sandbox_name>
```

- Install/setup software inside container

- Convert sandbox to image and back again:
```
apptainer build <container_name>.sif <sandbox_name>
apptainer build --sandbox <sandbox_name> <container_name>.sif
```

Build your own container – from definition file

```
Bootstrap: docker
From: ubuntu:latest

%post
  apt-get dist-upgrade
  apt-get update
  apt-get install -y python
  mkdir /test
  mv /python_sum.py /test

%files
  python_sum.py

%runscript
  exec "python" "/test/python_sum.py" "$@"
```

Uses pre-built Ubuntu container from Docker; many more bootstrap agents available

Defines what happens during installation (download software and libraries, create directories, …)

Can be used to copy files into container; (before %post section)

This is executed when container image is run (via `apptainer run` or directly)

Build container image from definition file:

```
apptainer build <container_name>.sif <definition_file>
```

Build your own container – for GPUs

Apptainer natively supports running GPU-enabled applications inside a container.

On Alex/TinyGPU: GPU device libraries are automatically bind-mounted into container, no additional option necessary.

Requirements:

- Host has working installation of GPU driver and CUDA libraries (Alex, TinyGPU)
- CUDA version of application inside container must be compatible with host installation

If you encounter problems with missing GPU-support, try commands `run`/`shell`/`execute` with `--nv` option, e.g. `apptainer run --nv <container_name>`.

Additional hints

- Per default, all file systems (/home, …) are mounted inside a container.
  - To prevent mounting any file systems: `apptainer run -contain <container_name>`
  - Specify different home directory: `apptainer run -H $HOME/my-container-home <container_name>`

- Pulled container images are by default saved to `$HOME/.apptainer/cache`. Set environment variable `$APPTAINER_CACHEDIR` to different location, e.g. `$WORK`.

- Using MPI inside containers is not recommended, as it requires the exact version of the host MPI implementation including all dependencies (e.g. Slurm, Infiniband libraries, …) to work properly.

- Apptainer User Guide: https://apptainer.org/docs/user/latest/index.html

Software installation for users

SPACK

SPACK

- Package manager for supercomputers
- Project started by Livermore National Lab (LLNL)
- Easy installation of scientific software
- Easily swap compilers and build options
- Target specific microarchitectures (for optimizations)
- Written in Python
- Integration with common `module` system

Homepage: https://spack.io

Repository with package recipes: https://github.com/spack/spack

SPACK at NHR@FAU systems

- Uses some OS-provided packages
- Basic packages are partly pre-installed in a platform-specific repo

```
$ module load user-spack
$ spack --help
usage: spack [-hkV] [--color {always,never,auto}] COMMAND ...

A flexible package manager that supports multiple versions,
configurations, platforms, and compilers.
[…]
$ module avail
---- $WORK/USER-SPACK/share/spack/modules/linux-almalinux8-x86_64 ----
autoconf/2.69-oneapi2022.1.0-ogngd4o        libevent/2.1.12-oneapi2022.1.0-urjh4xz
automake/1.16.5-oneapi2022.1.0-v63ckpi      libiconv/1.16-oneapi2022.1.0-caghp63
nettle/3.4.1-oneapi2022.1.0-7epkve6
```

- Modules `000-all-spack-pkgs/<version>` show all pre-build packages
- There might be different SPACK versions per cluster

SPACK - Compilers

- List available compilers:

```
$ spack compiler list
==> Available compilers
-- dpcpp almalinux8-x86_64 -------------------------------------
dpcpp@2023.0.0

-- gcc almalinux8-x86_64 ----------------------------------------
gcc@11.2.0  gcc@8.5.0

-- intel almalinux8-x86_64 --------------------------------------
intel@2021.8.0  intel@2021.7.0

-- oneapi almalinux8-x86_64 -------------------------------------
oneapi@2023.0.0
```

- Use unknown compiler:

```
$ module load gcc/12.1.0
$ spack compiler find
==> Added 1 new compiler to
$HOME/.spack/linux/compilers.yaml
    gcc@12.1.0
```

SPACK – Package information

- Get available versions:

```
$ spack versions hwloc
==> Safe versions (already checksummed):
  master  2.7.1  2.6.0  2.4.1  2.3.0  2.1.0  2.0.3  2.0.1  1.11.13  1.11.11 […]
==> Remote versions (not yet checksummed):
==> Warning: Found no unchecksummed versions for hwloc
```

- More info (description, versions, variants, dependencies, …):

```
$ spack info hwloc
AutotoolsPackage:   hwloc

Description:
    The Hardware Locality (hwloc) software project. […]

Homepage: https://www.open-mpi.org/projects/hwloc/

Preferred version:
    2.8.0      https://download.open-mpi.org/release/hwloc/v2.8/hwloc-2.8.0.tar.gz
```

SPACK – Package selection syntax

- Check resolved dependencies (with default settings):

```
$ spack spec hwloc                  # or spack spec hwloc@2.7.1
Input spec
--------------------------------
hwloc

Concretized
--------------------------------
hwloc@2.7.1%gcc@11.2.0~cairo~cuda~gl~libudev+libxml2~netloc~nvml~opencl+pci~rocm+shared
arch=linux-almalinux8-icelake
    ^libpciaccess@0.16%gcc@11.2.0 arch=linux-almalinux8-icelake
        ^libtool@2.4.7%gcc@11.2.0 arch=linux-almalinux8-icelake
```

Unclear why 2.7.1 but preferred is 2.8.0

SPACK – Package selection syntax

- Check resolved dependencies (with own compiler):

```
$ spack spec hwloc@2.7.1 % gcc@12.1.0
Input spec
--------------------------------
hwloc@2.7.1%gcc@12.1.0

Concretized
--------------------------------
hwloc@2.7.1%gcc@12.1.0~cairo~cuda~gl~libudev+libxml2~netloc~nvml~oneapi-level-
zero~opencl+pci~rocm build_system=autotools libs=shared,static arch=linux-almalinux8-
icelake
    ^libpciaccess@0.16%gcc@12.1.0 build_system=autotools arch=linux-almalinux8-icelake
        ^libtool@2.4.7%gcc@12.1.0 build_system=autotools arch=linux-almalinux8-icelake
```

- Add cuda variant:

```
$ spack spec hwloc@2.7.1 % gcc@11.2.0 +cuda    # gcc@12.1.0 incompatible with cuda
hwloc@2.7.1%gcc@11.2.0~cairo+cuda~gl~libudev+libxml2~netloc~nvml~oneapi-level-
zero~opencl+pci~rocm
   ^cuda@11.8.0%gcc@11.2.0~allow-unsupported-compilers~dev
```

SPACK – Package selection syntax

- SPACK allows software installation including dependencies for users
  - … in various version …
  - … with specific compiler …
  - … for a specific micro-architecture …
  - ... with various build variants …
  - … integrated in the `module` system of NHR@FAU

- The `user-spack` module is limited
  - No environments (`spack load/unload …`)
- Why not simply `git clone …/spack`?
  - Does not use cluster-specific configuration
  - Cannot re-use pre-build packages