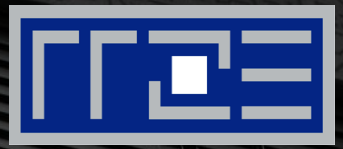
REGIONALES RECHENZENTRUM ERLANGEN [RRZE]

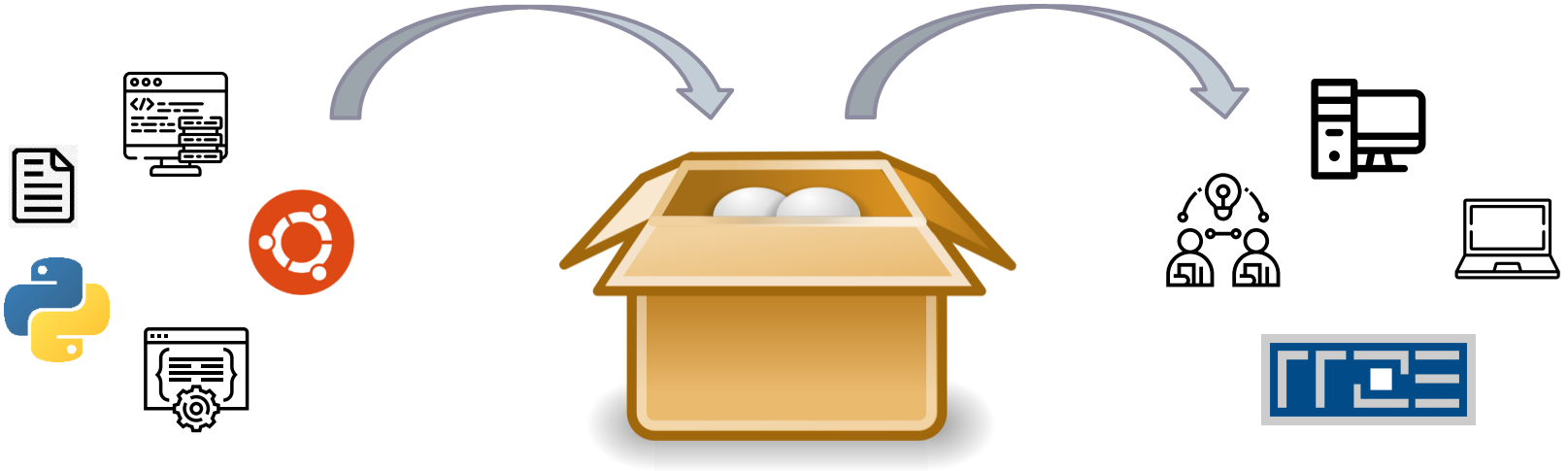


Introduction to Software Containers with Singularity

HPC Services, RRZE

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.



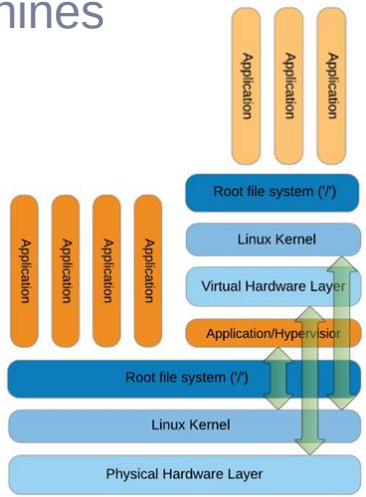
What can I do with Containers?

- Portability
  - Single container file, easy to transport and share
  - Run a pre-built application from Singularity Hub or Docker Hub without installing anything
- Bring your own Environment
  - Run application built for a different Linux distribution on host OS
  - Run commercially supported code requiring particular environment (either in container or outside!)
  - Use static environment (→ fund once, never update software development model)
  - Run legacy code on old operating systems
- Reproducible science
  - Entire application can be contained and archived/distributed for others to replicate
  - Reproduce environment of workflow created by someone else
- Package complicated software stacks
  - Easily verifiable via checksum or signature for version control
  - Use for analysis pipelines to run on different platforms and produce the same result everywhere

Containers vs. Virtual Machines

Virtual Machines

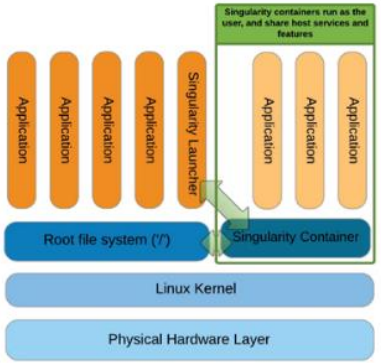
- Install every component of OS, including kernel
- Flexible (e.g. Windows VM on Mac)
- Quite large and resource hungry



Mainly used for long running interactive sessions with many different applications; not suitable for HPC

Containers

- Share kernel with the host OS
- Less flexible (Linux container must run on Linux host OS)
- More lightweight and faster, less overhead



Best suited for running only one or two applications, non-interactively; more suitable for HPC applications

Container Frameworks

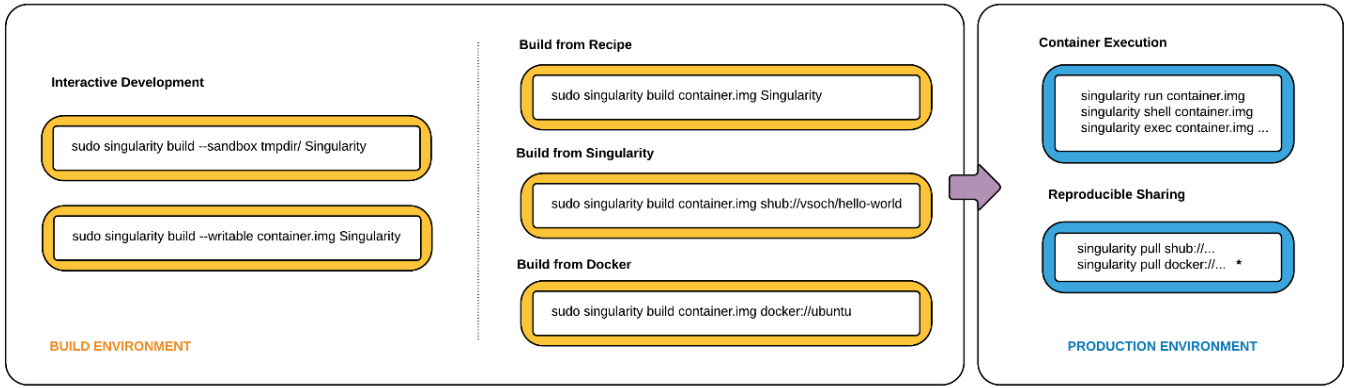
- Most popular container framework
- Built for running multiple containers on a single system
- Fully isolate each container from others and from host system
- Not suitable for HPC platforms, also due to security concerns
- Many pre-built containers available, e.g. on Docker Hub

Best suited for DevOPs teams providing cloud-native micro-services to users

- Specifically designed for HPC
- Each application (more or less) has its own container
- No full isolation from other containers or host system
- No root access on production system necessary
- Can convert Docker containers to Singularity and run containers directly from Docker Hub

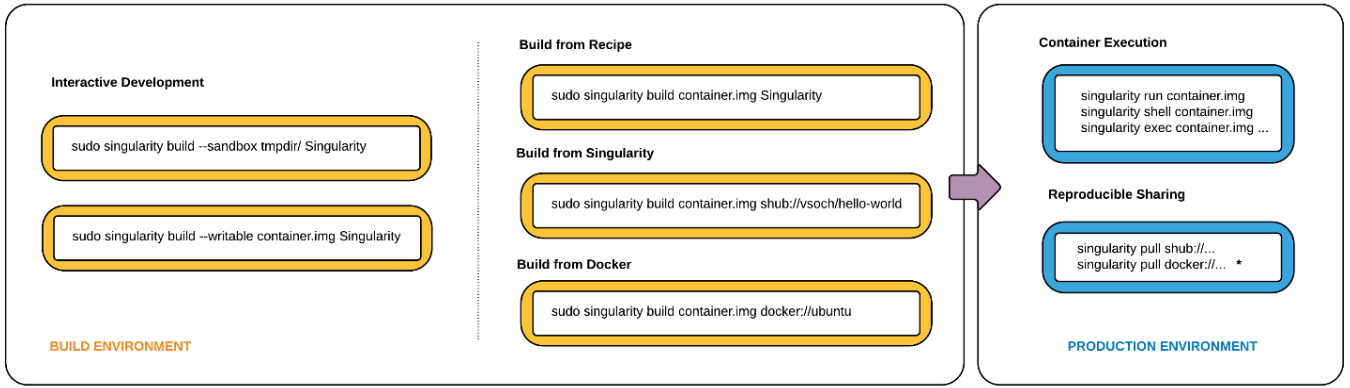
Best suited for running scientific software in an HPC environment

Basic usage



- Use pre-built containers or build them yourself.
- Building containers from scratch (interactively or via definition file) requires root access, so build them on your local machine.
- Run/shell/import of a (pre-built) container in a production environment is possible as a normal user.
- Generally: you are the same user inside the container than outside!
- Container images are build immutable to preserve reproducibility.

Basic usage



- No performance penalties from using containers.
- External file systems are automatically mounted in container (at RRZE: /apps, /home, /lxfs) as well as /dev for GPUs and Infiniband/Omni-Path network.
- Supports GPU-dependent applications within containers.
- Containers can be run through any job scheduler (Slurm, Torque, ...)

  → job script just calls `singularity run/exec`
- Possible to run MPI applications, but a bit more involved to setup.

Basic usage

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

Available Commands:
  build       Build a Singularity image
  cache       Manage the local cache
  exec        Run a command within a container
  inspect     Show metadata for an image
  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
  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 Singularity
  […]

Run 'singularity --help' for more detailed usage information.
```

Using existing containers – Sources

- Singularity can convert and run containers in many different formats, including Docker containers
- Some popular places for pre-built containers:
  - The Singularity Container Library, developed and maintained by Syslabs
  - Docker Hub, developed and maintained by Docker
  - Quay.io, developed and maintained by RedHat
  - NGC, developed and maintained by Nvidia
  - … many more

Security disclaimer:
As with all software, only download and execute if it comes from a trusted source! Don't build containers from untrusted sources or run them as root!

Using existing containers – Security

Before you run an unfamiliar/pre-built container:

- Review runscript: `singularity inspect --runscript <container_name>`
- Use the `--no-home` and/or `--containall` options

For building a container from recipe or from scratch:

- On systems with recent kernels (>= Ubuntu 18.04), use the `--fakeroot` feature
- Build inside a Linux VM

Dockerhub official /certified images

- Official images: reviewed, scanned for vulnerabilities
- Certified images: baseline testing, best practice guidelines

Signed/verified Singularity images:

- No official review, but proof that image was not tampered with.
- Anyone can sign an image, trust of maintainer required!

Using existing containers – Example

Live demo

- Download/pull container, e.g.:

```
singularity pull docker://ubuntu
```

- Enter containers with shell:

```
singularity shell <container_name>
```

- Execute containerized commands with exec

```
singularity exec <container_name> <command>
```

- Running container with run

```
singularity run <container_name>
```

or

```
./<container name>
```

→ Use run command in batch script on clusters!

Build your own container

- Requires Linux-based operating system with root/fakeroot access, so use your local machine or a VM
- Install Singularity and Go first (https://sylabs.io/guides/3.5/admin-guide/installation.html)

Possible example for an interactive workflow:

1. Create a writable container (called a sandbox)
2. Shell into the container with the `--writable` option and tinker with it interactively
3. Record changes in your definition file
4. Rebuild the container from the definition file if something goes wrong
5. Iteratively change your container until you are happy with the results
6. Rebuild the container from the final definition file as a read-only singularity image format (SIF) image for use in production

Build your own container - Example

Live demo

- Create sandbox:

```
singularity build --fakeroot --sandbox <sandbox_name> docker://ubuntu
```

- Enter (writable) container with shell:

```
singularity shell --fakeroot --writable <sandbox_name>
```

- Build container image from definition file:

```
singularity build --fakeroot <container_name>.sif <definition_file>
```

- Convert sandbox to image and back again:

```
singularity build <container_name>.sif <sandbox_name>
singularity build --sandbox <sandbox_name> <container_name>.sif
```

Build your own container – Example

Live demo

```
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 (Singularity container library, debootstrap, yum,…)

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 `singularity run` or directly)

More details on definition files: https://sylabs.io/guides/3.5/user-guide/definition_files.html

Build your own container – for GPUs

Live demo

Singularity natively supports running GPU-enabled applications inside a container.
Commands like run/shell/execute can take a `--nv` option, which will setup the container's environment to use an NVIDIA GPU and the basic CUDA libraries, e.g.

```
singularity run --nv <container_name>
```

Requirements:

- Host has working installation of GPU driver and CUDA libraries (@RRZE: TinyGPU, GPU nodes in emmy)
- CUDA version of application inside container must be compatible with host installation

Special on TinyGPU: GPU device libraries are automatically bind-mounted into container, just execute your container via `singularity run <container_name>`

Application cases at RRZE

- Tensorflow for AI applications (https://www.anleitungen.rrze.fau.de/hpc/special-applications-and-tips-tricks/tensorflow/)
- OpenPose: container for portability between different HPC facilities; includes GPU and CPU options, and also different GPU generations (http://peter-uhrig.de/openpose-with-nvcaffe-in-a-singularity-container-with-support-for-multiple-architectures/)
- Container with different OS: used to fix software bug which only occurs for specific configuration → much faster to setup than VM
- … and hopefully more soon!

Are you using containers for your application?
Or are you planning to?
Tell us your reasons, experiences and thoughts about the topic!

Further resources

- Official Singularity User Guide (https://sylabs.io/guides/3.5/user-guide/)
- Tutorial with examples from NIH HPC group: https://github.com/NIH-HPC/Singularity-Tutorial
- Example definition files: Singularity source code, examples subdirectory or https://sylabs.io/guides/3.5/user-guide/definition_files.html
- Tensorflow for AI applications (https://www.anleitungen.rrze.fau.de/hpc/special-applications-and-tips-tricks/tensorflow/)
- MPI application: https://sylabs.io/guides/3.5/user-guide/mpi.html

- .. and many more, just use your search engine of choice!