TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

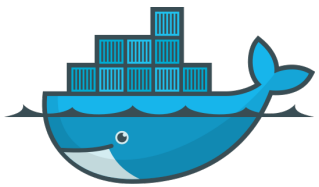
Fakultät Mathematik, Institut für Wissenschaftliches Rechnen

SCIENTIFIC COMPUTING IN CONTAINERS

Dr. Simon Praetorius

Container Meetup – Dresden, 26. Nov. 2019

Scientific Computing in Containers



Overview:

1. Scientific Computing
2. High-Performance Computing
3. Reproducibility

Toolset:

- Docker (`docker.io`)
- Singularity (`sylabs.io`)
- Gitlab (`gitlab.com`)

Containers for Scientific Computing

"Generally container technologies have been designed to solve a single primary use case for the enterprise: **micro-service virtualization**"

Containers for Scientific Computing

"Generally container technologies have been designed to solve a single primary use case for the enterprise: **micro-service virtualization**"

This is **NOT** the Scientific Computing or High-Performance Computing use case!

Containers for Scientific Computing

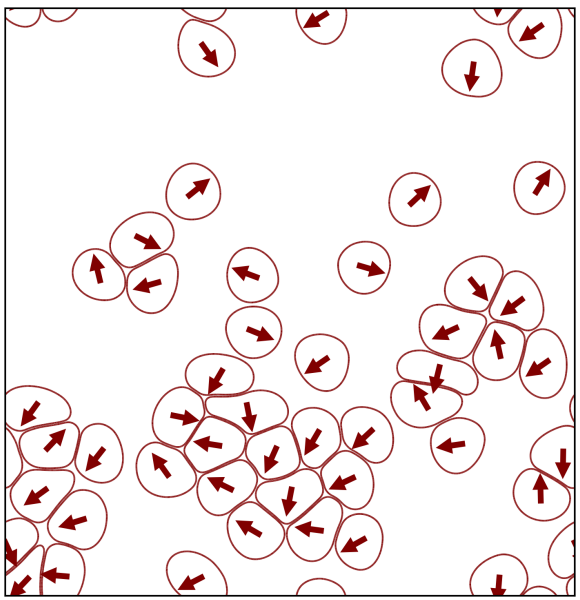
"Generally container technologies have been designed to solve a single primary use case for the enterprise: **micro-service virtualization**"

This is **NOT** the Scientific Computing or High-Performance Computing use case!

So... what is our use case?

Example: Collective Behavior of Deformable Cells

Modeling and Simulation:



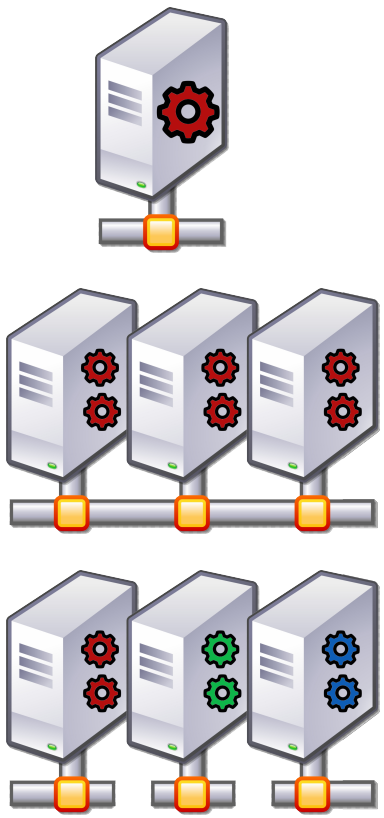
- Each cell described by individual physical model
- Interaction and deformation driven by contact and motion
- Self-propelled cells
- Simulation of each cell on single core
- Communication between cores to implement interaction of cells

Marth, Voigt, *Interface Focus*, (2016)
Praetorius, Voigt, *NIC Symposium* (2018)
Wenzel, Praetorius, Voigt, *J. Chem. Phys.*, (2019)

Scientific Computing

Thee Scenarios in our institute:

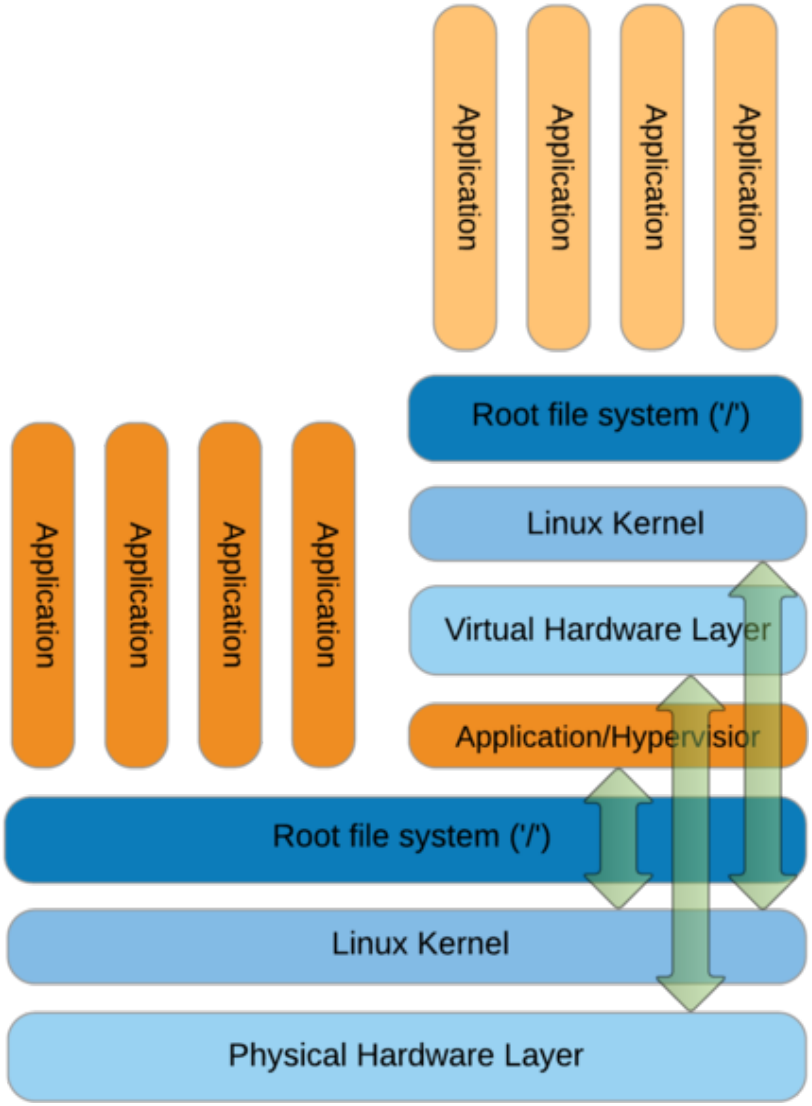
1. Small, single core computation:
   - Convergence studies
   - Verification of numerical algorithms
   - Development and test of the model
   - Study of 1-2 cells
2. Large scale highly parallel jobs:
   - 100/1,000/10,000 cores for 1 simulation setup
   - Days – weeks of runtime
   - Up to TB of data produced
   - 1000 cells collectively
3. Parameter studies:
   - Many small jobs (with just a few cores)
   - Variation of parameter landscape
   - Postprocessing of parameter dependency
   - Determine model parameters, e.g. 48 cells



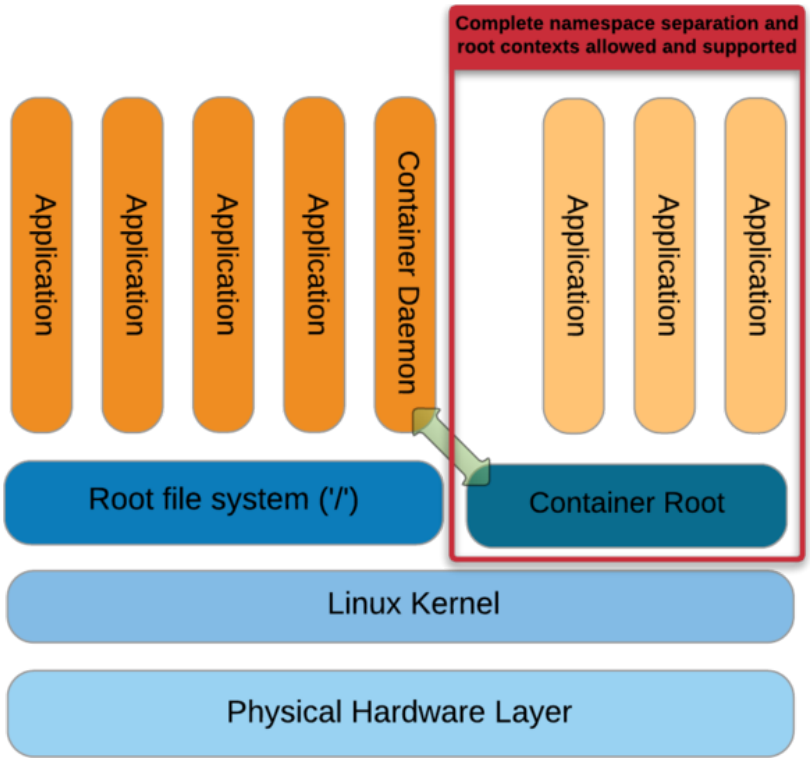
High-Performance Computing

- Performance! Do we get near-native application performance?
- Can we use optimized libraries of HPC environment, e.g. Customized OpenMPI.
- Is containerization allowed on the HPC cluster at all? Security concerns...
- Workspaces for Data-management
- Combination of (different) Compute-Nodes
- Classical Dependency-Management by Environmental Modules

Full Virtualization vs. Containers



Virtual Machine



General Container

High-Performance Computing

Limitations of VM and general Containers

- VM-based applications have indirect access to the host kernel and hardware via the guest OS and hypervisor → significant performance overhead
- Trusted users running trusted containers
- Not designed to support batch-based workflows (e.g. SLURM)
- Not designed to support tightly-coupled, highly distributed parallel applications (MPI).

High-Performance Computing

Solution: Alternative Container Frameworks

Shifter, Singularity, udocker, Charliecloud, ...

Singularity

2015 Developed at Lawrence Berkeley National Laboratory (Gregory Kurtzer)

April 2016 First release

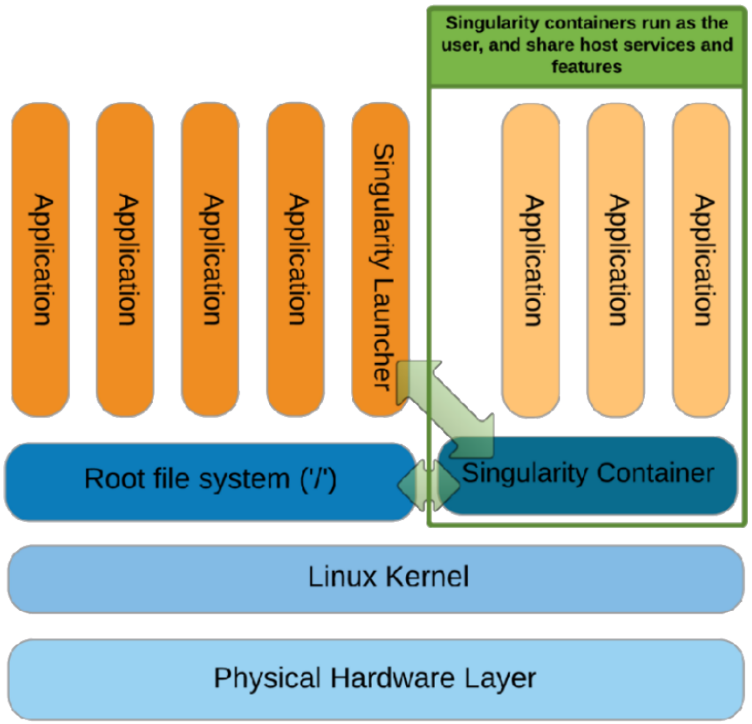
2016,2017 Several awards in the HPC community, e.g. “Best HPC Programming Tool or Technology” (2017)

2018 Software maintained and developed by sylabs.io

Big Users
- TACC (Texas)
- Oak Ridge Titan (Tennessee)
- GSI Helmholtz Center Heavy Ion (Darmstadt)
- Open Science Grid + Cern
- Taurus @ TUD

Features of Singularity

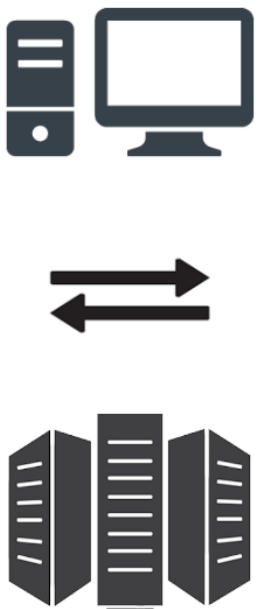
- Each container is a single image file
- No root owned daemon processes
- No user contextual changes or root escalation allowed; user inside container is always the same user who started the container
- Supports HPC hardware: Infiniband, GPUs, Intel Omni-Path Architecture.
- Supports HPC applications: MPI, ressource managers (e.g. SLURM)



Singularity Container

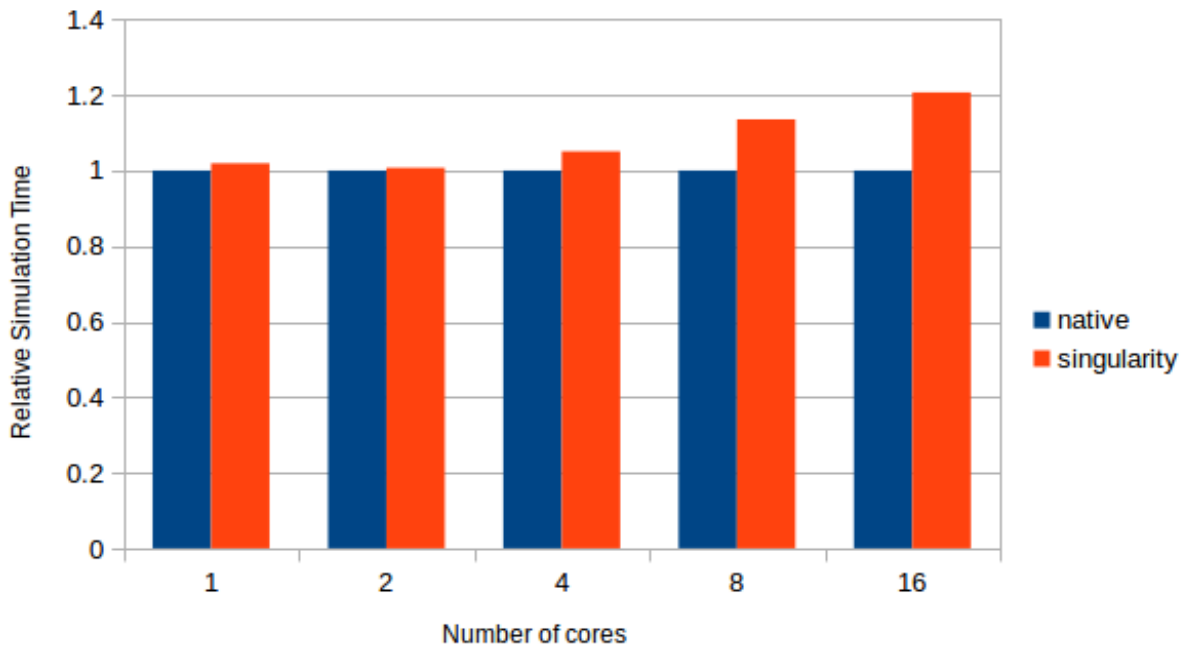
Singularity Workflow

1. **Build** your Singularity containers on a local system where you have root or sudo access
   - Build from definition file (recipe), similar to Dockerfile
   - Convert an existing docker container
   - Remote builder
2. **Transfer** your Singularity containers to the HPC system where you want to run them
   - Copy image file to cluster
   - Push to and pull from *SingularityHub* or another registry
3. **Run** your Singularity containers on that HPC system
   - With simple user rights.



Singularity Performance

Benchmark: Comparison of native code to singularity:



(Very preliminary results)

Singularity Definition File vs. Dockerfile

```
Bootstrap: docker
From: debian:buster

%labels
  Author simon.praetorius@tu-dresden.de
  Version v0.1

%post     # Step 1: prepare the debian environment
  apt-get update -y
  apt-get install -y --no-install-recommends \
    build-essential ca-certificates cmake \
    gcc-9 g++-9 gfortran-9 git git-lfs pkg-config \
    mpi-default-bin mpi-default-dev petsc-dev
    # ... and more debian packages
  apt-get clean
  rm -rf /var/lib/apt/lists/*

%post     # Step 2: Install Dune dependencies
  mkdir /dune && cd /dune
  git clone https://[...]/core/dune-common.git
  git clone https://[...]/core/dune-geometry.git
  # ... download more dune modules
  ./dune-common/bin/dunecontrol --builddir=build all

%environment
  export DUNE_CONTROL_PATH=/dune
  export PATH=/dune/dune-common/bin:$PATH

%runscript
  dunecontrol --builddir=build $@
```

Singularity Definition File vs. Dockerfile

```
Bootstrap: docker
From: debian:buster

%labels
  Author simon.praetorius@tu-dresden.de
  Version v0.1

%post     # Step 1: prepare the debian environment
  apt-get update -y
  apt-get install -y --no-install-recommends \
    build-essential ca-certificates cmake \
    gcc-9 g++-9 gfortran-9 git git-lfs pkg-config \
    mpi-default-bin mpi-default-dev petsc-dev
    # ... and more debian packages
  apt-get clean
  rm -rf /var/lib/apt/lists/*

%post     # Step 2: Install Dune dependencies
  mkdir /dune && cd /dune
  git clone https://[...]/core/dune-common.git
  git clone https://[...]/core/dune-geometry.git
  # ... download more dune modules
  ./dune-common/bin/dunecontrol --builddir=build all

%environment
  export DUNE_CONTROL_PATH=/dune
  export PATH=/dune/dune-common/bin:$PATH

%runscript
  dunecontrol --builddir=build $@
```

```
FROM debian:buster


LABEL MAINTAINER="simon.praetorius@tu-dresden.de"
LABEL VERSION="v0.1"

# Step 1: prepare the debian environment
RUN apt-get update -y \
 && apt-get install -y --no-install-recommends \
      build-essential ca-certificates cmake \
      gcc-9 g++-9 gfortran-9 git git-lfs pkg-config \
      mpi-default-bin mpi-default-dev petsc-dev \
      # ... and more debian packages
 && apt-get clean \
 && rm -rf /var/lib/apt/lists/*

# Step 2: Install Dune dependencies
RUN mkdir /dune && cd /dune \
 && git clone https://[...]/core/dune-common.git \
 && git clone https://[...]/core/dune-geometry.git \
    # ... download more dune modules
 && ./dune-common/bin/dunecontrol --builddir=build all

ENV DUNE_CONTROL_PATH=/dune
ENV PATH=/dune/dune-common/bin:$PATH

ENTRYPOINT ["dunecontrol", "--builddir=build"]
CMD ["--current", "all"]
```

Build and Run the Singularity Container

Build the Container

```
# build the singularity container in the .sif format locally
sudo singularity build container.sif definition.def

# remote build
singularity remote login --tokenfile [TOKENFILE]
singularity build --remote container.sif definition.def
```

Run the Container

```
# run the dunecontrol script to compile a dune application
singularity run container.sif --current all

# or use container as executable
container.sif --current all

# Several executables can be provided as app in the container
singularity run --app dunecontrol container.sif --current all

# Direct execution of commands
singularity exec container.sif dunecontrol --opts=/dune/dune.opts --current all

# Shell environment
singularity shell container.sif
```

Singularity on HPC

Runscript `run.sh`

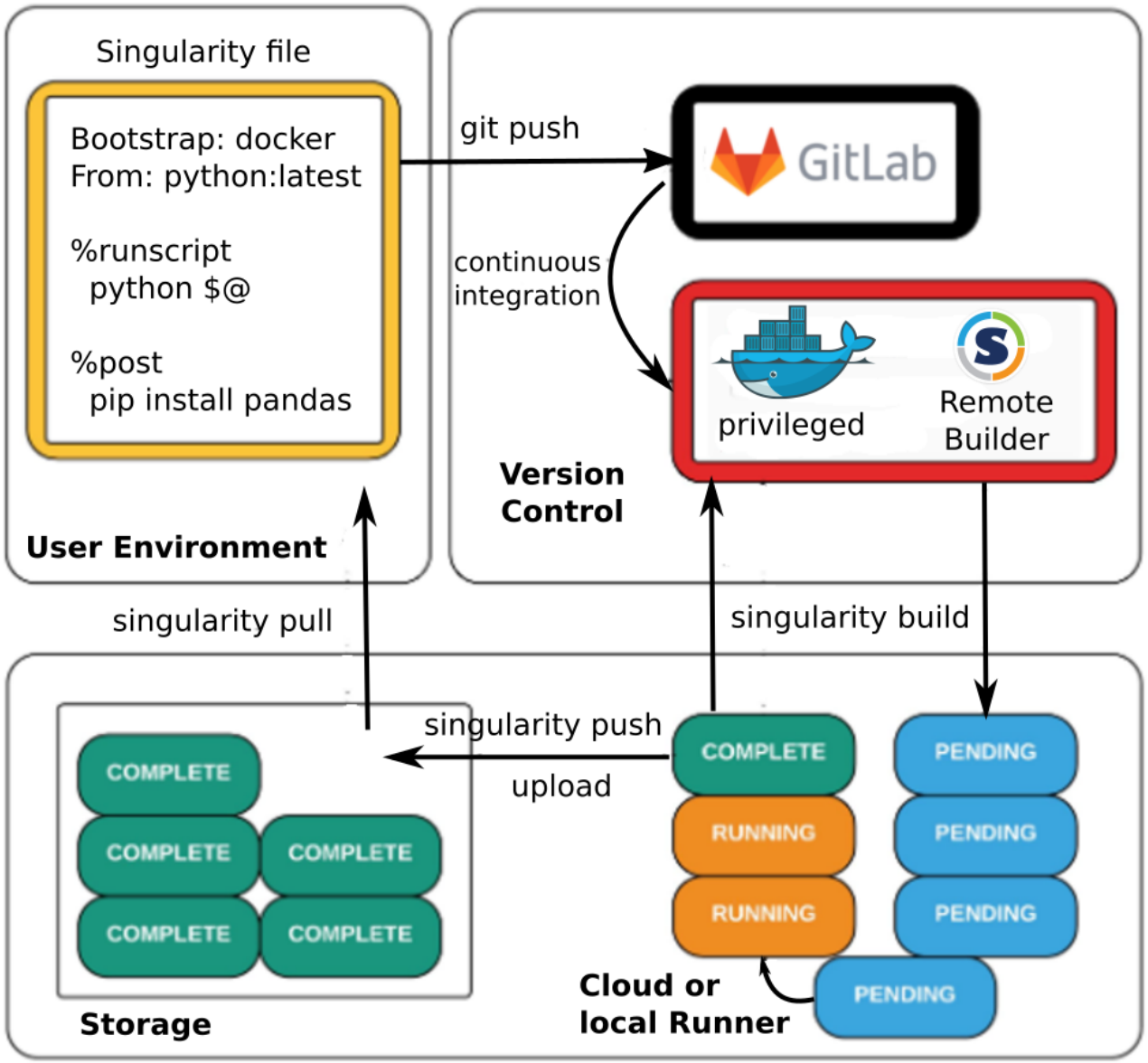
```
#!/bin/bash
#SBATCH --time=01:00:00
#SBATCH --output=log-%j.out
#SBATCH --error=log-%j.err

MPI_DIR=/sw/installed/OpenMPI/3.1.3-GCC-8.2.0-2.31.1
export PATH=$MPI_DIR/bin:$PATH
export LD_LIBRARY_PATH=$MPI_DIR/lib:$LD_LIBRARY_PATH

srun singularity run --bind [OUTPUT_DIR] container.sif app [...]
```

```
sbatch -N 1 -n 24 run.sh
```

Singularity automatic build workflow



Sochat V., Prybol C, Kurtzer GM. Singularity Hub: Registry for Reproducible Containers (manuscript in progress)

Build Singularity Container inside Docker

Dockerfile

```
FROM golang:1.13-alpine3.10 as builder

RUN apk update
RUN apk add --no-cache [... build dependencies]

ENV SINGULARITY_VERSION=3.5.0
LABEL Maintainer simon.praetorius@tu-dresden.de
RUN # ...Download and build singularity
RUN mconfig -p /usr/local/singularity && \
 && make -C builddir && make -C builddir install

# Multi-stage build
FROM alpine:3.10
LABEL Maintainer simon.praetorius@tu-dresden.de
COPY --from=builder /usr/local/singularity /usr/local/singularity
RUN apk add --no-cache [... run dependencies]
ENV PATH=/usr/local/singularity/bin:$PATH
```

```
docker build -t [HUB]/singularity:3.5 dir/
docker push [HUB]/singularity:3.5
```

Build Singularity inside Docker

Gitlab CI Script:

```
image: [HUB]/singularity:3.5

before_script:
  - singularity remote login --tokenfile ${SINGULARITY_TOKEN}

build:
  script:
    - singularity build [--remote] container.sif definition.def
    - singularity push -U container.sif library://[NAMESPACE]/container:latest
```

Gitlab Runner configuration:

```
[[runnser]]
  name = "singularity-builder"
  url = "https://gitlab.[HOST].de"
  token = "XXXXXXXXXX"
  executor = "docker"
  [runners.docker]
    privileged = true
    allowed_images = ["[HUB]/singularity:3.5"]
    ...
```

Reproducibility

Workflow allows to...

- Save history and tag version of Singularity definition files
- Combine with tagging of project source code
- Automatic build and upload of singularity images
- Images everywhere available in latest version

Difficulties:

- Fix versions of software inside container
  - Solution: build software stack inside container with EasyBuild
- Store code inside or outside of image?
  - Better: fix version in Git

Reproducibility of data and results:

- Same binary and library (sha1sum)
- Same simulation results on different host

Summary & Challenges

Summary

- Singularity, a container format for HPC
- Local builds, remote runs
- Parallelization with good performance

Challenges

- MPI is possible but not simple to install
  - Compatible version on host and in container!
- Performance overhead is small, but for large scale computations it is measurable
  - Needs more testing, maybe finer adjustment of libraries
- Root privileges required to build container
  - Needs privileged mode for docker container to build in CI, maybe fakeroot.
- Container image files are large
  - Need to separate build and execution containers.

References

- W. Marth, A. Voigt: *Collective migration under hydrodynamic interactions - a computational approach.* Interface Focus, (2016)
- D. Wenzel, S. Praetorius, A. Voigt: *Topological and geometrical quantities in active cellular structures.* J. Chem. Phys. (2019)
- S. Praetorius, A. Voigt: *Collective Cell Behavior - a Cell-Based ParalleliSation Approach for a Phase Field Active Polar Gel Model.* NIC Symposium, (2018)
- Greg Kurtzer: *keynote slides at HPC Advisory Council*, Stanford, (2017)
- Marty Kandes: *An Introduction to Singularity: Containers for Scientific and High-Performance Computing*, XSEDE Webinar, (2019)