August 22, 2023

Matthew Sgambati

Software Quality Assurance for High Performance Computing Containers utilizing MVAPICH2 and the MOOSE framework

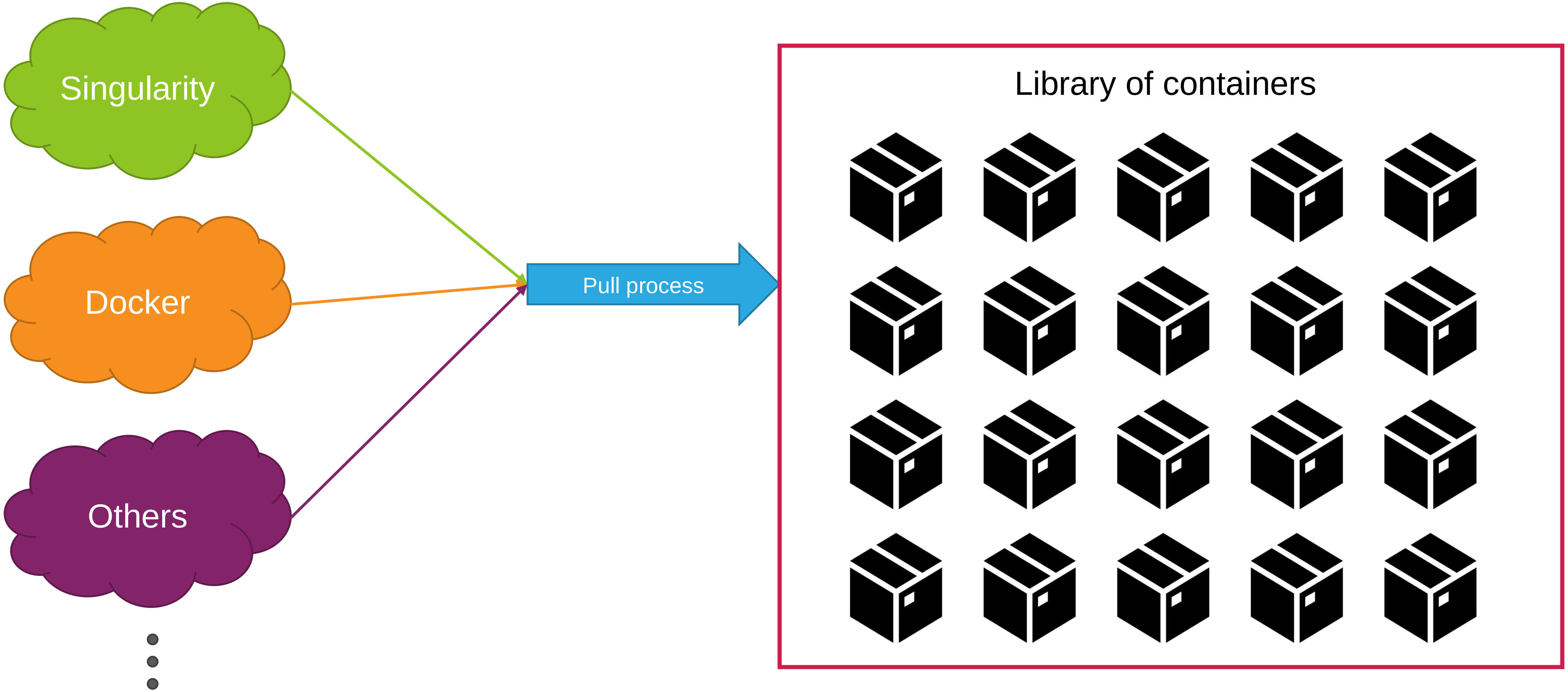
MUG 2023

INL/CON-23-74293

Why containers?

- Verification and Validation: Is my code running correctly still? Is it passing all unit tests and giving the same results as my last few papers?
- Reproducibility: Can I run my same application 10 years later? Can I rebuild my container 10 years later?
- Portability across multiple supercomputer architectures: Can I run the same container on multiple supercomputers with different networks and drivers *without rebuilding and without sacrificing performance*?
- Simplification: one environment, customized packages (independence from system administrators); make it easier on the user: Can I work without needing intervention from the system administrator?
- Security (insulation against package versioning issues): Does a security update on a supercomputer break my code?

Containers – Creation



Singularity

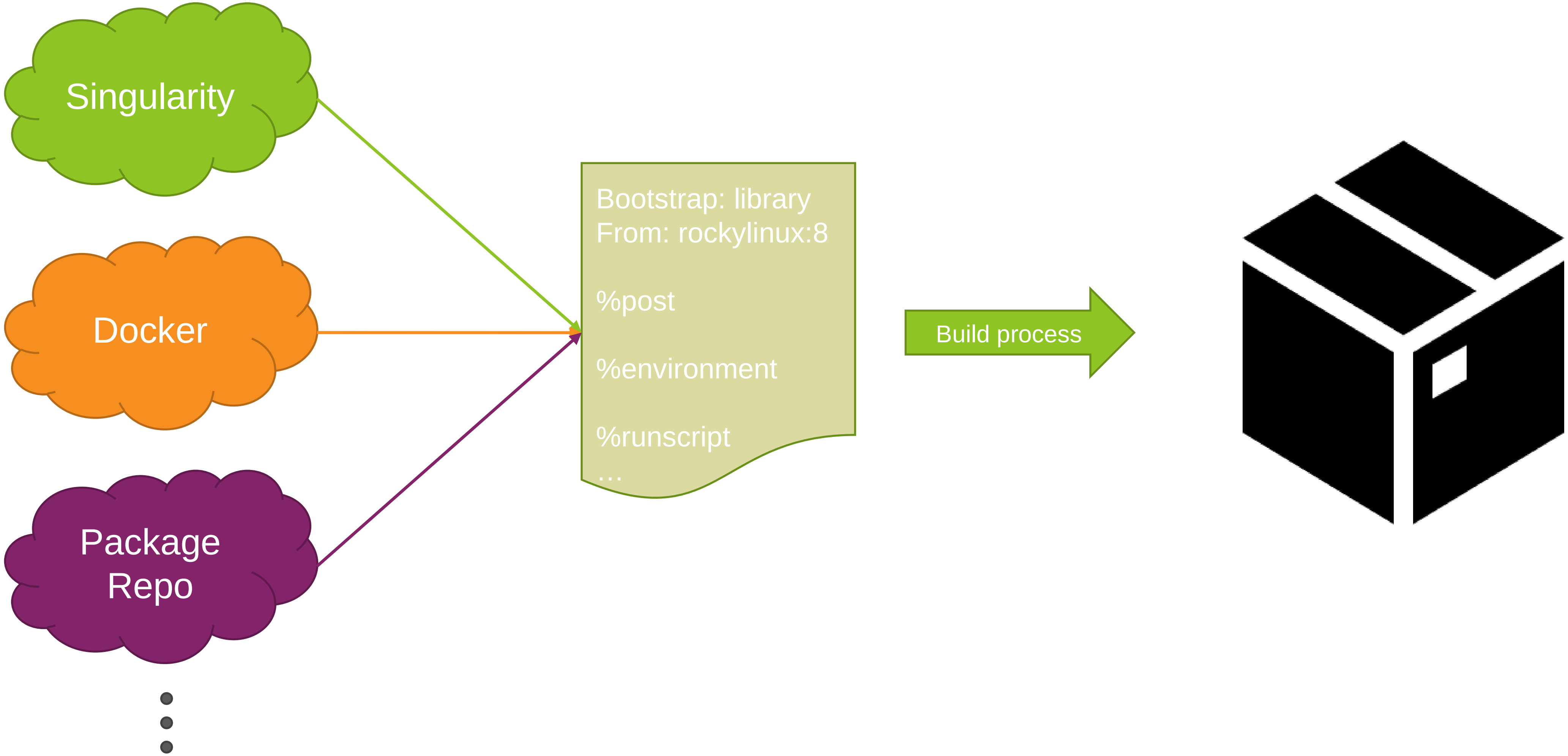
Docker

Others

Pull process

Library of containers

Create a container with def file



Singularity

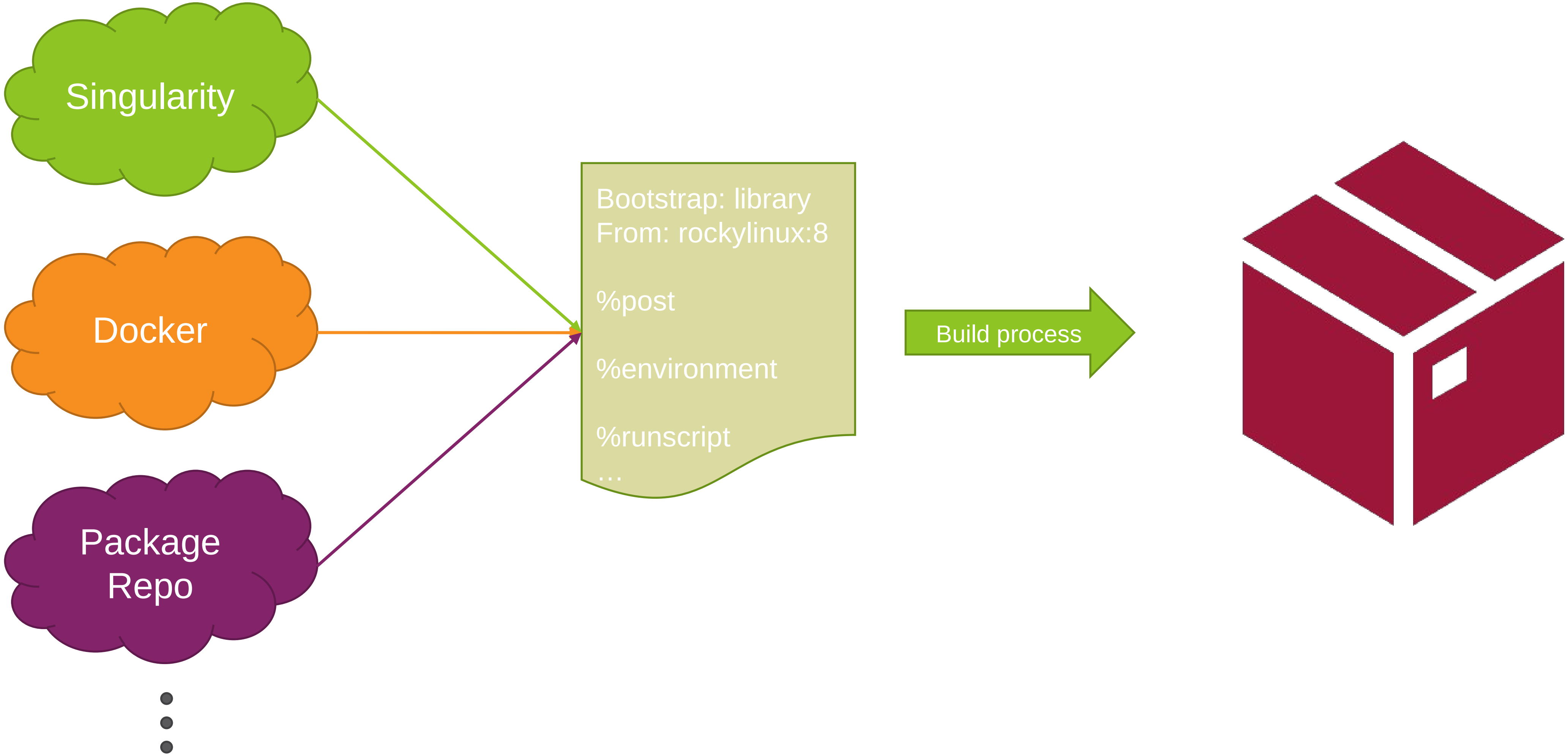
Docker

Package Repo

```
Bootstrap: library
From: rockylinux:8

%post

%environment

%runscript
…
```

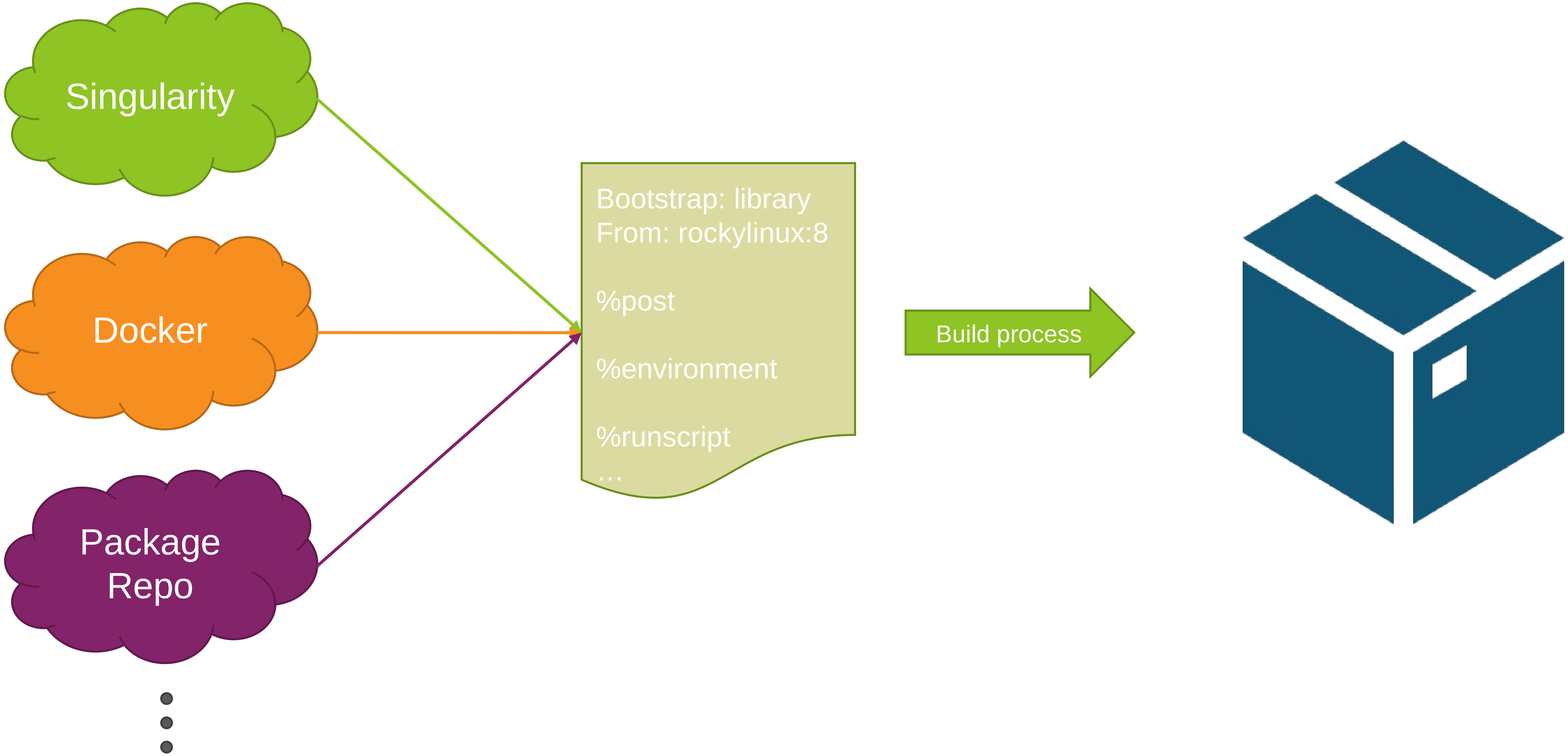
Build process

Create a container – Reproduceable?



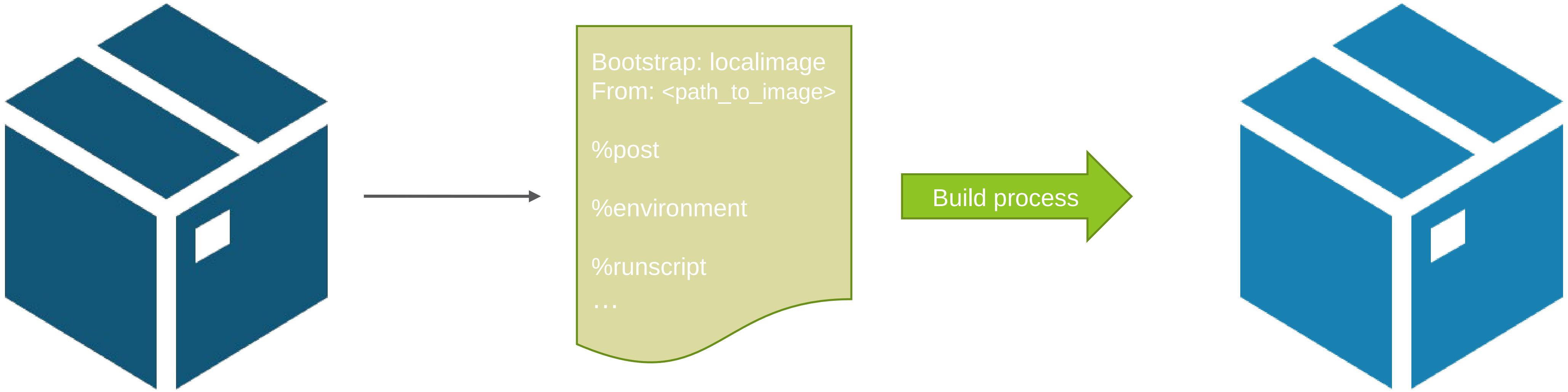
```
Bootstrap: library
From: rockylinux:8

%post

%environment

%runscript
…
```

Create a container – Reproduceable?
Layer system



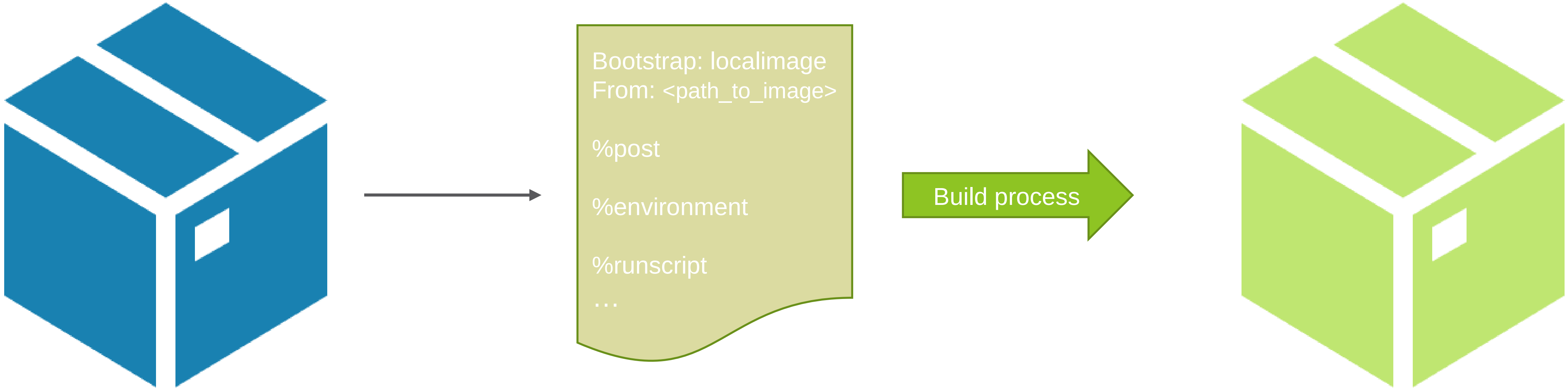
```
Bootstrap: library
From: rockylinux:8

%post

%environment

%runscript
…
```

Create a container – Reproduceable?
Layer system



```
Bootstrap: localimage
From: <path_to_image>

%post

%environment

%runscript
…
```

Create a container – Reproduceable?
Layer system



```
Bootstrap: localimage
From: <path_to_image>

%post

%environment

%runscript
…
```

Build process

Containers with MPI and IB – Two Approaches

- Hybrid
  - System MPI
  - Container MPI
  - System IB
  - Container IB

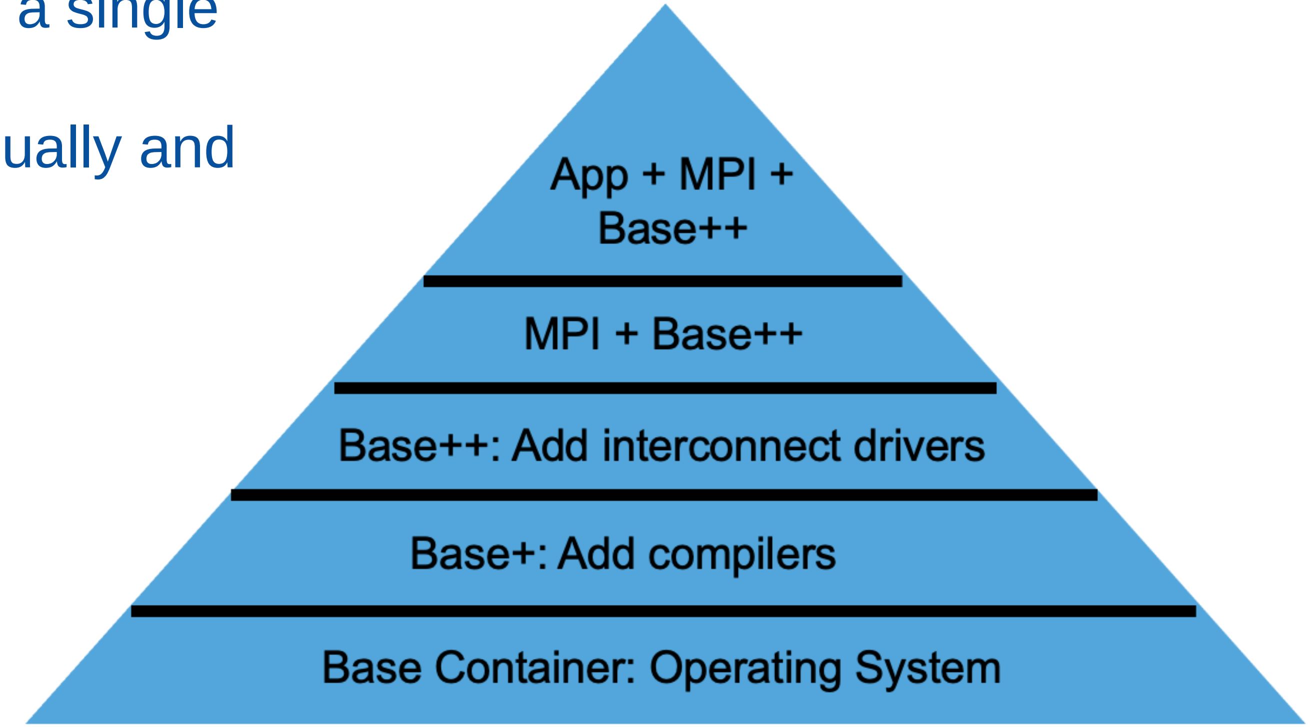
- Bind
  - System MPI
  - System IB

NOTE:

This requires the Host OS and Container OS to be compatible

Container Strategy

- There are three key components to the strategy:
  - HPC application containers built via layers
  - Key components are grouped into a single layer
  - Each layer can be updated individually and reproducibly via local mirrors



App + MPI + Base++

MPI + Base++

Base++: Add interconnect drivers

Base+: Add compilers

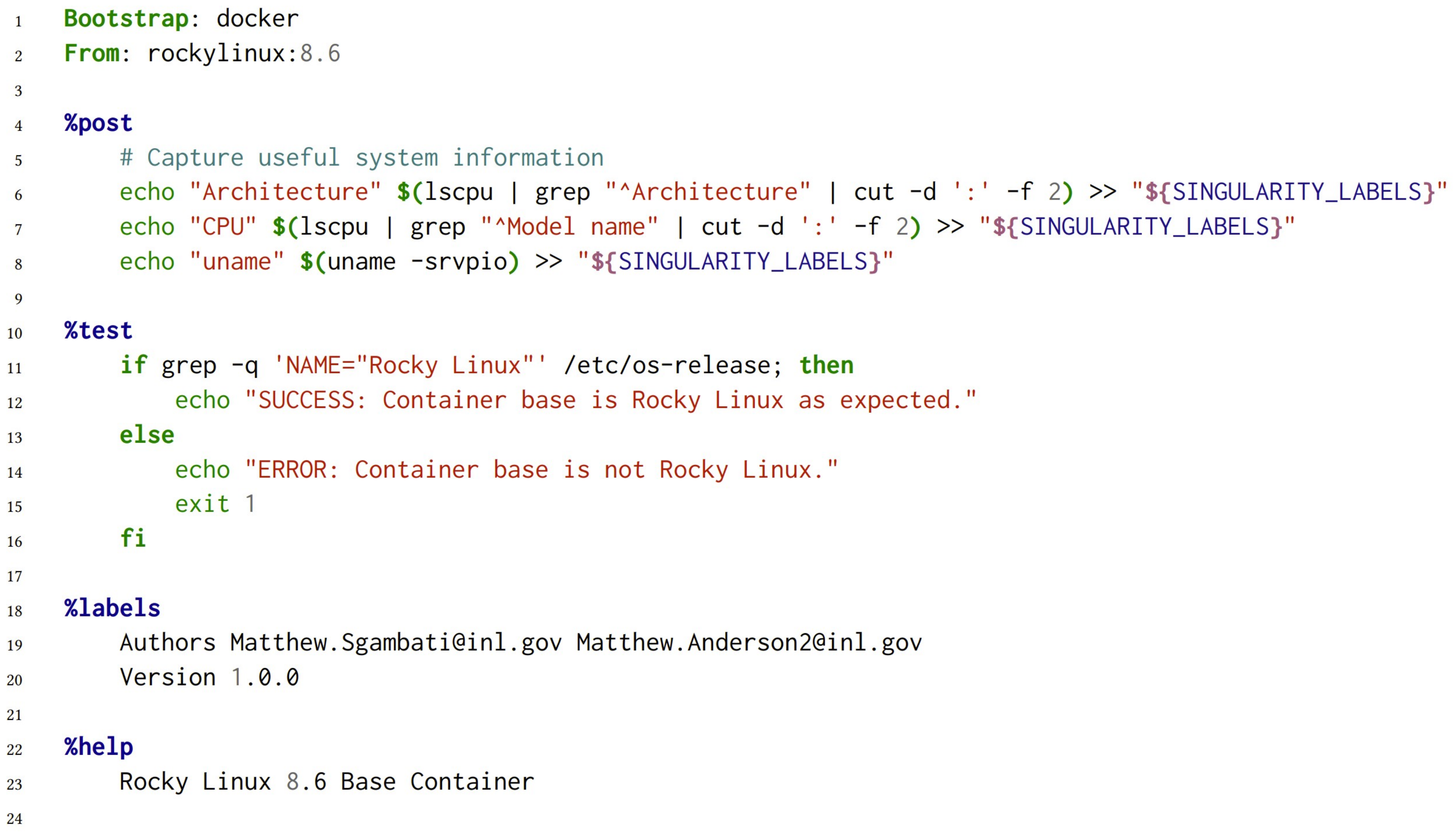
Base Container: Operating System

Container Strategy – Assumptions

- Host systems will use a long-term support (LTS) version of drivers and/or software stacks when possible
- Host system administrators will install an ABI-compatible version of MPI if one does not exist on the target system

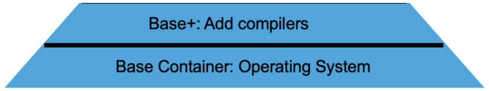
Base Container Definition File

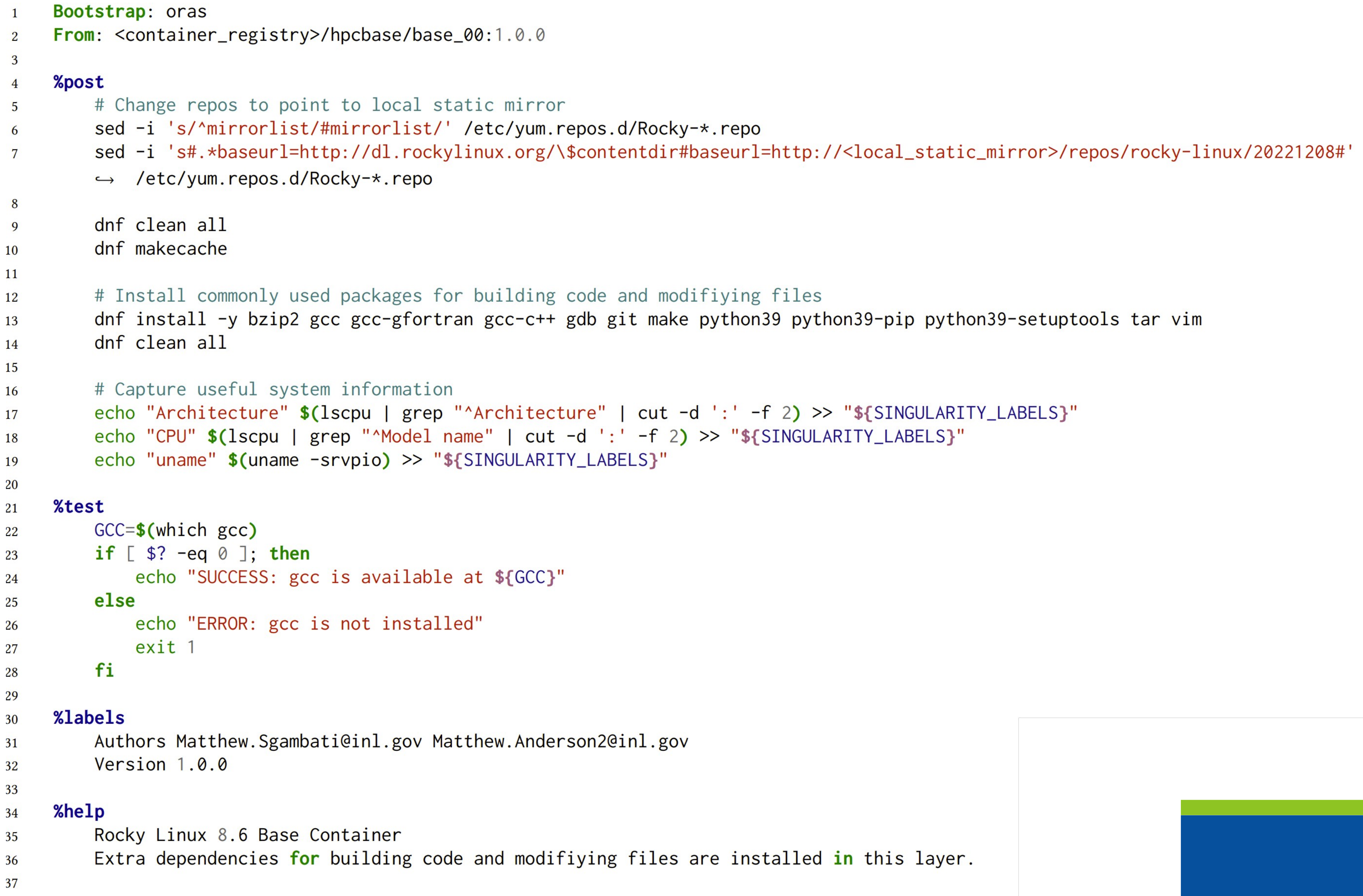
Base Container: Operating System



```
Bootstrap: docker
From: rockylinux:8.6

%post
    # Capture useful system information
    echo "Architecture" $(lscpu | grep "^Architecture" | cut -d ':' -f 2) >> "${SINGULARITY_LABELS}"
    echo "CPU" $(lscpu | grep "^Model name" | cut -d ':' -f 2) >> "${SINGULARITY_LABELS}"
    echo "uname" $(uname -srvpio) >> "${SINGULARITY_LABELS}"

%test
    if grep -q 'NAME="Rocky Linux"' /etc/os-release; then
        echo "SUCCESS: Container base is Rocky Linux as expected."
    else
        echo "ERROR: Container base is not Rocky Linux."
        exit 1
    fi

%labels
    Authors Matthew.Sgambati@inl.gov Matthew.Anderson2@inl.gov
    Version 1.0.0

%help
    Rocky Linux 8.6 Base Container

```

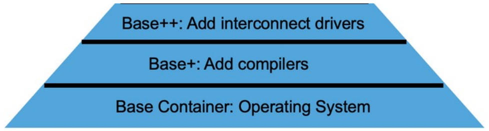
Base+ Container Definition File

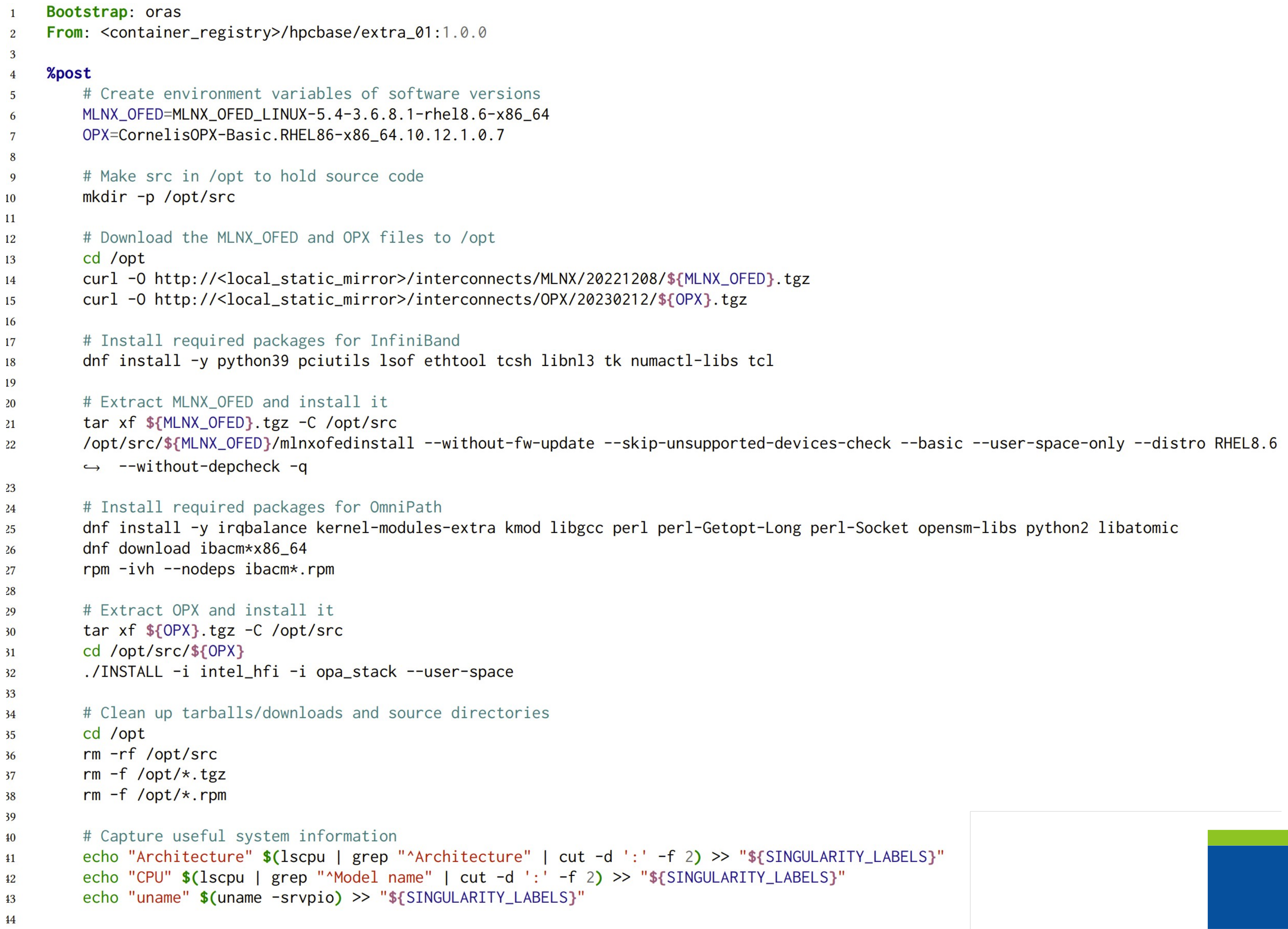




```
Bootstrap: oras
From: <container_registry>/hpcbase/base_00:1.0.0

%post
    # Change repos to point to local static mirror
    sed -i 's/^mirrorlist/#mirrorlist/' /etc/yum.repos.d/Rocky-*.repo
    sed -i 's#.*baseurl=http://dl.rockylinux.org/\$contentdir#baseurl=http://<local_static_mirror>/repos/rocky-linux/20221208#' /etc/yum.repos.d/Rocky-*.repo

    dnf clean all
    dnf makecache

    # Install commonly used packages for building code and modifiying files
    dnf install -y bzip2 gcc gcc-gfortran gcc-c++ gdb git make python39 python39-pip python39-setuptools tar vim
    dnf clean all

    # Capture useful system information
    echo "Architecture" $(lscpu | grep "^Architecture" | cut -d ':' -f 2) >> "${SINGULARITY_LABELS}"
    echo "CPU" $(lscpu | grep "^Model name" | cut -d ':' -f 2) >> "${SINGULARITY_LABELS}"
    echo "uname" $(uname -srvpio) >> "${SINGULARITY_LABELS}"

%test
    GCC=$(which gcc)
    if [ $? -eq 0 ]; then
        echo "SUCCESS: gcc is available at ${GCC}"
    else
        echo "ERROR: gcc is not installed"
        exit 1
    fi

%labels
    Authors Matthew.Sgambati@inl.gov Matthew.Anderson2@inl.gov
    Version 1.0.0

%help
    Rocky Linux 8.6 Base Container
    Extra dependencies for building code and modifiying files are installed in this layer.
```

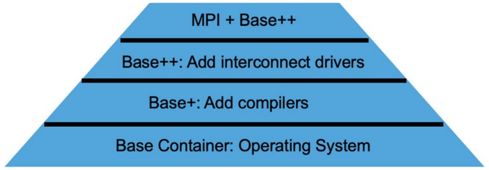
Base++ Container Definition File

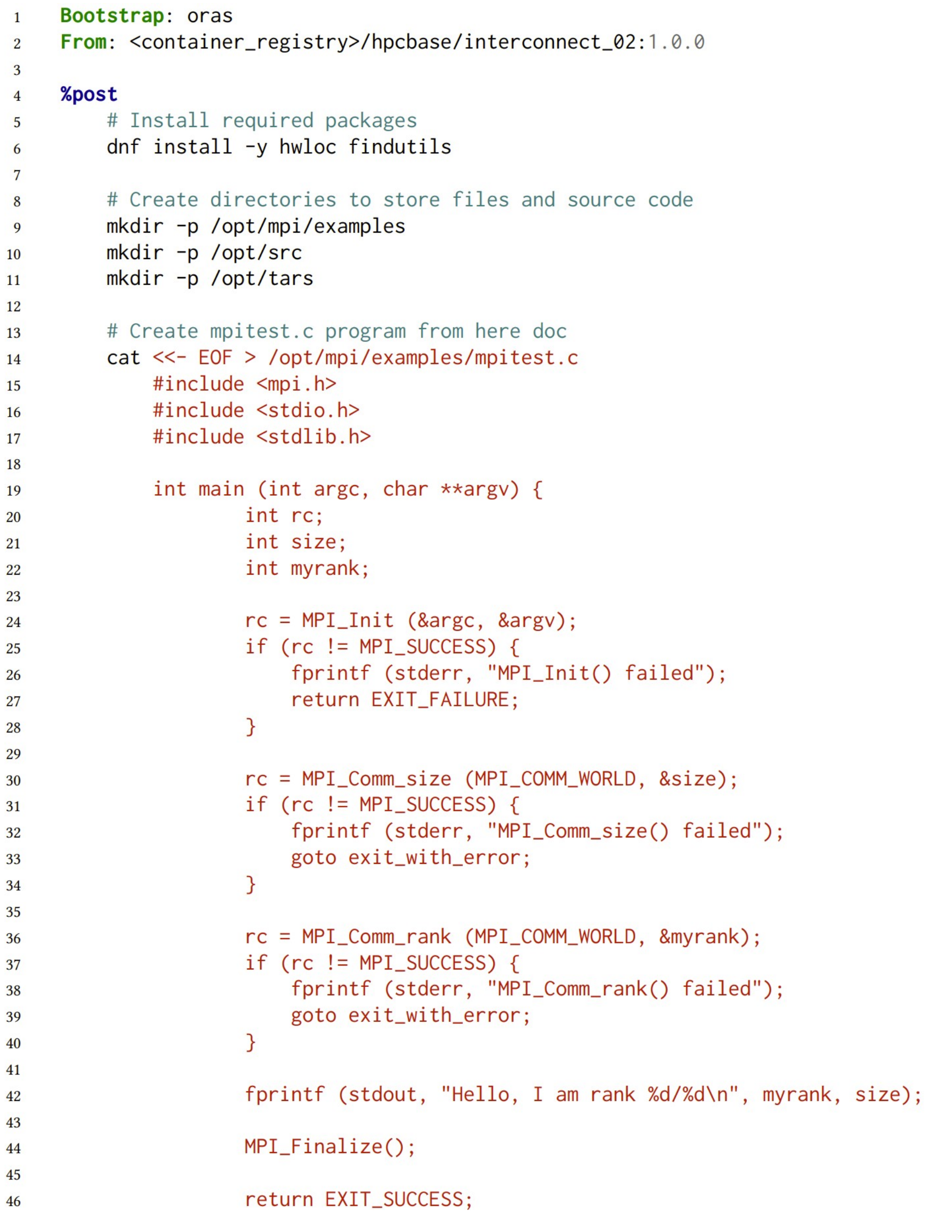

Base++: Add interconnect drivers
Base+: Add compilers
Base Container: Operating System



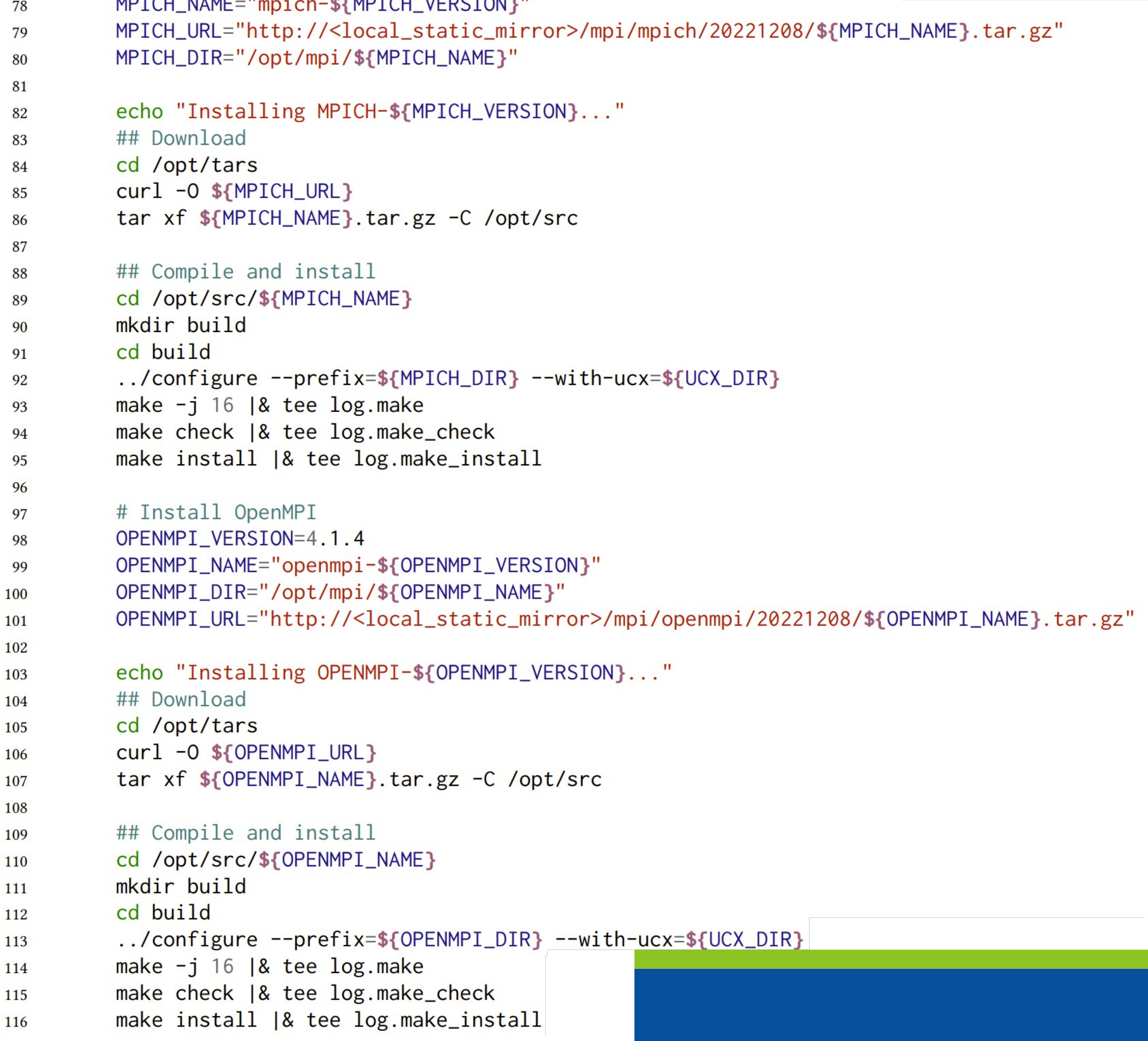
```
Bootstrap: oras
From: <container_registry>/hpcbase/extra_01:1.0.0

%post
    # Create environment variables of software versions
    MLNX_OFED=MLNX_OFED_LINUX-5.4-3.6.8.1-rhel8.6-x86_64
    OPX=CornelisOPX-Basic.RHEL86-x86_64.10.12.1.0.7

    # Make src in /opt to hold source code
    mkdir -p /opt/src

    # Download the MLNX_OFED and OPX files to /opt
    cd /opt
    curl -O http://<local_static_mirror>/interconnects/MLNX/20221208/${MLNX_OFED}.tgz
    curl -O http://<local_static_mirror>/interconnects/OPX/20230212/${OPX}.tgz

    # Install required packages for InfiniBand
    dnf install -y python39 pciutils lsof ethtool tcsh libnl3 tk numactl-libs tcl

    # Extract MLNX_OFED and install it
    tar xf ${MLNX_OFED}.tgz -C /opt/src
    /opt/src/${MLNX_OFED}/mlnxofedinstall --without-fw-update --skip-unsupported-devices-check --basic --user-space-only --distro RHEL8.6
    ↪  --without-depcheck -q

    # Install required packages for OmniPath
    dnf install -y irqbalance kernel-modules-extra kmod libgcc perl perl-Getopt-Long perl-Socket opensm-libs python2 libatomic
    dnf download ibacm*x86_64
    rpm -ivh --nodeps ibacm*.rpm

    # Extract OPX and install it
    tar xf ${OPX}.tgz -C /opt/src
    cd /opt/src/${OPX}
    ./INSTALL -i intel_hfi -i opa_stack --user-space

    # Clean up tarballs/downloads and source directories
    cd /opt
    rm -rf /opt/src
    rm -f /opt/*.tgz
    rm -f /opt/*.rpm

    # Capture useful system information
    echo "Architecture" $(lscpu | grep "^Architecture" | cut -d ':' -f 2) >> "${SINGULARITY_LABELS}"
    echo "CPU" $(lscpu | grep "^Model name" | cut -d ':' -f 2) >> "${SINGULARITY_LABELS}"
    echo "uname" $(uname -srvpio) >> "${SINGULARITY_LABELS}"

```

MPI + Base++ Container Definition File


MPI + Base++
Base++: Add interconnect drivers
Base+: Add compilers
Base Container: Operating System

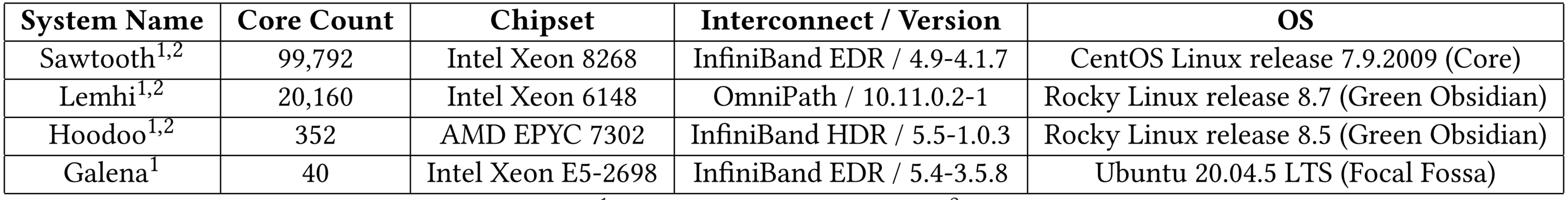


```
Bootstrap: oras
From: <container_registry>/hpcbase/interconnect_02:1.0.0

%post
    # Install required packages
    dnf install -y hwloc findutils

    # Create directories to store files and source code
    mkdir -p /opt/mpi/examples
    mkdir -p /opt/src
    mkdir -p /opt/tars

    # Create mpitest.c program from here doc
    cat <<- EOF > /opt/mpi/examples/mpitest.c
        #include <mpi.h>
        #include <stdio.h>
        #include <stdlib.h>

        int main (int argc, char **argv) {
                int rc;
                int size;
                int myrank;

                rc = MPI_Init (&argc, &argv);
                if (rc != MPI_SUCCESS) {
                    fprintf (stderr, "MPI_Init() failed");
                    return EXIT_FAILURE;
                }

                rc = MPI_Comm_size (MPI_COMM_WORLD, &size);
                if (rc != MPI_SUCCESS) {
                    fprintf (stderr, "MPI_Comm_size() failed");
                    goto exit_with_error;
                }

                rc = MPI_Comm_rank (MPI_COMM_WORLD, &myrank);
                if (rc != MPI_SUCCESS) {
                    fprintf (stderr, "MPI_Comm_rank() failed");
                    goto exit_with_error;
                }

                fprintf (stdout, "Hello, I am rank %d/%d\n", myrank, size);

                MPI_Finalize();

                return EXIT_SUCCESS;
```



```
    # Install MPICH
    MPICH_VERSION=3.4.3
    MPICH_NAME="mpich-${MPICH_VERSION}"
    MPICH_URL="http://<local_static_mirror>/mpi/mpich/20221208/${MPICH_NAME}.tar.gz"
    MPICH_DIR="/opt/mpi/${MPICH_NAME}"

    echo "Installing MPICH-${MPICH_VERSION}..."
    ## Download
    cd /opt/tars
    curl -O ${MPICH_URL}
    tar xf ${MPICH_NAME}.tar.gz -C /opt/src

    ## Compile and install
    cd /opt/src/${MPICH_NAME}
    mkdir build
    cd build
    ../configure --prefix=${MPICH_DIR} --with-ucx=${UCX_DIR}
    make -j 16 |& tee log.make
    make check |& tee log.make_check
    make install |& tee log.make_install

    # Install OpenMPI
    OPENMPI_VERSION=4.1.4
    OPENMPI_NAME="openmpi-${OPENMPI_VERSION}"
    OPENMPI_DIR="/opt/mpi/${OPENMPI_NAME}"
    OPENMPI_URL="http://<local_static_mirror>/mpi/openmpi/20221208/${OPENMPI_NAME}.tar.gz"

    echo "Installing OPENMPI-${OPENMPI_VERSION}..."
    ## Download
    cd /opt/tars
    curl -O ${OPENMPI_URL}
    tar xf ${OPENMPI_NAME}.tar.gz -C /opt/src

    ## Compile and install
    cd /opt/src/${OPENMPI_NAME}
    mkdir build
    cd build
    ../configure --prefix=${OPENMPI_DIR} --with-ucx=${UCX_DIR}
    make -j 16 |& tee log.make
    make check |& tee log.make_check
    make install |& tee log.make_install
```

Container Systems for Testing



System Name	Core Count	Chipset	Interconnect / Version	OS
Sawtooth[1,2]	99,792	Intel Xeon 8268	InfiniBand EDR / 4.9-4.1.7	CentOS Linux release 7.9.2009 (Core)
Lemhi[1,2]	20,160	Intel Xeon 6148	OmniPath / 10.11.0.2-1	Rocky Linux release 8.7 (Green Obsidian)
Hoodoo[1,2]	352	AMD EPYC 7302	InfiniBand HDR / 5.5-1.0.3	Rocky Linux release 8.5 (Green Obsidian)
Galena[1]	40	Intel Xeon E5-2698	InfiniBand EDR / 5.4-3.5.8	Ubuntu 20.04.5 LTS (Focal Fossa)

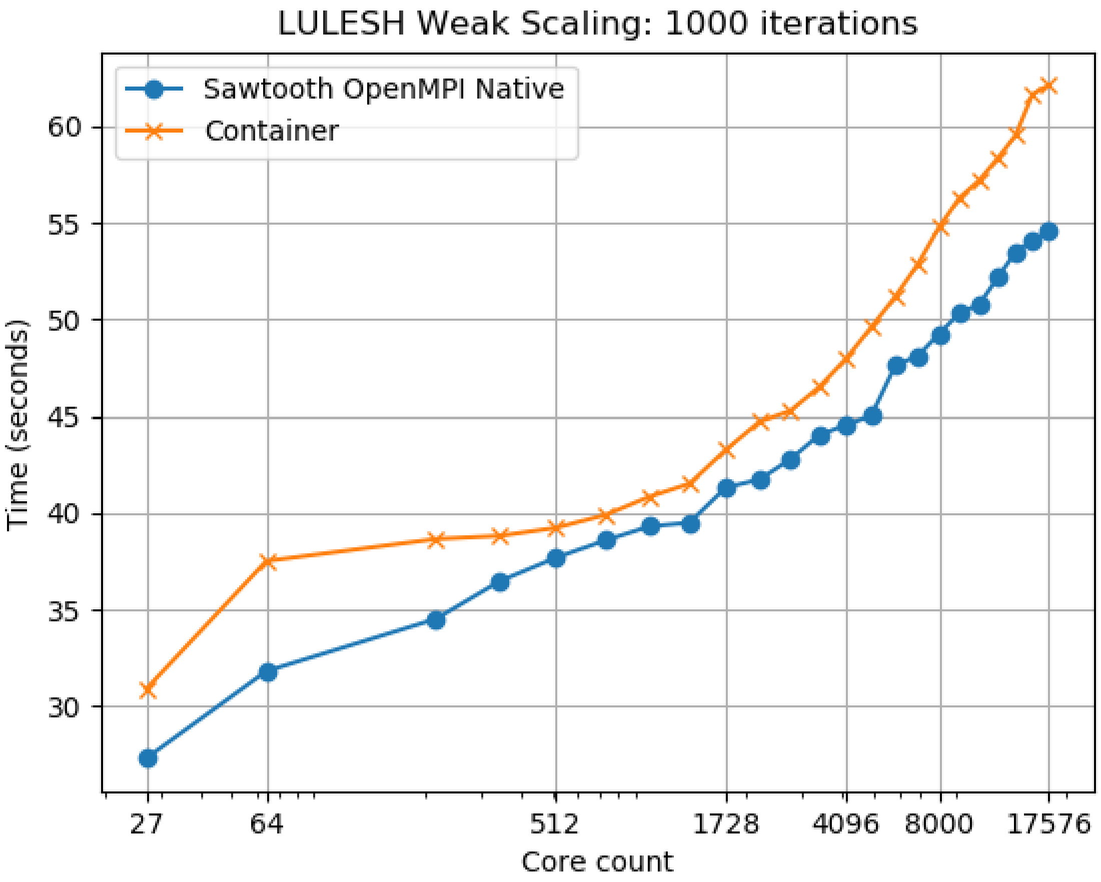
Sawtooth

Lemhi

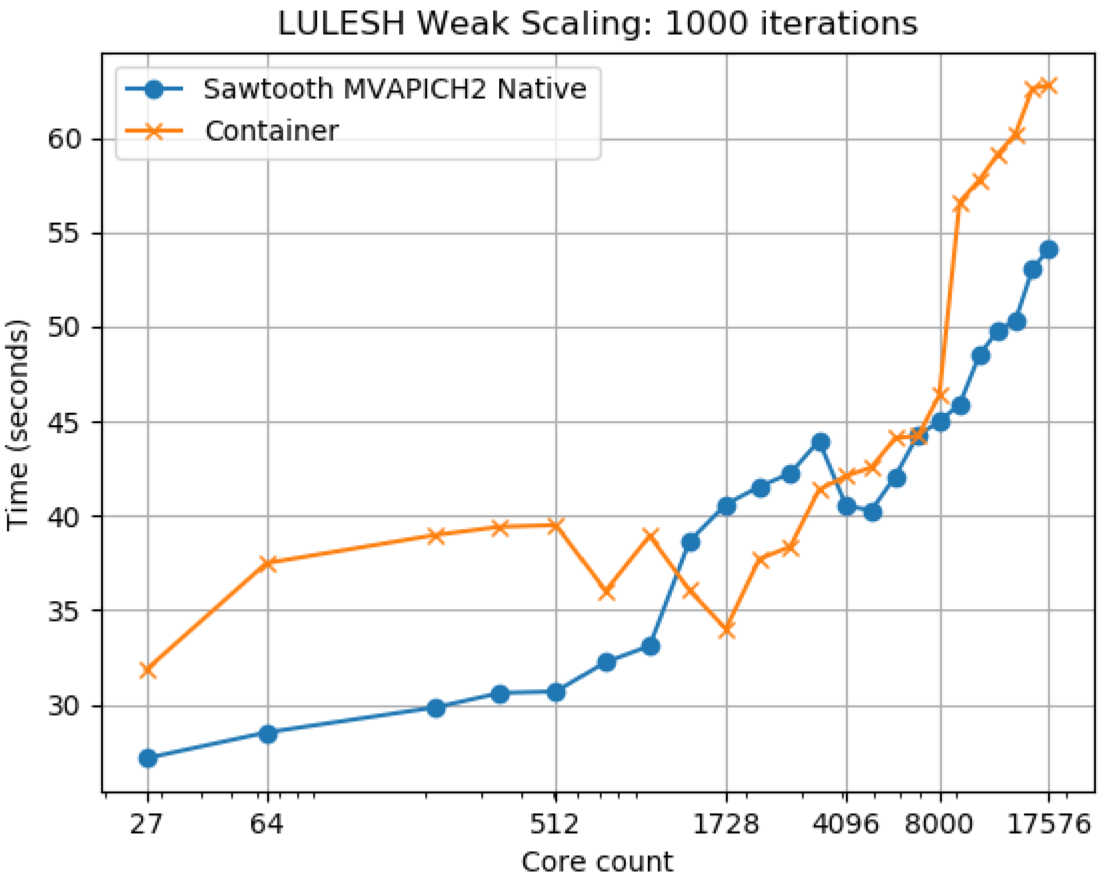
Hoodoo

Galena

LULESH – Sawtooth

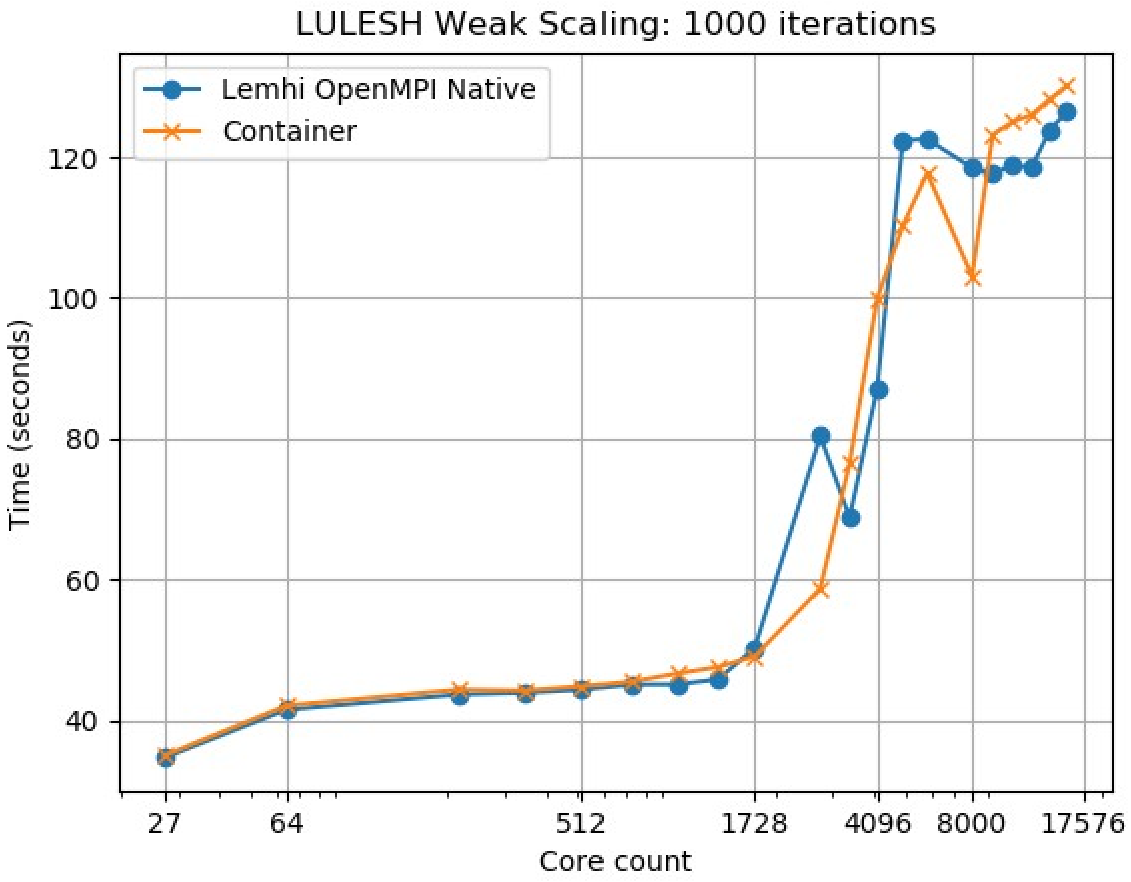


OpenMPI 4.1.4

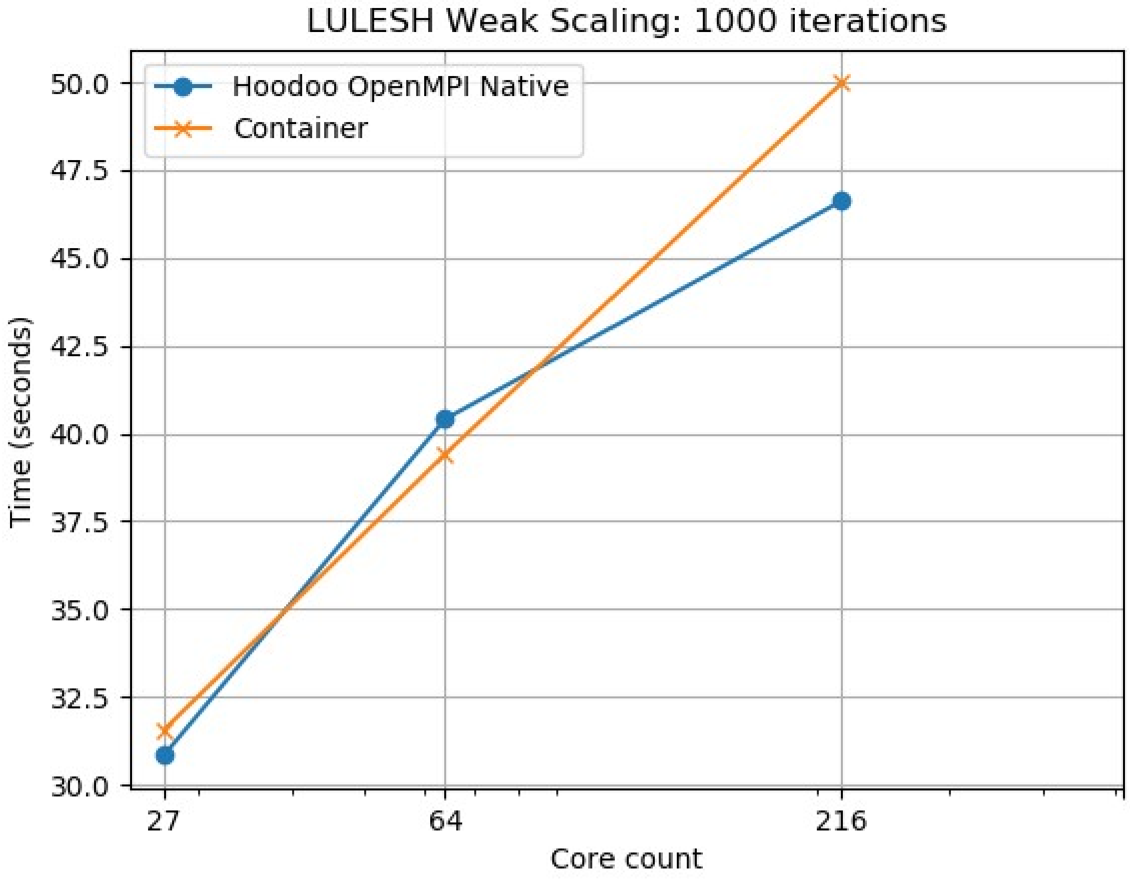


MVAPICH2 2.3.5

LULESH – Lemhi and Hoodoo

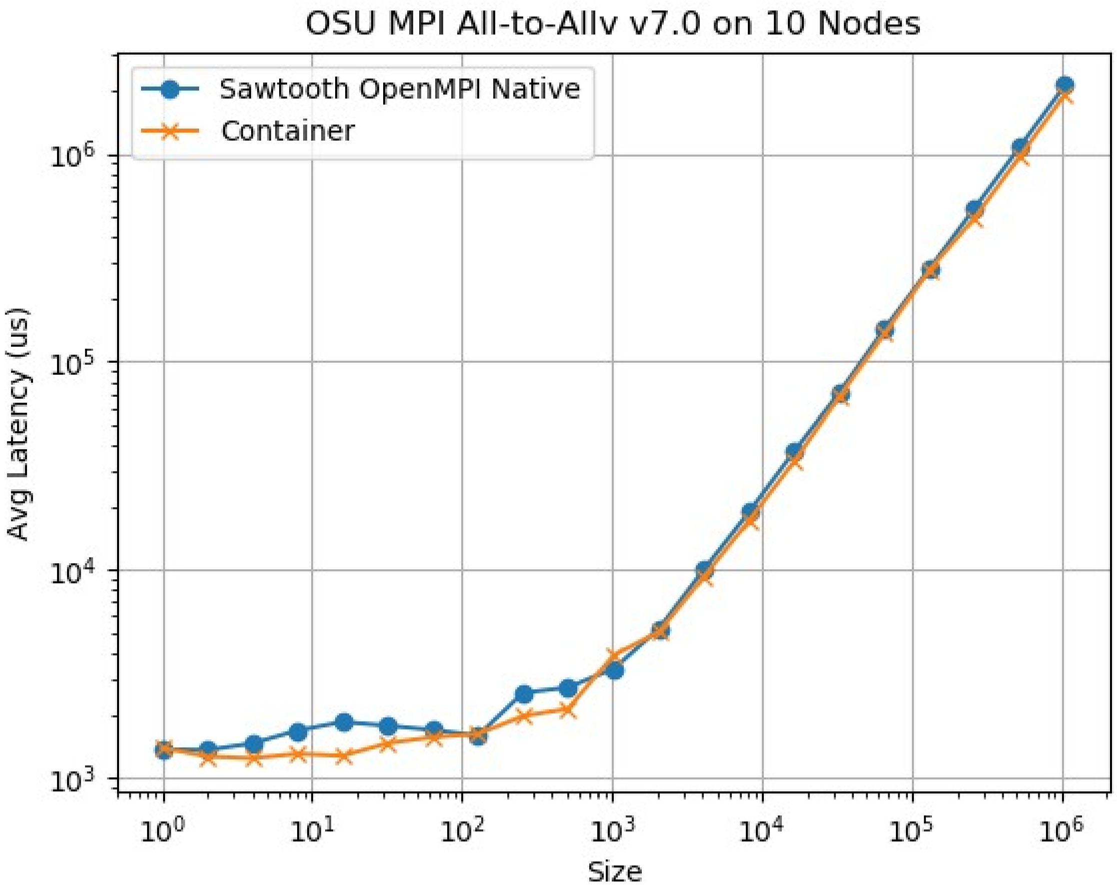


Lemhi
OpenMPI 4.1.1

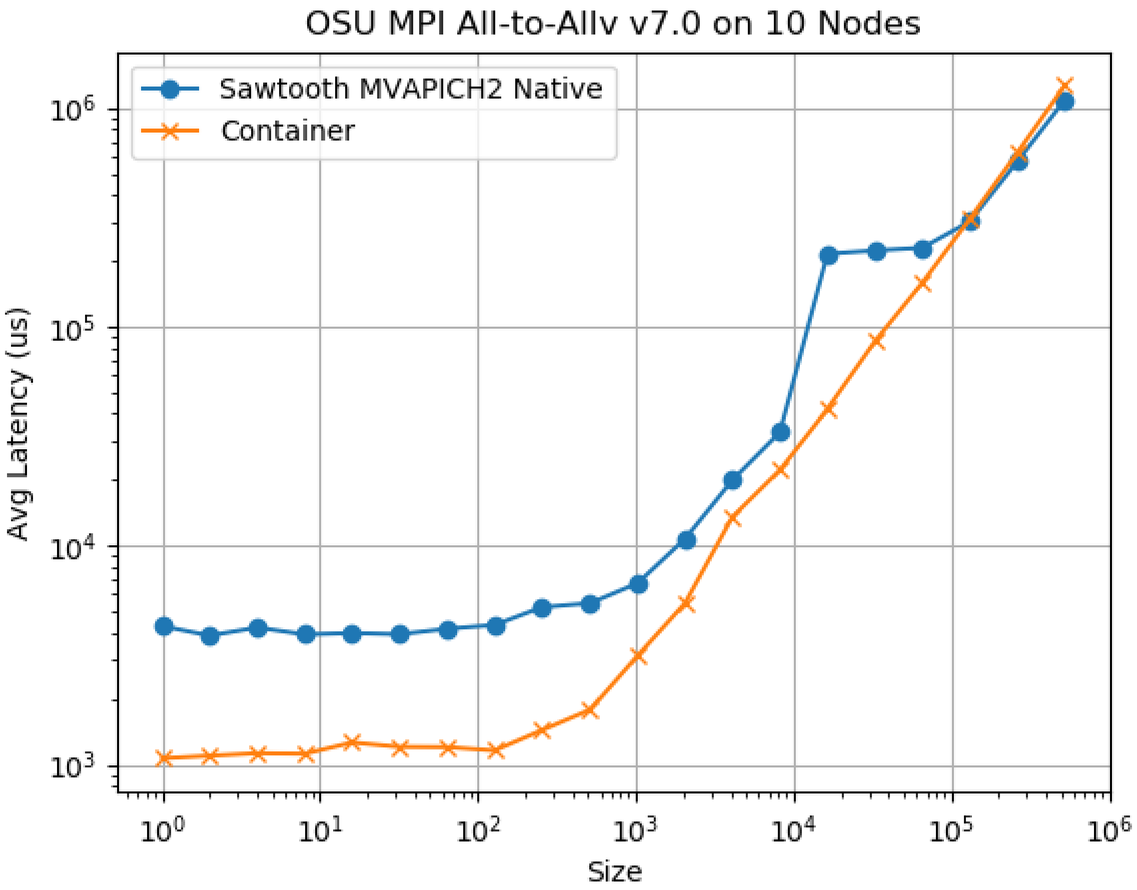


Hoodoo
OpenMPI 4.0.5

OSU MPI All-to-Allv 10 Nodes – Sawtooth

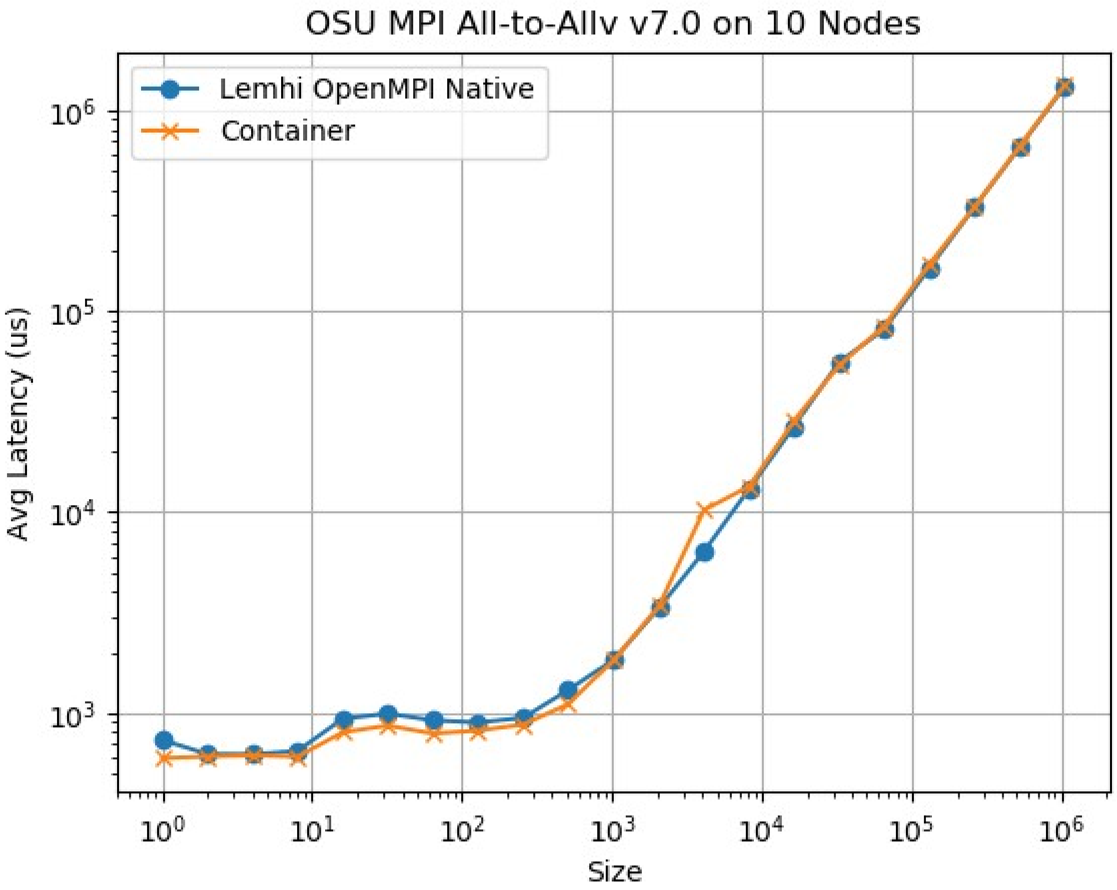


OpenMPI 4.1.4

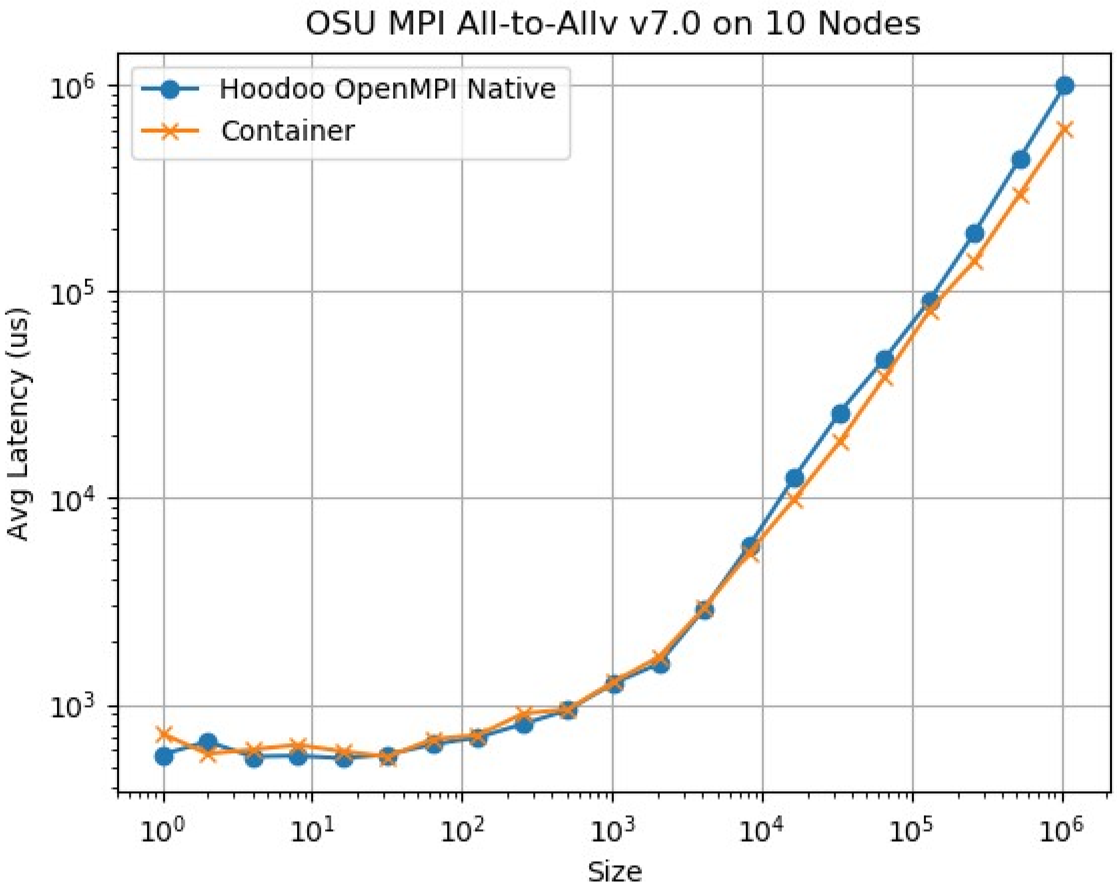


MVAPICH2 2.3.5

OSU MPI All-to-Allv 10 Nodes – Lemhi and Hoodoo

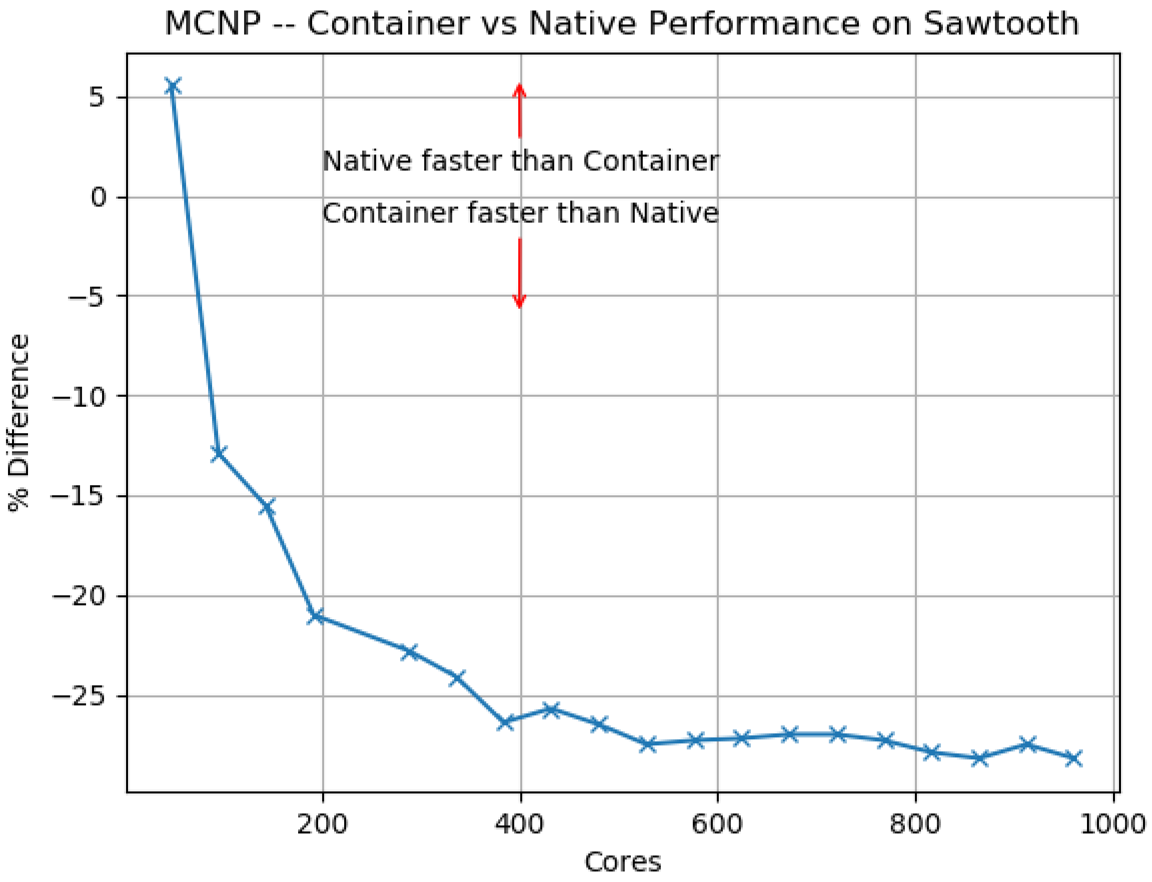


Lemhi
OpenMPI 4.1.1

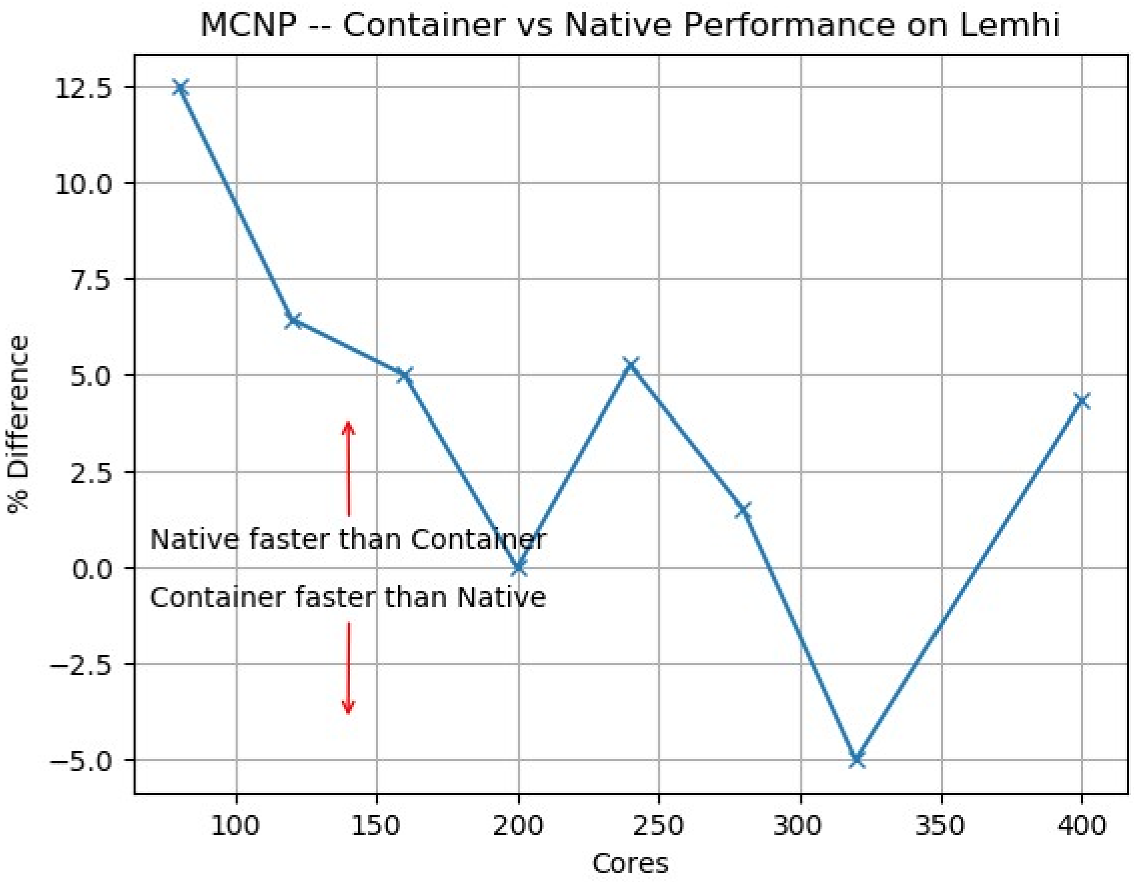


Hoodoo
OpenMPI 4.0.5

MCNP – Container vs Native Performance

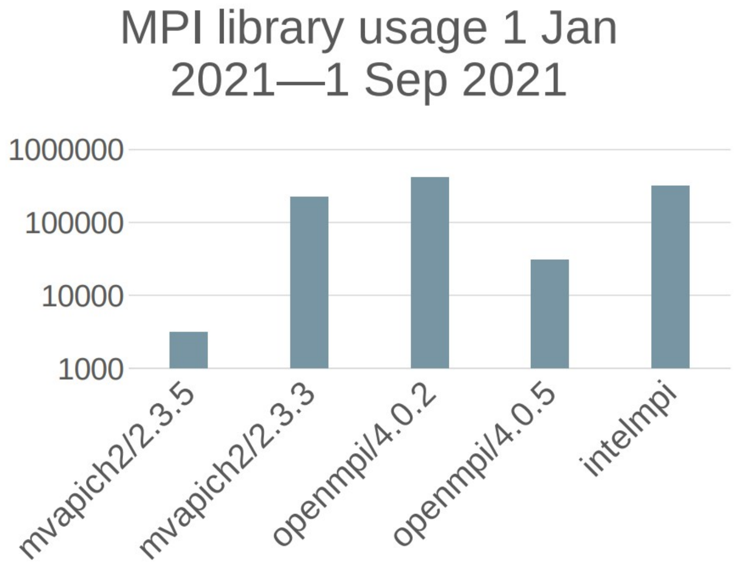


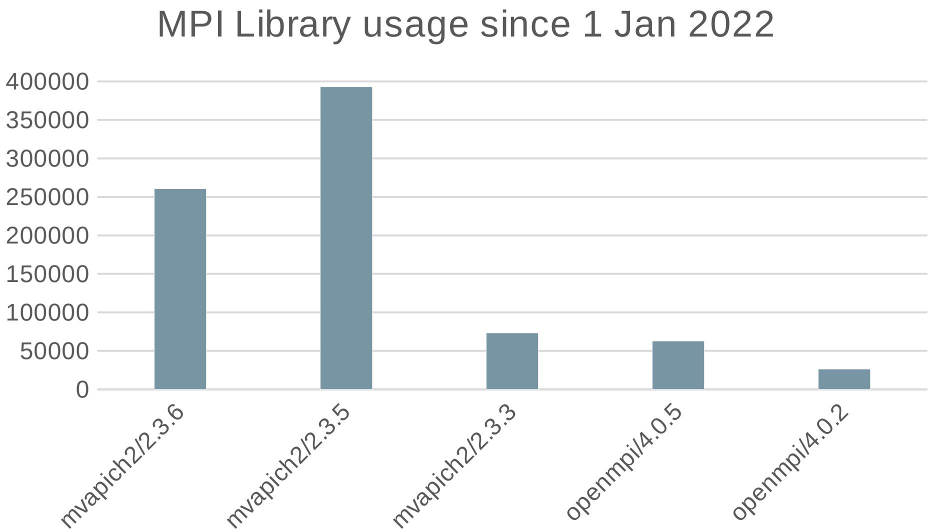
Sawtooth
OpenMPI 4.1.4

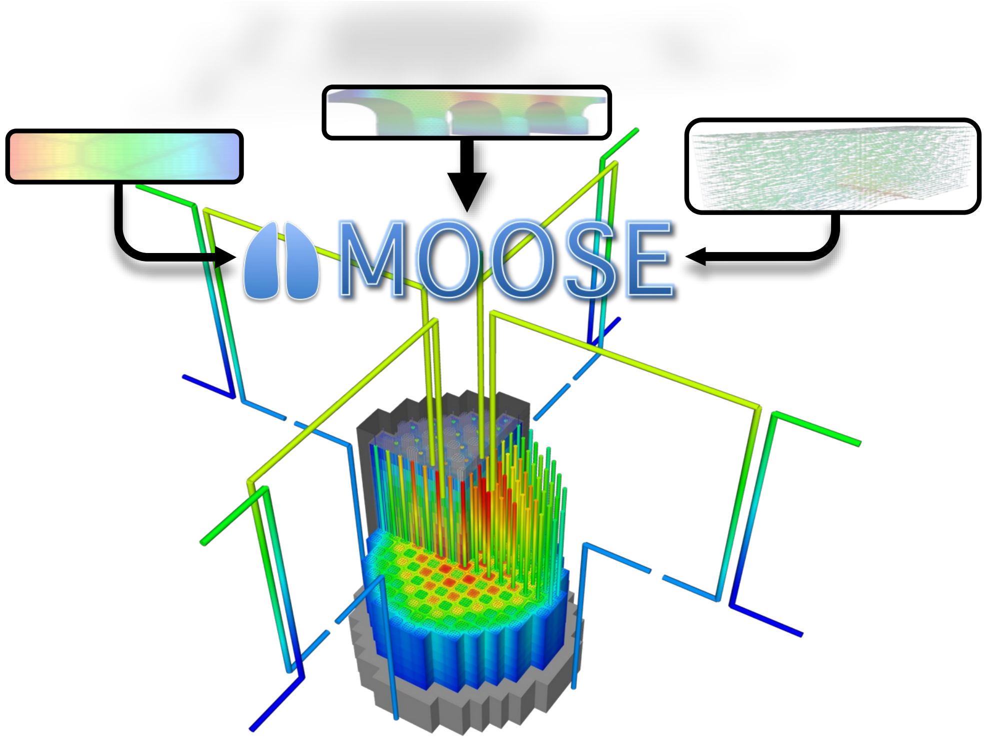


Lemhi
OpenMPI 4.1.1

MVAPICH2 and MOOSE



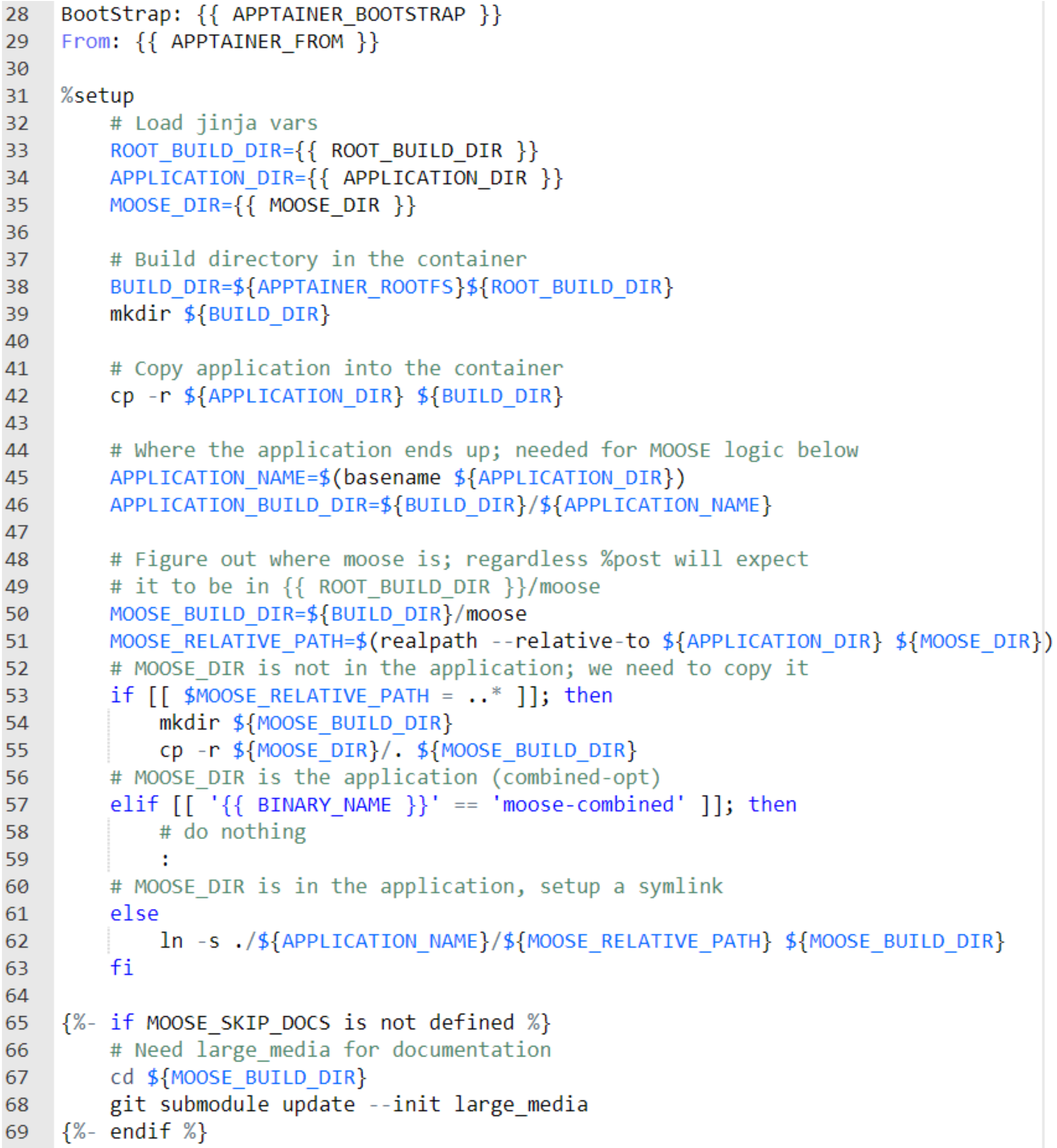




Multiphysics Object-Oriented Simulation Environment

- Open-source finite element framework
  - Basis for most nuclear modeling and simulation tools developed at INL

MOOSE Container layers – outcome



```
BootStrap: {{ APPTAINER_BOOTSTRAP }}
From: {{ APPTAINER_FROM }}

%setup
    # Load jinja vars
    ROOT_BUILD_DIR={{ ROOT_BUILD_DIR }}
    APPLICATION_DIR={{ APPLICATION_DIR }}
    MOOSE_DIR={{ MOOSE_DIR }}

    # Build directory in the container
    BUILD_DIR=${APPTAINER_ROOTFS}${ROOT_BUILD_DIR}
    mkdir ${BUILD_DIR}

    # Copy application into the container
    cp -r ${APPLICATION_DIR} ${BUILD_DIR}

    # Where the application ends up; needed for MOOSE logic below
    APPLICATION_NAME=$(basename ${APPLICATION_DIR})
    APPLICATION_BUILD_DIR=${BUILD_DIR}/${APPLICATION_NAME}

    # Figure out where moose is; regardless %post will expect
    # it to be in {{ ROOT_BUILD_DIR }}/moose
    MOOSE_BUILD_DIR=${BUILD_DIR}/moose
    MOOSE_RELATIVE_PATH=$(realpath --relative-to ${APPLICATION_DIR} ${MOOSE_DIR})
    # MOOSE_DIR is not in the application; we need to copy it
    if [[ $MOOSE_RELATIVE_PATH = ..* ]]; then
        mkdir ${MOOSE_BUILD_DIR}
        cp -r ${MOOSE_DIR}/. ${MOOSE_BUILD_DIR}
    # MOOSE_DIR is the application (combined-opt)
    elif [[ '{{ BINARY_NAME }}' == 'moose-combined' ]]; then
        # do nothing
        :
    # MOOSE_DIR is in the application, setup a symlink
    else
        ln -s ./${APPLICATION_NAME}/${MOOSE_RELATIVE_PATH} ${MOOSE_BUILD_DIR}
    fi

{%- if MOOSE_SKIP_DOCS is not defined %}
    # Need large_media for documentation
    cd ${MOOSE_BUILD_DIR}
    git submodule update --init large_media
{%- endif %}
```

- Utilizes the ”MPI + Base++” container as a base for continuous integration (non-HPC) and most HPC execution
- Improved reproducibility and portability
- Integrated into CI/CD for building workflows
  - Reduced job builds to 20 per week, which are all automated
- Simplified build process

Conclusion

- This strategy has shown the following:
  - Portability
    - Across multiple supercomputer architectures
  - Reproducibility
    - Due to layers and local mirrors
  - Traceability
    - Def files only have most recent changes
  - Simplification
    - HPC staff only needs to focus on MPI ABI and interconnect series compatibility
  - Security
    - Insulation against host system updates

Questions?

INL

Idaho National Laboratory

Battelle Energy Alliance manages INL for the U.S. Department of Energy’s Office of Nuclear Energy. INL is the nation’s center for nuclear energy research and development, and also performs research in each of DOE’s strategic goal areas: energy, national security, science and the environment.

WWW.INL