Title: Docker Containers for MCNP® Development

Author(s): Zukaitis, Anthony J.

Intended for: Report

Issued: 2024-10-21 (rev.1)

Los Alamos
NATIONAL LABORATORY

Docker Containers for MCNP® development

Anthony J. Zukaitis

LA-UR-24-31167
October 17, 2024

1 Introduction

Containers are a revolutionary technology in software development and deployment that provides a lightweight, portable environment for ensuring consistency across multiple computing environments. In anticipation of the MCNP [1] 6.3.1 release, two Docker [2] container images have been released on DockerHub [3] for general use. The MCNP source code is not included in the images, and users are still required to obtain it through RSICC [4]. The images produced by Docker are compliant with the OCI (Open Container Initiative) standards [5], ensuring compatibility with other container engines such as Podman [6] or Kubernetes' CRI-O [7].

Initially, the images are stored under the author's personal space on DockerHub (docker.io/azukaitis), but they will be relocated to a dedicated MCNP group space once approved. In the future, they will also be available through the registry feature of the https://github.com/lanl/mcnp-containers project.

The use of Docker provides a pre-configured environment for building and running MCNP, ensuring reproducibility of results across various host architectures. This significantly improves consistency when running MCNP on different systems. Notably, executables and installers from the Docker images have successfully passed the MCNP development branch testing suite on x86-64 architectures, including Windows, macOS, and Linux operating systems.

Furthermore, testing has demonstrated compatibility with macOS Docker in emulation mode on the latest Apple Mac M2 Ultra hardware, ensuring robust support even on the latest platforms.

In this document, we will provide a step-by-step guide to using the Docker images across multiple platforms. Additionally, we will present performance numbers for building and running the MCNP test suite.

2 Container Image Overview

Before diving into the specifics of the MCNP container image, it's essential to define some key terminology related to containers to avoid confusion.

1. **Container Instantiation (or Container Runtime)**: This refers to the process of running a container from an existing container image. This involves creating a new instance of the container, which is then managed by the container engine.

2. **Container Image**: A container image is a pre-packaged template that includes the application code, dependencies, and other necessary components. It is used to create instances of containers.

3. **Image Builder**: The image builder is the tool responsible for creating a container image from a set of instructions or a base image. Popular image builders include Dockerfile, Buildah [8], and Kaniko [9].

4. **Container Engine (or Runtime Environment)**: A container engine is the runtime environment that allows containers to run on a host system. Popular container engines include Docker, Podman, rkt, and cri-o. The container engine ensures process isolation, manages resources, and provides networking support.

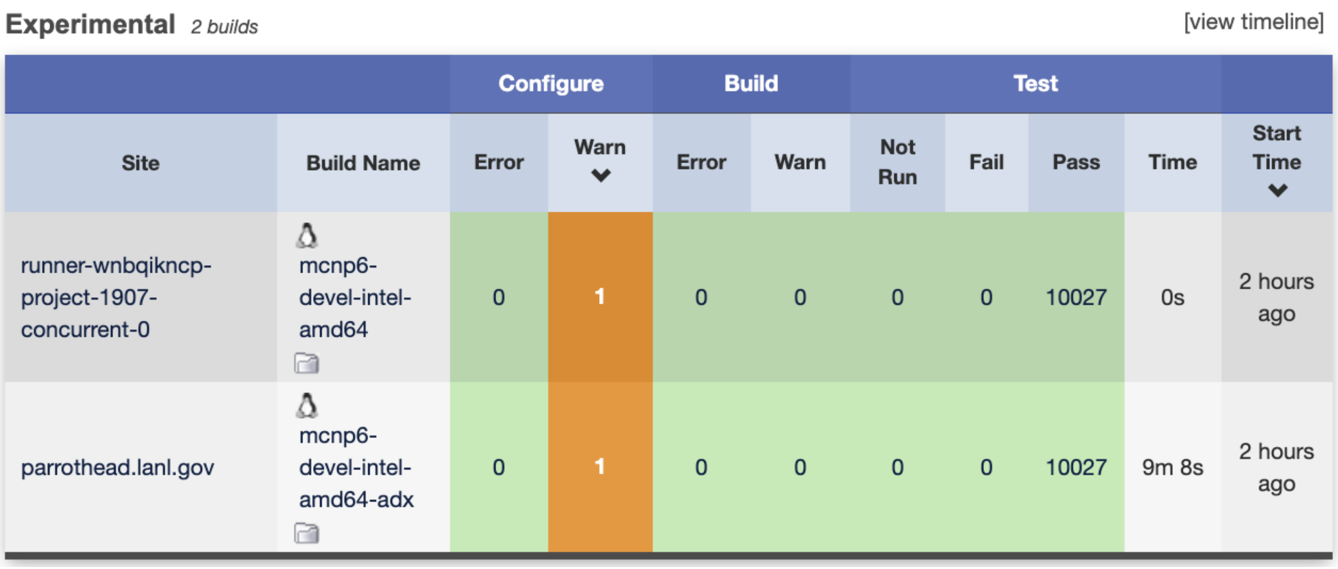
2.1 Dependencies

The base layer of the MCNP developer image is built upon the CentOS 7 image. Although CentOS 7 is an older Linux distribution, it is required for compatibility with our LANL Red Hat 7 servers. This CentOS image serves as the foundation, representing a minimal Linux environment stripped down to reduce its size.

To create the base image, we used the yum package manager to install basic system utilities necessary for building additional software. If a package did not exist, it was compiled from source and installed within the image. An example Dockerfile is attached in Appendix A.

The following key packages were included in the container image:

- `gcc/g++/gfortran` 11.3.0
- `cmake` 3.28.3
- `OpenMPI` 4.1.1
- `HDF5` 1.14.2
- `python` 3.12.3
- `lyx` (git hash: 16660d12) for compiling the MCNP manual
- `qt` 6.3.2
- `Intel Classic` 2021.9.0



Experimental *2 builds* [view timeline]

		Configure		Build		Test				
Site	Build Name	Error	Warn	Error	Warn	Not Run	Fail	Pass	Time	Start Time
runner-wnbqikncp-project-1907-concurrent-0	mcnp6-devel-intel-amd64	0	1	0	0	0	0	10027	0s	2 hours ago
parrothead.lanl.gov	mcnp6-devel-intel-amd64-adx	0	1	0	0	0	0	10027	9m 8s	2 hours ago

Figure 2-1: Two runs reported on our CDash server for the intel installer on a host machine (`mccoy`) as well as inside a minimized base CentOS 7 image (runner-wnb...)

All packages were installed in `/usr/local` within the container image, and their respective versions or git hashes were stored in the install path. To ensure proper integration of these tools, environment variables such as `LD_LIBRARY_PATH`, `CMAKE_PREFIX_PATH`, and `C_INCLUDE_PATH` were updated to include the appropriate paths.

By default, the user's shell environment is set to source the `intel_setvars.sh` script, which configures the environment variables required for the Intel compilers.

This image was able to not only pass all tests but also build the MCNP installers in multiple configurations. These installers produced inside of the container, were tested on external hardware as well as inside a minimal container. Two of the testing results for one of the installers on an external linux host and inside a minimal container are shown in Figure 2-1.

3 MCNP developer container usage

By default, the username within the container is set to be mcnp-developer. This mcnp-developer user has administrator privileges and can do passwordless sudo commands within the container. This means that once one has instantiated a container, they are free to install or build any packages desired. Any modifications within the container will not modify the MCNP developer image. If using a modified version of the container image is desired from an instantiated container, the "docker commit" command can be used to save your modified version of the MCNP developer image.

The default compiler environment in the developer image is intel classic 2021.9.0. To use the gcc 11.3.0 compiler, set the following environment compiler flags:

- CC=gcc
- CXX=g++
- FC=gfortran

In order to run problems or tests, install the mcnp data on the host using the MCNP data manager [10] and set the DATAPATH environent variable to the install location on the host.

Podman has also shown to be a viable open source product developed by Redhat. If Podman is desired, all of the following examples should work seemlessly by switching the docker command to podman.

3.1 Container Usage on Linux

Docker is best installed using the the native package manager for your linux distribution. Although the MCNP developer docker image is centos based, the host linux distribution is irrelevant.

Once Docker or Podman is installed, use the "docker pull" command to download the MCNP 6.3.1 developer image.

```
bash> docker pull docker.io/azukaitis/mcnp-developer-centos7-openmpi:6.3.1
6.3.1: Pulling from azukaitis/mcnp-developer-centos7-openmpi
93485e7767db: Already exists
f1f85fab4d91: Already exists
fd27e14ade45: Pull complete
Digest: sha256:685de73ada466b3e6b7e27041ec55d4e0274440c734f202cfac162b1da88ba38
Status: Downloaded newer image for azukaitis/mcnp-developer-centos7-openmpi:6.3.1
docker.io/azukaitis/mcnp-developer-centos7-openmpi:6.3.1
```

From time to time, these images will be updated as bugs are addressed and features are added. Therefore the hashes of the layers and checksum may change over time.

A good practice for using Docker and the MCNP developer image is to create a unique working directory in the users home directory on the host system (mywork). In this directory, copy the MCNP source code into that directory.

```
bash> cd $HOME
bash> mkdir mywork
bash> cp -r my-MCNP-source-code $HOME/mywork/MCNP
```

The container will be started with the following flags keeping in mind that your home directory in the container will be `/home/mcnp-developer`:

- xhost +localhost: allows the localhost access to the X server.

MCNP developer container usage

- -it: interactve and assign a tty
- --rm : remove the container once exited
- --volume $DATAPATH:/mcnpdata : mounts the host mcnp data directory to /mcnpdata inside the container.
- --env DATAPATH=/mcnpdata : sets the DATAPATH environment variable inside the container to /mcnpdata inside the container.
- --env DISPLAY=host.docker.internal:0 : sets the DISPLAY environment variable to a special value that Docker uses to forward X11 display fuctionality.
- --name mcnp-container : sets the name of the container to mcnp-container.
- --volume $HOME/mywork:/home/mcnp-developer/mywork : mounts your unique working directory mywork in the /home/mcnp-developer/mywork directory inside the container.

Using this command with the "--" flags can be repeated multiple times with out worry about creating multiple containers and needing to manage them. This is for pure convenience for the examples in this report.

```
bash> xhost +localhost
bash> docker run -it --rm  \
         --volume $DATAPATH:/mcnpdata \
         --env DATAPATH=/mcnpdata \
         --env DISPLAY=host.docker.internal:0 \
         --name mcnp-container \
         --volume $HOME/mywork:/home/mcnp-developer/mywork \
         docker.io/azukaitis/mcnp-developer-centos7-openmpi:6.3.1 /bin/bash

:: initializing oneAPI environment ...
   bash: BASH_VERSION = 4.2.46(2)-release
   args: Using "$@" for setvars.sh arguments:
:: compiler -- latest
:: debugger -- latest
:: dev-utilities -- latest
:: mpi -- latest
:: tbb -- latest
:: oneAPI environment initialized ::

[mcnp-developer@8db90d07c665 ~]$
```

On Windows, the DOS and Powershell environments have different ways to dereference environment variables. Therefore the docker run command must be modified to do so. In addition, the continue line character is different. For example in DOS, the command would be:

```
bash> docker run -it --rm ^
         --volume %DATAPATH%:/mcnpdata ^
         --env DATAPATH=/mcnpdata ^
         --env DISPLAY=host.docker.internal:0 ^
         --name mcnp-container ^
         --volume %HOME%/mywork:/home/mcnp-developer/mywork ^
         docker.io/azukaitis/mcnp-developer-centos7-openmpi:6.3.1 /bin/bash

:: initializing oneAPI environment ...
   bash: BASH_VERSION = 4.2.46(2)-release
   args: Using "$@" for setvars.sh arguments:
:: compiler -- latest
:: debugger -- latest
:: dev-utilities -- latest
:: mpi -- latest
:: tbb -- latest
:: oneAPI environment initialized ::

[mcnp-developer@8db90d07c665 ~]$
```

(On Windows, xhost is not available as part of the Windows versions of the X11 server)

Next we will use cmake to configure, build, and run the regressions suite as one normally would [11]. This is the 6.3.0 build guide since the 6.3.1 build guide has not been released at this time. In the example below, extranious output is replaced with "***".

```
[mcnp-developer@8db90d07c665 ~]$ cd mywork/MCNP
[mcnp-developer@8db90d07c665 ~]$ mkdir build; cd build
[mcnp-developer@f3849f55a536 build]$ cmake ../MCNP  -DCMAKE_BUILD_TYPE=Release
-- CMAKE_BUILD_TYPE is Release
-- The C compiler identification is Intel 2021.9.0.20230302
-- The CXX compiler identification is Intel 2021.9.0.20230302
-- The Fortran compiler identification is Intel 2021.9.0.20230302
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
***
-- Configuring done (43.9s)
-- Generating done (2.8s)
-- Build files have been written to: /home/mcnp-developer/mywork/MCNP/build
[mcnp-developer@8db90d07c665 ~]$ make -j $(nproc)
***
[100%] Linking CXX static library libmcnp_Fortran.a
[100%] Built target mcnp_Fortran
[100%] Building Fortran object CMakeFiles/mcnp.dir/Source/src/main.F90.o
[100%] Linking Fortran executable MCNP
[100%] Built target mcnp
[mcnp-developer@8db90d07c655 ~]$ ctest -j $(nproc) -R regression
Test project /home/mcnp-developer/mywork/MCNP/build
          Start   79: mcnp.regression.inp07.initialize
          Start   93: mcnp.regression.inp09.initialize
***
          Start  406: mcnp.regression.inp155.produces.inp155r.h5
1508/1510 Test  #406: mcnp.regression.inp155.produces.inp155r.h5
     ...................................................   Passed    0.01 sec
1509/1510 Test  #405: mcnp.regression.inp155.diff.inp155m.inp155.mctl
     ..............................................   Passed    0.04 sec
1510/1510 Test  #404: mcnp.regression.inp155.diff.inp155o.inp155.outp
     ..............................................   Passed    0.05 sec

100% tests passed, 0 tests failed out of 1510
```

It is noted here that we use the container's "nproc" command to determine the number of jobs to build and test with. This ensures we are using the maximum number of resources allocated to the container. On systems with many cores and mounted file systems, the number may be needed to be set explicitly to something smaller. Using Docker on the MacOS operating system, one may also have to increase the amount of memory allotted to the container in the Docker->Settings->Resources menu. This is to account for the amount of memory the intel compiler requires per job during compilation.

Once the MCNP executable is built, one can simply add the full path of the directory to the PATH environment variable. To run your executable.

```
[mcnp-developer@8db90d07c665 ~]$ export PATH=/home/mcnp-developer/mywork/MCNP/build:$PATH
[mcnp-developer@8db90d07c665 ~]$ which MCNP
~/mywork/MCNP/build/MCNP
```

To run your personal input files, the container will need access to them from the host. This is most easily accomplished by putting them into the mounted "mywork" directory. Or to keep the MCNP source code separate, one could just mount an additional directory from the host that contains the input files. In this example, we copied inp04.inp to the mywork directory on the host.

```
[mcnp-developer@7a4d4d997d8d mywork]$ MCNP i=inp04.inp
          Code Name & Version = MCNP, 6.3.1, development
          Copyright Triad National Security, LLC/LANL/DOE - see LICENSE file

     _/      _/        _/_/_/       _/      _/       _/_/_/         _/_/_/
    _/_/  _/_/      _/             _/_/    _/       _/    _/     _/
   _/  _/  _/      _/             _/  _/  _/       _/_/_/       _/_/_/
  _/      _/      _/             _/    _/_/       _/           _/    _/
 _/      _/        _/_/_/       _/      _/       _/             _/_/

 MCNP ver=6.4.0, ld=06/12/24  09/19/24 10:23:05
Source version = devel-679dad5c3e
  comment.  Physics models disabled.
 comment. using random number generator 1, initial seed = 19073486328125
***
 ctm =        0.00   nrn =           1206364
 dump    2 on file runtpe.h5   nps =        1000   coll =            46182
 tally data written to file mctal
 mcrun  is done
```

3.2 VSCode

During testing, we also were able to integrate the container with VSCode with the Remote Development extension. Using the "Other containers" pulldown menu, select the container. Then select the right arrow to connect to the existing container that was created with the commands above. Next open the folder inside the container where your MCNP source code is located. Once connected, just start a terminal and issue your cmake and ctest commands.

4 Performance

The performance of various components of the developer workflow has been broken down into the following subsections. When comparing these results, please note that the host system's LANL required endpoint protection/security solutions, such as Nessus, CrowdStrike Falcon, and other security software may impact performance.

4.1 Windows

On the Windows platform, we used Docker version 20.10.7 to run our experiments. The host system was running on a dual socket Intel Xeon Platinum processor with 88 cores in total. The results of our experiments are presented in Table 4-1. We compared the timing performance of building and testing MCNP between the host system and the Docker container environment. These numbers should be taken with a grain of salt as the Intel Windows compiler and Intel Linux compiler are completely different entities.

Step	**command**	**Host**	**Container**
Configure	cmake ../	130s	51s
Build	cmake –build . – -j 44	143s	98s
All Tests	ctest -j 44	960s	586s

Table 4-1: Timing comparisons between the host and Docker container environments for building MCNP on Windows.

Overall, our results show that the Docker continer environment was faster than the host system in terms of building and testing MCNP.

4.2 Linux

On the Linux platform, we used Docker version 1.13.1 to run our experiments. The host system was running on a dual socket Intel Xeon Platinum 8160 processor with 88 cores in total.

The results of our experiments are presented in Table 4-2. We compared the timing performance of building and testing MCNP between the host system and the Docker container environment.

Step	**command**	**Host**	**Container**
Configure	cmake ../	58s	57s
Build	cmake –build . – -j 88	94s	94s
All Tests	ctest -j 88	343s	360s

Table 4-2: Timing comparisons between the host and Docker container environments for building MCNP on Linux.

Here, it was measured that the Docker container environment performed similarly to the host system in terms of building MCNP, but only slightly slower when running tests.

4.3 MacOS M2 Ultra

On the MacOS platform, we used Docker version 20.10.7 to run our experiments. The host system was running on an Apple ARM M2 ultra processor with 23 cores in total. We used the MCNP developer image to build and test our code. This capability fills a large need when one wants to develop on the Mac since the Intel compiler does not support the Mac M(x) architechture. Using Dockers internal emulation mode, all of the tests past and performance is good. Since there was no host software dependency stack for intel to compare with, we used the homebrew[12] package manager on the host to install the GNU compiler suite and other necessary dependencies.

The results of our experiments are presented in Table 4-3. We measured the timing performance of building and testing MCNP using the Docker x86-64 emulation capability. In general, the GNU compiler is always much faster than the Intel compiler since it tends to optimize muych less. None the less, the x86 emulated intel container times are comparable to the host Windows intel timings with only half of the cores.

Step	**command**	**Intel Container**	**Host GNU compiler**
Configure	cmake ../	67s	33s
Build	cmake –build . – -j 22	131s	54s
All Tests	ctest -j 22	1020s	360s

Table 4-3: Timing comparisons of building MCNP on MacOS using Docker container between the Intel Compiler in the mcnp developer container and the GNU compiler suite on the host.

5 Conclusions

Overall, our experience with the Docker container engine system was quite pleasing. We found that taking a capability designed to be isolated and integrating it with the host can be managed with a few developer-side scripts. The X11 graphics side has been inconsistent across all platforms in recent years, but we were rewarded by seeing it finally have a consistent interface now.

In addition to our technical accomplishments, we also noted some key considerations for users of Docker and containerization technologies. For example, the Docker licensing structure can be complex, which may be a concern for those seeking open-source solutions. In such cases, Podman is an excellent alternative to Docker.

Moreover, on Windows platforms, having both Podman and Docker installed on the same system can be challenging due to how Windows manages the underlying virtual machines needed for these capabilities. Therefore, careful consideration should be given to the choice of installing both containerization technologies.

Looking ahead, our future work will focus on transitioning away from production Red Hat legacy support and modern Linux distributions to enable native package managers to handle many dependencies. This effort will also involve creating multi-platform images that directly support both ARM and x86 architectures. To achieve this goal, we plan to migrate MCNP from the Intel compiler to the GCC compiler.

By taking these steps, we aim to improve the performance, efficiency, and maintainability of our containerized applications, while also ensuring compatibility with a broader range of platforms and architectures.

References

[1] Joel Aaron Kulesza et al. *MCNP® Code Version 6.3.0 Theory & User Manual*. Tech. rep. LA-UR-22-30006, Rev. 1. Los Alamos, NM, USA: Los Alamos National Laboratory, Sept. 2022. DOI: 10.2172/1889957. URL: https://www.osti.gov/biblio/1889957.

[2] Docker, Inc. *Docker Documentation*. Accessed: 2024-09-04. 2024. URL: https://docs.docker.com/.

[3] Inc. Docker. *Docker Hub: The World's Leading Service for Finding and Sharing Container Images*. Accessed: 2024-09-16. 2024. URL: https://hub.docker.com.

[4] Radiation Safety Information Computational Center. *RSICC: Collecting, Analyzing, and Disseminating Radiation Transport and Safety Software*. Accessed: 2024-09-16. 2024. URL: https://rsicc.ornl.gov.

[5] Open Container Initiative. *Open Container Initiative: Image Specification*. https://specs.opencontainers.org/image-spec. Accessed: 2024-09-04. 2024.

[6] Inc. Red Hat. *Podman: A Tool for Managing OCI Containers and Pods*. Accessed: 2024-09-16. 2024. URL: https://podman.io.

[7] *Kubernetes*. Accessed: 2024-09-16. https://kubernetes.io/.

[8] Inc. Red Hat. *Buildah: A tool that facilitates building OCI container images*. https://github.com/containers/buildah. Accessed: 2024-09-26. 2024.

[9] Google LLC. *Kaniko: Build Container Images In Kubernetes*. https://github.com/GoogleContainerTools/kaniko. Accessed: 2024-09-26. 2024.

[10] Colin James Josey and Jeremy Lloyd Conlin. *LANL Nuclear Data Manager Format Specification v1.0*. Tech. rep. LA-UR-22-28306. Los Alamos, NM, USA: Los Alamos National Laboratory, Aug. 2022. DOI: 10.2172/1880470. URL: https://www.osti.gov/biblio/1880470.

[11] Jeffrey S. Bull et al. *MCNP® Code Version 6.3.0 Build Guide*. Tech. rep. LA-UR-22-32851, Rev. 1. Los Alamos, NM, USA: Los Alamos National Laboratory, Dec. 2022. DOI: 10.2172/1906011. URL: https://www.osti.gov/biblio/1906011.

[12] Homebrew Team. *Homebrew: The Missing Package Manager for macOS (or Linux)*. Available at: https://brew.sh/. 2024.

A Appendix A - Dockerfiles

Below is the latest Dockerfile used to create the mcnp-developer-centos7-openmpi image 6.1.3. This was used to build the images before the end of life of CentOS 7. After this date, the images were patched by hand to fix deprecated yum mirrors. When the original image was built from this dockerfile, it depended upon on the the Software Collections (SCL) repository which appears to be no longer supported. The image has been subsequentally patched to disable this repository and others that refer to to the disabled repository site `mirrorlist.centos.org`.

```
FROM --platform=linux/amd64 centos:7 as base

# Ensure proxies are set correctly
ARG http_proxy
ENV http_proxy $http_proxy
ENV HTTP_PROXY $http_proxy
ARG https_proxy
ENV https_proxy $https_proxy
ENV HTTPS_PROXY $https_proxy
ARG no_proxy
ENV no_proxy $no_proxy

RUN yum update -y

RUN yum install -y centos-release-scl \
      nc epel-release gcc \
      coreutils grep util-linux-ng procps-ng \
      findutils psmisc sed gawk which sudo perl \
      sudo file patchelf bzip2 wget

RUN yum update -y \
    && yum install -y \
       rh-git218-git \
       rh-git218-git-lfs

ENV PKG=/opt/rh/rh-git218/root
ENV PATH=${PKG}/usr/bin:$PATH
ENV LD_LIBRARY_PATH=${PKG}/usr/lib64:${PKG}/usr/lib:${LD_LIBRARY_PATH}
ENV MANPATH=${PKG}/usr/share/man:$MANPATH
RUN git --version
RUN git-lfs --version

RUN echo "[oneAPI]" >> /etc/yum.repos.d/oneAPI.repo \
    && echo "name=Intel oneAPI repository" >> /etc/yum.repos.d/oneAPI.repo \
    && echo "baseurl=https://yum.repos.intel.com/oneapi" >> /etc/yum.repos.d/oneAPI.repo \
    && echo "enabled=1" >> /etc/yum.repos.d/oneAPI.repo \
    && echo "gpgcheck=1" >> /etc/yum.repos.d/oneAPI.repo \
    && echo "repo_gpgcheck=1" >> /etc/yum.repos.d/oneAPI.repo \
    && echo "gpgkey=https://yum.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-PRODUCTS.PUB
    && yum update -y \
    && yum install -y \
      intel-oneapi-compiler-fortran-2023.1.0 \
      intel-oneapi-compiler-dpcpp-cpp-and-cpp-classic-2023.1.0

WORKDIR /tmp
ENV GCC_VERSION=11.3.0
ENV PREFIX=/usr/local/gcc-11.3.0
ENV PATH=$PREFIX/bin:$PATH
ENV LD_LIBRARY_PATH=$PREFIX/lib64:$PREFIX/lib:$LD_LIBRARY_PATH
RUN wget http://ftp.gnu.org/gnu/gcc/gcc-$GCC_VERSION/gcc-$GCC_VERSION.tar.gz \
    && tar -xzf gcc-$GCC_VERSION.tar.gz \
    && rm gcc-$GCC_VERSION.tar.gz

# Build and install GCC
RUN cd gcc-$GCC_VERSION \
    && ./contrib/download_prerequisites \
    && mkdir build \
    && cd build \
    && ../configure --prefix=$PREFIX --enable-languages=c,c++,fortran --disable-multilib

RUN yum install -y make

RUN cd gcc-$GCC_VERSION/build \
    && make -j$(nproc) \
    && make install \
    && cd / \
    && rm -rf /tmp/*

RUN g++ --version
RUN gfortran --version
RUN gcc --version

RUN source /opt/intel/oneapi/setvars.sh; ifort -v; icc -v; icpc -v

ENV PKG=/opt/rh/rh-git218/root
ENV PATH=${PKG}/usr/bin:$PATH
ENV LD_LIBRARY_PATH=${PKG}/usr/lib64:${PKG}/usr/lib:${LD_LIBRARY_PATH}
ENV MANPATH=${PKG}/usr/share/man:$MANPATH
RUN git --version
RUN git-lfs --version

ENV PKG=/opt/rh/httpd24/root
ENV PATH=${PKG}/usr/bin:$PATH
ENV LD_LIBRARY_PATH=${PKG}/usr/lib64:${PKG}/usr/lib:${LD_LIBRARY_PATH}
ENV MANPATH=${PKG}/usr/share/man:$MANPATH
RUN curl --version

WORKDIR /tmp
RUN curl -LO https://www.openssl.org/source/openssl-1.1.1w.tar.gz \
    && tar -xzvf openssl-1.1.1w.tar.gz \
    && cd openssl-1.1.1w \
    && ./config --prefix=/usr/local --openssldir=/usr/local shared \
    && make \
    && make install

ENV LD_LIBRARY_PATH=/usr/local/lib64:${LD_LIBRARY_PATH}

RUN openssl version

WORKDIR /tmp
ENV PYTHON_VERSION=3.12.3

RUN yum install -y zlib-devel \
    && curl -LO https://www.python.org/ftp/python/${PYTHON_VERSION}/Python-${PYTHON_VERSION}.tgz
    && tar -xzf Python-${PYTHON_VERSION}.tgz \
    && mkdir build; cd build \
    && ../Python-${PYTHON_VERSION}/configure \
         --prefix=/usr \
         --with-openssl=/usr/local \
         --enable-shared \
         --with-ensurepip=install \
    &&  ldconfig

RUN cd /tmp/build \
    && make -j$(nproc) \
    && make install \
    && python3 --version \
    && cd /tmp; rm -rf /tmp/*

ENV LD_LIBRARY_PATH=/usr/lib:${LD_LIBRARY_PATH}

RUN python3 --version

RUN python3 -m pip install requests

RUN python3 -c "import requests"

WORKDIR /tmp

# Download CMake 3.24.3 source and install
```

Appendix A - Dockerfiles

```
ENV CMAKE_VERSION=3.24.3
ENV CMAKE_ROOT=/usr/local/cmake-${CMAKE_VERSION}
RUN curl -LO https://github.com/Kitware/CMake/releases/download/v${CMAKE_VERSION}/cmake-${CMAKE_
RUN tar -xf cmake-${CMAKE_VERSION}.tar.gz \
    && cd cmake-${CMAKE_VERSION} \
    && ./configure --prefix=${CMAKE_ROOT} -- -DCMAKE_USE_OPENSSL=OFF \
    && make -j $(nproc) \
    && make install

ENV PATH="${CMAKE_ROOT}/bin:$PATH"
RUN cmake --version

# Download and install openmpi 4.1.1
ENV OPENMPI_VERSION=4.1.1
ENV OPENMPI_DIR=/usr/local/openmpi-${OPENMPI_VERSION}
WORKDIR /tmp

RUN curl -LO https://download.open-mpi.org/release/open-mpi/v4.1/openmpi-${OPENMPI_VERSION}.tar.
RUN tar -xf openmpi-${OPENMPI_VERSION}.tar.gz \
    && rm openmpi-${OPENMPI_VERSION}.tar.gz \
    && cd openmpi-${OPENMPI_VERSION} \
    && ./configure --prefix=${OPENMPI_DIR} \
    && make -j$(nproc) >/dev/null \
    && make install >/dev/null \
    && cd / \
    && rm -rf /tmp/*

# Set environment variables for OpenMPI
ENV PATH="${OPENMPI_DIR}/bin:${PATH}"
ENV LD_LIBRARY_PATH="${OPENMPI_DIR}/lib64:${LD_LIBRARY_PATH}"
ENV LD_LIBRARY_PATH="${OPENMPI_DIR}/lib:${LD_LIBRARY_PATH}"
ENV MANPATH=${OPENMPI_DIR}/share/man:${MANPATH}
ENV C_INCLUDE_PATH=${OPENMPI_DIR}/include

# Verify OpenMPI installation
RUN mpicc --version && mpicxx --version && mpifort --version

# Download and install HDF5
WORKDIR /tmp
ENV HDF5_VERSION=1.14.2
ENV HDF5_DIR=/usr/local/hdf5-${HDF5_VERSION}

RUN curl -LO https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.14/hdf5-${HDF5_VERSION}/src/h

RUN tar -xf hdf5-${HDF5_VERSION}.tar.gz \
    && rm hdf5-${HDF5_VERSION}.tar.gz \
    && mkdir build \
    && cd build \
    && cmake ../hdf5-${HDF5_VERSION} \
       -DCMAKE_INSTALL_PREFIX=${HDF5_DIR} \
       -DCMAKE_BUILD_TYPE=Release \
       -DHDF5_BUILD_TOOLS=ON \
       -DHDF5_ENABLE_PARALLEL=OFF \
       -DHDF5_ENABLE_Z_LIB_SUPPORT=ON \
       -DHDF5_BUILD_EXAMPLES=OFF \
    && cmake --build . --parallel $(nproc) \
    && cmake --install . >/dev/null \
    && cd / \
    && rm -rf /tmp/*

# Set environment variables for HDF5
ENV PATH="${HDF5_DIR}/bin:${PATH}"
ENV LD_LIBRARY_PATH="${HDF5_DIR}/lib:${LD_LIBRARY_PATH}"
ENV LD_LIBRARY_PATH="${HDF5_DIR}/lib64:${LD_LIBRARY_PATH}"
ENV MANPATH=${HDF5_DIR}/share/man:${MANPATH}
ENV C_INCLUDE_PATH=${HDF5_DIR}/include

WORKDIR /tmp
ENV LYX_VERSION=16660d12
ENV LYX_ROOT=/usr/local/lyx-${LYX_VERSION}

ENV CC=/usr/local/gcc-11.3.0/bin/gcc
ENV CXX=/usr/local/gcc-11.3.0/bin/g++
RUN yum install -y qt5-qtbase-devel qt5-qtsvg-devel \
    && git clone https://git.lyx.org/repos/lyx.git \
    && cd lyx; git reset --hard ${LYX_VERSION}; cd .. \
    && mkdir build \
    && cd build \
    && cmake ../lyx -DCMAKE_INSTALL_PREFIX=${LYX_ROOT} -DLYX_INSTALL=ON \
    && cmake --build . --parallel $(nproc) >/dev/null \
    && cmake --install . >/dev/null \
    && cd / \
    && rm -rf /tmp/* \
    && yum clean all \
    && cd ${LYX_ROOT}/bin \
    && ln -s ./lyx2.4 ./lyx

ENV PATH=${LYX_ROOT}/bin:${PATH}
ENV MANPATH=${LYX_ROOT}/share/man:${MANPATH}
RUN lyx --version

# Install ninja for Qt6
```

```
WORKDIR /tmp
RUN curl -LO https://github.com/ninja-build/ninja/archive/master.tar.gz \
    && tar -xf master.tar.gz \
    && cd ninja-master \
    && ./configure.py --bootstrap \
    && mv ninja /usr/local/bin \
    && rm -rf /tmp/*

RUN ninja --version

RUN yum install -y \
       dbus at-spi2-atk at-spi2-core \
       mesa-libGLU-devel libXrender-devel libX11-devel \
       libXi-devel \
       libxkbcommon-devel libxkbcommon-x11-devel \
       *xcb* \
       libfontconfig1-dev \
       libfreetype6-dev

ENV QT_VERSION=6.3.2
ENV QTDIR=/usr/local/qt-${QT_VERSION}
WORKDIR /tmp
RUN curl -LO https://download.qt.io/archive/qt/6.3/${QT_VERSION}/single/qt-everywhere-src-${QT_V
    && cmake --version \
    && tar -xf qt-everywhere-src-${QT_VERSION}.tar.xz \
    && rm qt-everywhere-src-${QT_VERSION}.tar.xz \
    && mkdir build \
    && cd build \
    && ../qt-everywhere-src-${QT_VERSION}/configure \
          -prefix ${QTDIR} \
          -opensource -confirm-license \
          -release \
          -skip qtwebengine \
          -no-icu \
          -xcb \
          -nomake examples \
          -nomake tests \
    && cmake --build . --parallel $(nproc) \
    && cmake --install . >/dev/null \
    && cd / \
    && rm -rf /tmp/* \
    && yum clean all

# Reset environment variables for Qt
ENV PATH="${QTDIR}/bin:${PATH}"
ENV LD_LIBRARY_PATH="${QTDIR}/lib:${LD_LIBRARY_PATH}"
ENV LD_LIBRARY_PATH="${QTDIR}/lib64:${LD_LIBRARY_PATH}"
ENV MANPATH=${QTDIR}/share/man:${MANPATH}
ENV C_INCLUDE_PATH=${QTDIR}/include:${C_INCLUDE_PATH}

# Download CMake 3.28.3 source and install
WORKDIR /tmp
ENV CMAKE_VERSION=3.28.3
ENV CMAKE_ROOT=/usr/local/cmake-${CMAKE_VERSION}
ENV PATH="${CMAKE_ROOT}/bin:$PATH"
RUN curl -LO https://github.com/Kitware/CMake/releases/download/v${CMAKE_VERSION}/cmake-${CMAKE_
RUN tar -xf cmake-${CMAKE_VERSION}.tar.gz \
    && cd cmake-${CMAKE_VERSION} \
    && ./configure --prefix=${CMAKE_ROOT} -- -DCMAKE_USE_OPENSSL=OFF \
    && make -j $(nproc) \
    && make install \
    && cd /tmp \
    && rm -rf ./* \
    && cmake --version

RUN python3 -m pip install h5py setuptools matplotlib

RUN adduser mcnp-developer \
    && usermod -a -G wheel mcnp-developer \
    && echo '%wheel ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers \
    && yum clean all

SHELL ["/bin/bash","-l","-c"]
WORKDIR /home/mcnp-developer
RUN echo "source /opt/intel/oneapi/setvars.sh" >> .bashrc
USER mcnp-developer
ENV HDF5_USE_FILE_LOCKING=FALSE
ENV OMP_STACKSIZE=256M
ENV FC=ifort
ENV CC=icc
ENV CXX=icpc
```