ARCHER2-eCSE02-13 Technical Report
Containerisation of NEMO Employing Singularity (CoNES)

James Harle[1] and Chris Wood[2]
[1] National Oceanography Centre, Southampton. jdha@noc.ac.uk
[2] EPCC, University of Edinburgh

Abstract

The traditional method for running an Ocean General Circulation Model (OGCM) on High Performance Computing (HPC) platforms is to compile the source code using the native libraries on the host machine. The ease at which this task is accomplished will depend on the user's knowledge of the HPC platform and the availability of the required libraries. The resulting OGCM executable will be unique to the HPC platform it was compiled on (and quite possibly to the user who compiled it). In the case of the NEMO OGCM, there is, in addition to a large user community, a dispersed developer community, each relying on their local HPC platform. By introducing containerisation to the build process of applications run on HPC platforms, one can create a reproducible environment in which these codes can be developed and executed. We use the open-source Singularity software to develop methods of containerisation for the NEMO OGCM. Several approaches are explored and tested on the ARCHER2 UK National HPC facility.

Keywords: ARCHER2, eCSE, NEMO, Singularity, Containers, Ocean Modelling

Introduction

Containerisation has become an increasingly popular method for software deployment, especially in cloud computing environments. Over the last few years containerisation has also been gaining traction in the High Performance Computing (HPC) community with its benefits of flexibility and reproducibility. By deploying a container image on an HPC platform the traditional methods of: transferring code, setting up, debugging, error resolution and library conflict/installation are avoided. As discussed in [1] containerised computing provides a convenient application for the scientist, minimising any adaptation required to the HPC environment. It becomes possible to move between environments and scale up or down in a consistent and reproducible manner e.g. from a local cluster to a national HPC facility, or even the cloud, with a different run-time environment. In addition, exotic applications that are not traditionally supported on HPC platforms can also be accommodated through the use of containers. In an effort to make reproducible scientific results we are increasing making use of code repositories and official data archives. With the use of containers, peers can reproduce the entire package. The container provides the means to reconstruct the exact runtime environment the model code was executed in. This concept of full reproducibility not only makes it easier to appraise or build on scientific studies, but it can also benefit the development cycle of the code. Another benefit of containerisation is its use as a teaching aid or for outreach, where knowledge of compilers and machine architectures is limited; it provides a packaged solution accelerating the setup process, allowing the focus to be on the science.

At present there are few examples of software containers being employed in the ocean modelling community. A recent study by Cheng et al. [2]: System for High-resolution

prediction on Earth-to-Local Domains (SHiELD), looked at the containerisation of a unified atmospheric model for weather-to-seasonal prediction using both Docker and Singularity. They discuss many of the benefits of containerisation we have just outlined. One of the other points they raise is that the container may also be used to release software whilst protecting innovative schemes, and therefore intellectual property. Encapsulating a scientific code or application (along with their dependencies) as a container image provides an attractive, consistent and lightweight method for their efficient deployment on a wide variety of computer systems.

In this report we present a method for the containerisation of the Nucleus for European Modelling of the Ocean (NEMO) framework and explore the possible benefits that this approach may have for the ocean modelling community. We begin with an overview of the NEMO framework and the Singularity container software. Next, we outline the approach taken in deploying a series of test containers on the UK's ARCHER2 national HPC facility. We then evaluate the performance of the containerised code versus the *bare metal* equivalent. Finally, we summarise the current landscape of container technologies and the merits of employing containers for the running Ocean General Circulation Model (OGCM) simulations.

Framework

NEMO

The Nucleus for European Modelling of the Ocean (NEMO) is a framework for ocean and climate modelling [3]. The ocean component of NEMO is a primitive equation model employed for a range of idealised, regional and global ocean circulation studies. It provides a flexible tool for studying the ocean and the wider earth climate system over a wide range of space and time scales [4]. At present, standard global OGCM configurations using NEMO are of the horizontal resolution of 1/12^{o} and 75 levels in the vertical. This roughly equates to five trillion active grid cell and is typically run on O(1000) cores. Regional OGCM simulations, on the whole, consume more modest resource. A workhorse of the National Oceanography Centre, UK, is the Atlantic Margin Model and has a horizontal resolution of 7km (AMM7). This has around 3 million active grid cells and can be comfortably accommodated on O(100) cores. Coupled to NEMO is XIOS, a standalone application that manages the diagnostic outputs. For the purpose of this study XIOS is treated as part of the NEMO framework.

Singularity

Like many of the other containers available, Singularity was developed to provide a reproducible and mobile environment in which to run applications (see Kurtzer et al [5] for full details). One of the main uses of Singularity is to provide a means of containerisation to the scientific computing and HPC communities. The open source code is designed to run as a non-privileged user (root-less) and does not have a persistent control daemon. While previous geophysical studies employing containers have made use of Docker, e.g. [6], the use of Singularity has benefits in a HPC environment where the scientist will almost likely not have root permissions. In addition, the container image can be encrypted to secure applications, models, and data, and there is also support from public/private key signing and guarantees of immutability. From a practical standpoint the use Singularity allows the reproducible build, share, and archive of applications, models and data from workstations, to HPC, to the edge, securely leveraging GPUs, FPGAs, high-speed networks, and filesystems [7].

While Singularity can build containers in several different formats the default is Singularity Image Format (SIF) files and are compressed and immutable. This file contains prebuilt applications and dependencies, in addition to any other files and data, that are to be used by a Singularity container to construct a runtime environment and run the application.

Application

There is no set way to approach the containerisation and will be, ultimately, guided by the application and/or end goal. In addition, the SIF file can be relatively lightweight, containing only the executable and therefore requiring additional libraries, data and inputs to be *bound* to the container at runtime. For this study we explore the containerisation of the NEMO application and all its dependencies, *binding* the AMM7 configuration and forcing files to the container at runtime. We also explore the *Hybrid* and *Bind* approach to message passaging when deploying containers. During the build process NEMO is compiled with a choice of MPI libraries. When the container is deployed in *Hybrid* mode the MPI processes outside to the container are handle by the host, working in conjunction with the MPI in the container. In *Bind* mode the container relies solely on the host's MPI libraries, which are bound to the container at runtime.

Approach

While Singularity container images are often talked about as *portable*, i.e. infrastructure agnostic, in reality the interconnect and MPI implementation is specific to each HPC system prevents such generality. In order for the container application to be deployed, it must either have the same MPI libraries as the host or at the very least compatible application binary interfaces (ABI). In the case of the ARCHER2 system the MPICH and openMPI libraries are available by default; while openMPI is commonly used it is not part the MPI ABI initiative, whereas MPICH is. The ARCHER2 system has been configured to maximise performance using the MPICH software. We chose to implement both an openMPI and MPICH version of the container image, in order to cover the needs of most of the NEMO community.

Our approach is two step for efficiency. The first is to build a SIF file of the dependencies as closely aligned to the host system as possible. The second it to then use this as the *bootstap agent* in building the NEMO SIF file. The first SIF file is relatively large and time invariant, so it only has to be built once. The second step is lightweight and may have to be recompiled several times depending on the user's needs.

In this study we are only building a SIF file of the NEMO application. All the configuration files and input data required to run OGCM are external and not included in the SIF file. That is not to say they that they couldn't be. However, we chose this approach to demonstrate that different configurations can by mounted at runtime allowing the user a flexible approach, whilst rendering the core application immutable. If we were publishing a scientific report, then including the configuration (and possibly the input data) in the SIF file may be a more desirable approach to take.

Build Process

The Singularity *definition file* provides a recipe to build the reproducible SIF file. It contains information about the base OS, software to compile and environment setup. The following

stripped-down example illustrates the process of building a NEMO/XIOS SIF. In practice a two-step build was employed in the CoNES project. The first step is to *containerise* the OS and libraries required to build NEMO/XIOS. The second is to bootstrap that SIF setup in the build process of the NEMO/XIOS SIF file. This can then be tidied, removing any redundant build files and system libraries and reducing the size of the container image. Here, we present an overview of simplified key points:

The header of the first definition file specifies the *bootstrap agent* used to create the base image. In this instance the OS built is Ubuntu 20.4.

```
Bootstrap: library
From: ubuntu:20.04

%files
    input_files/NEMO_in /input_files/NEMO_in
```

The next section, `%files`, lists the external files on the host system requires to build the SIF file. This is a simple *namelist* file, `NEMO_in`, which provides variables that allow the user to customise the build process:

```
MY_SRC=                         # If blank no need to do anything
NEMO_VERSION=4.0.4              # Check that VERSION is 4.0.[2-6], 4.0_HEAD or trunk
XIOS_REVISION=                  # Use default value if empty
NEMO_COMPONENTS='OCE'           # Which NEMO components to build OCE/ICE/TOP etc
CPP_KEYS=                       # Any additional compiler keys to include?
MPI=                            # Which MPI implementation to use MPICH | OMPI
```

In the `%post` section, the base OS is defined along with mandatory binaries. Any relevant dependencies not available via `apt-get` (MPI, HDF5 and netCDF) are built from source. The following is truncated for brevity:

```
%post

    ##
    # Install apt-get binaries, build necessary dependencies
    ##

    apt install -y locales #locales-all
    locale-gen en_GB en_GB.UTF-8 # en_US en_US.UTF-8

    apt install -y software-properties-common
    add-apt-repository universe
    apt update

    apt install -y python \
...
    if [ "$MPI" = "MPICH" ]
    then

         apt install -y libfabric-dev

         wget http://www.mpich.org/static/downloads/3.4.2/mpich-3.4.2.tar.gz
         tar -xvzf mpich-3.4.2.tar.gz -C mpi --strip-components 1
         rm mpich-3.4.2.tar.gz
         cd mpi

         ./configure CC=gcc CXX=g++ FC=gfortran --prefix=/opt/mpi/install
         make
```

```
        make install

    elif [ "$MPI" = "OMPI" ]
    then
...
```

Next the `%environment` section defines the path to the HDF libraries required by the container at runtime.

```
%environment

    export LD_LIBRARY_PATH=/opt/hdf5/install/lib:$LD_LIBRARY_PATH
```

Once the first definition file (`Singularity.nemo_baseOS`) has be written, building the SIF file is trivial:

```
sudo singularity build nemo_baseOS.sif Singularity.nemo_baseOS
```

To now build the NEMO SIF file a second definition file (`Singularity.nemo`) is written using the first image as the *bootstrap* agent:

```
Bootstrap: localimage
From: nemo_baseOS.sif

%files
    input_files/NEMO_in /input_files/NEMO_in
    input_files/MY_SRC.tar.gz /input_files/MY_SRC.tar.gz
    input_files/setup_nemo /input_files/setup_nemo
    input_files/arch_files /input_files/arch/nemo/arch-files
```

We now include several other input files: `MY_SRC.tar.gz` contains any updated source files required to build NEMO; `setup_nemo` is the NEMO/XIOS build script, which checks out the source code and builds NEMO/XIOS using the `arch_files` compiler directives for the container environment.

Finally, NEMO and XIOS are compiled using the previously imported setup script from `%files`:

```
%post

    # compile NEMO/XIOS
    /input_files/setup_nemo -x /nemo -w /nemo -m singularity -v $NEMO_VERSION
```

Finally, `%runscript` defines the action taken when the container is executed. As both NEMO and XIOS have been built, there are checks to see which is required.

```
%runscript

    if [[ $1 == 'nemo' ]]
    then
        /opt/nemo/nemo
    else
        /opt/xios/xios
    fi
```

As before we can now build the NEMO SIF file:

```
sudo singularity build nemo.sif Singularity.nemo
```

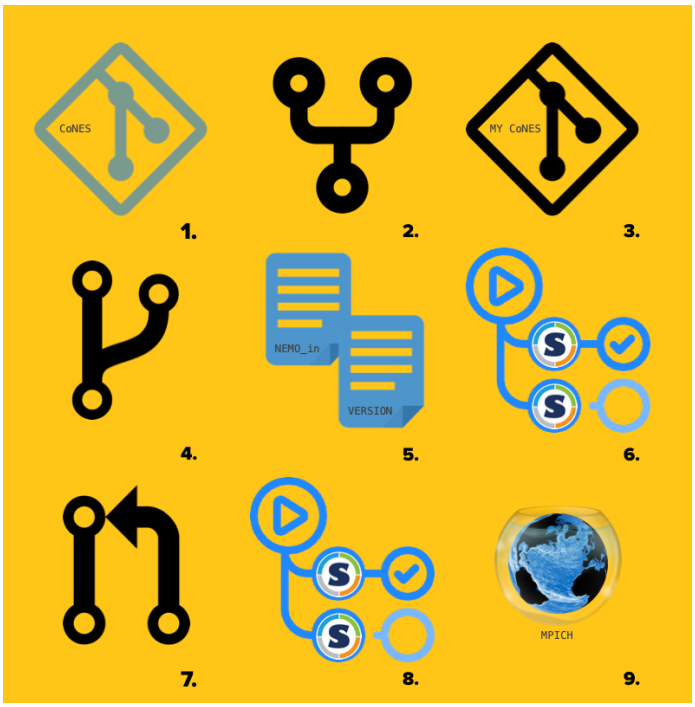
Which can now be deployed on our host HPC system.

Most HPC clusters now support the use of Singularity, while Linux and OSX also have native builds. Windows architectures require Singularity to run within a Virtual Machine. The command requires `sudo` just as installing software on your local machine requires *root* privileges. If this is not an option the SIF file can either be built as *fakeroot* on the host system, or alternatively, for example, via GitHub Actions.

GitHub Actions Build

To improve the access to users not familiar with building containers, alternative workflows may be employed. We demonstrate one such workflow through the use of GitHub Actions [8]. If building locally is not an option, then it is possible to build and release Singularity containers on GitHub. Singularity Deploy developed by Vanessa Sochat [9] has been modified to allow users to fork the GitHub CoNES repository and, using GitHub Actions, build and release a bespoke NEMO singularity container in much the same manner as described previously. The build process takes order 15 minutes which is comparable to local builds. An individual NEMO SIF file build can be created using the following steps:

1. Navigate to the CoNES GitHub repository (github/NOC-MSM/CoNES)
2. Fork the repository into:
3. Your own GitHub repository.
4. Create and checkout a new branch.
5. Edit the `NEMO_in` and `VERSION` files according to your requirements, opening a pull request at the same time.
6. A GitHub Actions test is triggered to check the container can be successfully built.
7. Merge back to main/master.
8. GitHub Actions build is triggered.
9. A container is published under the repository's Releases.



As *root* and *fakeroot* access were not permitted on the ARCHER2 HPC facility the NEMO SIF files were either built using the command line approach on local clusters or workstations, or by employing the GitHub Actions approach.

Experimental Setup

At the time of the study there were two supported compilers on the ARCHER2 system: Cray and GNU, with the choice of two transfer protocols (OFI, UCX) for the available MPI libraries: MPICH and openMPI. With the time and compute resource available we conducted the following array of experiments using the NEMO AMM7 configuration:

Bare-Metal
- Cray/MPICH/OFI
- GNU/MPICH/UCX
- GNU/openMPI/UCX

SIF (Hybrid MPI)
- GNU/MPICH
- GNU/MPICH/UCX
- GNU/MPICH/OFI
- GNU/openMPI/UCX

SIF (Bind MPI)
- GNU/MPICH/ OFI
- GNU/openMPI/UCX

Each were performed at various levels of optimisation (O0-O3) and over a range of core counts (25-711). As the system openMPI netCDF/HDF libraries did not appear to work, a fresh install had to be built on the host for the bare-metal tests. Each experiment was run for a model month and timing outputs taken directly from the model's internal diagnostics. The runscripts used in submission to the workload schedular can be found in the project repository [10].

Performance

As a fixed NEMO configuration with which to run the experimental ensemble was chosen, we are limited to reporting the performance only in terms of the strong scaling. We were unable to perform simulations on a single core, so performance is measured relative to 25 cores. We performed several experiments to evaluate the computational performance of the NEMO containers against their bare metal counterparts.

Figure 1 shows the strong scaling for GNU-MPICH-OFI Hybrid container, across all levels of optimisation and a range of core counts. This figure is very similar in form to that of the other container and bare metal members. In general, at optimisation level O1 or greater, the AMM7 simulation runs at between at around 10 hours per model year on 25 cores, dropping to around 15 minutes per model year on 711 cores. There is a degree of *super-scaling* over the lower core counts. It is unclear as to why this happens, but the results are consistent across all ensemble members.

Figure 1 Strong Scaling: GNU-MPICH-OFI-Hybrid. Red-orange axis/lines show speed relative to 25 cores (black line being the ideal). O0-O3 are the levels of code optimisation. Blue-green lines show the number of hours to complete 1 model year of simulation.

To give an overview of all ensemble members at O1 optimisation we present a performance matrix in Figure 2a. The relative performance of each simulation is measured against the bare-metal GNU-MPICH-UCX member. At this optimisation level the bare-metal Cray-MPICH-OFI is the best performing member across all core counts. At higher core counts the percentage gain in performance generally increases. The Hybrid containers are within 5-10% of their bare-metal counterparts, although the GNU-MPICH Hybrid container exhibits a significant performance drop as core placement moves to multiple nodes. While the performance of the GNU-MPICH-OFI Bind container is similar to that of its Hybrid equivalent, the GNU-openMPI-UCX Bind container mimics more its bare-metal counterpart, suffering at higher core counts.

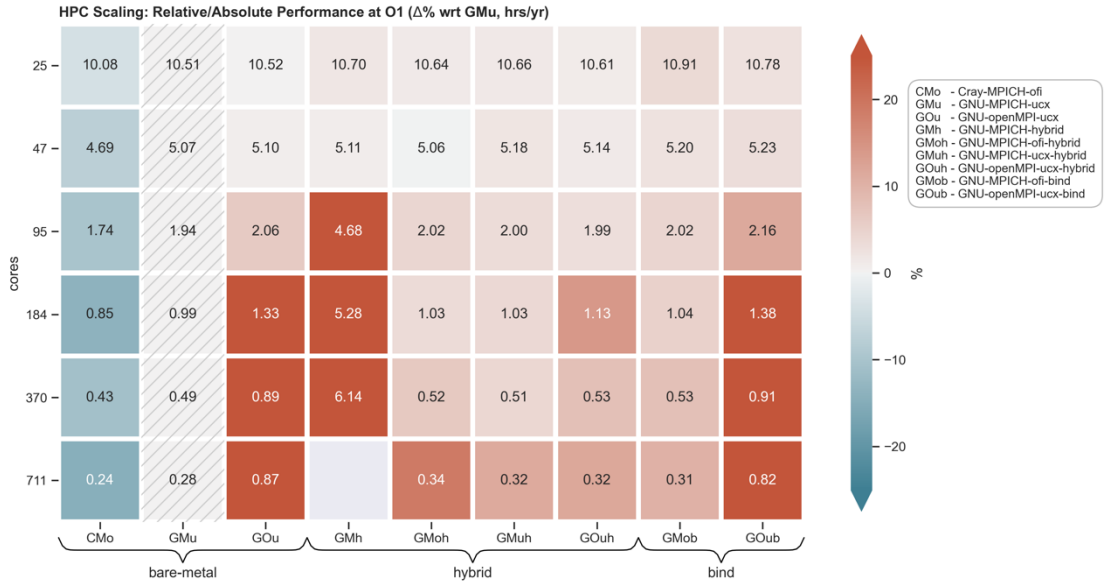


Figure 2a Relative/Absolute performance of each experiment at optimisation level O1. Performance gain is relative to bare-metal GNU-MPICH-UCX. Values in each box give the number of hours to complete one model year of simulation.

At O3 optimisation (Figure 2b) the bare-metal Cray-MPICH-OFI performance is surprisingly reduced relative to the O1 optimisation at higher core counts. Overall, the bare-metal GNU-MPICH-UCX member has the best performance. The NEMO containers are around 10% slower than the reference bare-metal member, but interestingly comparable to bare-metal Cray-MPICH-OFI at higher core counts. The bare-metal GNU-openMPI-UCX and GNU-openMPI-UCX Bind container suffer at higher core counts as in the O1 optimisation case.

Figure 2b Relative/Absolute performance of each experiment at optimisation level O3. Performance gain is relative to bare-metal GNU-MPICH-UCX. Values in each box give the number of hours to complete one model year of simulation.

Discussion

In this study we have detailed an approach to containerise the NEMO OGCM. It is by no means the only approach, but it provides the most flexibility for the user. By separating out the application and configuration, the user has the option to modify the parameter settings of a simulation while the code itself remains immutable. However, if a container were to be published as part of a scientific study, it may be desirable to include the configuration, and even forcing data and postprocessing tools, within the container image to maintain strict reproducibility.

The performance of the containers is marginally slower than their bare-metal counterparts. ARCHER2 is an HPE Cray EX supercomputing system with fully optimised compilers, parallel libraries and interconnects; it is therefore unsurprising that bare metal cases perform better. Although it is interesting that, at O3 optimisation, the performance of the bare metal Cray-MPICH-OFI drops off at higher core counts, and is comparable to the containerised members. It is also unclear as to why there is a slowdown of the bare-metal GNU-openMPI-UCX and GNU-openMPI-UCX Bind members at high core counts. This would suggest an issue with the host implementation of openMPI, although the GNU-openMPI-UCX Hybrid member appears to be unaffected by this.

In terms of the different methods of message passing when using containers, the Hybrid approach works well as long as the MPI within the container is compiled with a compatible transfer protocol backend. This is demonstrated in the slow down observed with the GNU-MPICH Hybrid member when the core count spans more than a single node. While the GNU-MPICH-OFI Bind member has similar performance as the Hybrid members, there is a little more overhead in defining the runscript for this method. Paths to the MPI installation on the host must be identified and bound to the correct directories in the container. In contrast the Hybrid approach it is very similar to running native MPI applications. In both cases the MPI used to compile the application in the container must be compatible with the host's MPI implementation. The only perceivable benefit of the Bind method is that it reduces the SIF file size, but only marginally.

In addition to the performance tests presented, the level of output from the application was also assessed. It was shown to have very little impact on the performance of the containers as did the size of the SIF file itself. In the initial phase of the study, we employed a one-step build process with very little house keeping. The original SIF files were of the order 300-400MB; the final, slimmed down versions, being under 100MB.

For most applications the ease of use, portability and reproducibility benefits of containerisation will outweigh the small performance hit. The performance may well be increased further with optimisation of the container build and runtime options, something we did not have time to explore. However, for large complex one-off production runs the fully optimised and tuned bare metal case will still be desirable. In the context of NEMO, the users will have to weigh up the merits of containerisation versus highly optimised native compiled source code on a case-by-case basis. As part of a development cycle the use of a container facilitates the portability of the code. During a maintenance period on the ARCHER2 system, we were able to transfer, setup and run the AMM7 NEMO container on the National Oceanography Centre's local HPC cluster with minimal overhead. The local cluster only had

IntelMPI available, but due to the MPI ABI we were able to run the GNU-MPICH-UCX Hybrid/Bind members without intervention.

The AMM7 configuration, in hindsight, is too small to fully test scalability of the containerised NEMO code. Cheng et al. [2], reported performance drop off at very large core counts, something we were unable to confirm. In order to fully evaluate performance, with both strong and weak scaling, over a wide range of core counts, NEMO's GYRE configuration should be employed.

Presentations

The CoNES project has been presented to the following:

- National Oceanography Centre's Marine Systems Modelling Group
- Plymouth Marine Laboratory's Biogeochemical Modelling Group
- NEMO System Team
- Informally at EPCC
- RSECon22 [11]
- UK Met Office
- NEMO developers party

Code Availability

The code to generate NEMO Singularity containers currently resides on GitHub: https://github.com/NOC-MSM/CoNES. The code referenced in this report has also been tagged and released as 10.5281/zenodo.8305515.

Acknowledgements

This work was funded under the embedded CSE programme of the ARCHER2 UK National Supercomputing Service (http://www.archer2.ac.uk). The authors thank Michael Bareford and Andy Turner for their input and feedback during the course of this project.

References

[1] Melton, J. R., Arora, V. K., Wisernig-Cojoc, E., Seiler, C., Fortier, M., Chan, E., and Teckentrup, L.: CLASSIC v1.0: the open-source community successor to the Canadian Land Surface Scheme (CLASS) and the Canadian Terrestrial Ecosystem Model (CTEM) – Part 1: Model framework and site-level performance, Geosci. Model Dev., 13, 2825–2850, https://doi.org/10.5194/gmd-13-2825-2020, 2020.

[2] Cheng, K.-Y., Harris, L. M., and Sun, Y. Q.: Enhancing the accessibility of unified modeling systems: GFDL System for High-resolution prediction on Earth-to-Local Domains (SHiELD) v2021b in a container, Geosci. Model Dev., 15, 1097–1105, https://doi.org/10.5194/gmd-15-1097-2022, 2022.

[3] www.nemo-ocean.eu (last accessed 23/08/23)

[4] Madec G, Bourdallé-Badie R, Chanut J, Clementi E, Coward A, Ethé C, et al. NEMO ocean engine. 2019 Oct 24 [cited 2022 Apr 20]; Available from: https://zenodo.org/record/3878122

[5] Kurtzer GM, Sochat V, Bauer MW. Singularity: Scientific containers for mobility of compute. PLoS One. 2017 May 11;12(5):e0177459. doi: 10.1371/journal.pone.0177459. PMID: 28494014; PMCID: PMC5426675.

[6] Hacker, J. P., J. Exby, D. Gill, I. Jimenez, C. Maltzahn, T. See, G. Mullendore, and K. Fossell, 2017: A Containerized Mesoscale Model and Analysis Toolkit to Accelerate Classroom Learning, Collaborative Research, and Uncertainty Quantification. *Bull. Amer. Meteor. Soc.*, **98**, 1129–1138, https://doi.org/10.1175/BAMS-D-15-00255.1.

[7] https://apptainer.org (last accessed 23/08/23)

[8] https://github.com/features/actions (last accessed 23/08/23)

[9] https://github.com/singularityhub/singularity-deploy (last accessed 23/08/23)

[10] https://github.com/NOC-MSM/CoNES (last accessed 23/08/23)

[11] https://virtual.oxfordabstracts.com/#/event/public/3101/submission/89 (last accessed 23/08/23)