Project Title	Hybrid Eco Responsible Optimized European Solution
Project Acronym	HEROES
Grant Agreement No.	956874
Start Date of Project	01.03.2021
Duration of Project	24 Months
Project Website	heroes-project.eu

D2.2 Workflows containers

Work Package	**D2.2, Workflow containers**
Lead Author (Org)	**Jose E. Torres (HPCNow!)**
Contributing Author(s) (Org)	**Jose E. Torres (HPCNow!)**
Reviewed by	**Davide Pastorino (DoIT)** **Emanuele Viale (DoIT)** **Jorik Remy (UCit)** **Sablin Amon (UCit)**
Approved by	**Management Board**
Due Date	**M12**
Date	**08/04/2022**
Version	**V1.4**

Dissemination Level

X	PU: Public
	PP: Restricted to other programme participants (including the Commission)
	RE: Restricted to a group specified by the consortium (including the Commission)
	CO: Confidential, only for members of the consortium (including the Commission)

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

Versioning and contribution history

Version	Date	Author	Notes
0.1	01/02/2022	Jose E Torres (HPCNow!)	Start document
0.2	02/02/2022	Jose E Torres (HPCNow!)	Created document structure
0.3	03/02/2022	Jose E Torres (HPCNow!)	Added draft information
0.4	09/02/2022	Jose E Torres (HPCNow!)	Added draft information
0.5	15/02/2022	Jose E Torres (HPCNow!)	Added draft information
0.6	16/02/2022	Jose E Torres (HPCNow!)	Edited sections
0.7	17/02/2022	Jose E Torres (HPCNow!)	Edited sections
0.8	22/02/2022	Jose E Torres (HPCNow!)	Added code examples
0.9	23/02/2022	Jose E Torres (HPCNow!)	Edited sections
1.0	24/02/2022	Jose E Torres (HPCNow!), Pierre Puigdomenech (HPCNow!)	Review and added comments, edited sections
1.1	25/02/2022	Jose E Torres (HPCNow!), Davide Pastorino (DoIt), Benjamin Depardon (UCit)	Review and added comments
1.2	28/02/2022	Jose E Torres (HPCNow!), Davide Pastorino (DoIt)	Review and added comments
1.2	01/03/2022	Jose E Torres (HPCNow!)	Edited sections
1.4	02/03/2022	Jose E Torres (HPCNow!)	Completed references and List of Figures
1.5	03/03/2022	Jose E Torres (HPCNow!)	Review corrections
1.6	04/03/2022	Jose E Torres (HPCNow!)	Review corrections
1.7	08/03/2022	Sablin Amon (HPCNow!)	Review
1.8	09/03/2022	Sablin Amon (HPCNow!)	Review
1.9	11/03/2022	Benjamin Depardon (UCit), Sablin Amon (UCit)	Review
2.0	14/03/2022	Jose E Torres (HPCNow!)	Review corrections

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

2.1	15/03/2022	Jose E Torres (HPCNow!)	Review corrections
2.2	16/03/2022	Jose E Torres (HPCNow!)	New Section + Review corrections
2.4	23/03/2022	Jose E Torres (HPCNow!)	Complemented sections + reviewed corrections.
2.5	24/03/2022	Jorik Remy (UCit)	Review
2.6	25/03/2022	Jose E Torres (HPCNow!)	Updated image
3.0	08/04/2022	Corentin Lefèvre (Neovia)	Final version approved by the Management Board
3.0	23/05/022	Alysée Cibil (Neovia)	Modified version (disclaimer changed) following the comments of the mid-term review

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

Table of Contents

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

List of Figures

List of Tables

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

References and Applicable Documents

[1] HEROES Project website, https://heroes-project.eu/

[2] Containerization Explained, IBM Website, https://www.ibm.com/cloud/learn/containerization

[3] Singularity, https://sylabs.io/singularity/

[4] Docker, https://www.docker.com/

[5] MPI, Message Passing Interface Wikipedia, https://en.wikipedia.org/wiki/Message_Passing_Interface

[6] Open MPI, Open MPI Project website, https://www.open-mpi.org/

[7] MPICH, MPICH Project website, https://www.mpich.org/

[8] ORTED, orted man page, https://www.open-mpi.org/doc/v4.0/man1/orted.1.php

[9] Building Singularity Images, https://sylabs.io/guides/2.6/user-guide/build_a_container.html

[10] Singularity Definition files, https://sylabs.io/guides/2.6/user-guide/container_recipes.html

[11] Building Singularity images, https://sylabs.io/guides/2.6/user-guide/build_a_container.html

[12] Docker and Singularity compatibility, https://sylabs.io/guides/2.6/user-guide/build_a_container.html#downloading-a-existing-container-from-docker-hub

[13] OpenFOAM Official website, https://openfoam.org/

[14] Centos Operating System website, https://centos.org

[15] Slurm official website, https://slurm.schedmd.com/

[16] Artificial Intelligence Definition, https://en.wikipedia.org/wiki/Artificial_intelligence

[17] Machine Learning, https://en.wikipedia.org/wiki/Machine_learning

[18] TensorFlow website, https://www.tensorflow.org/

[19] RYAX website, https://ryax.tech/

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

[20] Nextflow website, https://nextflow.io/

[21] EAR website, https://www.bsc.es/research-and-development/software-and-apps/software-list/ear-energy-management-framework-hpc

https://on-demand.gputechconf.com/gtc/2018/presentation/s8368-containerizing-deep-learning-with-singularity.pdf

https://chiroptical.dev/blog/building-tensorflow-gpu-images-for-hpc

https://www.intel.com/content/dam/www/public/us/en/documents/presentation/hpc-containers-singularity-advanced.pdf

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

Acronyms and Abbreviations

Terminology/Acronym	Description
AI	Artificial Intelligence
API	Application Programming Interface
ARM	Advanced RISC Machines
CAD	Computer Aided Design
CAE	Computer-aided engineering
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
ELF	Executable Linkable Format
GID	Group Identifier
GPU	Graphics Processing Unit
HEROES	Hybrid Eco Responsible Optimized European Solution
HPC	High-Performance Computing
ML	Machine Learning
MPI	Message Passing Interface
OS	Operating System
OFED	Open Fabrics Enterprise Distribution
ORTED	Open RTE User-Level Daemon
PMI	Process Management Interface
SDF	Singularity Definition File
SIF	Singularity Image File
SME	Small and medium-sized enterprises
UID	User Identifier

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

Executive Summary

The HEROES [1] project is aiming at developing an innovative European software solution allowing industrial and scientific user communities to easily submit complex Simulation and ML (Machine Learning) workflows to HPC (High Performance Computing) Data Centres and Cloud Infrastructures. It will allow them to take informed decisions and select the best platform to achieve their goals on time, within budget and with the best energy efficiency.

This document presents an approach for the containerization [2] of the software used on the CAE and AI workflows selected as examples for the project.

The other tasks of the HEROES project rely on the workflows provided by this Work Package as examples to provide an end-to-end prototype at the end of the project.

Existing and new workflows will be designed, deployed, and operated to allow AI/HPC users to focus on their daily work taking advantage of encapsulating their applications inside reusable container images.

The development of this document and steps is closely linked to the optimisation work of WP3.

The scripts, templates and code described in this document will be constantly reviewed and adapted to satisfy all emergent needs of the HEROES platform throughout the life of the Project.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

1 Introduction

The objective of this document is to understand the process of containerizing the applications used on the workflows that will be taken as examples in the HEROES project.

Containerization [2] is defined as a form of operating system virtualization through which applications are run in isolated user spaces called containers, all using the same shared operating system (OS).

A container is essentially a fully packaged and portable computing environment.

The HEROES platform uses containerization in all the steps of the workflows during the execution phase. These containers are managed by the Application and Container Management & Orchestration module, described in detail in D3.1.

The use of containers enables the immutability of the execution environment and the portability of the software, independently of the destination of the execution. The technology chosen to achieve this is Singularity [3], a well proven solution inside HPC and Cloud environments.

More information about this technology is described in Section 2.

1.1 Why do we need containers?

- Simplify application building
- Application isolation
- Faster application deployment
- Validation and reproducibility of results
- Server consolidation/Server efficiency
- Deployment on bare metal or virtual machines

1.2 Benefits of containers

- Lightweight
- Low overhead
- Easier application sharing
- Reproducibility in different computing environments
- Container images facilitates modification, distribution, and execution

1.3 Container Limitations

- Architecture compatibility: always limited to CPU architecture and binary format (ELF) - (you could add a level of hardware virtualization)

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

- Portability: containers in general are portable, but there are limitations
- Glibc / Kernel compatibility support
- Any other kernel / user land API compatibility (e.g., OFED)
- Bit rot: containers are subject to stagnation just as any operating system is
- Performance: there “may be” a slight theoretical performance penalty with utilizing kernel namespaces
- File paths: the file system is different inside the container than outside the container

1.4 Containers Basic Concepts

A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another.

Some of the terminology talking about containers is:

- **Build or definition file:** is a file defining how to build an image and their content
- **Image:** file that contains the application and all its dependencies
- **Container:** is an instantiation of an image
- **Registry:** a server where images are stored

Containers can be built to include all the programs, libraries, data and scripts such that an entire application can be contained and either archived or distributed for others to replicate no matter what version of Linux they are presently running.

Once an image container is built by a user, it can be shared over the entire HEROES platform using the Data Management module, that stores the created container images and their information.

This feature allows code reusability and collaborative development, allowing the access and use of a vast catalogue of pre-built container images by the HEROES users, simplifying the process to deploy applications.

1.5 General container architecture

Containers use a form of operating system (OS) virtualization. They leverage features of the host operating system to isolate processes and control the processes access to CPUs, memory, and disk space.

Each container shares the host OS kernel and, usually, the binaries and libraries, too. Sharing OS resources, such as libraries, significantly reduces the need to reproduce the operating system code—a server can run multiple workloads with a single operating system installation.

Generally, the container daemon is a root owned daemon which will separate out all possible namespaces to achieve a fully emulated separation from host and other containers.

Figure 1 presents the difference between the execution of a virtual machine and of a container (Docker in this case).

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

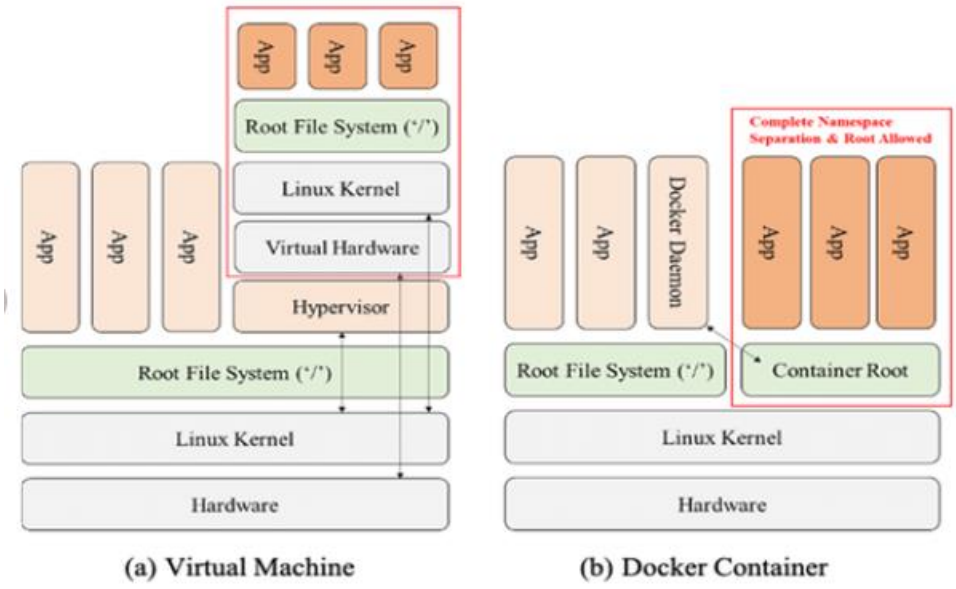

Figure 1. Container architecture diagram

2 Singularity

Singularity is a container technology. It allows the user to create and run containers that package up pieces of software in a way that is portable and reproducible. By software we mean that almost everything, from the operative system up, can be customized and run regardless the hosting OS.

Singularity was created to run complex applications on HPC clusters in a simple, portable, and reproducible way. It is an open-source project, with a friendly community of developers and users. The user base continues to expand, with Singularity now used across industry and academia in many areas of work.

2.1 Why Singularity?

- Verifiable reproducibility and security, using cryptographic signatures, an immutable container image format, and in-memory decryption.
- Integration over isolation by default. Easily make use of GPUs, high speed networks, parallel filesystems on a cluster or server by default.
- Mobility of compute. The single file SIF container format is easy to transport and share.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

- A simple, effective security model. You are the same user inside a container as outside, and cannot gain additional privilege on the host system by default.
- Directly import Docker [4] images and seamlessly run them with the additional features provided by Singularity.
- Consolidating a workflow into a Singularity container simplifies distribution and repeatability of scientific results.

Table 1 presents a comparison between Singularity and Docker. Singularity is very well fitted for the use cases of HEROES: it has been built from the ground up for HPC/AI use cases and with security as a strong requirement.

Table 1. Features between Singularity and Docker container technologies

Feature	Singularity	Docker
Multiple containers can be run on same hardware	Yes	Yes
Can be created and destroyed quickly	Yes	Yes
Do not need entire OS, only a core run time	Yes	Yes
Transferable to other machines easily	Yes	Yes
Image Format	Single file	Layered file
Use with HPC schedulers	Yes	No
Native support for MPI	Yes	No
Support for GPUS	Yes	No
Root owned Daemon process	No	Yes

2.2 Singularity runtime

When you run a container, the processes in the container will run as your user account, this happens because there is no root daemon process and no escalation of privileges within the container, just because the Singularity execution binary (sexec/sexec-suid) is executed via execv() provoking that the process itself is replaced by the process inside the container.

Singularity dynamically writes UID and GID information to the appropriate files within the container, and the user remains the same inside and outside the container.

Another fact is that the container file system is mounted using the nosuid option, and processes are started with the PR_NO_NEW_PRIVS flag set. This means that even if you run sudo inside your container, you won't be able to change to another user, or gain root privileges by other means. This approach provides a secure way for users to run containers and greatly simplifies things like reading and writing data to the host system with appropriate ownership.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

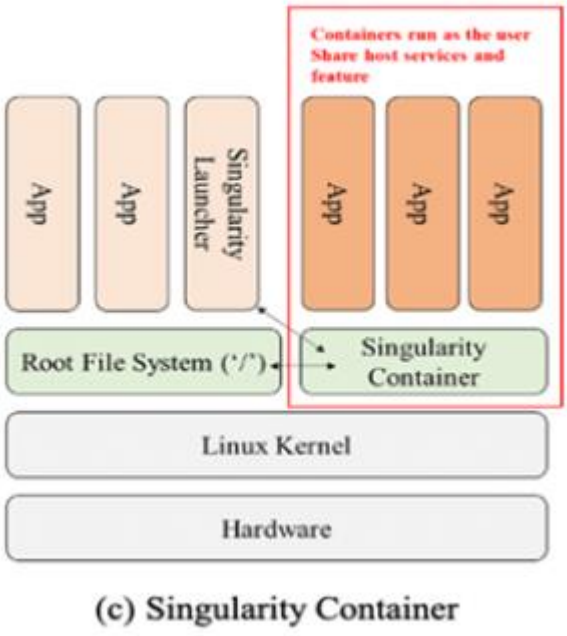

Figure 2. Singularity architecture diagram

Additional blocks are in place to prevent users from escalating privileges once they are inside of a container.

Namespaces are created depending on configuration and process requirements, where Singularity only isolates the mount namespace, and bind mount several host directories such as $HOME and /tmp into the container at runtime by default.

These measures allow users to interact with the host system from within the container in sensible ways, where the CLONE_FS namespace is used to virtualize completely the new root filesystem.

2.3 Using Singularity containers

In this section is described some of the commands that can be used on Singularity to perform different actions using containers and images.

- Obtaining a shell:

```
singularity shell lolcow_latest.sif
```

- Exec: run a given command inside the container:

```
singularity exec lolcow_latest.sif fortune
```

- Download an image

```
singularity pull library://godlovedc/funny/lolcow
```

- Run the .sif file image

```
singularity run lolcow_latest.sif
```

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

2.4 Inspecting Singularity containers

The Singularity container images are defined by files called definition files. Singularity provides commands to check this definition file from a built Singularity image.

```
singularity inspect --deffile lolcow_latest.sif
```

2.5 Singularity MPI

The Message Passing Interface (MPI) [5] is a standard extensively used by HPC applications to implement various communications across compute nodes of a single system or across compute platforms.

Singularity provides support for the two major Open-Source implementations, OpenMPI [6] and MPICH [7].

Although there are several ways of carrying this out, the most popular way of executing MPI applications installed in a Singularity container is to rely on the MPI implementation available on the host.

Here are described some of the points of the OpenMPI /Singularity workflow in detail (see Figure 3):

- The MPI launcher (e.g., mpirun, mpiexec) is called by the resource manager or the user directly from a shell.
- Open MPI then calls the process management daemon (ORTED [8]).
- The ORTED process launches the Singularity container requested by the launcher command, as such mpirun.
- Singularity builds the container and namespace environment.
- Singularity then launches the MPI application within the container.
- The MPI application launches and loads the OpenMPI libraries.
- The OpenMPI libraries connect back to the ORTED process via the Process Management Interface (PMI).

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

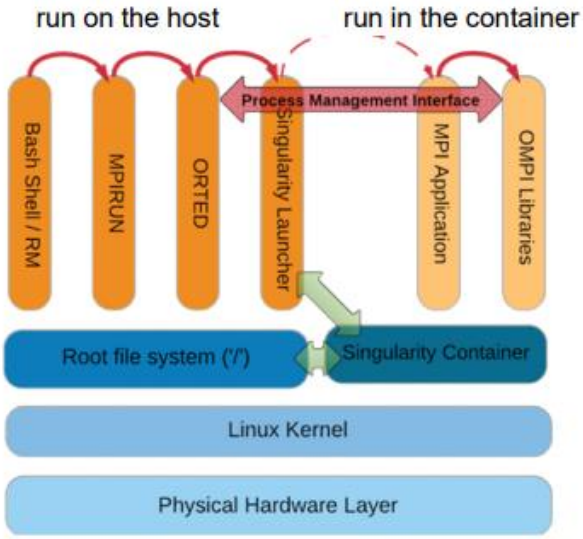

Figure 3. Singularity MPI architecture diagram

Assuming the container with MPI and the application is already built, the mpirun command to start the application looks like:

```
$ mpirun -n <NUMBER_OF_RANKS> singularity exec <PATH/TO/MY/IMAGE>
</PATH/TO/BINARY/WITHIN/CONTAINER>
```

3 Building images

As a platform that is widely used in the scientific/research software and HPC communities, Singularity provides great support for reproducibility.

If a Singularity image is built for some scientific software, others are able to reproduce exactly the same environment again.

Singularity follows the “Configuration as code” approach and a container configuration can be stored in a file which can then be committed to a version control system alongside other code.

This file can then be used to reproduce a container with the same configuration at some point in the future.

In HEROES, Singularity images are integrated by the Workflows and Job execution module, based on Nextflow [20], allowing the execution of processes using Singularity images as instantiated containers with all the software needed for the entire workflow execution.

Once the images were built, the resulting files were distributed across the HEROES Remote pseudo-filesystem (HRFS), more detailed on the D3.2 Deployment Suite, publishing the required images to execute the workflow on all the machines involved on the complete execution.

To use Singularity images on the HEROES platform, no extra configuration is needed in the image itself, as the Workflows and Job execution module allows to execute any kind of image transparently for the end-user.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

3.1 Approaches

There are various approaches to building Singularity [9] images:

- **Building within a sandbox:** The image is built interactively within a sandbox environment. This means obtaining a shell within the container environment and installing and configuring all the packages and code before exiting the sandbox and converting it into a container image.
- **Building from a Singularity Definition File:** In this approach, the entire Singularity images are auto-described on a single text file. Later, the Singularity image is built following all the instructions and commands defined inside this file.

Definition files are small text files, but container files may be very large multi-gigabyte files that are difficult and time consuming to move around.

This makes Definition files ideal for storing in a version control system along with their revisions.

Currently, HEROES platform assumes that the user provides the Singularity images files built by themselves, uploading all required Singularity Image Files (SIF) as part of the dataset used on the workflow execution.

All the data provided initially by the user, is copied and stored by the Data Management module, and stored on the assigned directories on the HRFS, allowing to reach and use these images from directories where the user has permissions.

For the moment, a service on HEROES to offer to build the images for the users is still under analysis and is not implemented. A possible approach is to provide an internal Gitlab service with a CI/CD pipeline to trigger the build of a Singularity image when the code of a Singularity definition file is provided.

3.2 Creating a Singularity definition file (SDF)

A Singularity Definition File, SDF [10] is a text file that contains a series of statements that are used to create a container image.

In line with the “Configuration as Code” approach mentioned above, the SDF can be stored in a code repository alongside the application code and used to create a reproducible image.

This means that for a given commit in the repository, the version of the SDF present at that commit can be used to reproduce a container with a known state.

Singularity uses an SDF to bootstrap a new container.

An example of an SDF could look like this:

```
Bootstrap: docker
From: ubuntu:20.04

%post
    apt-get -y update && apt-get install -y python

%runscript
    python -c 'print("Hello World! Hello from our custom Singularity
image!")'
```

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

A definition file has a number of optional sections, specified using the % prefix, that are used to define or undertake different configurations during different stages of the image build process.

3.2.1.1 Bootstrap line

The Bootstrap line is similar to prefixing an image path when using the Singularity pull command.

A range of different bootstrap options are supported: registries, DockerHub, etc.

3.2.1.2 %post section

This section is where you can download files from the internet with tools like git and wget, install new software and libraries, write configuration files, create new directories, etc.

3.2.1.3 %runscript section

This section is used to define a script that should be run when a container is started based on this image using the singularity run command.

3.2.1.4 %setup section

During the build process, commands in this section are first executed on the host system outside of the container after the base OS has been installed.

3.2.1.5 %files section

In this section, allows to copy files into the container with greater safety than using the %setup section.

3.2.1.6 %environment section

This section allows you to define environment variables that will be set at runtime.

3.2.1.7 %labels section

This section is used to add metadata to the file (/.singularity.d/labels.json) within your container.

3.2.1.8 %startscript section

Similar to the %runscript section, the content of the %startscript section is written to a file within the container at build time.

3.2.1.9 %runscript section

The content of this section is written to a file within the container that is executed when the container image is run (either via the singularity run command or by executing the container directly as a command). When the container is invoked, arguments following the container name are passed to the runscript. This means that you can (and should) process arguments within your runscript.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

3.2.1.10 %help section

Any text in the %help section is transcribed into a metadata file in the container during the build.

3.2.1.11 %test section

This section runs at the very end of the build process to validate the container using a method of your choice.

3.3 Building the image

The image can be built using this command [11]:

```
singularity build my_test_image.sif my_test_image.def
```

The above command requests the building of an image based on the **my_test_image.def** file with the resulting image saved to the **my_test_image.sif** file.

Note that administrative privileges are required to build the image.

The result of the previous command is a **my_test_image.sif** Singularity image file in the current directory.

3.4 Notes about created images

3.4.1.1 Inspecting the image

This command shows how a container image was built, obtaining the definition file from the image file.

```
singularity inspect --deffile lolcow_latest.sif
```

You can drop the standard output to a file to obtain the SDF of the container:

```
singularity inspect --deffile lolcow_latest.sif >> lolcow_latest.def
```

These commands are very useful to understand how an image was built, allowing us to edit it directly or modify it to obtain a different behaviour from this base.

3.4.1.2 Docker compatibility

Singularity allows the user to build images directly from Docker [4] containers, allowing a Docker Hub endpoint to build an image from the registry [12].

```
sudo singularity build container.img docker://ubuntu
```

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

3.4.1.3 Cluster platform configuration for running Singularity containers

It is recommended to move the created .sif file to a platform with an installation of Singularity, rather than attempting to run the image pulling the container from the registry.

This avoids problems related to limited connectivity and machines with restricted access to the Internet domain.

From the HEROES platform, all the connections to the HPC machines are achieved using the SSH protocol, and the outbound connections from the login nodes are restricted, so, pulling the Singularity images from an external registry is not always allowed.

In order to provide the images required to execute the different steps of a workflow, the images should be copied previously from the HEROES platform as a source, using just tools that relies on the SSH connectivity.

In this process, the users don't need to copy manually anything to the remote locations, as the HEROES platform takes care on copying the required images automatically before executing the workflow.

This image files movement is managed by the Workflows and Job management module in combination with the Data Management Module inside the HEROES platform.

3.4.1.4 Signing containers

Singularity supports signing containers. This allows a digital signature to be linked to an image file. This signature can be used to verify that an image file has been signed by the holder of a specific key and that the file is unchanged from when it was signed.

4 CAE Containers

4.1 Introduction

The numerical simulation of the Computational Fluid Dynamics is a very well-known strategy to optimize engineering designs in multiple fields inside Computational Aided Engineering.

Both the aerodynamical and the thermal behaviour can be predicted, leading to an optimal manufacturing design.

High Performance Computing (HPC) simulation tools are capable to run these simulation routines in a time compatible with the time to market of any product developed by manufacturing SMEs or big companies.

In this regard, HPC simulation tools can be seen as enabling technologies for the use of CFD and increasing the market share of any related product.

In particular, for any CFD model several steps are needed to be accomplish one after the other, before obtaining the final result, some of these steps are requiring manual interaction of the user.

The objective of HEROES is to help CFD users to accomplish the final result in an easier way, while optimising the HPC platform and tools.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

The ultimate objectives are:

- To reduce the human interaction by using automation during manual steps
- To optimise the use of HPC resources
- To reduce the time needed to accomplish one CFD workflow

In this context, the use of containerized applications makes it possible to define concrete environments, allowing the same execution to be ported and reproduced on different computational locations.

As we defined on the D2.1, for the selected CAE workflow to exemplify how a CFD user workflow is executed on the HEROES platform, the OpenFoam program is required, as is needed to execute the proposed example to run an aerodynamic simulation against a 3D model.

In the next sections is explained how the application can be defined and containerized to be used later on the HEROES platform using the Singularity technology, and how can be tested locally before uploading it to the platform.

4.2 Containerized OpenFoam

For obtaining a Singularity container image including a valid OpenFoam [13] version, a definition file (**heroes-openfoam-7.def**) is created with a baseline based on a Centos [14] Docker image.

On the post section, additional dependencies are installed, like MPI and the OpenFoam version to include.

```
Bootstrap: docker
From: centos:7

%help
    This recipe provides an OpenFOAM-7 environment installed
    with GCC and OpenMPI-4.

%post
    ### Install prerequisites
    yum groupinstall -y 'Development Tools'
    yum install -y wget git openssl-devel libuuid-devel

    ### OpenFOAM version
    pkg=OpenFOAM
    vrs=7

    ### Install under /opt
    base=/opt/$pkg
    mkdir -p $base && cd $base

    ### Download OF
    wget -O - http://dl.openfoam.org/source/$vrs | tar xz
    mv $pkg-$vrs-version-$vrs $pkg-$vrs

    ### Download ThirdParty
    wget -O - http://dl.openfoam.org/third-party/$vrs | tar xz
```

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

```
    mv ThirdParty-$vrs-version-$vrs ThirdParty-$vrs

    ### Change dir to OpenFOAM-version
    cd $pkg-$vrs

    ### Get rid of unalias otherwise singularity fails
    sed -i 's,FOAM_INST_DIR=$HOME\/$WM_PROJECT,FOAM_INST_DIR='"$base"',g'
etc/bashrc
    sed -i 's/alias wmUnset/#alias wmUnset/' etc/config.sh/aliases
    sed -i '77s/else/#else/' etc/config.sh/aliases
    sed -i 's/unalias wmRefresh/#unalias wmRefresh/' etc/config.sh/aliases
    ### Compile and install
    . etc/bashrc
    ./Allwmake 2>&1 | tee log.Allwmake

    ### Clean-up environment
    rm -rf platforms/$WM_OPTIONS/applications
    rm -rf platforms/$WM_OPTIONS/src

    cd $base/ThirdParty-$vrs
    rm -rf build
    rm -rf gcc-* gmp-* mpfr-* binutils-* boost* ParaView-* qt-*

    strip $FOAM_APPBIN/*

    ### Source bashrc at runtime
    echo '. /opt/OpenFOAM/OpenFOAM-7/etc/bashrc' >>
$SINGULARITY_ENVIRONMENT

%test
    . /opt/OpenFOAM/OpenFOAM-7/etc/bashrc
    icoFoam -help

%runscript
    echo
    echo "OpenFOAM installation is available under $WM_PROJECT_DIR"
    echo
```

4.3 Image creation

The build of the previous definition file can be achieved by this command, obtaining a valid Singularity image (**heroes-openfoam-7.sif**) including all the software stack needed to run OpenFoam.

```
singularity build heroes-openfoam-7.sif heroes-openfoam-7.def
```

4.4 Execution

The following example shows how to execute a sample OpenFOAM command using the container image, confirming that the image is built including the required toolset:

```
singularity exec heroes-openfoam-7.sif simpleFoam -help
Usage: simpleFoam [OPTIONS]
options:
  -case <dir>       specify alternate case directory, default is the cwd
  -fileHandler <handler> override the fileHandler
```

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

```
  -hostRoots <(((host1 dir1) .. (hostN dirN))> slave root directories (per
host) for distributed running
  -libs <(lib1 .. libN)> pre-load libraries
  -listFunctionObjects List functionObjects
  -listFvOptions List fvOptions
```

4.5 Singularity OpenFoam + Slurm

This section shows how is possible to use the Singularity image to launch a Slurm [15] job to execute an OpenFoam program:

```
##!/bin/bash
#SBATCH --job-name=slurm-heroes-openfoam
#SBATCH --output=slurm-heroes-openfoam_%j.out
#SBATCH --cpus-per-task=1
#SBATCH --gres gpu:1
#SBATCH --time=1:00:00

OF_IMG=heroes-openfoam-7.sif
OF_DIR=$WORK/openfoam/

mkdir $SCRATCH/motorBike_data
cp -v motorBike.tar.gz $SCRATCH/motorBike_data/

module load singularity

srun singularity exec --home $WORK:/home --bind $LSTOR:/tmp $OF_IMG \
    SimpleFoam $SCRATCH/motorBike_data/ --batch_size=128 \
                                        --max_steps=100000
```

5 AI/ML Containers

5.1 Introduction

Artificial intelligence (AI) [16] refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions.

The ideal characteristic of AI [15] is its ability to rationalize and take actions that have the best chance of achieving a specific goal.

Machine learning (ML) [17] is a subfield of artificial intelligence. The goal of ML is to make computers learn from the data that an end user gives them and adapt to new data without being assisted by humans. The resulting program, consisting of the algorithm and associated learned parameters, is called a trained model.

Within ML, Deep learning techniques enable this automatic learning through the absorption of huge amounts of unstructured data such as text, images, or video.

The objective of HEROES is to help ML users to accomplish the final result in an easier way, while optimising the ML environment and tools.

The ultimate objectives are:

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

- To reduce the human interaction by using automation during manual steps.
- To optimise the use of HPC resources.
- To reduce the time needed to accomplish one ML workflow.

In this context, the utilization of containerized applications allows to define concrete ML environments, allowing to port and reproduce the same execution on different computational locations.

As we defined on the D2.1, for the selected ML workflow to exemplify how a machine learning user workflow is executed on the HEROES platform, the Tensorflow library is required, as is needed to execute the proposed example to run an object recognition process against a video file.

In the next sections is explained how the application can be defined and containerized to be used later on the HEROES platform using the Singularity technology, and how can be tested locally before uploading it to the platform.

5.2 Containerized Tensorflow

For obtaining a Singularity container image including a valid Tensorflow [18] version, a definition file (**heroes-tensorflow-latest.def**) is created with a baseline based on the official Tensorflow Docker image.

This file is based on the official Docker image from Tensorflow, later, the post section includes additional dependencies.

```
bootstrap: docker
from: tensorflow/tensorflow

%help

   This Singularity definition contains a TensorFlow installation

%post

  pip install pillow matplotlib urllib3 opencv-python
  python -c "import platform; print('Python: ',platform.python_version())"
  python -c "import tensorflow as tf; print('TensorFlow: ',tf.__version__)"

%environment

   export LC_ALL=C

%runscript
```

5.3 Image creation

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

The build of the previous definition file can be achieved by this command, obtaining a valid Singularity image (**heroes-tensorflow.sif**). Including all the software stack needed to run Tensorflow.

```
singularity build heroes-tensorflow-latest.sif heroes-tensorflow -
latest.def
```

5.4 Execution

Following example shows how to execute a sample Tensorflow command using the container image:

```
singularity exec heroes-tensorflow -latest.sif python -c import tensorflow
as tf; print('TensorFlow: ',tf.__version__)
```

5.5 Singularity Tensorflow + Slurm

This section shows how is possible to use the Singularity image to launch a Slurm job to execute a Tensorflow program:

```
##!/bin/bash

#SBATCH --job-name=slurm-heroes-tensorflow
#SBATCH --output=slurm-heroes-tensorflow_%j.out
#SBATCH --cpus-per-task=1
#SBATCH --gres gpu:1
#SBATCH --time=1:00:00

TF_IMG=heroes-tensorflow.sif
OF_DIR=$WORK/tensorflow/mobilenet_ssd_v2

mkdir $SCRATCH/tensorflow_data
cp -Ra $PROJECTDIR/tensorflow_data/ $SCRATCH/tensorflow_data/

module load singularity

srun singularity exec --home $WORK:/home --bind $SCRATCH:/tmp $TF_IMG
python $OF_DIR/ssd_mobilenet_v2_coco.py SCRATCH/tensorflow_data/
```

6 HEROES containers and workflows integration

6.1 Nextflow introduction

The importance of the Workflows and Job Management module was widely described on the D2.1, assuming the use of RYAX [19] as the Workflow Manager tool in a first iteration.

In recent changes over the architecture and the software used, RYAX was replaced by Nextflow [20], as it offers some native integrations for some technologies used on the HEROES platform and some other benefits.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

Some points that are considered on using Nextflow are:

- Fast prototyping
- Unified parallelism. Based on the dataflow programming model simplifying writing complex distributed pipelines.
- Stream oriented. allowing to handle complex stream interactions easily.
- Support for different jobs schedulers
- Reproducibility and portability using Singularity and Docker

6.2 Singularity on Nextflow

As this document describes the containerization of the application on HEROES platform, this section will focus on showing how Singularity can be used and integrated on Nextflow workflows.

Nextflow provides built-in support for Singularity. This allows to control the execution environment of the processes in your workflow by running in isolated containers along all their dependencies.

As we described in Section 3, Singularity images can contain any tool or piece of software you may need to carry out a process execution.

As is described before, the Singularity container image files provided by the users don't need to contain any software or configuration to be integrated on the HEROES platform.

All the configuration needed for the execution is defined outside them, in the different modules that conforms the platform, so the images should contain exclusively the software needed to run the applications or programs, keeping them simpler to configure, share and use.

Every time that a Nextflow workflow script launches a process execution, Nextflow will run it into a Singularity container created by using one specified image, in practice, Nextflow will automatically wrap your processes and run them by executing the singularity run command with the image you have provided and later is executed using the job scheduler that you specified.

Singularity images can be defined in a Nextflow configuration file, just as the example showed below:

```
process.container = '/path/to/singularity.img'
singularity.enabled = true
```

It is possible to specify a different Singularity image for each process definition in your workflow script if is needed for the global workflow execution.

For example, let's suppose you have two processes named foo and bar. You can specify two different Singularity images specifying them in the nextflow.config file as shown below:

```
process {
```

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

```
    withName:foo {
        container = 'image_name_1'
    }
    withName:bar {
        container = 'image_name_2'
    }
}
singularity {
    enabled = true
}
```

By default, when a container name is specified, Nextflow checks if an image file with that name exists in the local file system. If that image file exists, it's used to execute the container.

The Singularity configuration scope controls how Singularity containers are executed by Nextflow.

Different settings are available to configure Singularity scope on Nextflow:

- **enabled:** Turn this flag to true to enable Singularity execution (default: false).
- **engineOptions:** This attribute can be used to provide any option supported by the Singularity engine.
- **envWhitelist:** Comma separated list of environment variable names to be included in the container environment.
- **runOptions:** This attribute can be used to provide any extra command line options supported by the singularity exec.
- **noHttps:** Turn this flag to true to pull the Singularity image with http protocol (default: false).
- **autoMounts:** When true Nextflow automatically mounts host paths in the executed container. It requires the user bind control feature enabled in your Singularity installation (default: false).
- **cacheDir:** The directory where remote Singularity images are stored. When using a computing cluster, it must be a shared folder accessible to all computing nodes.
- **pullTimeout:** The amount of time the Singularity pull can last, exceeding which the process is terminated (default: 20 min).

6.3 Example containerized workflow diagram

Some of the purposes of using Singularity containers are described above in Section 2.1: portability, immutability, lightness, etc.

In HEROES, the approach is to containerize different steps of the workflows to take advantage of these benefits from containerization technology to run the desired applications and programs.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

In Figure 4**Erreur ! Source du renvoi introuvable.**, we display an example of a possible user workflow, marking in red squares the steps where potentially the containerized applications and programs are executed inside Singularity containers.

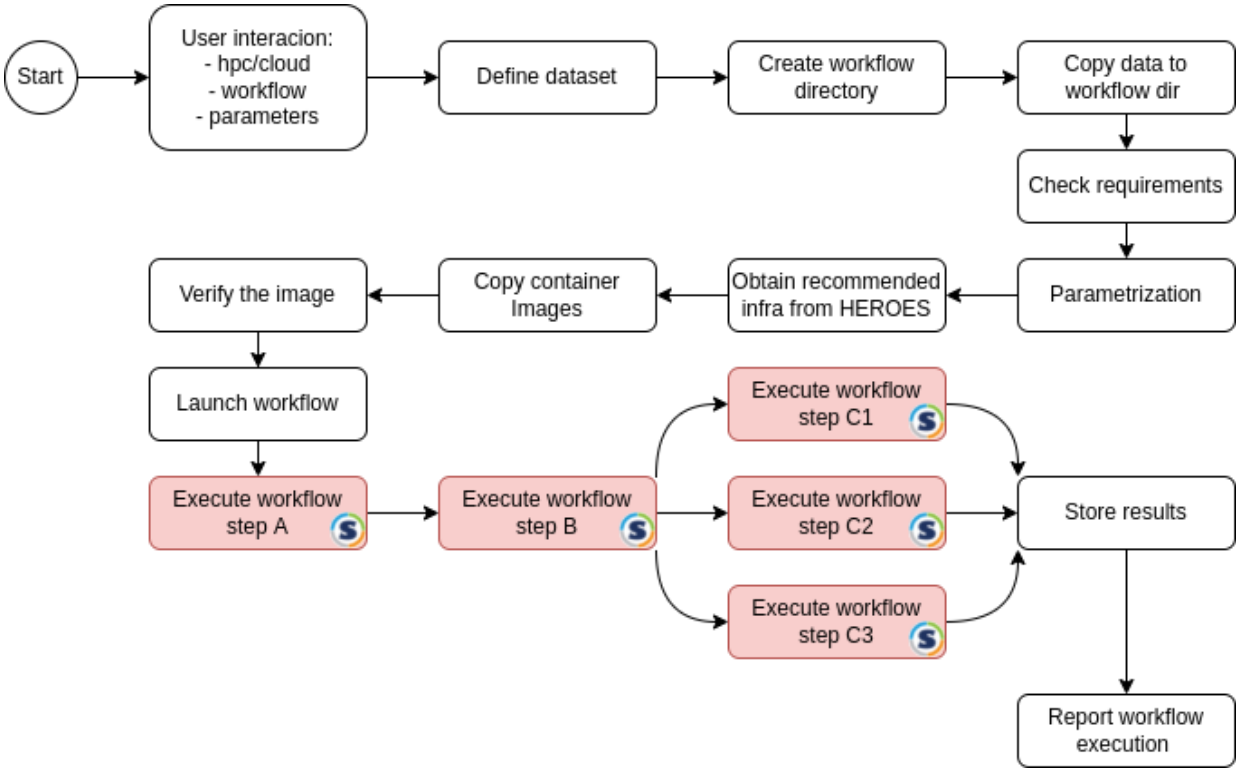

Figure 4. Singularity execution placement example

7 Conclusion

This document presented the containers and their definitions that the workflows in SINGULARITY project will use as examples throughout the project.

Two containers and their definitions are represented for two workflows:

- One including a typical program on the HPC field (airflow simulation through CFD simulations, OpenFoam containerized).
- One in the AI field (image recognition through machine learning, using Tensorflow).

An introduction on HPC containers, and the steps for define and build the container images on Singularity including the software used on the workflows, have been described, allowing an introductory definition and procedure that could be used to the image creation and application containerization for future steps in the project.

The application containerization for the workflows, represents a straight-forward mechanism to build, encapsulate, manage, and deploy functional HPC and ML applications inside workflows, simplifying the process for the end-users and obtaining some of the benefits on using containers like portability, immutability, etc.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.

The use of Singularity containers makes the execution of each step “independent” of the target platform.

The whole workflow execution using the containers here, will be handled by the Workflows and Job Management module described on the D2.1, integrated with other platform modules as the D3.2 Deployment Suite, to provide a proper environment to run the execution, and also the HEROES runtime.

After each scheduled container execution on a workflow, some logs and metrics are saved based on different parameters such as frequencies, power, etc. obtained by the Telemetry and accounting module, based mainly on the EAR suite integrated as the HEROES runtime.

On the D2.1 deliverable, we started considering RYAX as the initial Workflow manager, but lately was substituted by Nextflow, (described briefly on Section 7) as it’s an Open-Source solution and natively supports some of the technologies used on the HEROES platform.

The goal of SINGULARITY is to enable future developments and technology democratization, by providing user-friendly and flexible workflow tools. The main advantages of the application containerization using the SINGULARITY platform we envision are:

- Allow easy parameterized HPC simulations execution, allowing exploration of new models, parameters…
- The creation of containers for each step, allows their reutilization and generalization across the entire architecture.
- Simplification of usage for the end-users by providing “pre-packaged” workflows and applications. The usage of containerization makes the whole process of simulation easier, lowering the learning curve and even hiding the execution complexity.
- The containerization of applications allows to share and distribute complete images against other users on the platform, reducing installation and configuration times.

The HEROES project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956874. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Italy.