DESTINATION EARTH

Containerization and deployment of weather models on EuroHPC JU infrastructures

Massimo Gisonni, *Giuseppa Muscianisi (CINECA),*
Francesco Cola (CINECA/JSC), Thomas Geenen (ECMWF)

21st ECMWF workshop on high performance computing in meteorology, Bologna (Italy), 15-19 September 2025

Funded by the European Union

Destination Earth implemented by ECMWF esa EUMETSAT

CINECA

GOAL OF THE PROJECT

***PoC**: design of a workflow to containerize weather models & run them at performance in the clusters of the EuroHPC JU*

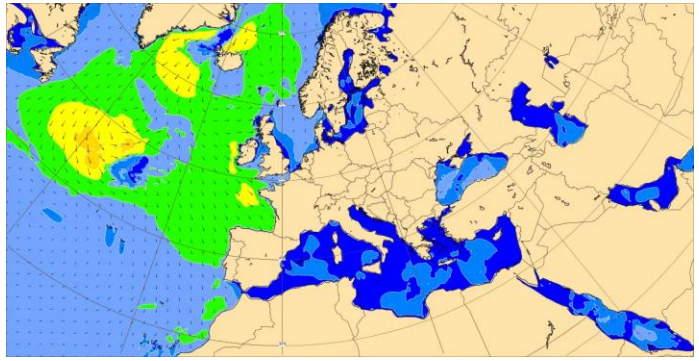
- A single immutable container image *(.sif)* holding the application's binary, the required libraries, interfacing at runtime to MPI

GOAL OF THE PROJECT

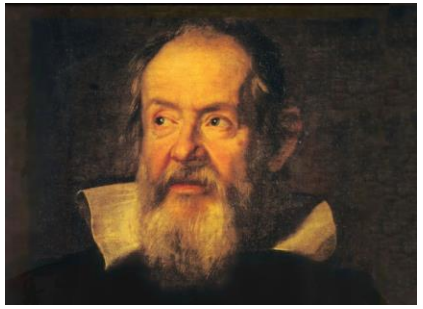
***PoC**: design of a workflow to containerize weather models & run them at performance in the clusters of the EuroHPC JU*

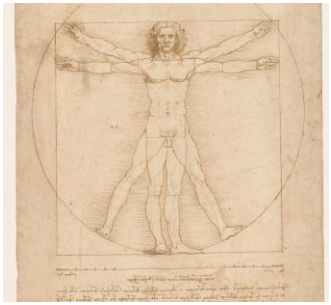
- A single immutable container image *(.sif)* holding the application's binary, the required libraries, interfacing at runtime to MPI

- **Target applications:**
  - *ecWAM* (Ocean Wave model)
  - *ecTrans* (Spectral transforms)
  - *RAPS20* (GPU support)
  - *RAPS21* (CPU only)



- **Target clusters:** EuroHPC Joint Undertaking
  - *GALILEO100* (CINECA)
  - *LEONARDO* (CINECA)
  - *MeluXina* (LuxProvide)
  - *LUMI* (CSC)





THE ASSETS OF CONTAINERS

Portability:

- **No need to rebuild** the same software for each system
- Different containers for different **architectures** *(x86 vs CRAY)*

Reproducibility:

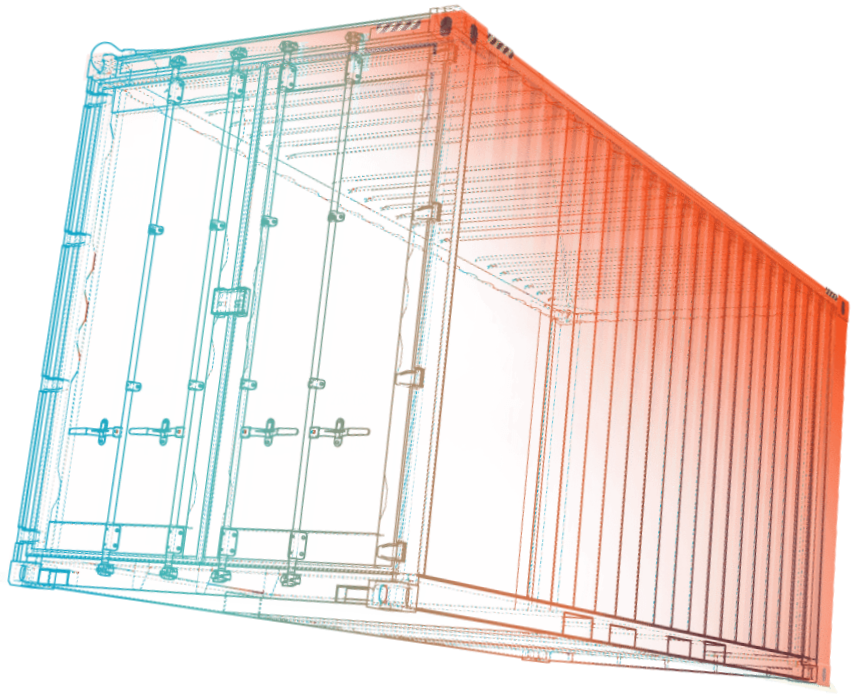
- **The same container** can be built in different sites from the **same recipe**
- Software stack inside the container is fixed

Legacy:

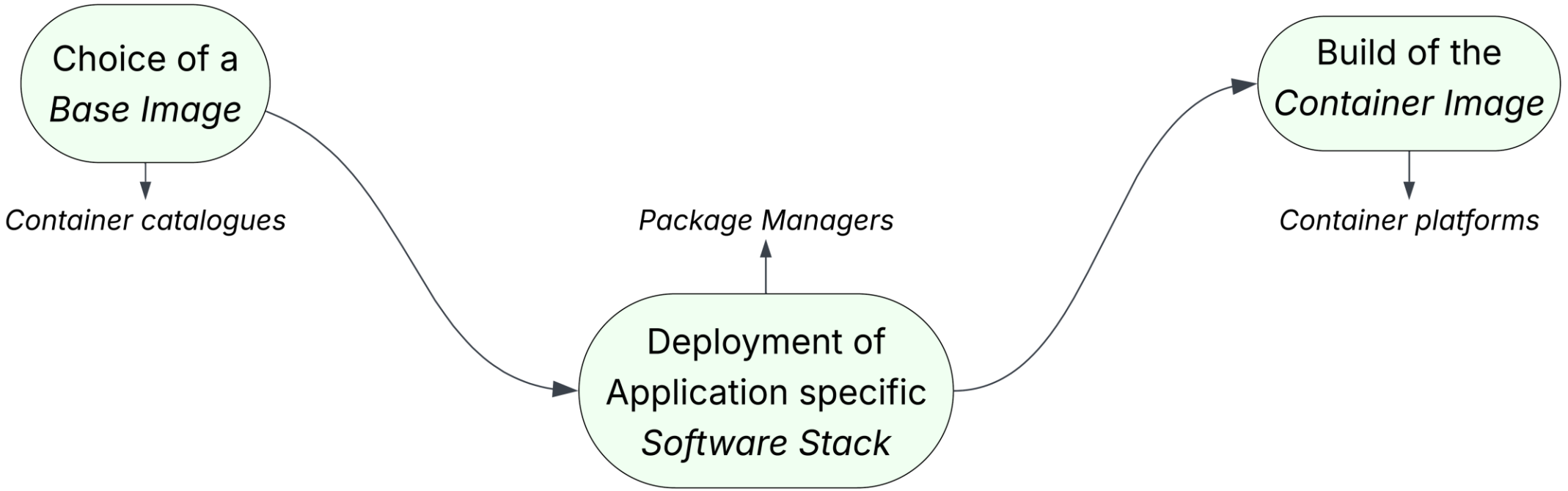
- **Testing old/new software stack** against old/new host software stack
- Quick deployment in case of issues in the host

Distribution:

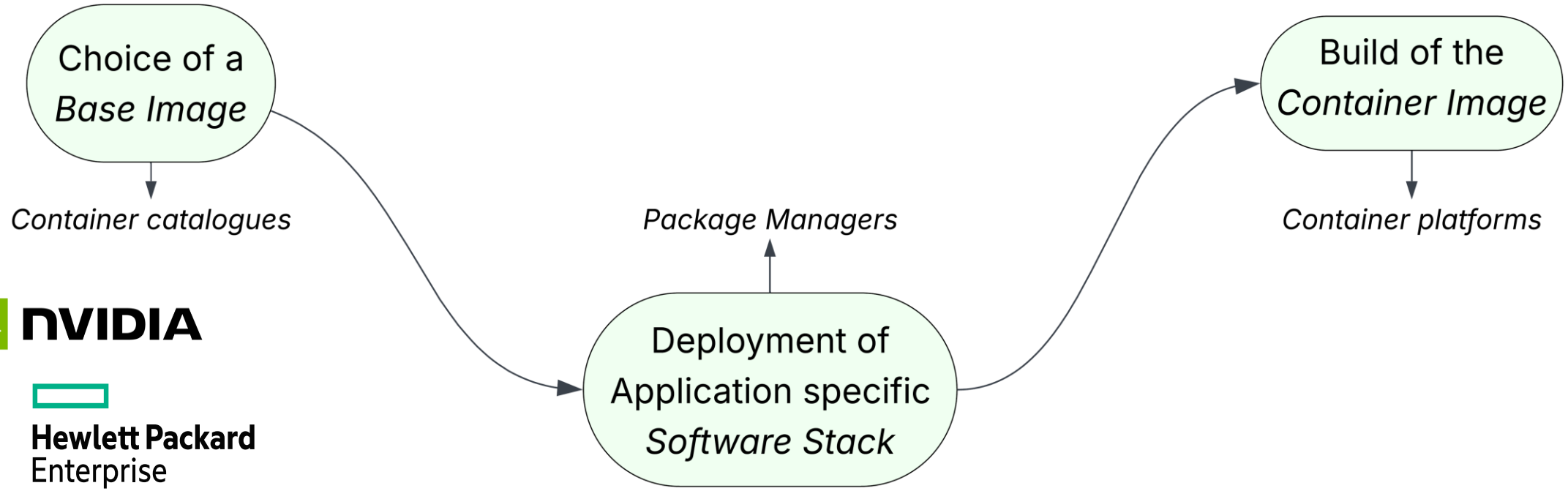
- Containers are **lightweight**, easily transferable
- **Integration on CI/CD pipeline** allows for controlled deployment



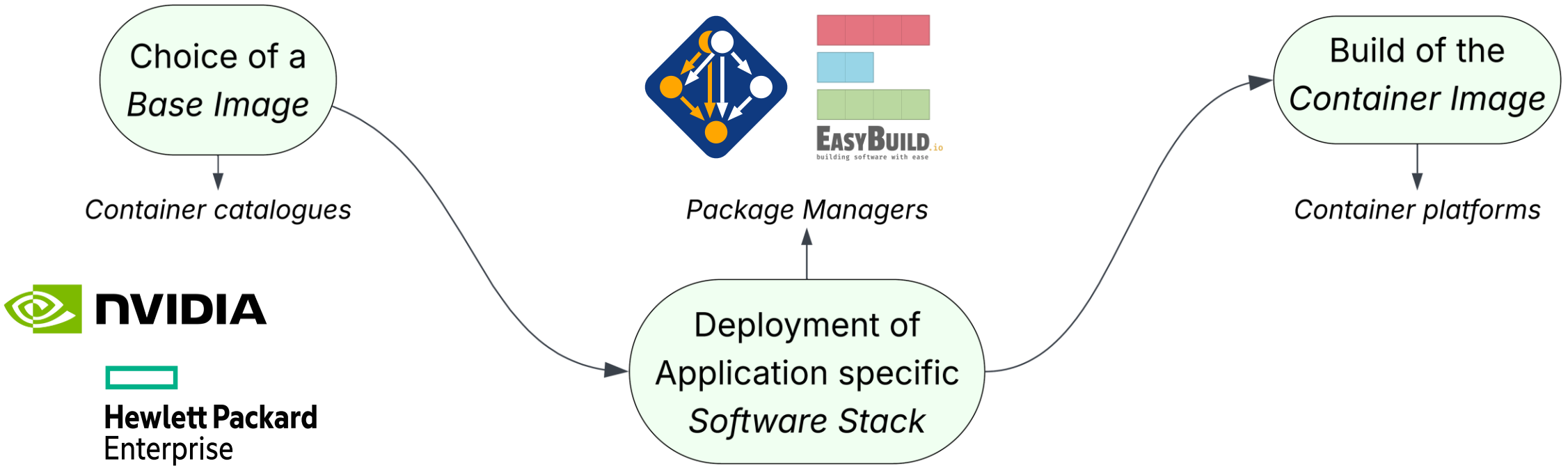
THE TOOLS OF THE PROJECT



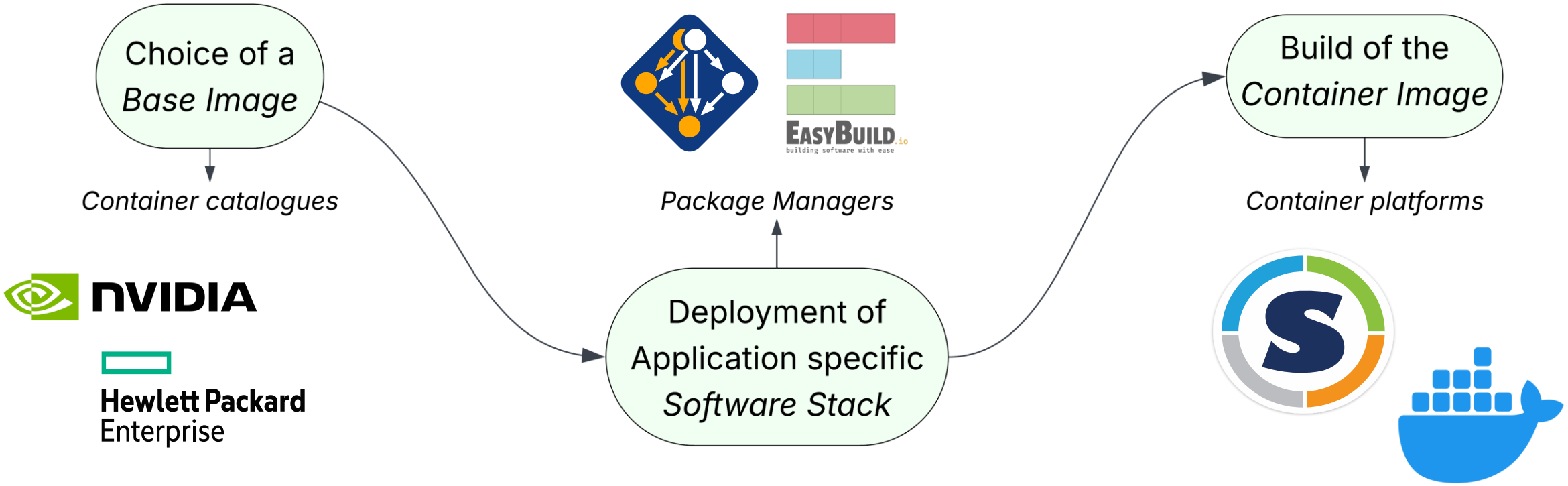
THE TOOLS OF THE PROJECT



THE TOOLS OF THE PROJECT



THE TOOLS OF THE PROJECT



THE TOOLS OF THE PROJECT: SINGULARITY

```
Bootstrap: docker
From: nvcr.io/nvidia/nvhpc:23.1-devel-cuda_multi-ubuntu22.04
Stage: build
```

HEADER

```
%files
    /model.tar.gz       /tmp
    /spack.yaml         /spack_env/spack.yaml

%post
    apt-get update && apt-get -y install python3 git wget
    tar -xzf model.tar.gz
    ./install_model.sh  --builddir=/build


%environment # allows to define environment variables
    export VARIABLE=MEATBALLVALUE


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


Bootstrap: docker
From: nvcr.io/nvidia/nvhpc:23.1-runtime-cuda_multi-ubuntu22.04
Stage: deploy

%files from build
    /build/bin/model        /deploy/model
```

SECTIONS

MULTISTAGE

THE TOOLS OF THE PROJECT: SPACK

Host compilers and pre-installed packages

```
config:
    source_cache: $env/source_cache
    install_tree: $env/install

    build_stage:
    - $env/build/stage
    - $env/spack-stage
```

```
compilers:
  - compiler:
      spec: gcc@=11.3.0
      paths:
        cc:   /bin/gcc
        cxx:  /bin/g++
        f77:  /bin/gfortran
        fc:   /bin/gfortran
      flags: {}
      operating_system: ubuntu22.04
      target: x86_64
```

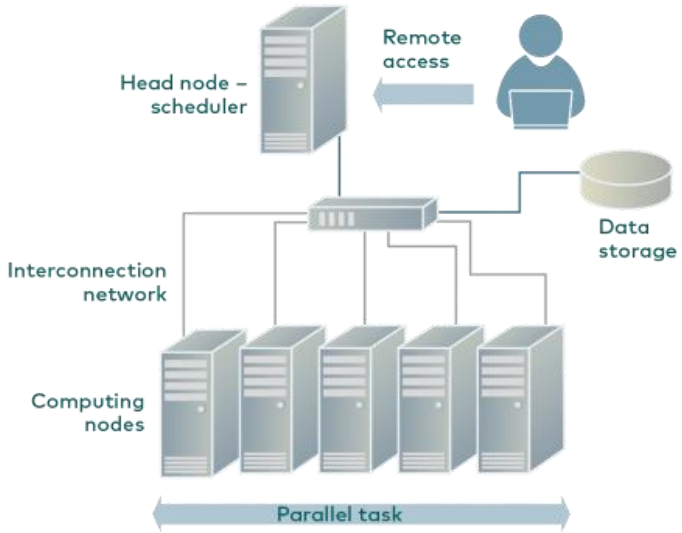
```
packages:
    cuda:
      externals:
      - spec: cuda@12.0.76
        prefix: /opt/cuda

    openmpi:
      externals:
      - spec: openmpi@4.1.5 %gcc@=11.3.0
        prefix: /opt/ompi
```

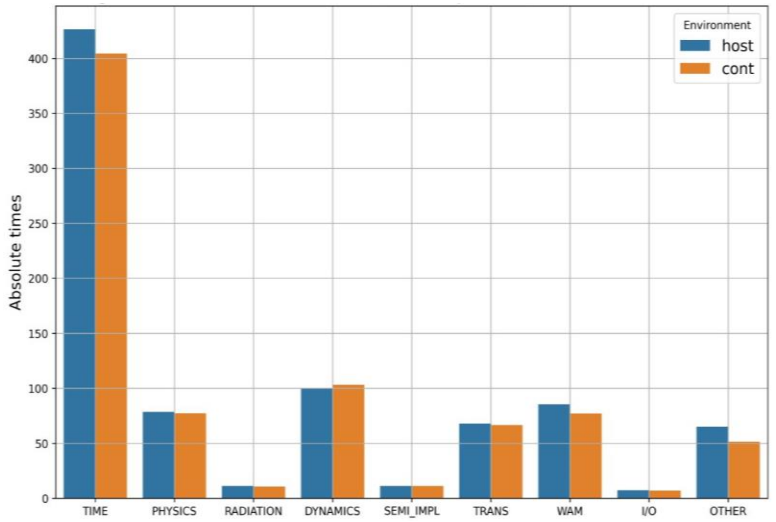
```
specs:
- hdf5@1.12.2              %nvhpc      -cxx +fortran +mpi
- zlib-ng                  %gcc
- netcdf-c@4.9.0           %nvhpc      +mpi +optimize
- netcdf-fortran@4.6.0     %nvhpc      build_system=autotools
- cmake@3.24


- package@version          %compiler   +options
```

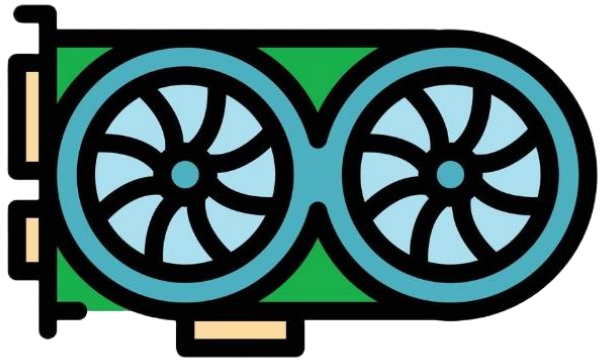
Packages to be installed

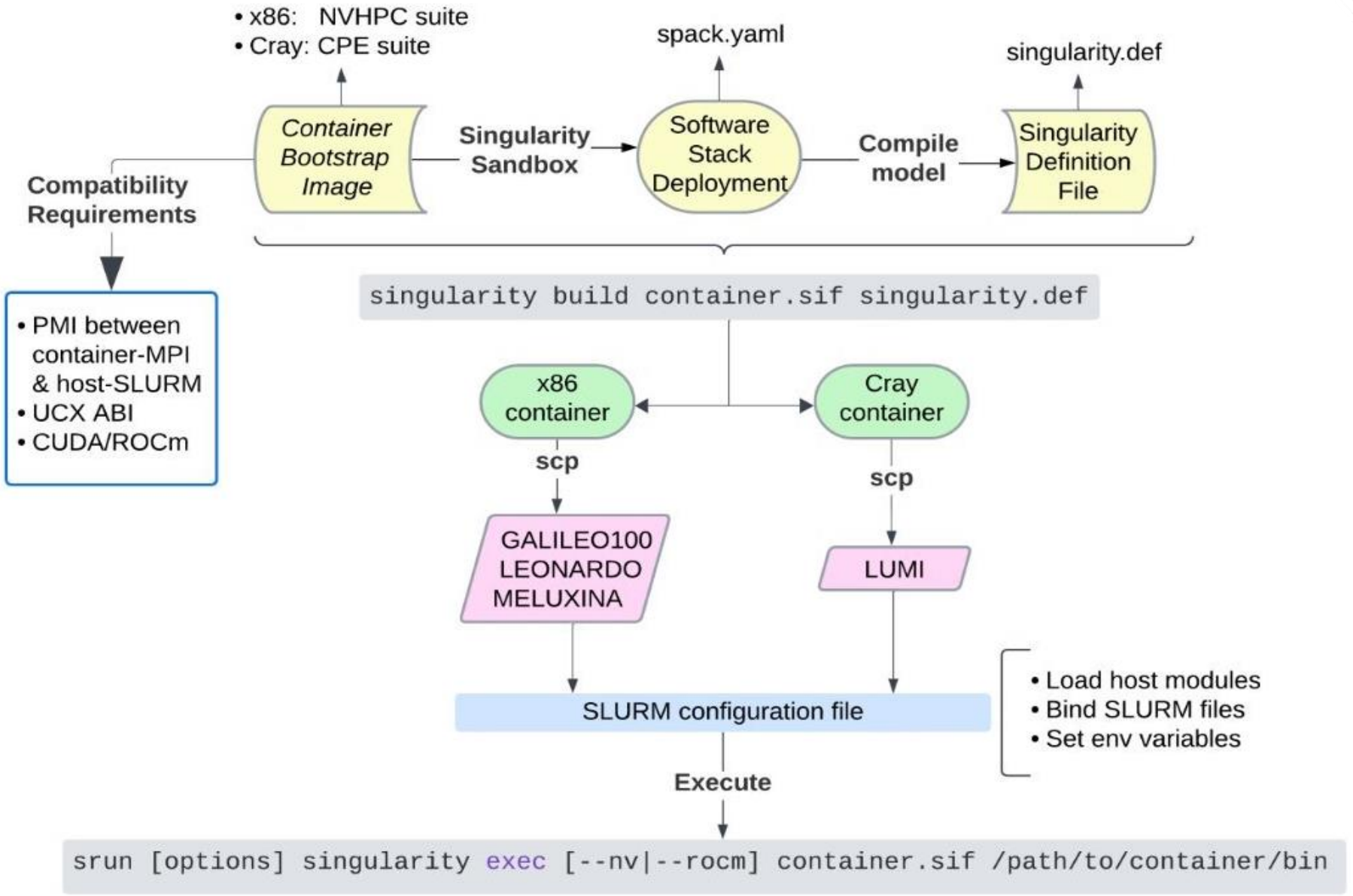
THE REQUIREMENTS

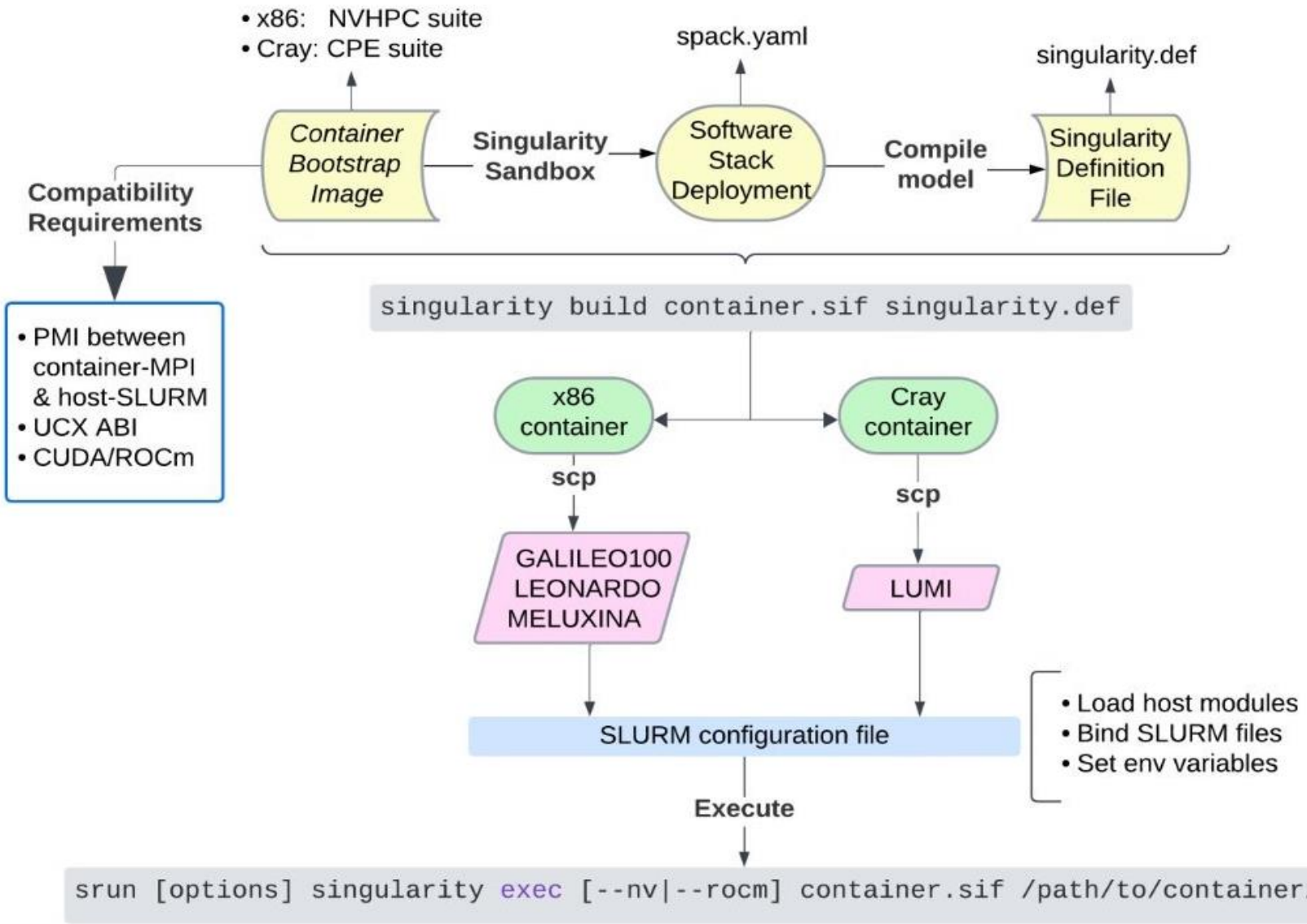
- *Integration in a HPC environment*
  - Integration with resource manager & host software stack
- *Run at performance wrt host*
  - Usage of bare metal capabilities (parallelism, infiniband)
- *Ensure usage of accelerators*
  - Correctly offloading to NVIDIA/AMD GPUs







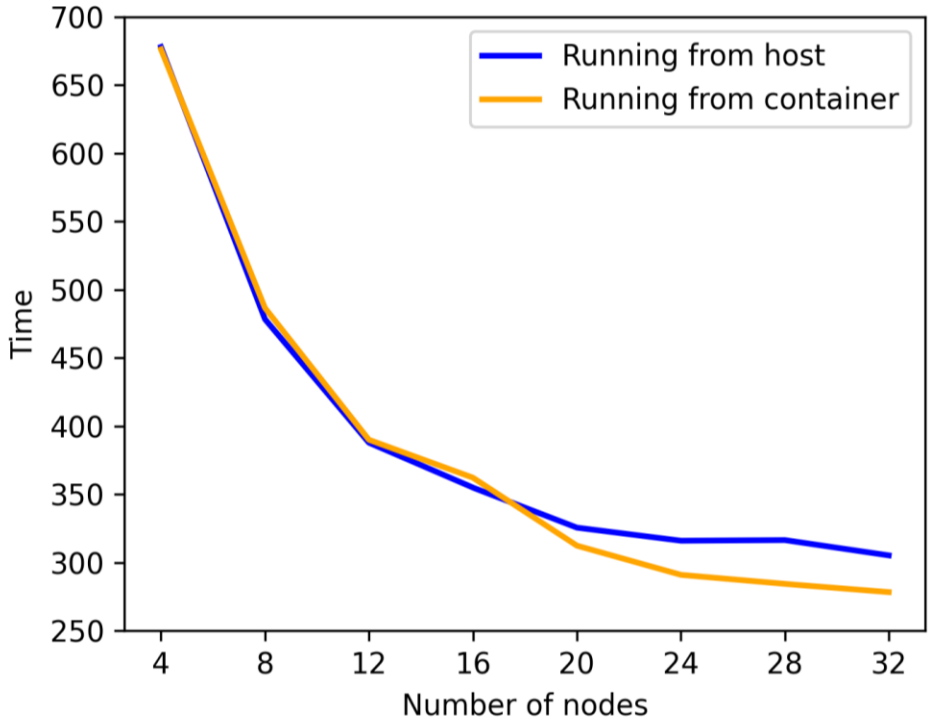

• x86: NVHPC suite
• Cray: CPE suite
spack.yaml
singularity.def
Container Bootstrap Image
Singularity Sandbox
Software Stack Deployment
Compile model
Singularity Definition File
Compatibility Requirements
• PMI between container-MPI & host-SLURM
• UCX ABI
• CUDA/ROCm
singularity build container.sif singularity.def
x86 container
Cray container
scp
scp
GALILEO100
LEONARDO
MELUXINA
LUMI
SLURM configuration file
• Load host modules
• Bind SLURM files
• Set env variables
Execute
srun [options] singularity exec [--nv|--rocm] container.sif /path/to/container/bin

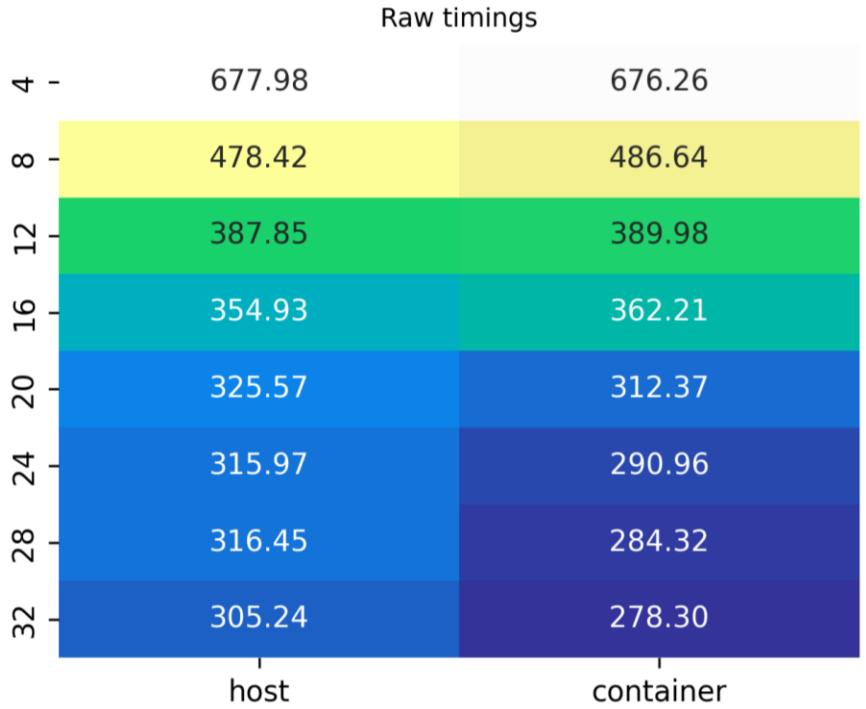


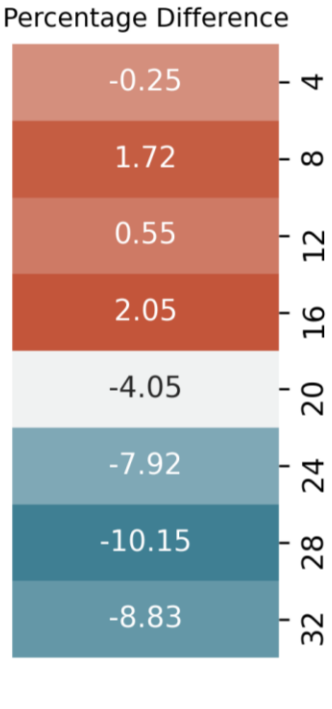
- *Build* on dedicated VMs/clusters allowing
- *Singularity to Docker* and viceversa
- Leveraging *base container images*:
  - *NVIDIA* images holds NVHPC suite (compiler, OpenMPI, CUDA)
  - *HPE* images holds CPE suite + help from *CSC* for ROCm integration
- *Reproducing host software stack:*
  - as faithfully as possible via *Spack*
- *Multistage Singularity builds:*
  - *lightweight* final container images
- *Singularity hybrid model* approach:
  - *Weak compatibility* requirements

Performance Assessment: ecWAM on GALILEO100

- Running on the O640 grid, CPU only, 48 threads per MPI task

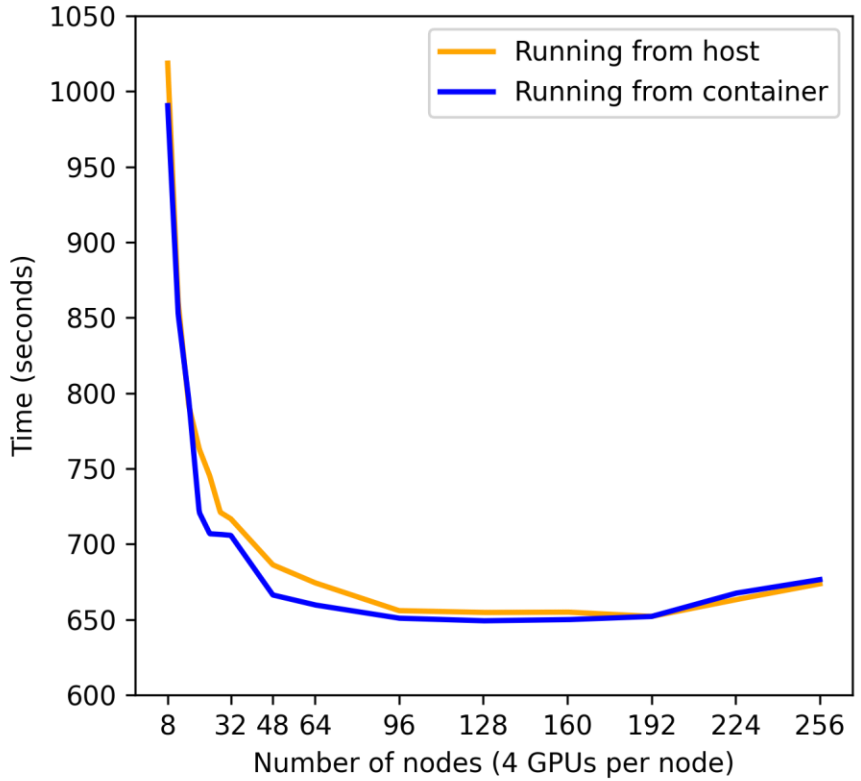






Performance Assessment: ecWAM on LEONARDO

- Running on the O1280 grid, GPU enabled, 4 MPI tasks per node, 8 threads per MPI task, 4 GPUs per node



Raw timings

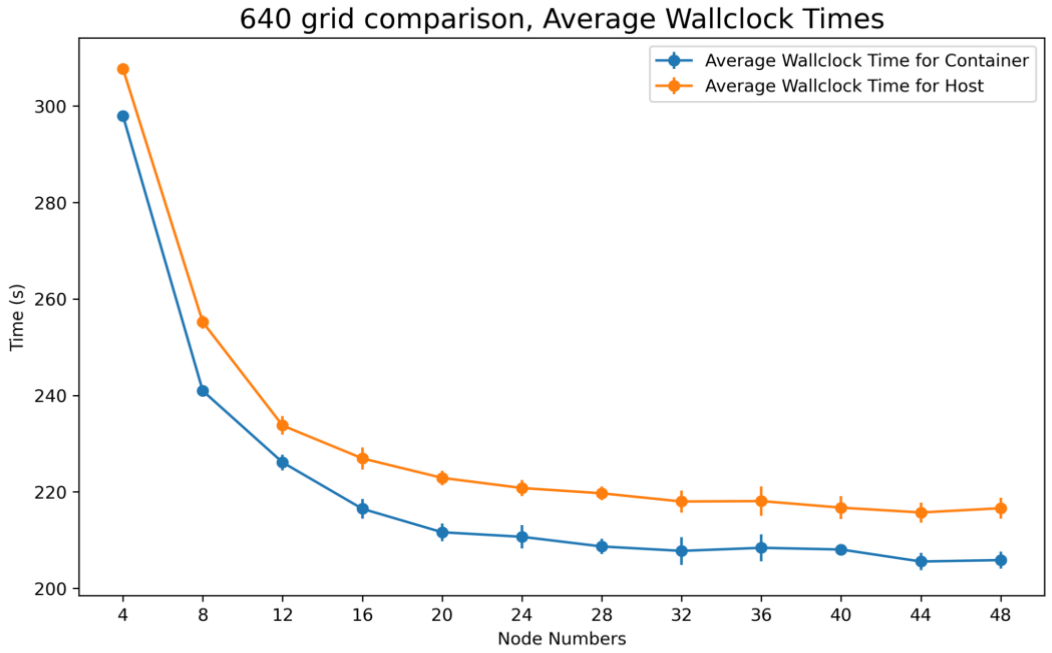
	host	container
8	1018.68	990.56
12	859.82	852.02
16	791.95	795.85
20	763.04	721.04
24	745.15	706.89
28	721.10	706.48
32	716.81	705.83
48	686.29	666.32
64	674.42	659.73
96	655.95	650.94
128	654.75	649.21
160	655.00	649.98
192	652.13	652.10
224	663.24	667.50
256	673.73	676.51

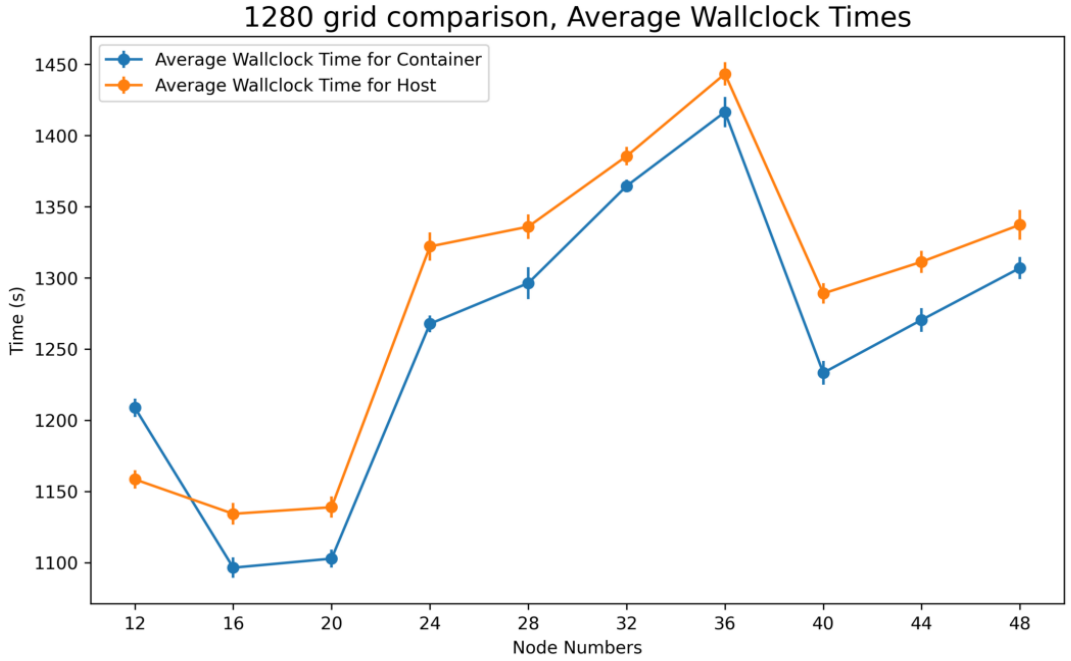
Percentage Difference

8	-2.76
12	-0.91
16	0.49
20	-5.50
24	-5.13
28	-2.03
32	-1.53
48	-2.91
64	-2.18
96	-0.76
128	-0.84
160	-0.77
192	-0.01
224	0.64
256	0.41

Performance Assessment: **ecWAM on MeluXina**

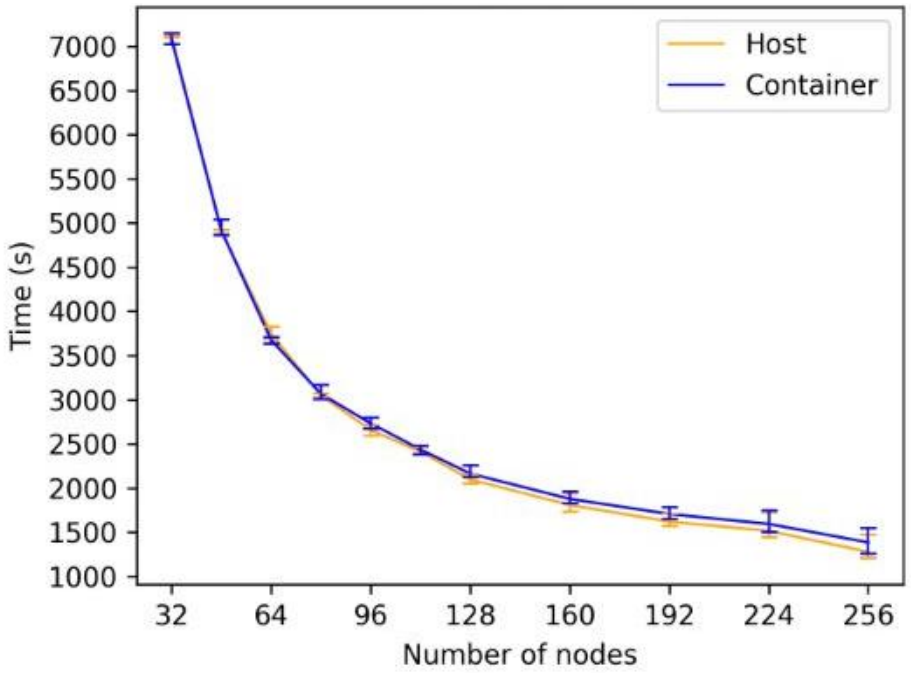
- Running on the 0640 & O1280 grid, GPU enabled, 4 MPI tasks per node, 8 threads per MPI task, 4 GPUs per node
- The ***very same*** container image deployed on LEONARDO was used

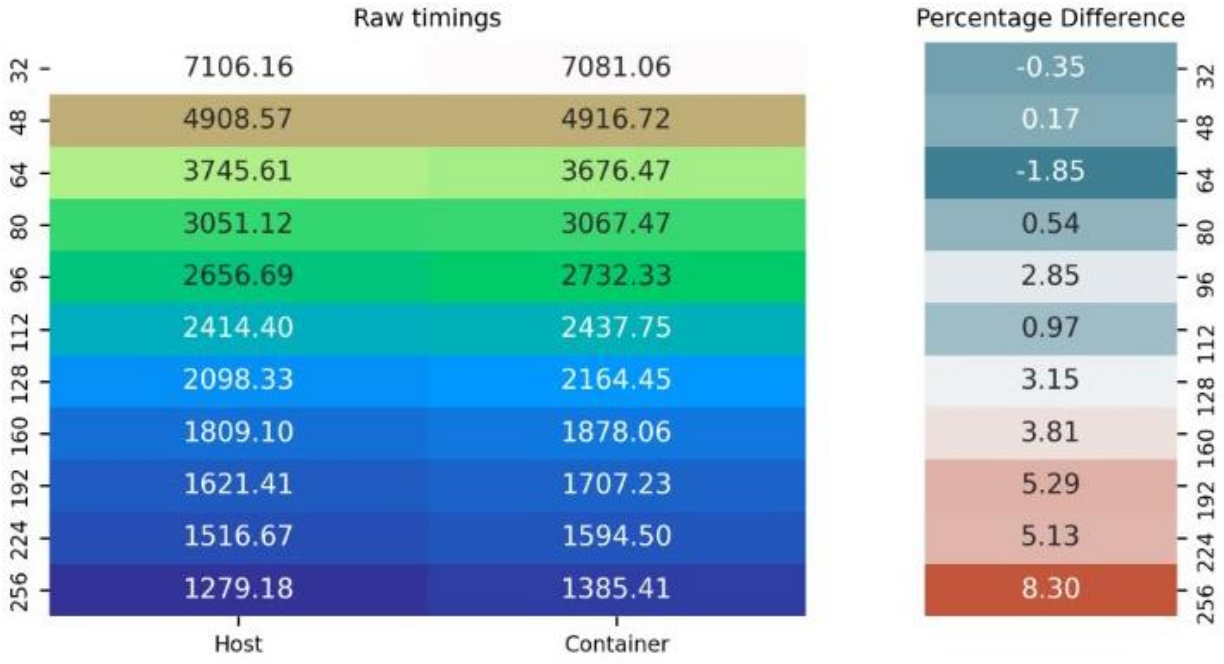




Performance Assessment: **RAPS20 on LEONARDO**

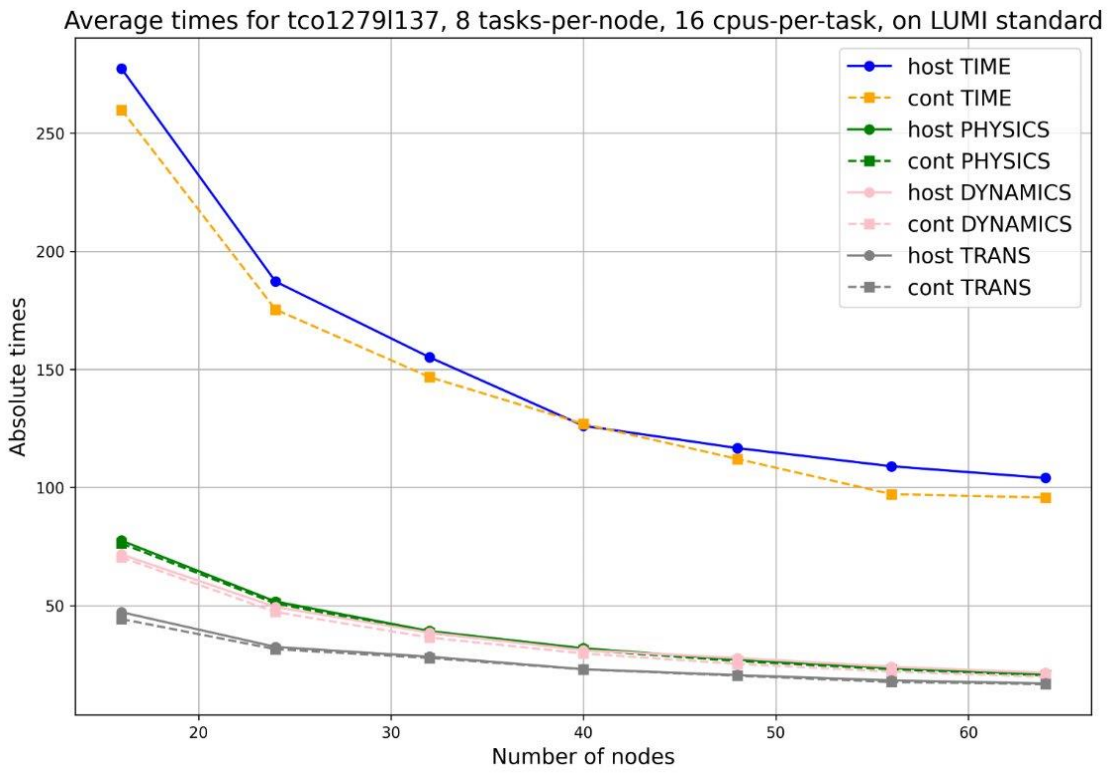
- Running *RAPS20* on tco2559l137, ecWAM on GPU, full nodes of LEONARDO booster partition

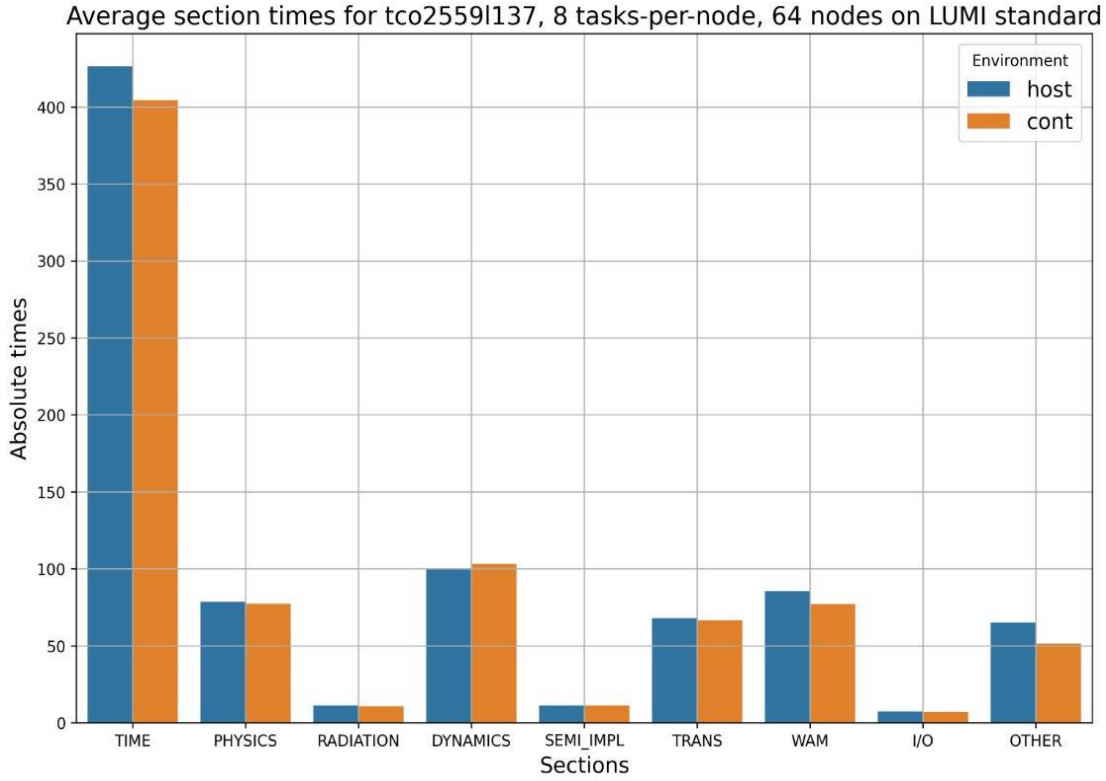




Performance Assessment: **RAPS21 on LUMI**

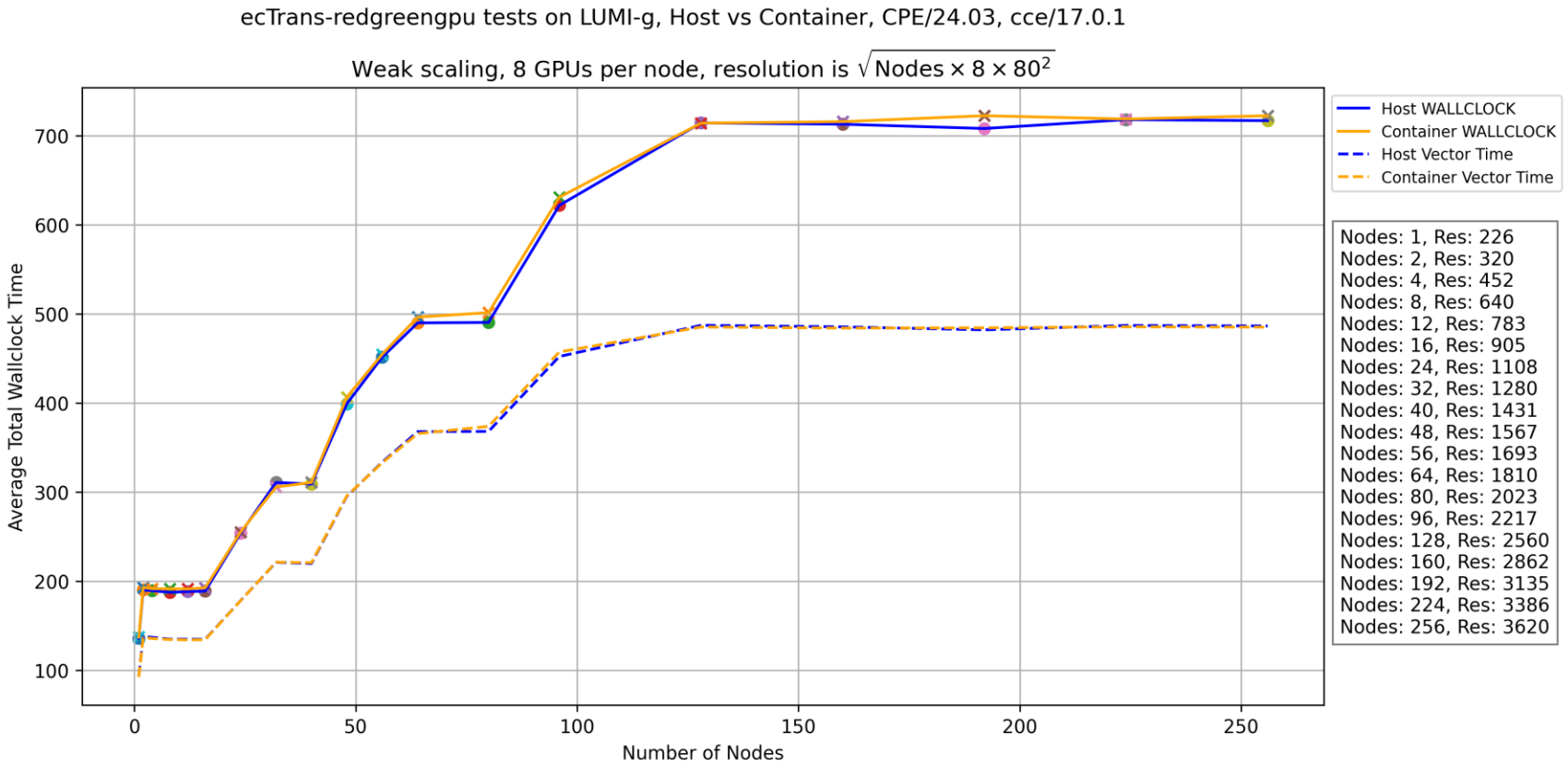
- Running *RAPS21* on full nodes of LUMI-C, tco1279l137 and tco2559l137 resolution





Performance Assessment: ecTrans on LUMI

- Running *ecTrans* (redgreengpu branch) on full nodes of LUMI-G, tco1279l137 and tco2559l137 resolutions



host_wt	cont_wt	%_diff_wt
135.84	137.28	1.06
190.28	193.04	1.45
189.39	191.57	1.15
187.48	191.52	2.16
188.29	191.35	1.63
188.92	192.57	1.93
253.93	255.21	0.50
310.82	305.76	-1.63
309.14	310.84	0.55
399.19	406.65	1.87
451.61	454.83	0.71
489.98	496.48	1.33
490.52	501.49	2.24
621.93	631.09	1.47
714.50	714.29	-0.03
713.02	715.91	0.41
708.21	722.55	2.03
718.14	718.84	0.10
717.00	722.53	0.77

Sanity Check: IFS_TEST_BITIDENTICAL=check

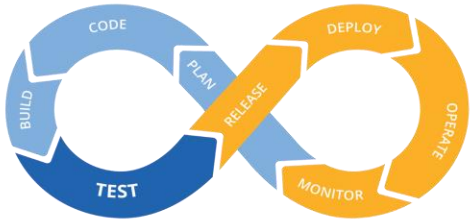
```
#------------------------------------------------------------------------------#
############################
### CONTAINER INTEGRATION ###
############################

  if [[ -n "${CONTAINER_ENV}" ]]; then
    if [[ ! -f "${CONTAINER_ENV}" ]]; then
        echo "Error: CONTAINER_ENV is set but file '${CONTAINER_ENV}' does not exist."
        exit 1
    fi
    echo "CONTAINER_ENV is set to '${CONTAINER_ENV}'. Reading variables..."
    source "${CONTAINER_ENV}"
    # Define the required variables (they must be declared in CONTAINER_ENV)
    required_vars=("CONTAINER_SW" "CONTAINER_FLAGS" "CONTAINER_GPU_FLAGS" "CONTAINER_IMAGE" "CONTAINER_BINARIES")
    # Check that all required variables exist (even if empty)
    for var in "${required_vars[@]}"; do
        if [[ ! -v $var ]]; then
            echo "Error: Required variable '$var' is not set in '${CONTAINER_ENV}'."
            exit 1
        fi
    done

    # Construct the LAUNCH command (as above without IFS_SOURCE_DIRS, LAUNCHER_OTHER_FLAGS to be appended at the end?)
    LAUNCH="${LAUNCHER} ${LAUNCHER_NPROC_FLAG} ${NPROC} ${LAUNCHER_NTHREAD_FLAG} ${NTHREAD} \
            ${LAUNCHER_MEM_FLAG} "

    # Concatenate container setup
    LAUNCH+="${CONTAINER_SW} ${CONTAINER_FLAGS} ${CONTAINER_GPU_FLAGS} \
            ${CONTAINER_IMAGE}"

    # Set the IFS executable to the container one
    # Extract basename (could be either ifsMASTER.SP or osmMASTER
    IFS_EXECUTABLE=$(basename "$IFS_EXECUTABLE")
    # Prepend RAPS bin folder of container
    IFS_EXECUTABLE=${CONTAINER_BINARIES}${IFS_EXECUTABLE}
  else
    echo "Launching Host installation"
    echo "Set variable CONTAINER_ENV to a file holding container variables to launch a containerized instance"
fi

#------------------------------------------------------------------------------#
```

Checked that the containerized RAPS21 image not only runs at performance, but runs meaningfully

- Ran the ctests of the *ifs-test suite* with a host-based installation
- Compared bitwise against a containerized version, with simple changes on the ifs-run script
- *Passed successfully*
- Failed using different compilers on host and container (as expected)
- Tested on LUMI, ECMWF ATOS, LEONARDO

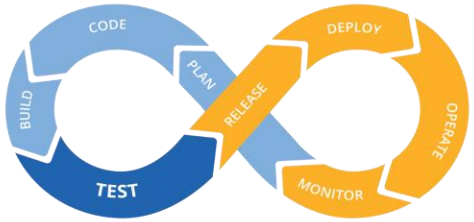
NEXT ON THE LIST

NEXT ON THE LIST



- ***CI/CD integration:*** container build and test workflows integrated on the CI/CD pipeline
- ***CW ---> Continuous Work!***
  - Containerize a new application takes time.
  - Containerize the same application for different versions/clusters does not take that much time

NEXT ON THE LIST

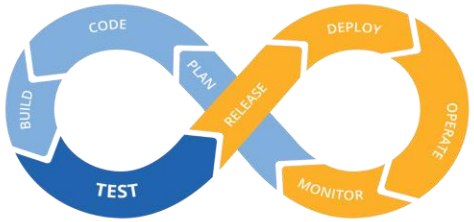


- ***CI/CD integration:*** container build and test workflows integrated on the CI/CD pipeline
- ***CW ---> Continuous Work!***
  - Containerize a new application takes time.
  - Containerize the same application for different versions/clusters does not take that much time

Ideally: *one container to rule them all*

- *Use the same container across all EuroHPC supercomputers*

NEXT ON THE LIST



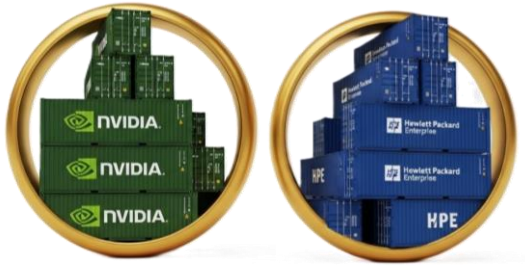
- ***CI/CD integration:*** container build and test workflows integrated on the CI/CD pipeline
- ***CW ---> Continuous Work!***
  - Containerize a new application takes time.
  - Containerize the same application for different versions/clusters does not take that much time

Ideally: *one container to rule them all*

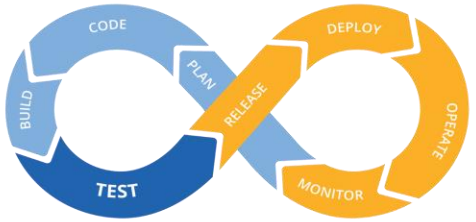
- *Use the same container across all EuroHPC supercomputers*

This project: *two sets of containers to rule some of them*

- *The same container is used across similar architectures*



NEXT ON THE LIST



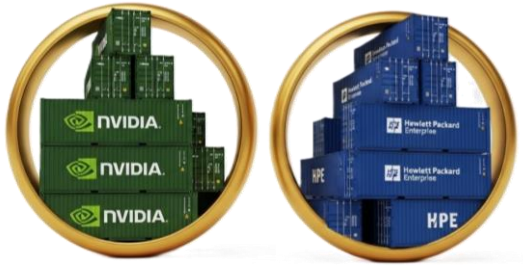
- ***CI/CD integration:*** container build and test workflows integrated on the CI/CD pipeline
- ***CW ---> Continuous Work!***
  - Containerize a new application takes time.
  - Containerize the same application for different versions/clusters does not take that much time

Ideally: *one container to rule them all*

- *Use the same container across all EuroHPC supercomputers*

This project: *two sets of containers to rule some of them*

- *The same container is used across similar architectures*



Practical?: *a set of container for each cluster*

- *Registry holding snapshots of applications and/or clusters software stack*

ECMWF

CINECA

Thanks for the attention!

QUESTIONS?