EC-EARTH Containerization

Abstract

The main goal for the EC-Earth team is to create and test, as a proof-of-concept, containerized versions of two simplified EC-Earth configurations: One for the current stable version EC-Earth3, and another for the next generation EC-Earth4, which is currently under development. EC-Earth3 is a well established version that is currently used for CMIP6 production runs, while EC-Earth4 is at an early development stage. Docker containers will be prepared for both configurations and test runs will be done on the Piz Daint machine at CSCS running under the Sarus container engine.

A short overview of the application

EC-Earth is an Earth System Model for providing climate information to climate services and to advance scientific knowledge of the Earth system, its variability, predictability and long-term changes resulting from external forcing. EC-Earth is composed of component models for different physical domains: ECMWF's IFS (EC-Earth4: OpenIFS) for the atmosphere, NEMO for the ocean, and the OASIS3-MCT coupler that provides data exchange between components. EC-Earth comprises further model components (e.g. for atmospheric chemistry, dynamic vegetation, carbon cycle, ...) that were not considered in this hackathon.

Break down of the containerization approach:

We installed Docker to build the container images for the models we wanted to containerize, we installed the EE edition/Community edition Ubuntu:

https://docs.docker.com/install/linux/docker-ce/ubuntu/

Download a docker image to start with, then we create a basic recipe to start adding all what we need (components, dependencies, etc) , we built the container and then executed the image first locally to check that everything was in place

Initially we are going to make use of GNU compilers to avoid the problem with the licenses, just for the initial approach , then we can replace these with other ones for optimization at a later stage.

We containerized two versions of the same model, the approach taken was more or less the same with small differences that we describe in the following sections.

Containerization of EC-EARTH 3

The overall idea is to use the container executable with all the code compiled, the container image that we will create will be based on the instructions of the following link:

https://sarus.readthedocs.io/en/latest/user/user_guide.html#native-mpi-support-mpich-based

We downloaded the sources of EC-earth3 and all related dependencies and we moved it into the container to make the initial tests and compile it.

How to create a container for EC-Earth3

Steps taken to build and run the container Prepare the source code and gribex tables:

```
> tar -czvf ec-earth.tar  ec-earth
> tar -czvf gribex_000370.tar.gz  gribex_000370
```

```
> sudo bash
```

Get the specific config files and Dockerfile

```
> git clone https://github.com/eth-cscs/ContainerHackathon.git
```

Build the container

```
> docker build -t ec-earth3-gnu-mpich:latest  -f ContainerHackathon/EC-Earth/Dockerfile.ecearth3 .
```

Check if the build is there:

```
> docker images
CONTAINER ID        IMAGE                   COMMAND             CREATED             STATUS              PO
b9bcc2ed0e92        ec-earth3-gnu-mpich     "/bin/bash"         7 minutes ago       Up 7 minutes
```

You can run an interactive section

```
> docker run --name ec-earth3 --rm -it ec-earth3-gnu-mpich /bin/bash
```

To see the running containers you can call:

```
> docker ps
CONTAINER ID        IMAGE                   COMMAND             CREATED             STATUS              PO
3b2f4e249701        ec-earth3-gnu-mpich     "/bin/bash"         39 minutes ago      Up 39 minutes
```

If you need to open an interactive session in a running container:

```
> docker exec  -it ec-earth3 bash
```

To simplify the exercise we chose ifs+amip configuration We first compiled the sources in the cto check all references were correctly configured, we changed the cfg files to match the target architecture and we also prepared the inidata files needed for making a basic run (ifs+amip). After the compilation was achieved, we deployed the container to the Piz-Daint cluster to start making some test runs and execute a basic test case, also copied the inidata files.

The run script was modified to prepare the rundir folder

save image to a tar file

```
> docker save ec-earth3-gnu-mpich > ec-earth3.tar
```

to import the image in daint

```
module load sarus
srun -C gpu -N1 --reservation=esiwace_1 sarus load <path-to-tar> <image-name>
```

```
> sarus images
REPOSITORY                TAG          DIGEST         CREATED               SIZE         SERVER
load/library/ec-earth3    latest       5cc1420b7d37   2019-12-05T09:23:53   387.25MB     load
```

Target Test chose TL159L62 : 19900101 + 1 day

Containerization of EC-Earth 4

EC-Earth4 comes with a different configuration and build system (compared to EC-Earth3), hence, the containerization approach is a bit different. Nevertheless, many dependencies are the same and the basic container building steps are similar. Ubuntu is used as the underlying base system and another layer with the configuration and build system scripts has been added.

Most EC-Earth users compile the model with Intel compilers and link to Intel MPI and the model is therefore very well tested in Intel environments. However, for the containerization, the Intel compiler and MPI library were not chosen. This is partly due to difficulties of handling licenses in containers, and also for the benefit of adding another well tested platform to EC-Earth. For that reason, the GNU compiler collection (GCC) and MPICH were chosen.

EC-Earth4 build configuration was relatively straightforward, once the appropriate Ubuntu packages were identified for all dependencies. For MPI, three options were actually tested: OpenMPI from Ubuntu packages, MPICH from Ubuntu packages, and MPICH version 3.1.4 from sources. The latter option due to the recommendations made for Sarus containers in the CSCS environment at Piz Daint.

After building an EC-Earth4 Docker container, the following test cases were run on a local machine:

- TL21L19 with GCC+OpenMPI (Ubuntu packages)
- TL21L19 with GCC+MPICH (Ubuntu packages)
- TL21L19 with GCC+MPICH 3.1.4 (from sources)
- TL159L91 with GCC+MPICH 3.1.4 (from sources)

All tests completed successfully. As a supplementary step, the Docker build configuration was added as one of the "supported" platforms in the EC-Earth4 configurations. The Docker image was transferred to Piz Daint and imported to Sarus. Due to the licensing issue with OpenIFS, no Docker repository was used and the image was exported from Docker in the form of a tar file and directly copied from machine to machine. Import and conversion to Sarus posed no problems at all. Running the converted image on the HPC is notably different from testing the original Docker image on the local machine. Running directly under Docker, the MPI is exclusively dealt with within the container and no workload management system is involved. On the HPC system, the interaction between MPI, container engine and workload manager needs to be considered. Particularly, there is one container started for each of the MPI ranks. This has an implication for the construction of the application container.

In its original form, the EC-Earth4 run script would call mpirun by itself within the container. On Piz Daint, the Hydra Process Manager used by MPICH would then detect the presence of Slurm (because of the env variables transferred from the host by Sarus), and try to call srun (which obviously is not present within the container). In order to run, we defined the environment variable `HYDRA_LAUNCHER=fork` to set Hydra to use a different launcher than Slurm's srun. This allowed us to Successfully ran test case TL21L19 with GCC+MPICH3.1.4 on a single container with 4 MPI ranks inside the container. However, parallel execution with multiple containers is out of reach for this image.

For testing at scale, the EC-Earth4 container was stripped of all initialisation tasks. Hence, the container was just providing the OpenIFS executable, leaving out any runtime initialisation. This allows to run one container instance per MPI process, as needed. A small script was developed for the initialization of the run directory, providing any initial data and namelist files needed. This script was run manually just before queueing of the job, which was sufficient for the experimental nature of this exercise. However, this needs further development for production-like scenarios. In this setup, Slurm will run the simulation and scale it to multiple containers.

With that setup, the smallest test case (TL21L19) was successfully run for a couple of time steps, using 4 MPI ranks (on 5 nodes). A series of tests was then run with the larger TL159L91 grid, simulating a ten day period. Elapsed time was measured for these tests for a varying number of nodes/processes:

- TL159L91, 10 days, 96 processes on 8 nodes: 18:26 elapsed job time
- TL159L91, 10 days, 192 processes on 16 nodes: 9:47 elapsed job time
- TL159L91, 10 days, 384 processes on 32 nodes: 5:05 elapsed job time

These tests conclude the containerization exercise for EC-Earth 4.

Conclusions

The experience was very positive, since at least in BSC were focusing on Singularity mostly, having the chance to test and see in action a different technology like Docker combined with Sarus was an interesting experience. EC-Earth 4 was successfully containerized and run at scale with a simplified test case (atmosphere-only) on Piz Daint. Containerization as such and deploying on the HPC was quite smooth and it was possible to focus, during the Hackathon, on the more involved question of designing a containerized application.

In general, the following steps appeared as crucial:

- Provide a development environment to build containers
- Port the application to the container (i.e. compiling inside the container)
- (re-)design the containerized application, i.e. split the tasks of the workflow
- Define the data flow in terms of files in directories (defining the container bind mounts)
- Test container locally in Docker
- Transfer and convert the Docker image to the target system (Sarus image on Piz Daint)

- Deal with multi-prog MPMD configuration

To a large degree, the containerization effort turned out to be a porting exercise. Importantly, this extends beyond the basic model build (i.e. compilation), and includes the porting of the workflow for the chosen test case. This emerged as a particular difficult task for EC-Earth 3, which includes many references to the complete CMIP6 work flow in the runtime environment (reading of forcing files, post-processing of output data). Stripping the experiment workflow of all those references and their dependencies is a complex task to deal with in limited time of the Hackathon.

Another important lesson is about the modularisation needed to containerize the model. Particularly running at scale on Piz Daint required to split up the tasks that are usually coupled within a run script. A number of hidden dependencies emerged while decoupling these tasks. Similarly, the explicit dependencies dictated by the container bind-mounts required some rethinking of the data workflow.