# How to Run Containers on the Intel Cluster

Antonio Carta

March 25, 2020

This guide describes my personal experience porting a python project from a single node to the Intel cluster `hpc2019.unipi.it`.

The use case presented here is a simple example of parallel computation. The python script that we want to run implements the training of a deep learning model, and we would like to run a model selection in parallel, with several instances of the same script running on different compute nodes of the cluster, each one with different parameters. This approach does not require communication between the jobs or communication between compute nodes since each configuration runs on a single node. If you are interested in multi-node training, you can look up horovod, a deep learning library that allows you to easily scale to multiple nodes.

In order to run the python script, the operating system must have a set of scientific libraries installed. Different users may require different version of the libraries, possibly incompatible between them. Since installing and managing all the dependencies on the cluster machines would be unfeasible, we will encapsulate the dependencies inside a Charliecloud container, which is a containerization mechanism designed for HPC systems.

## 1 Outline

The process of porting the experiment is articulated in 4 steps:

- export of your python environment
- creation of a charliecloud image
- creation of a slurm submission script
- scheduling of parallel jobs on the cluster with slurm

## 2 Creation of a Charliecloud Container

To create a container to run you python scripts, you must export your python environment. This will allow to easily install all the dependencies inside a container. You can use the command:

```
[language=bash]
        conda list -e > requirements.txt
```

Using the requirements, we can create a Dockerfile, which will be used to create our container and install the relevant dependencies. The Dockerfile is based on the Anaconda Docker image.

`Dockerfile`

```
FROM continuumio/miniconda3
ADD requirements.txt /requirements.txt
RUN conda install python=3.8
RUN conda install --file requirements.txt
RUN conda install pytorch torchvision cudatoolkit=10.1 -c pytorch
RUN pip install fbpca
WORKDIR /home/carta
```

Notice that despite having all the environment saved in the `requirements.txt` file, I had some problems during the installation and therefore I had to install pytorch separately after the other libraries.

The next step is the creation of the Charliecloud container from the Dockerfile. In my case, the container is called `antonio/lmn`

```
ch-build antonio/lmn .
ch-builder2tar antonio/lmn .
ch-tar2dir ./antonio.lmn.tar.gz .
```

For some unknown reasons, Charliecloud ignores the ENV directives defined in your Dockerfile. If you want to define additional environment variables for your container you can create an environment file that defines your custom variables. This file must be passed as an argument when running the container, as we will see below. In my case, I added to the PYTHONPATH the folder where my python library resides:

`config.env`

```
PYTHONPATH=.:/home/carta:/home/carta/cannon
```

Now we can easily run our python script with the command

```
ch-run --set-env=config.env
    --cd=/home/carta/docker_repo antonio.lmn
   /opt/conda/bin/python /home/carta/repo/main.py
```

where:

- `--cd` changes the directory

- `--set-env` set the environment variables

The container executed with `ch-run` is running on the head node.

# 3 Scheduling Jobs on the Cluster

Since we do not have direct access to the compute nodes, we must use `slurm` to schedule the jobs in parallel. `slurm` will take care of the allocation of the resources in a fair way for all the jobs.

To do this, we need to create a submission script `submit.sh`, which will be run by slurm:

`submit.sh`

```
#!/bin/bash
#
#SBATCH --job-name=/home/carta/docker_repo/docker_latest/test
#SBATCH --output=test.txt
#SBATCH --nodes=1
#
#SBATCH --array=1-8

ch-run --set-env=config.env
    --cd=/home/carta/docker_repo antonio.lmn
   /opt/conda/bin/python /home/carta/repo/main.py
   -- --id=$SLURM_ARRAY_TASK_ID
```

There are a couple of things to notice about the submission script:

- Comments prefixed by `#SBATCH` are used to pass options to the `sbatch` command. You can read more about `sbatch` options on the manual.
- The use of job arrays to easily manage all the jobs together.
- Specification of the resources for each job (`--node`).
- The argument `$SLURM_ARRAY_TASK_ID`, set by slurm to match the job id. In this case it is directly passed to the python script, which will use it to change the model's configuration.

To submit the jobs to the scheduler, you must run the command

```
sbatch submit.sh
```

You can check the status of your job with `squeue`, where you should see a list of your jobs in execution.

# 4 Final Notes

I hope you will find this helpful to start running your own experiments on the cluster. While this document describes my personal experience, using the steps described here and the official documentation it should be easy enough to adapt this procedure to your own use case.

# 5 Useful Links

These are some resources that you may find useful:

**Docker docs** `https://docs.docker.com/`

**Charliecloud docs** `https://hpc.github.io/charliecloud/index.html`

**slurm docs** ] `https://slurm.schedmd.com/documentation.html`