Introduction to supercomputing for AI

High performance computing (HPC) resources can be used to accelerate AI workflows. The EuroHPC Joint Undertaking (JU) offers free access to such resources to SMEs as well as larger companies. In this hands-on, you will learn:

- What is an HPC resource and how it is different from a cloud environment;
- How to connect to a cluster and explore resources;
- How to load necessary software and how workloads are managed;
- How to run a demo AI workflow based on Singularity.

⚙ Prerequisites

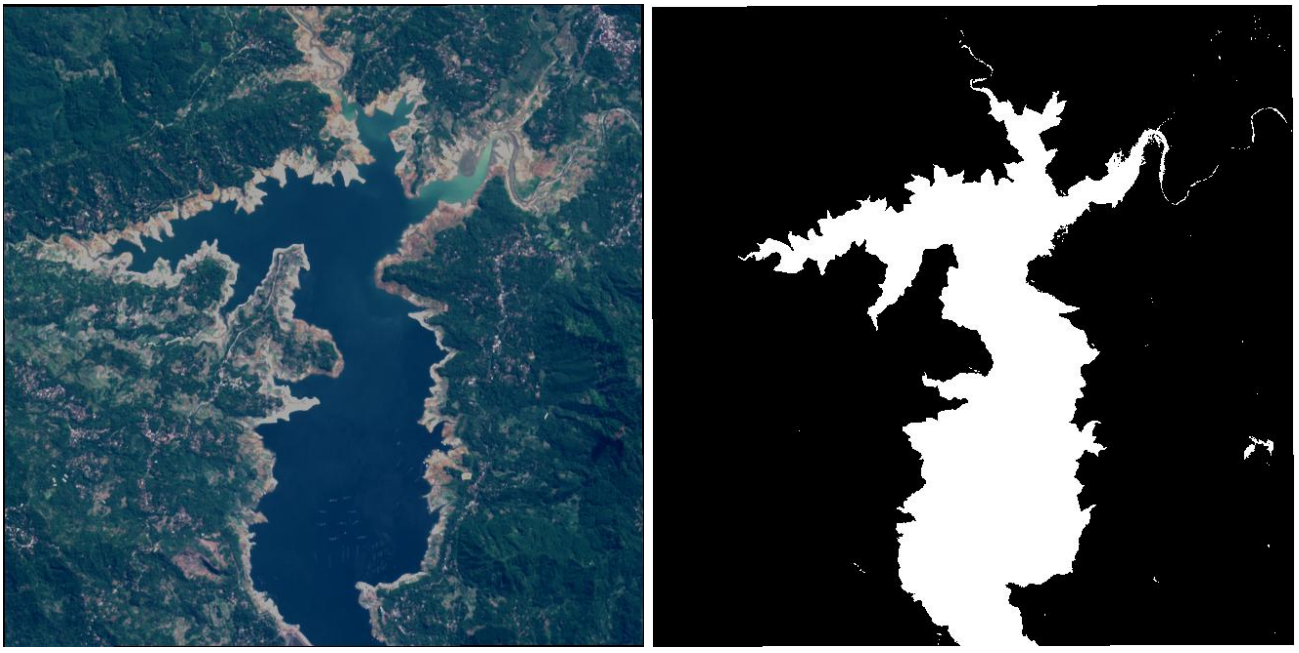
You will need to have credentials to access the PDC cluster. A working SSH client is needed: it is included on macOS and most Linux flavours; it is also available on Windows in the Powershell or under the Windows Subsystem for Linux (WSL).

Who is the course for?

This course is intended for data scientists that want to take advantage of higher computing power to perform their workflows. Some degree of familiarity with a command-line shell is recommended, but no expertise is required. No previous knowledge of supercomputing environments is required.

About the course

We will train a Unet model to be able to recognise water in satellite pictures. The source data comes from the Sentinel-2 satellite. Let us take the following picture for example:



The left panel shows the original image from Sentinel-2, whereas the right one was manually masked to highlight water regions.

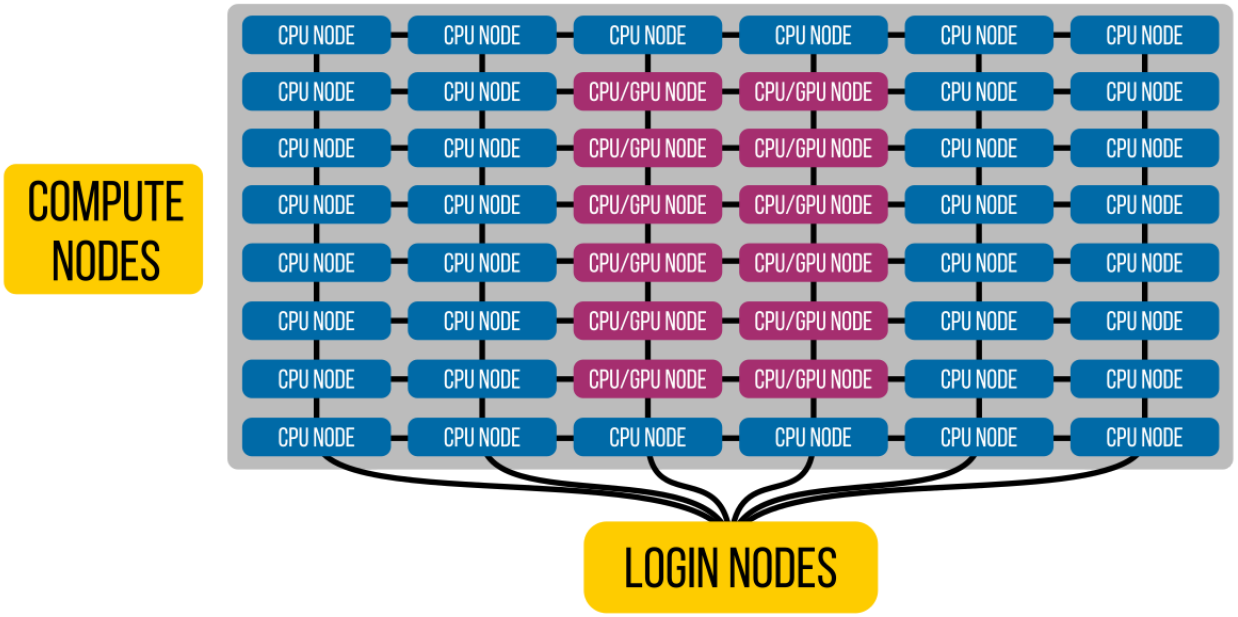
The objective is to train a Unet model to perform this automatically. The source code can be found at this repo. The example is based on Tensorflow and will be run using Singularity. The structure of the example is the following:

```
./supercomputing4ai_demo
├── build_singularity.def
├── images
│   └── generated-images
├── models
│   ├── serving
│   │   └── main.py
│   └── unet
│       ├── data
│       │   └── water
│       │       ├── Images
│       │       └── Masks
│       ├── main.py
│       └── result
│           ├── models
│           └── training
└── README.md
```

The `models` subfolder contains the model to be trained (`unet`) and the inference code (`serving`). Under `data`, the training dataset can be found, with the `Images` being some satellite images and `Masks` being the water-covered areas in those images. Upon running `main.py` in the `unet` folder, a Unet will be trained, producing a set of weights in the `models/` subfolder and training statistics (binary cross-entropy loss and accuracy). Inference is then performed with the `models/serving/main.py` script, which takes as an input an image and generates a mask of the water parts.

What is a supercomputer and how is it different from cloud?

A supercomputer (or *cluster*) is a high-performance computing infrastructure. The general idea is to have many (thousands) of machines, called *nodes*, working in parallel to increase performance. Each node can have one or several (usually two) multi-core CPUs, local RAM, and possibly accelerators such as GPUs. What distinguishes a cluster from a simple collection of computers is the high-speed connection between all the nodes, called *interconnect*. The speed and density of connections between nodes makes it possible for the nodes to seamlessly communicate and solve a problem in parallel. The general architecture of a cluster looks like below:



Schematics of a typical cluster.

The user accesses the cluster through the *login nodes*, which act as a gateway between the external world and the compute parts. They should not be used to run calculations. Usually a parallel file system is provided (such as Lustre), with independent servers (not shown in the figure) that manage distributed access from all the nodes. Moreover, each node usually has some temporary storage called *scratch* for fast access during computation. On Dardel specifically, each CPU node has two AMD EPYC CPUs with 64 physical cores each and RAM ranging from 256 GB to 2 TB. Each GPU node has one AMD EPYC and 4 AMD Instinct MI250X GPUs, each containing two GCDs (compute chips in AMD lingo); thus, each node has 8 GPUs.

How is HPC different from cloud?

While they both provide access to high-performance computational resources, a few important differences exist between cloud and HPC systems:

- An HPC resource is shared among several users; usually users do not have root access, unlike cloud where each user has their own sandboxed environment;
- Each workload pushed by a user (called *job*) contends for the available resources; usually, a queueing system is managed by a *job manager* (more on that later) which decides which jobs are going to run to maximise cluster usage. This means that a job may have to wait for resources before it is allowed to run.
- (Through EuroHPC) the cost of HPC does not scale with amount of resources used; with cloud it does.
- Engineering/scientific applications usually expect an HPC-like environment which is harder to set up in the cloud;
- Access to bare metal vs virtual machines.

Connecting to a HPC resource

If using SSH keys, once the keys are created and uploaded on the PDC interface, entering the cluster is as simple as:

```
$ ssh -Y <username>@dardel.pdc.kth.se
```

Which should get you into the PDC supercomputer. The `-Y` flag is used to be able to open graphical windows on the supercomputer, e.g. to visualise images. This will work only if you have a running local X server (if you are on Linux/WSL, you most likely do). Alternatively, you may choose to use Kerberos as an authentication method. To do that, you first need to ask for a Kerberos ticket:

```
$ kinit -f <username>@NADA.PDC.KTH.SE
```

After that, the SSH command looks like the following:

```
$ ssh -o GSSAPIAuthentication=yes -Y <username>@dardel.pdc.kth.se
```

More information about Kerberos can be found at this address.

⌨️ Type-Along

Let us check on which node we ended up. The name of the machine can be checked with the `hostname` command:

```
$ hostname
```

We can get a sense of the size of Dardel by using the `sinfo` command:

```
$ sinfo -s
PARTITION AVAIL  TIMELIMIT   NODES(A/I/O/T) NODELIST
gpu          up 1-00:00:00        49/9/4/62 nid[002792-002853]
main         up 1-00:00:00  604/256/112/972 nid[001012-001531,001756-001816,001818-
001819,001821-001896,001898-002007,002009-002023,002552-002567,002588-002759]
scania       up 4-00:00:00    22/187/15/224 nid[001532-001755]
scania-hf    up 4-00:00:00          0/3/1/4 nid[000011-000014]
memory       up 7-00:00:00        34/8/0/42 nid[000101-000118,001772-001779,002552-
002567]
shared       up 7-00:00:00        27/5/0/32 nid[001000-001011,002568-002587]
long         up 7-00:00:00        76/4/0/80 nid[001800-001819,002588-002647]
eggnog       up 7-00:00:00          4/0/0/4 nid[002536-002539]
supernova    up 14-00:00:0         5/6/5/16 nid[001817,001820,001897,002008,002540-
002551]
```

E.g. the `main` partition has 972 nodes, each containing 128 cores.

A general sense of the amount of work load can be gained with the `squeue` command, which shows all the jobs (running, queued):

```
$ squeue
```

Move between folders, ls, transferring to/from local storage

Upon logging in, you should be in your “home” folder, as reported by the prompt:

```
fiusco@login1:~>
```

where `login1` is the name of the host (in this case, the login node) and `~` represents the home folder. The full path of this directory can be printed using the `pwd` command (**p**rint **w**orking **d**irectory):

```
$ pwd
/cfs/klemming/home/f/fiusco
```

The contents of a directory can be listed with the `ls` command:

```
$ ls
Private  Public  spack-user
```

The `cd` (**c**hange **d**irectory) command can be used to navigate the filesystem.

Moving files/folders from/to the cluster can be achieved via the `scp` command to be run locally (i.e. not on the cluster):

```
$ scp [-r] /path/to/local/source user@dardel.pdc.kth.se:/path/to/destination
```

The optional `-r` flag is used to indicate recursive copying of whole folders and their contents.

⌨ Type-Along

In this workshop, our working folder will be in `/cfs/klemming/projects/supr/bustestingshared/`. You can create your own folder:

```
$ cd /cfs/klemming/projects/supr/bustestingshared
$ mkdir my_name
```

We can now clone the repository containing the material for the workshop:

```
$ cd my_name
$ git clone https://github.com/ENCCS/supercomputing4ai_demo
$ cd supercomputing4ai_demo
```

Available software and modules

A variety of software, libraries and compiler toolchains is usually available on most HPC resources. Usually a module system is used to access software provided by the cluster. A list of all available modules can be printed with the following syntax

```
$ module avail
```

The presence of a particular piece of software/library can be queried with:

```
$ ml spider <name>
```

This command will return all available versions of a module (if present), as well as specific instructions to load it if needed. The module can then be loaded with `module load <module_name(s)>`. For example, the *Julia* runtime can be loaded with:

```
$ ml PDC julia
```

All currently loaded modules can be printed with:

```
$ module list
```

⌨ Type-Along

We will use a Singularity container to get a Tensorflow environment. For that, we need a few modules:

```
$ ml PDC singularity
```

Queueing jobs and running a Singularity container

Most supercomputers use a system called SLURM to manage jobs and maximise cluster utilisation. There are two main workflows:

- Interactive mode, in which the user asks for resources, logs into the compute nodes and runs all they need to run
- Batch mode, in which the user prepares a `submit` script (usually Bash) that contains all the instructions to be executed. The script is then placed in the queue and runs without any further inputs.

Interactive resources can be requested with the `salloc` command in the following fashion:

```
$ salloc -n <n_cores> -t HH:MM:SS -A <allocation_number> -p <partition>
```

Most parameters are self-explanatory. `-t` is the wall time, `-A` is an allocation dependent on the project. `-p` is required by some clusters that have different types of nodes, e.g. Dardel has a set of nodes reserved for interactive use on a partition called `shared` and a set of nodes with GPUs (`gpu`). There are many other options that are explained here. Once the requested resources are granted, a MPI job can be executed with `srun -n <n_cores> my_command`; alternatively, the user can also ssh directly into the allocated node.

⌨️ Type-Along

Let's book a node with a GPU to run our script:

```
$ salloc -n 1 -t 00:50:00 -A pdc-bus-2024-8 -p gpu --gpus=1
```

Once we get our node, we can train our model:

```
$ srun -n 1 singularity exec --rocm -B ./models:/models
/cfs/klemming/projects/supr/bustestingshared/ENCCS/rocm_tensorflow/ python
models/unet/main.py
```

The `-B` flag is used to bind a directory on the filesystem to a directory in the container and the `--rocm` flag is used to expose the GPU to the container. The `rocm_tensorflow` folder contains a pre-built container with the necessary Python packages. We can inspect the accuracy and losses in the `results/training` folder, e.g. copying the images to our own laptop. Once the network is trained, we can perform inference:

```
$ srun -n 1 singularity exec --rocm -B ./models:/models,./images:/images
/cfs/klemming/projects/supr/bustestingshared/ENCCS/rocm_tensorflow python
models/serving/main.py water_body_17.jpg
```

You can open another local terminal (i.e. on your computer) and copy the images back for local visualisation:

```
scp -r
<yourusername>@dardel.pdc.kth.se:/cfs/klemming/projects/supr/bustestingshared/<yournam
/local/path
```

⌨️ Bonus - Batch mode job

It is often easier to submit a job to the general queue rather than waiting for an interactive resource. As an example, we can book two nodes and print their host name. Let's create the following submission script and call it *submit_test.sh*:

```
#!/bin/bash -l
#SBATCH -J test # Job name
#SBATCH -t 00:02:00 # Wall time
#SBATCH -A pdc-bus-2024-8 # Allocation number
#SBATCH -p main # Partition - in this case the normal (non-GPU) is fine
#SBATCH --nodes 2 # We want two different nodes
#SBATCH --ntasks-per-node=1 # We want to book only one core on each node - We need
just the hostname after all :)
#SBATCH --mail-type=ALL # Get an email when the job starts and when the job ends

# Now for the actual instruction to be executed
srun hostname > out.txt 2>&1 # Black magic to redirect both stdout and stderr to
the same file
```

We can now submit our short job to the queue:

```
$ sbatch submit.sh
```

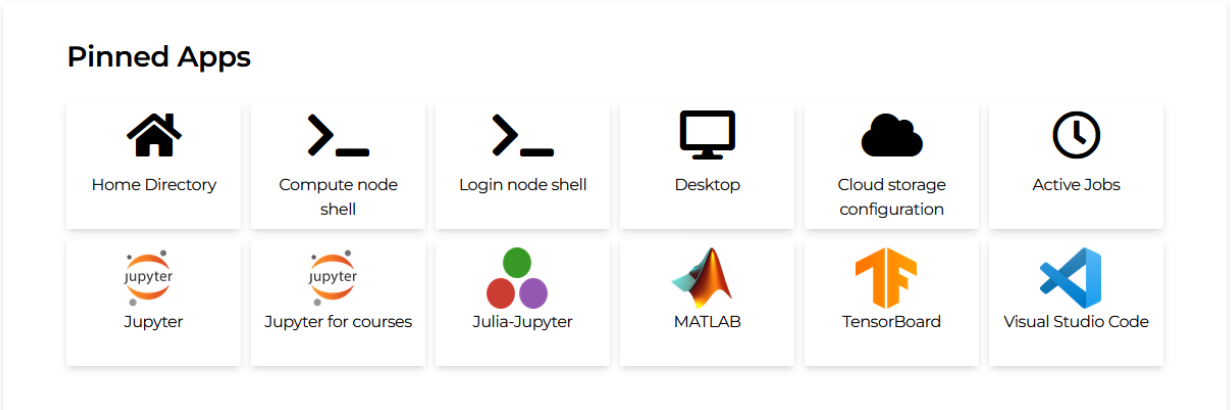
We can check the status in queue with:

```
$ squeue -u <username>
```

Further resources

While we have mainly looked at the “classic” way to login and interact with the cluster, newer systems have other GUI-based interfaces that you can use. For example:

- Dardel can use Thinlinc, which gives you a remote desktop on the cluster with the possibility to book nodes and launch apps directly there.
- LUMI has a web interface with a dashboard to launch common applications such as VS Code, Jupyter, Jupyter Julia. Each app has a launcher to book resources and run it on a compute node, with or without GPU.



In general, it is also possible to use Jupyter on any HPC resource with plain SSH, as detailed in our tutorial.

Quick Reference

Instructor's guide

Why we teach this lesson

Intended learning outcomes

Timing

Preparing exercises

e.g. what to do the day before to set up common repositories.

Other practical aspects

Interesting questions you might get

Typical pitfalls

See also

Further introductory material can be found on the Introduction to LUMI and HPC carpentry pages.

Credits

The lesson file structure and browsing layout is inspired by and derived from work by CodeRefinery licensed under the MIT license. We have copied and adapted most of their license text.

Instructional Material

- **Attribution** - You must give appropriate credit (mentioning that your work is derived from work that is Copyright (c) ENCCS and individual contributors and, where practical, linking to https://enccs.github.io/sphinx-lesson-template), provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **No additional restrictions** - You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

With the understanding that:

- You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.
- No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

Software

Except where otherwise noted, the example programs and other software provided with this repository are made available under the OSI-approved MIT license.