Human Brain Project

# HPAC Tutorial Hands-on Session

## Pollux 101 – OpenStack VM resources

In this tutorial, we will cover the introductory topics of accessing the VM resource Pollux, mainly using the Horizon web interface. This offers a relatively simple way of configuring and launching VM instances. You will have been given an account to login, you will also use this later to access the scalable compute resource Piz Daint. In addition to these instructions, each step references the slides that contain screenshots showing you what you should see.

Usually the first step is to configure the network and the router, as we are limited in time this has been done for you in advance 😊 However, if you would like details about these, these can be found on slides 36 – 41. We will be connecting to the network **HPAC_network** for the instances that are created. Note that all instances that are created will be deleted after a few days as we are using a test project space.

### Step 1: Slide 34

The first step is to access the Pollux web interface via the following address: https://pollux.cscs.ch/

You will be prompted to login, select authenticate using CSCS, and enter your studxx username and password. You will now see an overview of the compute resources available.

### Step 2: Slide 35

Next we want to look at the current instances in the project. To do this, select **Instances** from the **Compute** menu. There should only be one or two, as we are in the project space std02. In our project, we are allowed a maximum of 20 instances, so it is suggested that you pair up if you want, to reduce the load on the project space.

### Step 3: Slides 42 – 44

Before we create an instance, we have to create a key pair to allow us to connect to the VM. In order to do this, select **Key Pairs** under the **Compute** menu. Click on **Create Key Pair**, and put studxx as the name, replacing xx with your account number. Click **Create Key Pair**, and save the file to your laptop. Hopefully you should have already ensured that SSH is working on your machine, we will use this to securely connect to the VM that is created. Move the file you have just downloaded to the .ssh directory on your laptop.

### Step 4: Slides 45 – 50

Now that the network, router, and key pair are configured we can launch the instance. There are a number of steps now that allow us to configure the instance. For this tutorial we will use **Ubuntu**

**18.04** as the source. Click Yes on create new volume and delete volume on instance delete. For the instance flavour, select **m1.tiny**. For the network, select **HPAC_network** as the network that we will use. We can skip network ports, and for security groups we also select **HPAC_tutorial**. For **Key Pair**, select the one that you created previously that corresponds to your account details. We can now launch the instance.

Step 5: Slides 51 – 54

Once the instance is launched, we have to associate it with a floating IP in order to remotely connect to it. In order to do this, click **Instance** under the **Compute** menu where you should see your newly created instance. To the right, there is a drop-down menu, click **Associate Floating IP**. In the next screen, you will be told that no floating IPs are allocated, click to the **+** symbol. On the next screen, click **Allocate IP**. On the next screen an IP address will now be allocated, click **Associate**. We have now associated a floating IP to our instance. Copy this IP address as we will use it now.

Step 6: Slides 55 – 56

To check that the IP address has been currently setup and that we are able to communicate with it, we can ping it from the command line. To do this, open the command line on your laptop, and type:

**`ping 148.187.xx.xx`** (where xx.xx corresponds to your IP address)

If we receive a successful reply, we can then login to the instance using the SSH key generated before. To this, navigate to the folder that you saved the SSH key in, e.g. \.ssh, and type the following from the command line on your laptop:

**`ssh -i studxx.pem ubuntu@148.187.xx.xx`** (where xx.xx corresponds to your IP address)

Note that ubuntu is the default username for the cloud Ubuntu image that we are using. Once we have managed to enter the instance, we can set the password so that we are able to login using the console with the following command:

**`sudo passwd ubuntu`**

After which you will be prompted to enter a password

Step 7 – Slide 57

In the final step, we will now connect to the instance from the console in the web interface. To do this, we click on the name of the instance under **Instances** from the **Compute** menu, and click the **Console** tab. From there we are prompted to login. Using the password we just created, we can login with the ubuntu username. We have now managed to access the VM from the browser.

Step 8 – more advanced and also if we have time

OpenStack Object Store, known as Swift, offers cloud storage software so that you can store and retrieve lots of data with a simple API. One of the nice features it has is that access control lists

(ACLs) can be used to restrict who is able to share data with who. In the project space, four containers (file directories) have already been setup:

- Public_container – everyone can access
- container_51_60 – only accounts stud51 to stud60 can access
- container_51_60 – only accounts stud61 to stud70 can access
- container_51_60 – only accounts stud71 to stud80 can access

In this example we are going to setup the command line interface to Swift and show the use of the controlled access. You can use the Ubuntu instance that you created previously for this if you want. From the command line, type the following commands (each new line is a new command):

```
virtualenv openstack_cli

source openstack_cli/bin/activate

pip install -U pip setuptools

pip install -U python-openstackclient lxml oauthlib python-swiftclient

cd openstack_cli

git clone https://github.com/eth-cscs/openstack

source openstack/cli/pollux.env
```

If you have done this correctly, you will be prompted for your username and password. After this, you will now be able to see the containers and upload/download files from them. The following command shows the contents of container_71_80:

```
swift list container_71_80
```

The following command uploads a file called README.md to container_71_80:

```
swift upload container_71_80 ./README.md
```

The following command downloads a file called README.md from container_71_80:

```
swift download container_71_80 README.md
```

Using the above commands, create a text file called studxx (where xx is your account number), and upload it to the container that you have access to. Ignore the warning message, this is a known bug and your file will still be uploaded correctly. Once you have done this, download a text file from the container from another user account apart from your own. Now enjoy your coffee, you've earnt it 😊

Human Brain Project

# Piz Daint 101 – scalable compute resource

In this tutorial, we will cover the introductory topics of accessing the scalable compute resource Piz Daint via SSH, logging in via Ela, the module system and then finally submitting a simple batch job. In addition to these instructions, the slides contain screenshots showing you what you should see when you enter the commands. Commands that you will enter on your laptop are shown in **bold text**. If there is time, we will also introduce the container engine Sarus, developed for running containerised workflows at scale on Piz Daint.

## Step 1: Slide 67

The first step is to access the login node Ela. To do this, we will use the account details that you have been given. Open the command line on your laptop, and type the following command, replacing xx with the number of your account:

```
ssh studxx@ela.cscs.ch
```

Enter the password and press enter. You should now have enter, and will see a reminder for users of CSCS facilities. From the login node, you can access your $HOME storage directory, but cannot carry out any calculations.

## Step 2: Slide 68

To access the programming environments, you have to access Daint. To do this, type the following command on your laptop:

```
ssh daint
```

Enter the password and press enter. You should now be inside Piz Daint.

## Step 3: Slide 69

Piz Daint uses a module system. To see the modules that are currently loaded, type the following command on your laptop:

```
module list
```

This will return a list of the modules currently loaded.

## Step 4: Slide 70

There are a number of modules available on Piz Daint that you can use. To see the modules that are available, type the following command on your laptop:

```
module avail
```

This will return a long list of the modules that can be loaded.

Step 5: Slide 71

If we are interested in a particular module to use, e.g. Python, we can search for this rather than go through the long list of the modules that are available. As such, we will search for Cray Python modules, a version of Python adapted to work on Cray systems. To do this, type the following command on your laptop:

```
module avail cray-python
```

This will return a shorter list of four Cray Python modules that can be loaded.

Step 6: Slide 72

We can find out more about modules that we might be interested in using. In our case, we want to find out more about the Cray Python 3.6.5.7 module. To do this, type the following command on your laptop:

```
module show cray-python/3.6.5.7
```

This returns information about this module.

Step 7: Slide 73

Piz Daint offers the user two choices of compute node, multicore and hybrid. Based on our application, we may want to choose one or the other. If we want to use GPUs, then we should load the hybrid nodes modules. To do this, type the following command on your laptop:

```
module load daint-gpu
```

This allows us to use the hybrid nodes.

Step 8: Slide 74

If our application needs a lot of cores and does not need GPUs, we might instead be interested in using the 2x18 multicore nodes, e.g. if we have an application that takes advantage of OpenMP. Therefore we should load the multicore nodes module. To do this, type the following command on your laptop:

```
module load daint-mc
```

This allows us to use the multicore nodes.

Step 9: Slide 79

In order to use an application, we require data. As previously mentioned, there are different storage directories associated to the user on the CSCS scalable compute systems. The $USER directory is accessible from the login node, and uses GPFS. It is designed with reliability in mind, and is backed up with GPFS snapshots. In the next step, we will move a file from our local system to our $USER

directory. This directory can be accessed using the $HOME command. Close the connection to Daint and Ela, and carry out the following steps.

Firstly, create a small txt file on your laptop, and save it with the name HPAC_tutorial.txt , and create a directory on local system called HPAC_tutorial. Then from the command line on your local system type the following:

```
scp C:\HPAC_tutorial\HPAC_tutorial.txt studxx@ela.cscs.ch:/users/studxx/HPAC_tutorial.txt
```

You will be prompted to enter your password, after that you will see the status of the transfer. The file has now been copied to your $HOME directory. Log back into Ela and navigate to your $HOME directory.

Step 10: Slide 80

No jobs should be run on data in the $HOME directory, so we now have to copy it to the $SCRATCH directory, a fast workspace for running jobs. This is based on Lustre, and files older than 30 days are deleted from this space on a daily basis. As such, you should not store files here that you want to keep. In addition, no backups are taken of this, so any files you want to keep should be transferred once your job has completed. In this step, we are going to copy the HPAC_tutorial.txt file to the $SCRATCH workspace. As we are not able to access the $SCRATCH directory from Ela, ssh into Daint and then type the following:

```
scp $HOME/HPAC_tutorial.txt $SCRATCH/HPAC_tutorial.txt
```

Toy example: Slide 82

We will now look at how jobs are submitted to the compute system. For the sake of convenience, we use a batch script that contains information about the allocation and job. Here we can specify the number of nodes, type of node, job duration etc. In order to simplify this, there is a jobscript generator that you can use found here: https://user.cscs.ch/access/running/jobscript_generator/

Make sure you are in Daint, and working in your $SCRATCH directory. We will now try something a bit more advanced, and submit a job that prints a message from Python on different tasks on the node. In order to do this, first we need load a Python module. Looking back on previous steps, see if you are able to load the Cray Python Module 3.6.5.7 . Having done this, the next step is to select the hybrid nodes. Again, looking back at previous steps, see if you are able to load the hybrid module. The next step is to create a batch script. This can be done using a text editor on your local system, saving with the extension .sbatch . You can also directly create an .sbatch file in the $SCRATCH space using e.g. emacs or vim if you are comfortable with these tools.

Copy and modify the following text below into a text editor and save the file as studxx.sbatch (where xx is the number of your account).

```
#!/bin/bash -l
#SBATCH --job-name="studxx"
#SBATCH --mail-type=ALL
#SBATCH --mail-user=your email address
#SBATCH --time=0:01:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-core=2
```

```
#SBATCH --ntasks-per-node=24
#SBATCH --cpus-per-task=1
#SBATCH --partition=normal
#SBATCH --constraint=gpu
#SBATCH --hint=multithread

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export CRAY_CUDA_MPS=1

srun python -c \
'import os; \
print("Hello from Python task {} on node {}".format \
(os.environ["SLURM_PROCID"], os.environ["HOSTNAME"]))'
```

Copy this file to your $SCRATCH space using previously explained steps, and then finally execute the job with the following command:

**sbatch --reservation=HPAC_training studxx.sbatch**

In order to reduce waiting time, a reservation called HPAC_training has been setup for this course, and specify this at the time of execution. Normally you would have a short wait for the job to run, but with the reservation the job is run immediately.

Once the job has been executed, you should see a file named slurm-xxxxxxx.out in your $SRATCH space. Using the **vi** or **emacs** command inspect this file.

If you want, you can copy this output file to your $HOME directory or laptop, using the steps previously described.

Sarus container engine

Sarus is an OCI-compliant container engine engineered by CSCS, developed for the requirements of HPC. It has a user interface similar to Docker, and offers native performance through the use of hooks such as MPI and GPU. It enables researchers to run workflows developed on their laptop at scale on the Piz Daint scalable compute resources, and also enables a portable way to share workflows. In this tutorial, we will cover loading the Sarus module, pulling an image and then finally running it. We will demonstrate the MPI performance of Sarus using the OSU Micro-benchmarks which allows us to measure the performance of an MPI implementation, focusing on the latency and bandwidth tests.

Step 1: slide 90

As previously, the first step is to load the sarus module, and unload the module xalt. This is done by typing the following commands:

**module load sarus**

**module unload xalt**

The Sarus module is now loaded and ready for use.

Next we pull an image onto the system. The default location for pulling images from is DockerHub. In this example we pull the OSU image (ethcscs/osu-mb) that we will use measure the MPI performance by typing the following command:

```
srun -C gpu -N1 -t5 --reservation=HPAC_training sarus pull ethcscs/osu-mb:5.3.2-mpich3.1.4-ubuntu18.04
```

This pulls the following image from DockerHub: https://hub.docker.com/r/ethcscs/osu-mb

We can see the images that are available on the system, i.e. that have been pulled, typing the following command:

```
sarus images
```

Step 2: slides 91 – 94

After pulling the image onto the system, the next step is to run it. First we will run it using native MPI by typing the following command:

```
srun -C gpu -N2 -t2 --reservation=HPAC_training sarus run --mpi ethcscs/osu-mb:5.3.2-mpich3.1.4-ubuntu18.04 ./osu_latency
```

This command gives latency values. We can compare the numbers (lower is better) with a container that does not use native MPI support, but instead the non-optimised MPI coming from the image (notice the `--mpi=pmi2` option to `srun` and the absence of `--mpi`) by typing the following command:

```
srun -C gpu -N2 -t2 --mpi=pmi2 --reservation=HPAC_training sarus run ethcscs/osu-mb:5.3.2-mpich3.1.4-ubuntu18.04 ./osu_latency
```

Latency tests are carried out in a ping-pong fashion. The sender sends a message with a certain data size to the receiver and waits for a reply from the receiver. The receiver receives the message from the sender and sends back a reply with the same data size. Many iterations of this ping-pong test are carried out and average one-way latency numbers are obtained.

The next test we can carry out is the bandwidth, where higher is better. Bandwidth tests are carried out by having the sender sending out a fixed number (equal to the window size) of back-to-back messages to the receiver and then waiting for a reply from the receiver. The receiver sends the reply only after receiving all these messages. This process is repeated for several iterations and the bandwidth is calculated based on the elapsed time (from the time sender sends the first message until the time it receives the reply back from the receiver) and the number of bytes sent by the sender. The objective of this bandwidth test is to determine the maximum sustained data rate that can be achieved at the network level.

To test the bandwidth (higher is better) instead of latency, we can use `./osu_bw`, e.g. first with native MPI by typing:

```
srun -C gpu -N2 -t2 --reservation=HPAC_training sarus run --mpi ethcscs/osu-mb:5.3.2-mpich3.1.4-ubuntu18.04 ./osu_bw
```

and then comparing it to the results with non-native MPI by typing the following:

```
srun -C gpu -N2 -t2 --mpi=pmi2 --reservation=HPAC_training sarus run ethcscs/osu-mb:5.3.2-mpich3.1.4-ubuntu18.04 ./osu_bw
```