How to run a job on our computational ressources

Introduction

The rationale for this procedure is to ensure data security, to keep visibility on the cluster usage and prevent maxing out cluster resources.

Apptainer makes the container safe for all the other users : unlike Docker that needs to be run as root to be able to use GPUs, Apptainer is made for HPC usage, which prevents your container or the containers for other researchers to read your files, or erase them.

SLURM allows accounting of the submitted jobs to be able to keep visibility on the cluster usage, and will allows us to adapt its configuration along the way.

Environment

On either server Chacha or Disco, you have a symlink **datasets** in your home directory that is linked to the local storage of the server : its purpose is to give you a proper space to put all the data you will be working on.

You have also another symlink **shared_dataset** for jobs that needs to be run on several nodes : this filesystem is shared between nodes.

NOTE : By default, you are the only one seeing your data : If you are working as a team on these data, **please ask for a group creation** so we can add members in it and apply suitable permissions.

Containerize your application

To avoid having everyone installing their libraries installed on the system or on their user directly on the physical servers, we need you to keep them cleanly packed in a container : That way you can both install what you want inside this container, and you can do it without needing any root priviledge on the server you are sharing with other researchers.

See How to create a simple apptainer container

Run your application via SLURM

Now that you have your container ready in the Apptainer file format : for example application.sif

To run your job via SLURM you need to create a sbatch script :

(Note: for now we don't have Modules or LMod installed, we might add it later : hence the module commands commented)

Example for a serial job

```
#!/bin/bash
#SBATCH --job-name=application   # create a short name for your job
#SBATCH --nodes=1                # node count
#SBATCH --ntasks=1               # total number of tasks across all nodes
#SBATCH --cpus-per-task=1        # cpu-cores per task (>1 if multi-threaded tasks)
#SBATCH --mem=4G                 # total memory per node (4 GB per cpu-core is default)
#SBATCH --time=00:05:00          # total run time limit (HH:MM:SS)
#SBATCH --mail-type=begin        # send email when job begins
#SBATCH --mail-type=end          # send email when job ends
#SBATCH --mail-user=email@domain.org
#module purge
apptainer run application.sif <arg-1> <arg-2> ... <arg-N>
```

Example for a parallel MPI code

```
#!/bin/bash
#SBATCH --job-name=solar         # create a short name for your job
#SBATCH --nodes=1                # node count
#SBATCH --ntasks=4               # total number of tasks across all nodes
#SBATCH --cpus-per-task=1        # cpu-cores per task (>1 if multi-threaded tasks)
#SBATCH --mem-per-cpu=4G         # memory per cpu-core (4G per cpu-core is default)
#SBATCH --time=00:05:00          # total run time limit (HH:MM:SS)
#SBATCH --mail-type=begin        # send email when job begins
#SBATCH --mail-type=end          # send email when job ends
#SBATCH --mail-user=email@domain.org
#module purge
#module load openmpi/gcc/4.1.2
srun apptainer exec solar.sif /opt/ray-kit/bin/solar inputs.dat
```

Example using GPUs

```
#!/bin/bash
#SBATCH --job-name=tensorflow    # create a short name for your job
#SBATCH --nodes=1                # node count
#SBATCH --ntasks=1               # total number of tasks across all nodes
#SBATCH --cpus-per-task=4        # cpu-cores per task (>1 if multi-threaded tasks)
#SBATCH --mem-per-cpu=4G         # memory per cpu-core (4G per cpu-core is
default)
#SBATCH --time=00:05:00          # total run time limit (HH:MM:SS)
#SBATCH --gres=gpu:1             # number of gpus per node
#SBATCH --mail-type=begin        # send email when job begins
#SBATCH --mail-type=end          # send email when job ends
#SBATCH --mail-user=email@domain.org
#module purge
apptainer exec --nv ./tensorflow.sif python3 tensor.py
```

Note the **–nv** flag that allows you to use the GPU from your container without being root.

Example using GPU sharding

Sharding is a generic way of SLURM to use fragments of a GPU for a job, leaving room for other researchers, or running several jobs needing each one a fragment of GPU.

```
#!/bin/bash
#SBATCH --partition=Dance        # Partition to run the job on
#SBATCH --job-name=tensorflow    # create a short name for your job
#SBATCH --nodes=1                # node count
#SBATCH --ntasks=1               # total number of tasks across all nodes
#SBATCH --cpus-per-task=4        # cpu-cores per task (>1 if multi-threaded
tasks)
#SBATCH --mem-per-cpu=4G         # memory per cpu-core (4G per cpu-core is
default)
#SBATCH --time=02:00:00          # total run time limit (HH:MM:SS)
#SBATCH --gres=shard:24          # number of gpu shards to use
#SBATCH --mail-type=begin        # send email when job begins
#SBATCH --mail-type=end          # send email when job ends
#SBATCH --mail-user=email@domain.org
#module purge
apptainer exec --nv ./tensorflow.sif python3 tensor.py
```

Note the **–nv** flag that allows you to use the GPU from your container without being root. Here, the partition has been requested to be specifically run on the Disco node, which has max 80 shards. On Chacha the limit is 96 shards.

Example using an interactive shell

NOTE : Debugging your application directly on the ISC compute is to be avoided

To debug your application on a test Slurm + Apptainer infrastructure, you can use srun with the –pty argument to run your container :

```
# For a simple compute container :
srun --cpus-per-task=12 --time=3:00:00 --mem=24G --pty apptainer shell
/home/user.name/example_apptainer.sif
```

```
# Or a container using GPUs :
srun -G 1 --cpus-per-task=12 --time=3:00:00 --mem=24G --pty apptainer shell
--nv /home/user.name/example_apptainer.sif
```

Execute your batch file

Then you can run your sbatch script :

```
sbatch ./application_sbatch.sh
```

Storage considerations

Filesystems structure

On each server, your user is created with the following home directory :

```
ls -l /home/user.name
lrwxrwxrwx  1 root              root                51 Feb  4 10:21
.apptainer -> /data/disk01/apptainer/user.name/.apptainer/
lrwxrwxrwx  1 root              root                38 Feb  4 10:21 datasets
-> /data/space/datasets/user.name/
lrwxrwxrwx  1 root              root                31 Feb  4 10:21 results
-> /data/shared/user.name/results/
lrwxrwxrwx  1 root              root                31 Feb  4 10:22
shared_datasets -> /data/shared/user.name/shared_datasets/
```

- **.apptainer** is your cache directory for your containers, use `apptainer cache clean` every week to clean unused containers.
- **datasets** is your main directory for all your work material, do NOT use directly /home/user.name/ to put data or software on it : it would uselessly fill the root partition.
- **results** is located on a shared filesystem between all Slurm nodes to allow you to run jobs anywhere : as long as you write the output of your job in results, you will be able to access it whatever node the job had run on.
- **shared_datasets** is located on a shared filesystem between all Slurm nodes to allow you to run jobs anywhere : when your job needs to get the same data from several nodes, use this directory. **NOTE :** you will have less performances for fast reads/writes since it is a network shared filesystem : if this is a concern, please consider using the faster local datasets directory above and specify your work node in your sbatch using –nodelist=xxx.

Data retention

On the ISC Compute Center, you need to be aware of how much time data is kept on which location.

1. **Local datasets :** Data life is short here, for now cleaning is not automated but will probably be in the future. Useful for a large volume of data to be processed on one server, and with the fastest I/O available.

2. **Shared datasets and results :** Data life is short here, for now cleaning is not automated but will probably be in the future. Useful for a large volume of data to be processed on several servers, and with reasonable I/O. (with some network latency due to NFS)
3. **Filer mount :** Data life is long, and backuped : this is where you can store data, results and other files for a longer time. By default you don't have a filer mount, it is the Sinf who can create you a shared directory on their main filer (filer01.hevs.ch) on their infrastructure, which is backuped and also isolated with limited access. Create this filer mount with their "Demande d'obtention de droits d'accès réseau" request

NOTE : Please note the filer storage is slow, it is not advised to run a job directly using those data.

Cleaning

1. Users must regularly clean the Apptainer cache using `apptainer cache clean`. Every week is a good starting point, we might automate cleaning if this is not respected : please don't think the container is a persistent storage for your data, it is not.
2. Don't keep data directly on your /home/user.name/ directory : you are limited to 20GB on it just for convenience : no large file should be there.
3. When you create new apptainer containers, the .sif files can be quite big : don't store them in /home/user.name/, prefer using either datasets/ or if needed, archive them in your filer mount.

Good practice

Coding considerations

1. **Don't use VScode SSH plugin** to avoid constant file scanning on our servers :

- https://earlruby.org/2021/06/fixing-vscode-when-it-keeps-dropping-ssh-connections/
- https://stackoverflow.com/questions/71055834/ms-vscode-cpptools-taking-a-ton-of-cpu-usage
- https://github.com/microsoft/vscode-cpptools/issues/5362
- https://learn.microsoft.com/en-us/answers/questions/1221136/visual-studio-2022-clear-local-caches

1. Currently your vscode caches will be automatically removed from our servers.

From:
https://wiki.isc-vs.ch/ - **The ISC wiki**

Permanent link:
https://wiki.isc-vs.ch/doku.php?id=infra:howto:runjob&rev=1740660573

Last update: **2025/02/27 12:49**