Utiliser la plateforme PlaFRIM

2025-06-25

1 - Création du compte PlaFRIM

Pour créer un compte, il suffit de suivre ce lien

Création d'une clé ssh (demandée pour la création du compte)

Sur Mac, ou dans un Terminal Bash:

`ssh-keygen`

Ensuite, on va nous demander un code, qu'il faudra rentrer deux fois

Maintenant, si on va dans `.ssh/` on devrait avoir deux fichier:

- id_ed25519 (ou id_rsa) *la clé privée*
- id_ed25519.pub (ou id_rsa.pub) *la clé publique*

Ce qu'il va falloir utiliser pour créer le compte, c'est la clé publique, en faisant `cat id_ed25519.pub` puis en copiant toute la sortie par exemple.

2 - Associer la clé privé à la plateforme

Une fois la création du compte effectué, on doit recevoir un courriel contenant la confirmation de la création du compte ainsi que des instructions.

Pour ce faire, il faut edit le fichier de configuration sur notre machine locale `.ssh/config`

Puis y rajouter ces informations:

```
Host plafrim
        ForwardAgent yes
        ForwardX11 yes
        User identifiant
        ProxyJump identifiant@ssh.plafrim.fr

Host ssh.plafrim.fr
        IdentityFile ~/.ssh/id_ed25519 ou id_rsa #ici on met le nom de la clé
```

```
privé
        IdentitiesOnly yes
```

[instructions issues du mail]

Puis toujours sur la machine locale il faut charger la clé privé `ssh-add ~/.ssh/id_ed25519 (ou id_rsa)`
Celle-ci permettra de crypter / décrypter les communications entre plafrim et notre machine.

Maintenant on peut se connecter à la plateforme via ssh `identifiant@plafrim`, comme ce sera la première connexion, il est possible que le mot de passe soit demandé.

Par la suite il suffira de se connecter via `ssh plafrim`

3 - Télécharger le code depuis un repository

Pour pouvoir utiliser **GitLab**, pas besoin de paramétrer de clé ssh, juste activer le bon module

`module load tools/git/2.36.0`

Par contre, si on veut utiliser **GitHub**, il y a une configuration en plus a rajouter dans `.ssh/config`

```
Host github.com
    Hostname ssh.github.com
    Port 443
    User git
```

[1]

4 - Préparation de l'environnement

Créer son environnement python:

D'abord il faut importer un module avec la bonne **version python** (celle que l'on veut utiliser)

```
module load language/python/3.12
```

Ensuite on peut créer notre **venv** avec python 3.12

```
python3 -m venv NomDuVenv
```

Puis activer **l'environnement python**

```
source NomDuVenv/bin/activate
```

Une fois que c'est fait on peut installer les **librairies** nécessaires

```
pip install torch torchaudio torchvision

ou bien

pip3 install torch torchaudio torchvision
```

> Si il y a des **problèmes** pendant l'installation des librairies, la plupart du temps c'est dû à une erreur de compilation, bien souvent il suffit de changer de compilateur
>
> Et il faut à nouveau charger le module de compilation en question à **chaque fois** qu'on ouvre une nouvelle session plafrim

```
module load compiler/gcc/12.2.0
```

Si cette version ne fonctionne pas, ou n'est pas disponible, entrez juste

```
module load compiler/gcc/
```

puis TAB jusqu'à trouver une version qui convient

Une fois que c'est fait, on peut déjà lancer notre code comme sur notre machine, en ligne de commande.
Cela dit, on n'est pas encore connecté à une machine (Partie 5)

4 bis - (uniquement si on veut utiliser Guix pour les librairies)

Il faut créer manuellement les dossiers et le fichier suivant:
`~/.config/guix/channels.scm`

Et y mettre ces lignes:

```
(cons (channel (name 'guix-science-nonfree) (url "https://codeberg.org/guix-science/guix-science-nonfree.git") (introduction (make-channel-introduction "58661b110325fd5d9b40e6f0177cc486a615817e" (openpgp-fingerprint "CA4F 8CF4 37D7 478F DA05 5FD4 4213 7701 1A37 8446")))) %default-channels)
```

[1]

Ensuite, lancer `guix pull` (cela peut prendre de quelques minutes à quelques heures)

Vérifier les packages disponibles: `guix package --list-available | grep filtre`

Et on peut donc, si on gère les packages avec guix créer directement un script qui crée notre environnement

```
ENV_VARS := \

  LD_PRELOAD="/usr/local/cuda-12.3/targets/x86_64-
linux/lib/stubs/libnvrtc.so:\
/usr/local/cuda-12.3/targets/x86_64-linux/lib/libnvrtc-builtins.so.12.3:\
/usr/local/cuda-12.3/nsight-systems-2023.3.3/host-linux-x64/libstdc++.so.6"

GUIX_PACKAGES := \
sed\
coreutils\
findutils\
git\
grep\
bash\
python\
python-pytorch-with-cuda12\
python-numpy\
python-pandas\
python-matplotlib\
python-scipy\
python-pillow\
python-tqdm

.PHONY: dev-shell

dev-shell:
    @echo "Entering Guix shell with custom environment..."
```

```
        env $(ENV_VARS) guix shell $(GUIX_PACKAGES) --pure --
preserve=LD_PRELOAD --cores=0 --max-jobs=64

shell-execute:
    @echo "Executing command in Guix shell..."
    env $(ENV_VARS) guix shell $(GUIX_PACKAGES) --pure --preserve=LD_PRELOAD -
- $(CMD) --cores=0 --max-jobs=64

clean:
      @echo "Cleaning up..."
        rm -rf logs/ # Exemple : supprimer le répertoire de logs
```

[1]

Exemple pour chercher des paths spécifiques pour des packages: `find /usr/ -name "libnvrtc.so"`

5 - Obtenir une machine en intérractif

Commande pour demander une machine

`salloc -C "sirocco&zen3" -t 03:00:00 --exclusive` [1] [3]

Cette commande là permet de demander une machine avec au moins 22 coeurs de GPU, pendant 3 heures, et on réserve les ressources uniquement pour notre utilisateur

On peut ensuite se connecter via ssh à la machine qu'on nous propose

Exemple: `ssh sirocco23`

A partir de là, on est connecté à la machine, il suffit donc de retourner dans le dossier qui contient notre projet, activer le venv python, et lancer le job !

Un script slurm pour lancer des jobs automatiquement

```
#!/bin/bash

#SBATCH --job-name=model-train      # Nom de votre job Slurm
#SBATCH --output=logs/%x-%j.out      # Fichier de sortie standard (stdout)
#SBATCH --error=logs/%x-%j.err       # Fichier d'erreur standard (stderr)
#SBATCH --time=2-23:59:59 # Durée maximale d'exécution (2 jours, 23h, 59 min,
59 sec)
```

```
#SBATCH --exclusive        # Demande l'accès exclusif au nœud (attention si
beaucoup de jobs)
#SBATCH -C sirocco&zen3 # Spécifie le type de processeurs (à adapter aux
ressources de PlaFRIM)

make shell-execute CMD="python3 Modeles/code_plafrim.py"
```

[1], [2]

Informations supplémentaires

Par défaut, chaque utilisateur a un quota de 500Gb sur sa session.

Sources

[1] Documentation PlaFRIM
[2] Utiliser Slurm
[3] Nom et informations sur les machines issues également de [1]

	CPU	Memory	GPU	Storage
bora001-044	2x 18-core Intel CascadeLake	192GB		/tmp of 1 To
suroit01-22	2x 24-core AMD Zen4	256GB		
diablo01-04	2x 32-core AMD Zen2	256 GB		/tmp of 1 TB
diablo05	2x 64-core AMD Zen2	1 TB		/tmp of 1 TB
diablo06-09	2x 64-core AMD Zen3	1 TB		/scratch of 4 TB
zonda01-21	2x 32-core AMD Zen2	256 GB		
arm01	2x 28-core ARM TX2	256 GB		/tmp of 128 GB
sirocco07-13	2x 16-core Intel Broadwell	256 GB	2 NVIDIA P100	/tmp of 300 GB
sirocco14-16	2x 16-core Intel Skylake	384 GB	2 NVIDIA V100	/scratch of 750 GB
sirocco17	2x 20-core Intel Skylake	1 TB	2 NVIDIA V100	/tmp of 1 TB
sirocco18-20	2x 20-core Intel CascadeLake	192 GB	2 NVIDIA Quadro	
sirocco21	2x 24-core AMD Zen2	512 GB	2 NVIDIA A100	/scratch of 3.5 TB
sirocco22-25	2x 32-core AMD Zen3	512 GB	2 NVIDIA A100	/scratch of 4 TB
enbata01-02	2x 32-core AMD Zen4	384 GB	2 AMD MI210	
suet01	2x 28-core Intel IceLake	256 GB	2 Intel DataCenter GPU Flex 170	/scratch of 1 TB
kona01-04	64-core Intel Xeon Phi	96GB + 16GB		/scratch of 800 GB
brise	4x 24-core Intel Broadwell	1TB		/tmp of 280 GB
souris	12x 8-core Intel IvyBridge	3TB		