RÉPUBLIQUE FRANÇAISE

Liberté
Égalité
Fraternité

Inserm

La science pour la santé
From science to health

UTILISATION DE SLURM

AVEC LA PLATEFORME HPC INSERM

Version 0.2 – 26/03/2026

Auteur : Sébastien Aucouturier (ITQ)

RAPPEL : NOTIONS DE BASE SUR LE HPC ET SLURM

QU'EST-CE QU'UN HPC ?

Un HPC (High Performance Computing – Calcul Haute Performance) est un ensemble de plusieurs ordinateurs, appelés nœuds, utilisés pour réaliser des calculs complexes très rapidement.

Ces performances sont obtenues grâce à :

- La parallélisation des calculs
- La distribution des tâches sur plusieurs nœuds

Cela permet de traiter des volumes de calculs bien plus importants que sur un ordinateur classique.

EXEMPLES D'UTILISATION DES HPC

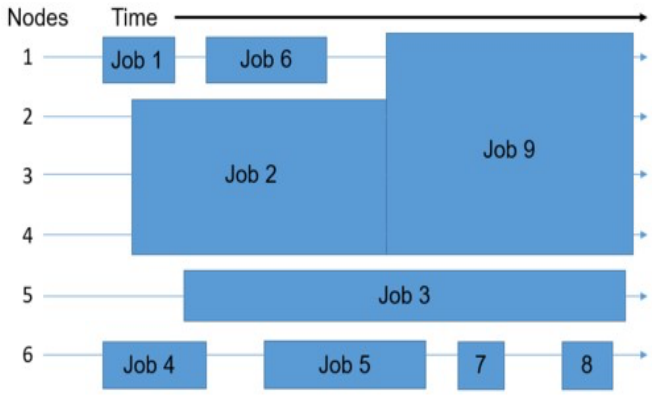
Les HPC sont utilisés dans de nombreux domaines :

- **Recherche scientifique** : climat, génomique, médecine, physique nucléaire
- **Conception de véhicules ou d'avions** : aérodynamique, crash-tests simulés
- **Intelligence artificielle** et entraînement de modèles de langage
- **Simulations complexes** : météo, volcanologie, sismologie, simulation 3D

QU'EST-CE QU'UN JOB ?

Sur un **HPC**, le travail ne se fait pas comme sur un ordinateur classique.

Au lieu d'exécuter directement un programme, on soumet un **job**.



Un **job** est un **programme ou un ensemble de commandes que l'on demande au cluster d'exécuter**.

Le système HPC décide :

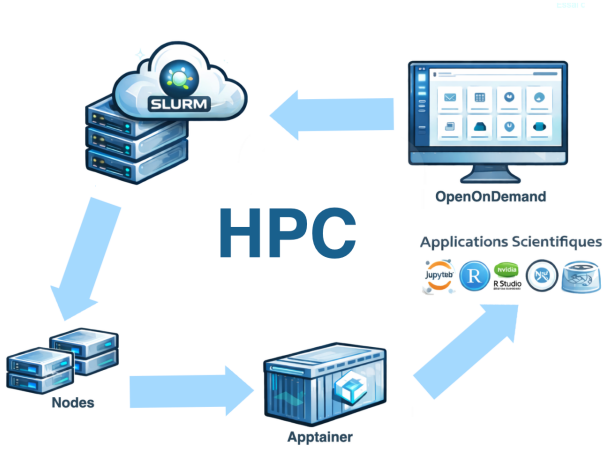
- **Où** exécuter le job
- **Quand** l'exécuter

En fonction des **ressources demandées** par l'utilisateur et des **ressources disponibles** sur le cluster.

Les jobs sont organisés dans **des files d'attente**.

ARCHITECTURE SIMPLIFIEE D'UN HPC

Un HPC est généralement composé de plusieurs éléments.



ORDONNANCEUR DE JOBS (SLURM)

L'**ordonnanceur** est le logiciel qui :

- Distribue les calculs
- Gère les ressources
- Planifie l'exécution des jobs

Sur le HPC Inserm, cet ordonnanceur est **SLURM**.

PORTAIL D'ACCES (OPENONDEMAND)

Le **portail OpenOnDemand** permet aux utilisateurs d'accéder plus facilement au cluster via une interface web.

Il permet notamment de :

- Lancer des sessions interactives
- Accéder à des applications scientifiques
- Gérer ses fichiers

Ces applications sont souvent déployées via **Apptainer** sur les nœuds de calcul sélectionnés par l'ordonnanceur.

APPLICATIONS SCIENTIFIQUES

Les utilisateurs peuvent lancer différentes applications scientifiques sur le cluster, par exemple :

- **JupyterHub**
- **RStudio**
- **NVIDIA NGC**
- Logiciels de simulation ou d'analyse de données

POURQUOI A-T-ON BESOIN D'UN ORDONNANCEUR DE JOBS ?

Sur un cluster de calcul, **de nombreux utilisateurs peuvent vouloir lancer des calculs en même temps**.

Sans organisation :

- Certaines tâches pourraient utiliser **toutes les ressources disponibles**
- D'autres utilisateurs ne pourraient **pas exécuter leurs calculs**
- L'**ordonnanceur de jobs** permet d'éviter cette situation.

Il :

- Organise l'utilisation des ressources du cluster
- Distribue équitablement les nœuds entre les utilisateurs
- S'assure que chacun puisse accéder à la puissance de calcul

Sur le HPC Inserm, cette tâche est assurée par **SLURM (Simple Linux Utility for Resource Management)**.

QUE FAIT SLURM LORSQUE VOUS SOUMETTEZ UN JOB ?

Lorsque vous soumettez un job, **SLURM** :

1. Cherche les **ressources nécessaires** (nœuds, CPU, mémoire)
2. Place votre job dans **une file d'attente** si les ressources ne sont pas immédiatement disponibles
3. Lance votre job sur le(s) **nœud(s) de calcul approprié(s)** lorsque les ressources deviennent libres
4. **Surveille l'exécution** du job
5. Vous informe lorsque le job est **terminé** ou s'il y a **une erreur**

POURQUOI SLURM EST INDISPENSABLE

Sans un système comme SLURM :

- Certains utilisateurs pourraient **monopoliser tout le cluster**
- Les ressources seraient **mal utilisées**
- Il serait **impossible de partager efficacement la puissance de calcul**

SLURM garantit donc une **utilisation équitable et efficace du cluster**.

VOCABULAIRE ESSENTIEL DE SLURM

Voici quelques termes importants que vous rencontrerez souvent.

JOB

Un **job** est une **demande de calcul envoyée au cluster**.

C'est l'unité de travail que vous soumettez à **SLURM**.

PARTITION (OU QUEUE)

Une **partition** est un **groupe de nœuds de calcul**.

Elle regroupe généralement des nœuds ayant des **caractéristiques similaires** :

- Mémoire
- CPU
- GPU

Chaque partition possède ses **propres règles** :

- Durée maximale d'un job
- Nombre maximal de nœuds
- Limites de ressources

NŒUD (NODE)

Un **nœud** est une **entité de calcul du cluster**.

Chaque nœud possède :

- Plusieurs **CPU**
- Une certaine quantité de **mémoire (RAM)**
- Éventuellement des **GPU**

TACHE (TASK / NTASKS)

Une **tâche** correspond généralement à **un processus en cours d'exécution**.

Autrement dit, c'est **une instance de votre programme** qui s'exécute sur le cluster.

UTILISATION VIA OPEN ONDEMAND

Sur le HPC Inserm, la plupart des utilisateurs accèdent au cluster via le portail **Open OnDemand (OOD)**.

Dans ce cas, le fonctionnement de **SLURM** est en grande partie **masqué** à l'utilisateur.

L'utilisateur n'a généralement pas besoin de manipuler directement les commandes SLURM (sbatch, squeue, etc.).

À la place, il utilise un portail qui lui permet de :

- Sélectionner l'**application scientifique** à lancer
- Choisir une **partition** (type de ressources à utiliser)
- Définir la **durée de la réservation**

Une fois ces paramètres choisis :

1. **Open OnDemand crée automatiquement un job SLURM**
2. Le job est envoyé à **SLURM**
3. SLURM **réserve les ressources nécessaires**
4. L'application est lancée sur le **nœud de calcul attribué**
5. L'utilisateur accède ensuite directement à l'application via son navigateur.

Ainsi :

- **Open OnDemand simplifie l'utilisation du cluster**
- **SLURM continue de gérer les ressources en arrière-plan**

Cette approche permet aux utilisateurs de **se concentrer sur leurs outils scientifiques**, sans avoir à maîtriser immédiatement toutes les commandes du système HPC.

POURQUOI FAUT-IL DEFINIR UNE DUREE MAXIMALE POUR UN JOB ?

Lorsque vous soumettez un job sur un HPC, vous devez généralement indiquer une **durée maximale d'exécution**.

Ce paramètre est obligatoire car il permet à l'ordonnanceur **SLURM** de **planifier efficacement les ressources du cluster**.

COMMENT SLURM UTILISE CETTE INFORMATION ?

Lorsque vous soumettez un job, SLURM doit décider :

- **Quand** lancer le job
- **Sur quels nœuds**
- **Pendant combien de temps les ressources seront occupées**

Connaître la durée maximale permet à SLURM :

- D'organiser la **file d'attente**
- D'optimiser l'utilisation des **nœuds de calcul**
- D'éviter les **temps morts sur le cluster**

C'est un peu comme réserver une salle de réunion, si vous indiquez que vous en avez besoin pendant **2 heures**, il est plus facile d'organiser les réservations suivantes.

CE QUI SE PASSE SI LE JOB DEPASSE LA DUREE DEMANDEE

Si votre programme dépasse la **durée maximale demandée**, SLURM **arrête automatiquement le job**.

Le calcul est alors interrompu.

C'est pourquoi il est important de :

- Estimer correctement la durée nécessaire
- Prévoir une marge raisonnable
- **Prévoir l'écriture régulière de résultats intermédiaires** (fichiers temporaires, checkpoints)

En effet, si le job est interrompu (fin de la durée maximale, panne, arrêt manuel), ces fichiers permettent :

- De **ne pas perdre l'intégralité du calcul**
- De **reprendre le calcul à partir du dernier état sauvegardé (si votre code le permet)**

Cette pratique est particulièrement recommandée pour les **calculs longs**, qui peuvent durer plusieurs heures ou plusieurs jours.

POURQUOI NE PAS TOUJOURS DEMANDER UNE DUREE TRES LONGUE ?

Il peut être tentant de demander **beaucoup plus de temps que nécessaire**, mais ce n'est généralement pas recommandé.

Une durée trop longue peut :

- **Retarder le démarrage de votre job**
- **Ralentir la planification du cluster**
- **Réduire l'efficacité globale du système**

En effet, SLURM doit trouver un créneau disponible **suffisamment long** pour exécuter votre job.

BONNE PRATIQUE

Essayez de demander une durée **la plus proche possible de la durée réelle du calcul**.

Cela permet :

- À votre job de **démarrer plus rapidement**
- À SLURM **d'optimiser l'utilisation du cluster**

QUAND UTILISER DIRECTEMENT LES COMMANDES SLURM ?

Le portail **Open OnDemand (OOD)** permet de lancer facilement des applications scientifiques via une interface web.

Dans la plupart des cas, cette interface suffit pour démarrer une session interactive ou utiliser des outils comme **Jupyter**, **code-server** ou d'autres environnements scientifiques.

Cependant, certaines situations nécessitent d'utiliser **directement les commandes SLURM** depuis le terminal.

C'est notamment le cas lorsque :

- Vous souhaitez **exécuter un script ou un programme personnalisé**
- Vous devez **lancer un grand nombre de calculs automatisés**
- Vous avez besoin de **paramétrer finement les ressources** (CPU, mémoire, nombre de tâches, etc.)
- Vous travaillez avec des **pipelines ou des scripts d'analyse complexes**
- Vous souhaitez **automatiser des traitements répétitifs**

Dans ces situations, l'utilisateur soumet un **job SLURM** via des commandes en ligne de commande.

Le principe reste le même que via Open OnDemand :

1. Vous décrivez les **ressources nécessaires**
2. SLURM place votre job dans une **file d'attente**
3. Le job est exécuté lorsque les **ressources deviennent disponibles**

La différence est que vous **contrôlez directement le comportement du job**, ce qui offre plus de **flexibilité et d'automatisation**.

Les sections suivantes présentent les **commandes SLURM essentielles** pour interagir avec le cluster.

ACCEDER AUX COMMANDES SLURM

Les commandes SLURM s'utilisent depuis un **terminal connecté au cluster HPC**.

Pour cela, deux méthodes sont généralement disponibles :

- **Connexion SSH** à un **nœud de login** du HPC
- Accès via **HPC Shell Access** dans le portail **Open OnDemand (OOD)**

Les **nœuds de login** sont des machines dédiées à :

- La préparation des jobs
- La soumission des calculs
- La gestion des fichiers

l est important de **ne pas exécuter de calculs lourds directement sur les nœuds de login**. Ces nœuds sont partagés par tous les utilisateurs et servent uniquement à **interagir avec le cluster**.

Les calculs doivent toujours être lancés sous forme de **jobs via SLURM**, qui se chargera de les exécuter sur les **nœuds de calcul**.

Une fois connecté au nœud de login (via SSH ou via **HPC Shell Access**), vous pouvez utiliser les **commandes SLURM** présentées dans la section suivante.

LES COMMANDES SLURM ESSENTIELLES

Lorsque vous travaillez directement avec SLURM depuis le terminal, quelques commandes suffisent pour gérer la majorité des situations.

Voici les **commandes les plus importantes à connaître**.

1. SBATCH — SOUMETTRE UN JOB

La commande sbatch permet de **soumettre un job au cluster**.

Le job est généralement décrit dans un **script bash** contenant :

- Les ressources demandées (directives #SBATCH)
- Les commandes à exécuter (votre code)

Les Instructions Essentielles pour SLURM (#SBATCH)

Dans SLURM, les **directives #SBATCH** permettent de **définir les paramètres de votre job**. Elles se placent **au début de votre script**, juste après le shebang (#!/bin/bash).

Voici quelques directives très courantes :

```
#SBATCH --job-name=MonCalcul
#SBATCH --output=Resultat_%j.out
#SBATCH --error=Erreur_%j.err
#SBATCH --time=00:10:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=500M
#SBATCH --partition=debug
```

Explications des directives #SBATCH

- *#SBATCH --job-name=MonCalcul*
  Donne un **nom à votre job** pour le retrouver facilement. Choisissez un nom descriptif.
- *#SBATCH --output=Resultat_%j.out*
  Nom du fichier où sera écrite la **sortie normale** de vos commandes (ce qui s'afficherait à l'écran).

%j est remplacé par l'**ID unique du job**, pratique pour ne pas écraser les résultats précédents.

- *#SBATCH --error=Erreur_%j.err*
  Nom du fichier où seront écrits les **messages d'erreur**.
  Très utile pour comprendre pourquoi un job a échoué.
- *#SBATCH --time=00:10:00*
  Temps maximum réservé pour le job (format hh:mm:ss).
  SLURM arrêtera votre job s'il dépasse cette durée.
  Commencez par une estimation large, puis affinez.
- *#SBATCH --nodes=1*
  Nombre de **nœuds de calcul** demandés. Pour débuter, 1 est souvent suffisant.
- *#SBATCH --ntasks=1*
  Nombre de **tâches** (processus) à lancer au total.
  Pour un programme simple non parallèle, 1 suffit.
- *#SBATCH --cpus-per-task=1*
  Nombre de **cœurs CPU réservés par tâche**.
  Pour un programme simple, 1 suffit.
- *#SBATCH --mem=500M*
  Quantité de **mémoire vive (RAM)** réservée pour le job.
  Vous pouvez utiliser G pour Gigaoctets, ex : --mem=4G.
- *#SBATCH --partition=debug*
  Partition (file d'attente) sur laquelle vous soumettez le job.

2. SQUEUE — VOIR LES JOBS EN COURS

La commande squeue permet de **voir les jobs en attente ou en cours d'exécution**.

Exemple :

```
squeue
```

Pour voir uniquement vos jobs : *squeue -u $USER*

Les informations affichées par SLURM

Lorsqu'on utilise squeue, les informations affichées incluent :

- ID du job
- partition
- utilisateur
- état du job
- durée d'exécution
- nœud utilisé

3. SCANCEL — ANNULER UN JOB

La commande scancel permet **d'arrêter un job**.

Exemple :

```
scancel 42
```

où 42 est l'ID du job.

Pour annuler tous vos jobs :

```
scancel -u $USER
```

4. SINFO — VOIR L'ETAT DU CLUSTER

La commande sinfo permet de **voir l'état des partitions et des nœuds du cluster**.

Exemple :

```
sinfo
```

La commande sinfo permet de connaître :

- Les **partitions disponibles**
- Le **nombre de nœuds**
- Leur **état**

5. SACCT — CONSULTER L'HISTORIQUE DES JOBS

La commande sacct permet de voir les informations sur les **jobs terminés**.

Exemple :

```
sacct
```

Consulter les détails d'un job spécifique

Pour voir les détails d'un job spécifique :

```
sacct -j 42
```

Consulter les détails d'un job

Cette commande permet de consulter :

- Le **temps d'exécution**
- La **mémoire utilisée**
- L'état **final du job**

6. RESUME RAPIDE DES COMMANDES SLURM

Commande	Utilité
sbatch	soumettre un job
squeue	voir les jobs en attente ou en cours
scancel	annuler un job
sinfo	voir l'état du cluster
sacct	consulter l'historique des jobs

7. EXEMPLE COMPLET DE WORKFLOW SLURM

Voici un exemple concret montrant **le cycle complet d'un job SLURM** : de la soumission à la consultation des résultats.

Ce workflow illustre le cycle classique :

- Créer un script sbatch avec vos ressources et commandes
- Soumettre le job avec sbatch
- Suivre son état avec squeue
- Consulter les résultats et l'état final avec sacct

7.1. EXEMPLE DE SCRIPT SBATCH

Créez un fichier mon_job.slurm :

```
#!/bin/bash
#SBATCH --job-name=TestCalcul
#SBATCH --output=Resultat_%j.out
#SBATCH --error=Erreur_%j.err
#SBATCH --time=00:10:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=2
#SBATCH --mem=1G
#SBATCH --partition=debug

echo "Job démarré sur le nœud : $(hostname)"
echo "Date et heure : $(date)"
```

```
# Exemple de commande à exécuter
python mon_script.py
```

7.2. SOUMETTRE LE JOB

Pour envoyer le job au scheduler SLURM :

```
sbatch mon_job.slurm
```

SLURM renverra un ID de job, par exemple :

```
Submitted batch job 42
```

7.3. VÉRIFIER L'ÉTAT DU JOB AVEC SQUEUE

Pour voir le job dans la file d'attente ou en cours :

```
squeue -u $USER
```

Exemple de sortie :

```
JOBID   PARTITION     NAME   USER   ST      TIME    NODES
42      debug   TestCalcul     alice   R       00:02:15        1
```

- ST = R signifie que le job est en exécution
- ST = PD signifie que le job est en attente

7.4. CONSULTER LES RESULTATS AVEC SACCT

Une fois le job terminé, vous pouvez obtenir les informations détaillées :

```
sacct -j 42
```

Exemple de sortie :

```
JobID   JobName     User    State   Elapsed
42      TestCalcul      alice   COMPLETED       00:03:12
```

- State : état final du job (COMPLETED, FAILED, CANCELLED, etc.)
- Elapsed : durée totale d'exécution

7.5. CONSULTER LES RESULTATS VIA LES FICHIERS .OUT ET.ERR

Rappelez-vous des directives --output et --error dans votre script :

```
#SBATCH --output=Resultat_test_%j.out
#SBATCH --error=Erreur_test_%j.err
```

SLURM a créé ces fichiers en remplaçant %j par l'ID de votre job. Exemple : si votre job a l'ID 42, vous trouverez dans le répertoire où vous avez soumis le script :

- Resultat_test_42.out : contient tout ce que vos commandes ont affiché en sortie standard (par exemple les echo).
- Erreur_test_42.err : contient tous les messages d'erreur générés par vos commandes. C'est le premier fichier à regarder si votre job a échoué (F).

7.6. POURQUOI MON JOB A-T-IL ECHOUE ? PISTES COURANTES

- Erreur dans le script SLURM : une faute de frappe dans une directive #SBATCH.
  - ⇨ SLURM le refuse souvent dès la soumission avec sbatch.
- Erreur dans vos commandes : votre programme a planté ou une commande Linux était incorrecte.
  - ⇨ Regardez le fichier .err.
- Temps dépassé : le job a duré plus longtemps que le --time demandé.
  - ⇨ Augmentez la limite de temps si nécessaire. Le fichier .out ou .err peut s'arrêter brusquement.
- Mémoire dépassée : le job a essayé d'utiliser plus de RAM que le noeud propose.
- Partition incorrecte / Ressources indisponibles : mauvais nom de partition ou demande de ressources irréaliste pour la partition choisie.
- Problème de fichiers : fichiers d'entrée non trouvés, permissions incorrectes…

PARTITIONS HPC INSERM

Pour le HPC Inserm, **4 partitions** ont été définies afin d'organiser et contrôler l'accès aux ressources de calcul, y compris les GPU.

Partition	Nœuds	Ressources par nœud	Usage recommandé
2xH100-96	1	2 GPU H100 NVL-96 Gb, 32 vCPU Mem: 256Go	Inférence de modèles LLM de grande taille (> 70b)
A40-48	2	1 GPU A40-48 Gb, 16 vCPU Mem: 128Go	Inférence de modèles LLM de petite taille (≤ 8b)
A100-10	14	1 GPU A100-10 Gb, 6 vCPU Mem: 48Go	Calcul scientifique nécessitant plusieurs nœuds GPU simultanément
CPU-16	4	16 vCPU Mem: 128Go	Calcul scientifique sans GPU
CPU-4	4	4 vCPU Mem: 32Go	Prototypage scientifique sans GPU

REGLES IMPORTANTES

Dans le cadre du HPC Inserm, un **nœud réservé** ne peut être utilisé que par **un seul utilisateur à la fois**.

Chaque réservation est **limitée à 30 heures maximum**.

Point crucial : au-delà de cette durée, vos résultats de calcul seront perdus si le job est interrompu.

Note : d'autres partitions sont en cours d'implémentation.

REGLES D'OR DU BON UTILISATEUR DE CLUSTER

Utiliser un cluster est un privilège partagé.

Voici quelques règles simples :

- Le nœud de connexion est sacré : N'y lancez jamais de calculs longs ou gourmands !
- Utilisez-le uniquement pour éditer des fichiers, compiler, soumettre des jobs (sbatch), et vérifier des résultats rapides.
- Soyez raisonnable : Ne demandez pas plus de temps, de CPU ou de mémoire que nécessaire. Cela bloque les ressources pour les autres. Estimez au mieux, et ajustez si besoin.
- Faites le ménage : Les espaces de stockage ne sont pas illimités. Supprimez régulièrement les fichiers de résultats ou temporaires dont vous n'avez plus besoin.

ET APRES ? POUR ALLER PLUS LOIN

JOBS INTERACTIFS : INTERAGIR DIRECTEMENT AVEC UN NŒUD DE CALCUL

Parfois, il est nécessaire de **se connecter directement à un nœud de calcul**, par exemple pour :

- **Déboguer** un job avant de le soumettre via sbatch
- **Tester un programme** sur les ressources exactes

Les commandes SLURM **srun** et **salloc** permettent ce type d'interaction.

SALLOC — ALLOUER UN NŒUD POUR UNE SESSION INTERACTIVE

salloc réserve un nœud ou plusieurs nœuds et vous ouvre une **session interactive** dessus.

Exemple :

```
salloc --partition=debug --nodes=1 --ntasks=1 --time=01:00:00
```

SRUN — EXECUTER UNE COMMANDE DIRECTEMENT SUR UN NŒUD

srun permet de lancer **une commande unique sur un nœud réservé**, sans ouvrir un shell interactif complet.

```
srun --partition=debug --nodes=1 --ntasks=1 --time=00:30:00 python mon_script.py
```

Notes : Il est également possible d'obtenir **un shell interactif sur un nœud** avec srun en utilisant l'option --pty.

```
srun --partition=debug --nodes=1 --ntasks=1 --time=01:00:00 --pty bash
```

Les mêmes règles de partition et ressources que pour sbatch s'appliquent.

UTILISATION DE TABLEAUX DE JOBS (JOB ARRAYS)

Lorsque vous devez **exécuter la même tâche sur de nombreux fichiers ou paramètres différents**, il n'est pas efficace de soumettre des dizaines ou des centaines de jobs séparément.

SLURM propose une fonctionnalité appelée **Job Array** qui permet de **lancer plusieurs instances du même script en une seule soumission**.

C'est particulièrement utile pour :

- Analyser **un grand nombre de fichiers de données**
- Tester **plusieurs paramètres d'un modèle**
- Traiter **plusieurs simulations indépendantes**
- Paralléliser des traitements identiques sur différents jeux de données

Par exemple, si vous avez **100 fichiers à analyser**, un Job Array permettra de lancer **100 tâches indépendantes**, chacune traitant un fichier différent.

PRINCIPE

On utilise la directive --array dans le script SLURM :

```
#SBATCH --array=1-100
```

Cela signifie que SLURM va créer 100 jobs similaires, avec un identifiant d'index allant de 1 à 100.

Chaque tâche peut récupérer son numéro grâce à la variable d'environnement :

```
$SLURM_ARRAY_TASK_ID
```

EXEMPLE DE SCRIPT

```
#!/bin/bash
#SBATCH --job-name=Analyse
#SBATCH --output=Result_%A_%a.out
#SBATCH --error=Erreur_%A_%a.err
#SBATCH --array=1-100
#SBATCH --time=01:00:00
#SBATCH --mem=2G
#SBATCH --partition=cpu

echo "Traitement du fichier numéro $SLURM_ARRAY_TASK_ID"

python analyse.py data_${SLURM_ARRAY_TASK_ID}.csv
```

Explications :

- %A = ID du job principal
- %a = index de la tâche dans le tableau
- $SLURM_ARRAY_TASK_ID = numéro de la tâche en cours

Chaque tâche va donc traiter un fichier différent :

```
data_1.csv
data_2.csv
data_3.csv
...
data_100.csv
```

AVANTAGES DES JOB ARRAYS

- Une seule soumission pour des dizaines ou centaines de jobs
- Gestion automatique par SLURM
- Exécution parallèle sur plusieurs nœuds
- Suivi simplifié dans squeue

LIMITER LE NOMBRE DE JOBS SIMULTANES

Il est possible de limiter le nombre de tâches exécutées en parallèle :

```
#SBATCH --array=1-100%10
```

Cela signifie :

- 100 tâches au total
- Maximum 10 exécutées en même temps

C'est utile pour éviter de saturer les ressources du cluster.

DEPENDANCES ENTRE JOBS

Dans certains workflows scientifiques, plusieurs étapes de calcul doivent être exécutées **dans un ordre précis**.

Par exemple :

1. Préparer ou nettoyer des données
2. Lancer un calcul principal
3. Analyser ou visualiser les résultats

Dans ce cas, il est important que **le job suivant ne démarre que lorsque le précédent est terminé**.

SLURM permet de gérer cela grâce aux **dépendances entre jobs**.

PRINCIPE

Lors de la soumission d'un job avec sbatch, on peut utiliser l'option :

```
--dependency
```

Cette option indique à SLURM qu'un job dépend d'un autre job.

Le nouveau job sera placé dans la file d'attente, mais il ne démarrera que lorsque la condition sera remplie.

Exemple simple

Supposons que vous avez deux scripts :

- preparation.slurm
- analyse.slurm

Vous soumettez d'abord le premier job :

```
sbatch preparation.slurm
```

SLURM renvoie un identifiant de job :

```
Submitted batch job 42
```

Vous pouvez ensuite soumettre le second job avec une dépendance :

```
sbatch --dependency=afterok:42 analyse.slurm
```

Cela signifie:

- Le job analyse.slurm ne sera lancé que si le job 42 se termine avec succès.

TYPES DE DEPENDANCES COURANTES

Option	Signification
after	Le job démarre **après que le job précédent a commencé**
afterok	Le job démarre **seulement si le job précédent se termine correctement**
afternotok	Le job démarre **seulement si le job précédent échoue**
afterany	Le job démarre **après la fin du job précédent, quel que soit le résultat**

EXEMPLE DE PIPELINE SIMPLE

On peut ainsi créer un pipeline de calcul automatisé :

```
job1=$(sbatch preparation.slurm | awk '{print $4}')
job2=$(sbatch --dependency=afterok:$job1 calcul.slurm | awk '{print $4}')
sbatch --dependency=afterok:$job2 analyse.slurm
```

Ce workflow garantit que :

- preparation.slurm s'exécute en premier
- calcul.slurm démarre seulement si la préparation réussit
- analyse.slurm démarre seulement si le calcul principal réussit

AVANTAGES DES DEPENDANCES

- Permet de créer des pipelines de calcul automatisés
- Évite d'exécuter des étapes si les données précédentes ne sont pas valides
- Simplifie la gestion de workflows complexes
- Réduit les erreurs humaines dans les enchaînements de jobs

PARTICULARITE DU CLUSTER HPC INSERM

HPC CLASSIQUE : PARTAGE DES RESSOURCES

Dans un **HPC classique**, les nœuds de calcul disposent généralement d'un **nombre très important de vCPU**.

Par exemple, un nœud CPU peut comporter **192 vCPU**.

Cette architecture est adaptée à un environnement fortement multi-utilisateur, dans lequel plusieurs utilisateurs **partagent simultanément les ressources d'un même nœud**.

Exemple sur un nœud disposant de **192 vCPU** :

- Un utilisateur réserve **32 vCPU**
- Un second **64 vCPU**
- Un troisième **8 vCPU**
- Les ressources restantes peuvent être allouées à d'autres utilisateurs

Ce modèle permet de **maximiser le taux d'occupation global du cluster**, au prix d'un **partage des ressources matérielles** entre utilisateurs.

L'APPROCHE DU HPC INSERM

L'architecture du **HPC Inserm** est différente.

Le cluster est déployé en **zone HDS (Hébergement de Données de Santé)**, dans un contexte où les exigences de **sécurité, d'isolation et de conformité** sont particulièrement fortes.

Pour cette raison, nous avons fait le choix d'un **modèle d'isolation stricte au niveau du nœud** :

Lorsqu'un utilisateur réserve un nœud, il en est **l'unique utilisateur pendant toute la durée de l'allocation**.

Ce choix architectural permet :

- D'éviter les problématiques de **co-tenancy**
- De garantir une **isolation forte des charges de travail**
- De simplifier certains aspects liés à la **sécurité et à la conformité**

DIMENSIONNEMENT DES NŒUDS

Dans ce contexte, des nœuds extrêmement volumineux (par exemple **192 vCPU**) seraient peu adaptés.

Un tel dimensionnement est pertinent dans un modèle **mutualisé**, mais il est **rarement exploitable efficacement par un seul utilisateur**.

Notre objectif est donc de trouver le **meilleur compromis entre** :

- Puissance de calcul
- Efficacité d'utilisation des ressources
- Adéquation aux usages scientifiques

dans un modèle où **un nœud est dédié à un utilisateur unique**.

Ainsi, tout en conservant la volonté de **mutualiser l'utilisation globale des ressources du cluster**, nos nœuds ont été dimensionnés de manière **plus équilibrée** :

- **Jusqu'à 32 vCPU** pour les configurations les plus importantes
- Notamment pour les nœuds intégrant des **accélérateurs GPU** (par exemple **2 × NVIDIA H100**)

Cette approche permet d'offrir **une puissance de calcul significative**, tout en conservant un dimensionnement cohérent avec un **modèle d'allocation exclusive des ressources**.

DUREE DES JOBS : DIFFERENCE AVEC LES HPC CLASSIQUES

Le modèle d'utilisation d'un **HPC classique** se reflète également dans les **durées de réservation autorisées pour les jobs**.

Dans un cluster mutualisé :

- La durée maximale d'un job peut atteindre **90 jours**
- La durée par défaut est souvent d'environ **7 jours**

Cette flexibilité est possible car les ressources d'un nœud sont **fractionnées et partagées entre plusieurs utilisateurs**.

L'ordonnanceur peut donc **optimiser l'occupation des ressources sur de longues périodes**.

Autrement dit, même si un job réserve une **partie des ressources pendant longtemps**, le reste du nœud peut continuer à être utilisé par d'autres utilisateurs.

Ce modèle favorise donc des **temps de réservation relativement longs**, tout en maintenant un **bon taux d'utilisation global du cluster**.

POLITIQUE DE RESERVATION DU HPC INSERM

Dans notre cas, le fonctionnement est différent.

Comme l'allocation d'un **nœud est exclusive à un seul utilisateur**, les ressources ne peuvent pas être partagées avec d'autres workloads pendant l'exécution du job.

Pour cette raison, nous avons défini une **durée maximale de job de 30 heures**.

Cette limite permet :

- De garantir une **rotation régulière des ressources entre utilisateurs**
- D'éviter **l'immobilisation prolongée de nœuds complets**

- De maintenir un **bon niveau de disponibilité du cluster**

Cette politique assure ainsi un équilibre entre performance, sécurité et accessibilité

COMPORTEMENT DU HPC INSERM EN CAS DE DEPASSEMENT DE MEMOIRE

Contrairement à un **HPC classique** où SLURM peut terminer le job pour dépassement mémoire (OOMKilled), l'infrastructure Inserm se comporte différemment :

La **couche système sous-jacente** (OS / gestionnaire du nœud) redémarre **l'ensemble du nœud** si la mémoire allouée est dépassée.

Le job en cours est alors **interrompu** et passe en **CANCELLED**.

SLURM **ne détecte pas directement** l'événement OOM, et le champ Reason du job **ne mentionne pas "OutOfMemory"**.

Un mécanisme spécifique a été mis en place pour que **l'information du dépassement mémoire apparaisse dans le champ Comment** après le redémarrage du nœud.

IMPLICATIONS POUR L'UTILISATEUR

- **Prévoir la mémoire nécessaire** : Évaluer la mémoire totale nécessaire pour vos jobs en fonction du nombre de vCPU.
- **Écrire des résultats temporaires** : Comme le job peut être interrompu brutalement, il est recommandé de sauvegarder **régulièrement les résultats intermédiaires**.
- **Surveiller les logs et le champ Comment** : Après un job annulé pour dépassement mémoire, consulter les informations pour identifier la cause réelle.
- **Tester sur des partitions plus petites** : Pour éviter d'utiliser tout un nœud inutilement, commencez par des jobs tests avec moins de vCPU et mémoire.

BONNES PRATIQUES

- **Réserver légèrement plus de mémoire** que votre estimation réelle pour réduire le risque d'interruption.
- **Fractionner les jobs lourds** : Si possible, diviser les calculs en plusieurs jobs plus petits pour limiter l'impact en cas d'interruption.
- **Automatiser les sauvegardes** : Scripts Python, R ou bash peuvent écrire des checkpoints dans des fichiers temporaires pour sécuriser les calculs.

Note : Dans ce modèle, la **sécurité et l'isolation des nœuds** priment sur la tolérance au dépassement mémoire. Il est donc essentiel de bien dimensionner vos jobs pour ne pas immobiliser un nœud complet.

APPTAINER — CONTENEURS POUR LE HPC

POURQUOI UTILISER DES CONTENEURS EN CALCUL SCIENTIFIQUE ?

LE PROBLEME CLASSIQUE : « ÇA MARCHE SUR MA MACHINE ! »

En développement scientifique, il est courant de rencontrer des problèmes liés aux dépendances et aux environnements logiciels :

- Votre code fonctionne parfaitement sur votre ordinateur, mais **échoue sur le cluster** :
  - Erreur de version de Python ou d'une bibliothèque cruciale
  - Bibliothèque manquante sur le système du cluster
  - Conflits de versions avec d'autres logiciels
- Vos collègues ne peuvent pas reproduire vos résultats car **leurs environnements diffèrent**

LES DEFIS DU LOGICIEL EN HPC

1. **Environnements complexes** : Les logiciels scientifiques nécessitent souvent des versions précises de Python, NumPy, MPI, ou d'outils spécialisés (bioinformatique, calcul parallèle…).
2. **Conflits** : Installer toutes ces dépendances pour tous les utilisateurs sur un cluster partagé est difficile. Une version qui fonctionne pour l'un peut casser le code d'un autre.
3. **Reproductibilité** : Pour garantir la fiabilité scientifique, il faut pouvoir **réexécuter la même analyse sur une autre machine** et obtenir le même résultat. Les conteneurs apportent une solution partielle mais efficace.

LA SOLUTION : LES CONTENEURS

Un conteneur logiciel est une **image standardisée** qui contient :

- L'application et toutes ses dépendances (bibliothèques, fichiers de configuration…)
- Éventuellement un système minimal nécessaire au fonctionnement
- Les instructions pour lancer le programme

Cette image peut être **déplacée et exécutée sur différentes machines** : votre poste local, le cluster ou le cloud, sans réinstaller toutes les dépendances.

Résultat :

- Meilleure portabilité
- Meilleure reproductibilité

POURQUOI APPTAINER/SINGULARITY AU LIEU DE DOCKER ?

1. **Sécurité** : exécution avec vos propres permissions utilisateur, sans privilège root. Crucial sur un cluster partagé.
2. **Intégration HPC** :

- Accès aux systèmes de fichiers partagés ($HOME, $SHARED)
- Support GPU Nvidia avec l'option --nv
- Fonctionnement fluide avec SLURM
    - SLURM gère l'allocation des ressources et la file d'attente
    - Apptainer garantit que l'environnement logiciel est cohérent sur tous les nœuds

3. **Portabilité et archivage** :

- L'image Apptainer est un **fichier unique (.sif)**, facile à copier et partager

COMMENT UTILISE APPTAINER SUR LE CLUSTER HPC

Sur le HPC, Apptainer peut être utilisé sur n'importe quel nœud de calcul. Pour exploiter pleinement toutes ses options, il est nécessaire de lui passer un certain nombre de paramètres, parfois un peu barbares. Heureusement, l'utilisation d'une variable d'environnement peut grandement nous faciliter la vie

```
APPTAINER_OPTION="-e --writable-tmpfs --sharens --nv"
```

Parametre	Description
-e	**Clean environment** : ignore l'environnement de l'utilisateur sur le système hôte. Seules les variables essentielles et celles définies dans le conteneur sont conservées. Cela évite les conflits avec les variables de votre session shell.
--writable-tmpfs	Crée un **système de fichiers temporaire en mémoire** qui est modifiable pendant l'exécution du conteneur. Idéal si votre conteneur est en lecture seule mais que vous avez besoin de modifier certains fichiers pendant le job. Les modifications disparaissent à la fin du conteneur.
--sharens	Partage les systèmes de fichiers montés (/home, /scratch, etc.) entre le conteneur et l'hôte de manière transparente. Permet à plusieurs jobs d'accéder aux mêmes fichiers si nécessaire.

LANCER UN CONTENEUR POUR EXECUTER UNE COMMANDE

Vous pouvez exécuter directement une commande à l'intérieur d'un conteneur :

```
apptainer exec $APPTAINER_OPTION mon_conteneur.sif python mon_script.py
```

- exec : exécute une commande dans le conteneur
- mon_conteneur.sif : fichier image Apptainer
- python mon_script.py : commande à exécuter dans le conteneur

OUVRIR UN SHELL INTERACTIF DANS LE CONTENEUR

Pour explorer le conteneur, lancer plusieurs commandes ou tester un programme :

```
apptainer shell $APPTAINER_OPTION mon_conteneur.sif
```

Vous obtenez un terminal interactif isolé dans le conteneur

Les modifications apportées à ce conteneur en cours d'exécution ne modifient pas l'image originale

Note : Pour obtenir un terminal interactif avec allocation de ressources SLURM, il est possible de combiner avec srun --pty.

MONTER DES REPERTOIRES DE VOTRE HPC DANS LE CONTENEUR

Par défaut, votre répertoire $HOME est monté automatiquement. Pour monter d'autres répertoires (ex: shared) :

```
apptainer exec $APPTAINER_OPTION --bind /shared:/mnt_conteneur.sif python mon_script.py
```

- --bind /shared:/mnt : lie le répertoire /shared du HPC à /mnt dans le conteneur

Ce qui permet de lire ou écrire des fichiers depuis le conteneur vers le cluster

UTILISATION AVEC SLURM

Pour exécuter un conteneur dans un job SLURM, insérez la commande Apptainer dans votre script .slurm :

```
#!/bin/bash
#SBATCH --job-name=mon_job_apptainer
#SBATCH --partition=cpu
#SBATCH --ntasks=1
#SBATCH --time=01:00:00
#SBATCH --mem=4G

APPTAINER_OPTION="-e --writable-tmpfs --sharens --nv"

apptainer exec $APPTAINER_OPTION mon_conteneur.sif python mon_script.py
```

ACCES AUX APPLICATIONS WEB DANS APPTAINER

Contrairement à **Docker**, où il est souvent nécessaire de **mapper un port** de l'hôte vers le conteneur (-p 3000:3000), **Apptainer/Singularity gère les réseaux différemment** dans un environnement HPC :

- Le conteneur **partage le réseau de l'hôte** par défaut.
- Cela signifie que les applications web ou serveurs lancés à l'intérieur du conteneur sont **accessibles directement** sur les ports locaux du nœud où le conteneur s'exécute.

Exemple :

Si votre application démarre sur le port 3000 dans le conteneur :

```
python app.py  # lance l'application sur le port 3000
```

Vous pourrez y accéder directement depuis le nœud où le conteneur est exécuté à l'adresse : http://localhost:3000

Remarque : Sur un HPC, localhost fait référence au nœud de calcul, pas à votre machine locale.

ACCEDER A VOTRE APPLICATION VIA OPEN ONDEMAND (OOD)

Sur un HPC comme celui de l'Inserm, les nœuds de calcul ne sont pas directement exposés à Internet. Pour utiliser une application web lancée depuis un conteneur Apptainer (ex. Python/Flask ou Java), il faut passer par l'interface Desktop d'Open OnDemand (OOD).

Étapes

1. Sélectionnez l'application Desktop depuis OOD et lancez la session.
2. Dans le Deskop interactif :
    1. Ouvez Un terminal celui-ci est connecté au nœud de calcul avec accès aux ressources CPU/GPU. 2.2 Lancer votre application dans le conteneur Apptainer

```
APPTAINER_OPTION="-e --writable-tmpfs --sharens --nv"
apptainer exec $APPTAINER_OPTION mon_conteneur.sif python mon_app.py
```

    2. Ouvrez Firefox depuis le Desktop OOD. 2.4 Saisissez l'URL : http://localhost:3000

Vous avez alors accès à votre application web exécutée dans Apptainer sur le nœud de calcul.

POINTS IMPORTANTS

- localhost fait référence au nœud, pas à votre machine locale.
- Chaque utilisateur peut lancer sa propre application web sur un port distinct, sans interférer avec les autres.
- Assurez-vous que le port choisi est libre sur le nœud.

DE DOCKER TO APPTAINER : PRET POUR LE HPC

Tu viens de passer des heures à créer **ta super image Docker**, avec toutes les bibliothèques, Python 3.11, Conda, et même le petit chat ASCII qui s'affiche au démarrage… et là, **sur le HPC, ça ne marche pas** !

Docker n'est pas installé sur le cluster et il ne peut pas tourner en root sur un environnement partagé.

Heureusement, **Apptainer** est là pour sauver la situation : il te permet de **convertir ton chef-d'œuvre Docker en image HPC-compatible (.sif)**, et de l'exécuter sur le cluster avec GPU, SLURM et tout le reste, **sans root et sans pleurer sur les dépendances manquantes**.

Il est fréquent de commencer avec une image Docker déjà prête, ou alors vous êtes un **expert des Dockerfile** .

Pour utiliser cette image sur le HPC, il faut **la convertir en image Apptainer (.sif)**.

1. Obtenir l'archive de l'image Docker Cette étape doit être réalisée **sur une machine où Docker est installé** (poste local, serveur personnel, cloud) :

```
docker pull python:3.11
docker save python:3.11 -o python_3.11.tar
```

2. **Convertir l'image au format Apptainer (.sif)**
   - Sur la machine ou le service disposant d'Apptainer :

```
apptainer build python_3.11.sif docker-archive://python_3.11.tar
```

   - Cette commande crée une **image unique .sif**, prêt à être utilisé sur le HPC.