Cómo solicitar recursos de GPU y CPU

Comprenda cómo está configurado nuestro clúster"

- **Para solicitar nodos de GPU o GPU primero debe comprender cómo está configurado su clúster**, por este motivo, le recomendamos que consulte la página inicial donde tenemos una descripción del clúster que le puede ayudar.
- Básicamente hay 2 arquitecturas: ***icelake*** (nodos de GPU) y ***sandy*** (nodos de CPU).
- En base a estas arquitecturas se sólicitan los recursos dentro del cluster.

Principalmente, hay 2 formas de especificar que recurso que se usará al enviar un trabajo usando ***constraints*** y usando ***gres***.

Constraints de slurm en TeideHPC

Los nodos pueden tener **features** asignadas por los administradores de TeideHPC. Los usuarios pueden especificar cuáles de estas *features* son requeridas por su trabajo utilizando la opción de **--constrains**. **Sólo los nodos que tengan características que coincidan con las restricciones del trabajo se utilizarán para satisfacer la solicitud.**

Para obtener una explicación más detallada de las restricciones, puede consultar la [documentación oficial de slurm] (https://slurm.schedmd.com/sbatch.html#OPT_constraint)

En el caso de TeideHPC, para averiguar la lista de contrain definidas en el cluster sólo ha de mirar la columna *AVAIL FEATURES* después de ejecutar el siguiente comando.

```
sinfo -o "%40N  %10c  %10m  %35f  %30G "
```

```
NODELIST                            CPUS       MEMORY     AVAIL_FEATURES                      GRES
node18109-1                         64         257214     ilk,gpu,a100                        gpu:a100:8
node2204-[3-4]                      20         31906      ivy                                 (null)
node17109-1,node17110-1,node18110-1,node  64         257214     ilk,viz,t4                          gpu:t4:1
node0303-2,node0304-[1-4],node1301-[1-4]  16         30000+     sandy                               (null)
node17102-1                         64         257214     ilk,gpu,a100,3g.20gb,2g.10gb,1g.5gb
```

```
gpu:3g.20gb:1(S:0),gpu:2g.10gb
node17101-1,node17103-1,node17104-1,node  64         257214
ilk,gpu,a100                      gpu:a100:4(S:0-1)
```

constrains	Tipo	Definición
ilk	arquitetura nodos	Permite seleccionar un nodo tipo icelake (tienen GPU)
sandy	arquitetura nodos	Permite seleccionar un nodo de cómputo con arquitectura sandy-bridge
ivy	arquitetura nodos	Permite seleccionar un nodo de cómputo con arquitectura ivy-bridge
viz	nodo de visualización	Permite seleccionar un nodo de GPU especializado en visualización (Nodos con Nvidia T4)
gpu	nodos de gpu	Permite seleccionar un nodo gpu (Nodos con NVidia A100)
a100	modelo gpu	Permite seleccionar un nodo gpu con NVidia A100 directamente
t4	modelo gpu	Permite seleccionar un nodo gpu con NVidia T4 directamente

Ejemplo de uso de constrains

Los siguientes comandos de sesión interactiva tienen su equivalente en *batch*. Sólo necesita añadir la directiva `#SBATCH --constraint=<constraint>` en tus scripts.

- Para un nodo de cómputo con arquitectura sandy (16 cores | 32/64 GB).

```
salloc -n 1 --cpus-per-task 16 -p express --mem 29000 --constrain=sandy
```

- Para un nodo de cómputo con arquitectura icelake (64 cores | 256 GB)..

```
salloc -n 1 --cpus-per-task 8  -p express --mem 10000 --constrain=ilk
```

- Para un nodo de GPU (icelake) con una GPU (Nvidia A100)

```
salloc -n 1 --cpus-per-task 8  -p express --mem 8000 --constrain=gpu[,a100]
```

- Para un nodo de GPU especializado en visualización (Nvidia Tesla T4).

```
salloc -n 1 --cpus-per-task 8  -p express --mem 8000 --constrain=viz[,t4]
```

Recursos genéricos (Generic Resource) *GRES*

En Slurm, **GRES significa Recurso Genérico**. GRES es una función que le permite especificar y administrar varios tipos de recursos genéricos, como GPU (Unidades de procesamiento de gráficos) dentro de un clúster informático.

La funcionalidad GRES de Slurm permite la asignación, la programación y el seguimiento eficientes de estos recursos para los trabajos enviados. Ayuda a garantizar que los recursos solicitados estén disponibles y sean utilizados correctamente por los trabajos que los requieren.

Para usar GRES de manera efectiva, debe comprender cómo está configurado su clúster, cómo se definen los tipos y cantidades de GRES disponibles y especificar los requisitos de GRES al enviar trabajos.

Para obtener los tipos de GRES definidos en el clúster, puede usar:

```
scontrol show config | grep Gres
```

```
GresTypes = gpu
```

Para conocer la lista de *GRES* en TeideHPC mira la columna *GRES* después de ejecutar este comando.

```
sinfo -o "%40N  %10c  %10m  %35f  %30G"
```

```
NODELIST                            CPUS       MEMORY     AVAIL_FEATURES                      GRES
node18109-1                         64         257214     ilk,gpu,a100                        gpu:a100:8
node2204-[3-4]                      20         31906      ivy                                 (null)
node17109-1,node17110-1,node18110-1,node  64         257214     ilk,viz,t4                          gpu:t4:1
node0303-2,node0304-[1-4],node1301-[1-4]  16         30000+     sandy                               (null)
node17102-1                         64         257214     ilk,gpu,a100,3g.20gb,2g.10gb,1g.5gb gpu:3g.20gb:1(S:0),gpu:2g.10gb
node17101-1,node17103-1,node17104-1,node  64         257214
ilk,gpu,a100                        gpu:a100:4(S:0-1)
```

Alternativamente, también se puede ver enumerando y filtrando el listado de nodos:

```
scontrol show nodes | egrep "NodeName|gres"
```

```
....
NodeName=node1315-4 Arch=x86_64 CoresPerSocket=8
NodeName=node2204-3 Arch=x86_64 CoresPerSocket=10
...
NodeName=node17101-1 Arch=x86_64 CoresPerSocket=32
  CfgTRES=cpu=64,mem=257214M,billing=64,gres/gpu=4,gres/gpu:a100=4
NodeName=node17102-1 Arch=x86_64 CoresPerSocket=32
  CfgTRES=cpu=64,mem=257214M,billing=64,gres/gpu=7,gres/gpu:1g.5gb=2,gres/gpu:
2g.10gb=1,gres/gpu:3g.20gb=1,gres/gpu:a100=3
NodeName=node17103-1 Arch=x86_64 CoresPerSocket=32
  CfgTRES=cpu=64,mem=257214M,billing=64,gres/gpu=4,gres/gpu:a100=4
NodeName=node17104-1 Arch=x86_64 CoresPerSocket=32
  CfgTRES=cpu=64,mem=257214M,billing=64,gres/gpu=4,gres/gpu:a100=4
...
```

As you can see, each node in cluster may have a different definition. For example this node has 4 GPUs NVidia A100.

```
NodeName=node17104-1 Arch=x86_64 CoresPerSocket=32
  CfgTRES=cpu=64,mem=257214M,billing=64,gres/gpu=4,gres/gpu:a100=4
```

and this node has 3 NVidia A100 and 1 partitioned GPU (Nvidia A100).

```
NodeName=node17102-1 Arch=x86_64 CoresPerSocket=32
  CfgTRES=cpu=64,mem=257214M,billing=64,gres/gpu=7,gres/gpu:1g.5gb=2,gres/gpu:
2g.10gb=1,gres/gpu:3g.20gb=1,gres/gpu:a100=3
```

Para entender cúal es el significado de `gres/gpu:1g.5gb=2,gres/gpu:2g.10gb=1,gres/gpu:3g.20gb=1,gres/gpu:a100=3` primero debe entender qué es MIG

Ejemplos de uso de *GRES*

- Para un nodo de GPU (icelake) con una GPU (Nvidia A100).

```
salloc -n 1 --cpus-per-task 8 -p express --mem 8000 --gres=gpu:a100=1
```

- Para una partición MIG de GPU (Nvidia A100) .

```
salloc -n 1 --cpus-per-task 8 -p express --mem 8000 --gres=gpu:2g.10gb=1
```

- Para GPUs especializadas en visualización (Nvidia Tesla T4).

```
salloc -n 1 --cpus-per-task 8 -p express --mem 8000 --gres=gpu:t4=1
```