

Centre de Calcul
de l'Institut National de Physique Nucléaire
et de Physique des Particules

La plateforme des notebooks Jupyter au CC-IN2P3

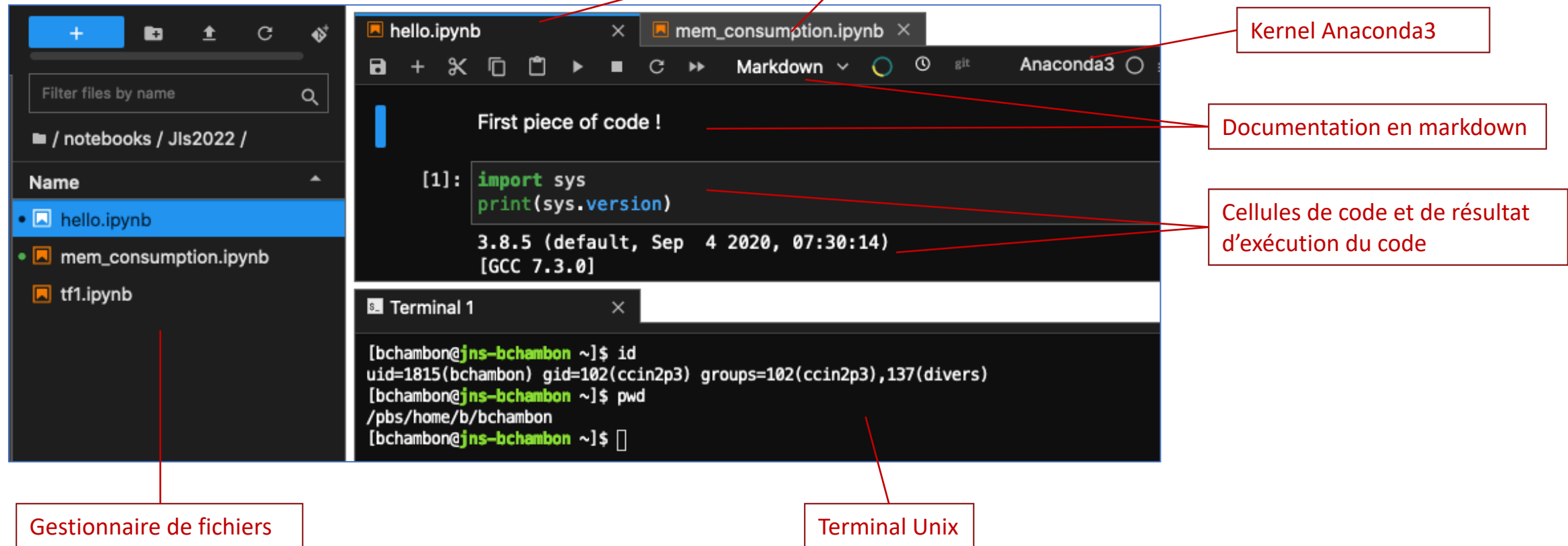
Bernard CHAMBON, Journées Informatiques de l'IN2P3/IRFU, 16 novembre 2022

- Introduction
- Architecture
- Mise à disposition des GPUs et de Dask+SLURM
- Cas d'usage fournis par nos utilisateurs
- Infrastructure matérielle - Chiffres relatifs à l'utilisation de la plateforme
- Perspectives et conclusion
- Annexes

- Objectif de cette plateforme
 - Fournir un service d'analyse interactive, via les notebooks Jupyter
 - Avec accès aux mêmes systèmes de stockage que ceux disponibles sur la plateforme d'accueil (cca.in2p3.fr)
 - Avec authentification via le SSO du CC-IN2P3

- Atouts des notebooks Jupyter
 - Simplicité
 - Navigateur WEB
 - Un même document pour du code, de la documentation, des résultats d'exécution
 - Disponibilité d'un terminal UNIX (sans faire ssh)
 - Diversité des langages utilisables, via les kernels (Jupyter = **Julia**, **Python**, **R**)
 - Richesse de l'écosystème via de nombreux widgets et/ou extensions

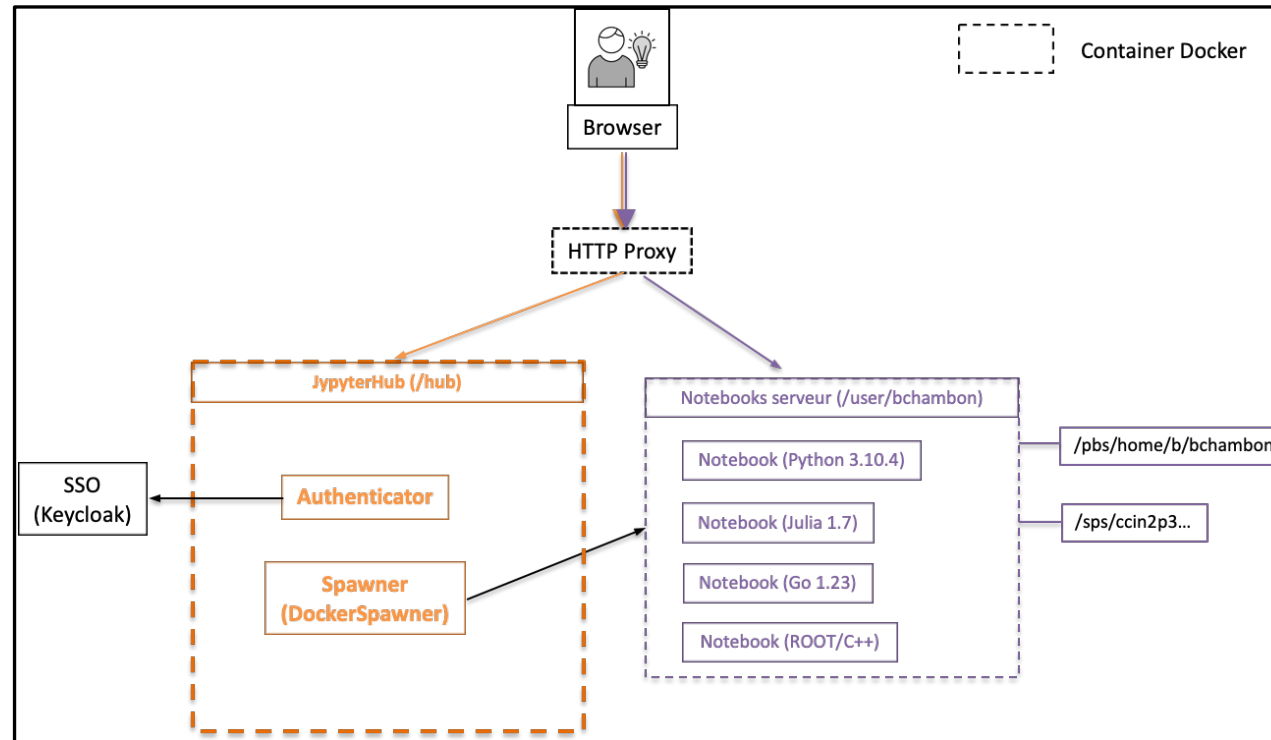
- Mon premier serveur de notebooks



The screenshot shows the JupyterLab interface with several components and annotations:

- 2 notebooks dans un serveur de notebooks**: Points to the tabs for `hello.ipynb` and `mem_consumption.ipynb`.
- Kernel Anaconda3**: Points to the `Anaconda3` kernel indicator in the top right of the notebook area.
- Documentation en markdown**: Points to the `Markdown` dropdown menu in the top right of the notebook area.
- Cellules de code et de résultat d'exécution du code**: Points to the code cell containing `import sys` and `print(sys.version)`, and its output: `3.8.5 (default, Sep 4 2020, 07:30:14) [GCC 7.3.0]`.
- Terminal Unix**: Points to the terminal window showing the output of `id` and `pwd` commands.
- Gestionnaire de fichiers**: Points to the file browser on the left side of the interface.

- Construite autour de **JupyterHub**
 - Composant permettant de définir l'authentification, de construire un formulaire d'options, d'instancier un serveur de notebooks
 - Configuration en Python



- Le service repose sur un cluster Docker avec l'orchestrateur Swarm pour placer les serveurs de notebooks sur les machines

- Accès au service
 - Ouvert à tout utilisateur disposant d'un compte calcul, mais certaines fonctionnalités nécessitent un accès privilégié (accès aux GPUs ou à Dask+SLURM)
 - Pas de limite de temps d'utilisation, mais les serveurs de notebooks IDLE sont monitorés (timeout de 3 jours | 1 jour pour respectivement un serveur de notebooks CPU | GPU)
- Authentification
 - Délégation de l'authentification (OAuth) auprès de du SSO Keycloak (certificat ou login/MdP du compte 'calcul')
- A propos du serveur de notebooks
 - Conteneur Docker basé sur une image préparée au CC et basée sur CentOS 7.6. (homogénéité avec les autres services du CC)
 - Container s'exécute avec les IDs (uid, gid) de l'utilisateur
 - Disponibilités des systèmes de stockage
 - Espace HOME dans PBS (ex /pbs/home/b/bchambon) , espaces GROUPES (ex /sps/ccin2p3) selon les groupes primaire et secondaires Spécifique à chaque utilisateur
 - Espaces THRONG /pbs/throng, SOFTWARE /pbs/software, CVMFS /cvmfs/xyz Identique pour tous les utilisateurs

- Ressources RAM et CPU
 - RAM
 - 2 GB par défaut, extensible sur demande
 - Par logon ou par groupe (si plusieurs groupes on considère le max, logon prioritaire sur le groupe).
 - Un widget permet à l'utilisateur de voir la consommation mémoire instantanée
 - CPU
 - Pas de limitation en nombre de CPUs
 - Mais un mécanisme de limitation est disponible et activable par logon ou par groupe (cas de consommation excessive)
- Monitoring des consommations
 - Utilisation de cAdvisor, Prometheus, Grafana pour la collecte, le stockage et la visualisation
 - Dashboard avec des métriques par logon ou par machine

Voir exemple
en annexe A1

Utilisation des GPUs

Utilisation de Dask+SLURM

- Objectif
 - Permettre de faire de l'analyse en utilisant les GPUs
- Moyen
 - Disposer du droit d'accès à cette fonctionnalité. En faire la demande, via un ticket
 - Accès à un formulaire d'options permettant de choisir le modèle et le nombre de GPUs ainsi que la RAM du serveur de notebooks (Voir slide suivant)
- L'utilisateur disposera alors
 - D'un serveur de notebooks restreint au nombre de GPUs choisis
 - Avec les principaux frameworks de machine learning (ML)
Pytorch, TensorFlow + TensorBoard + TensorFlow Probability + cuDNN
 - Rappel de la possibilité d'installer, dans un env. virtuel, d'autres logiciels compatibles avec les versions de CUDA et du driver

Formulaire d'options pour GPU et instantiation d'un serveur de notebooks

Formulaire d'options

My Notebooks Server Options

Compute engines CPU Only GPU

Memory (GB) ? 28

GPU model(s) K80

GPU(s) number ? 2

The GPU model **K80** provides :

- Hardware
 - 4 GPUs per host
 - 12 GB GPU-RAM per GPU
 - NVIDIA driver version 465.19.01
- Default software environment
 - CUDA 11.3 [cuda](#)
 - Pytorch 1.9.0 [pytorch](#)
 - TorchVision 0.10.0
 - TensorFlow 2.9.1 [tensorflow](#)
 - cuDNN 8.2.4
 - TensorFlow Probability 0.17.0
 - TensorBoard 2.9.0
 - CuPy 9.4.0 [cupy](#)
 - PyCUDA 2020.1 [pycuda](#)

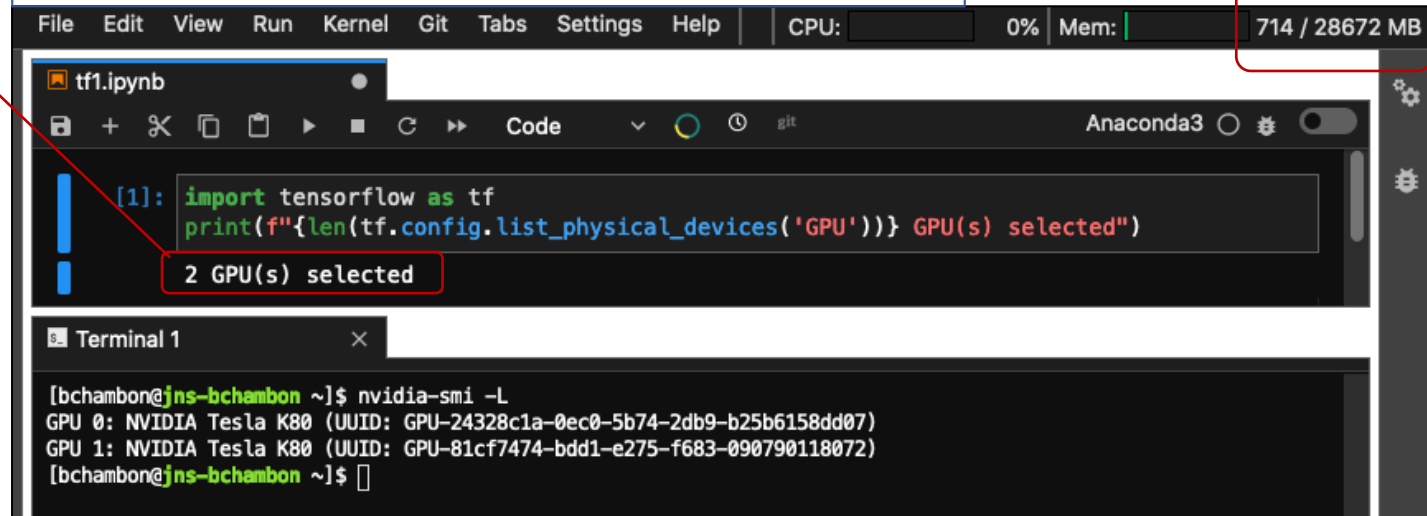
Launch My Notebooks Server

Sélection de 28 GB de RAM

Sélection de 2 GPUs

Rappel de la config hardware et software relativement au modèle de GPU choisi

Un serveur de notebooks GPU instancié avec 2 GPUs et 28 GB de RAM



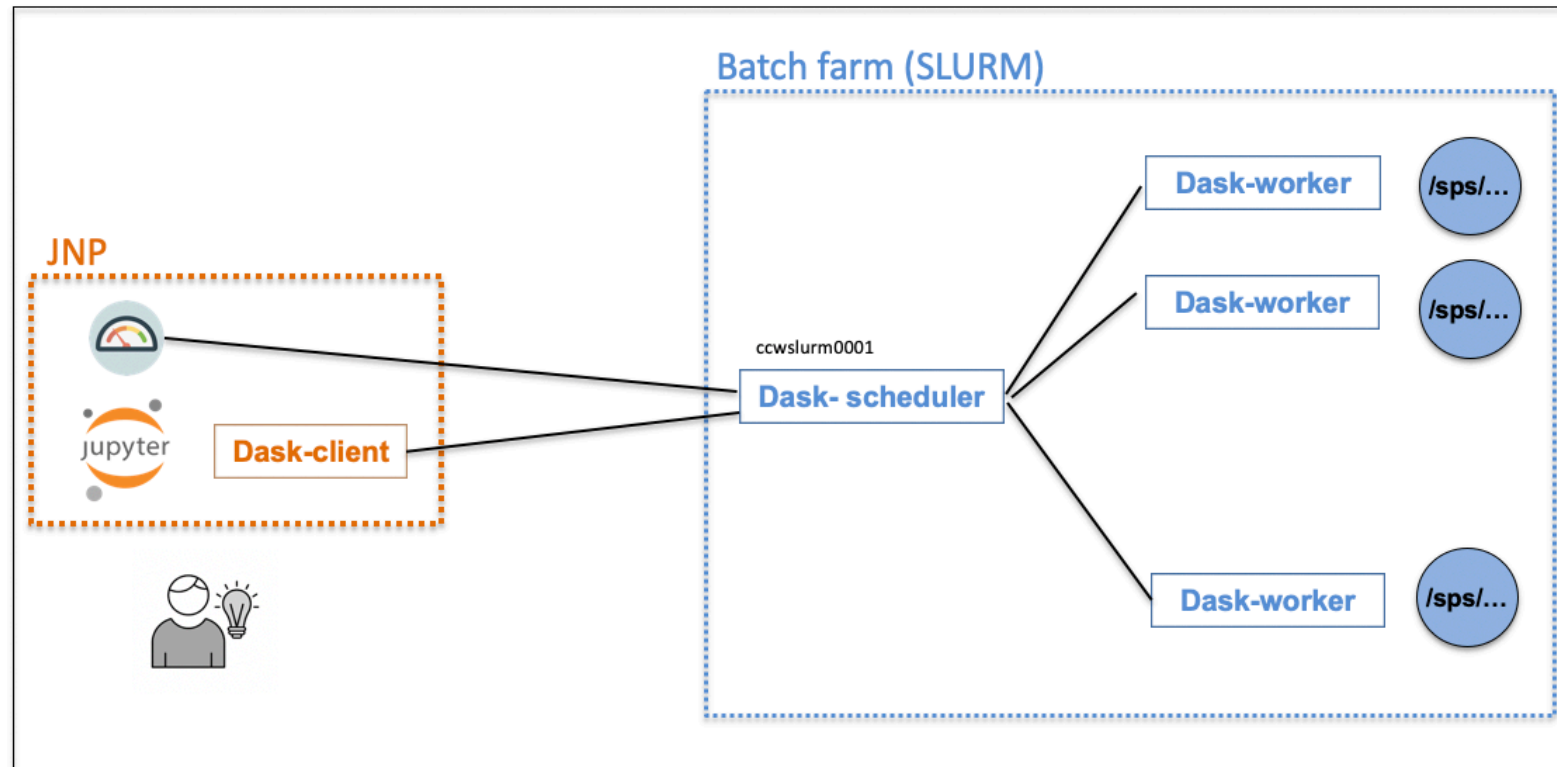
File Edit View Run Kernel Git Tabs Settings Help | CPU: 0% Mem: 714 / 28672 MB

```
[1]: import tensorflow as tf
print(f"{len(tf.config.list_physical_devices('GPU'))} GPU(s) selected")
```

2 GPU(s) selected

```
[bchambon@jns-bchambon ~]$ nvidia-smi -L
GPU 0: NVIDIA Tesla K80 (UUID: GPU-24328c1a-0ec0-5b74-2db9-b25b6158dd07)
GPU 1: NVIDIA Tesla K80 (UUID: GPU-81cf7474-bdd1-e275-f683-090790118072)
[bchambon@jns-bchambon ~]$
```

- Objectif
 - Permettre l'analyse interactive de gros volumes de données
 - Depuis le notebook (pour l'interactivité) avec utilisation des ressources de la ferme de batch SLURM (pour la performance)
 - En distribuant des tâches Dask sur, potentiellement, plusieurs centaines de jobs
- Architecture



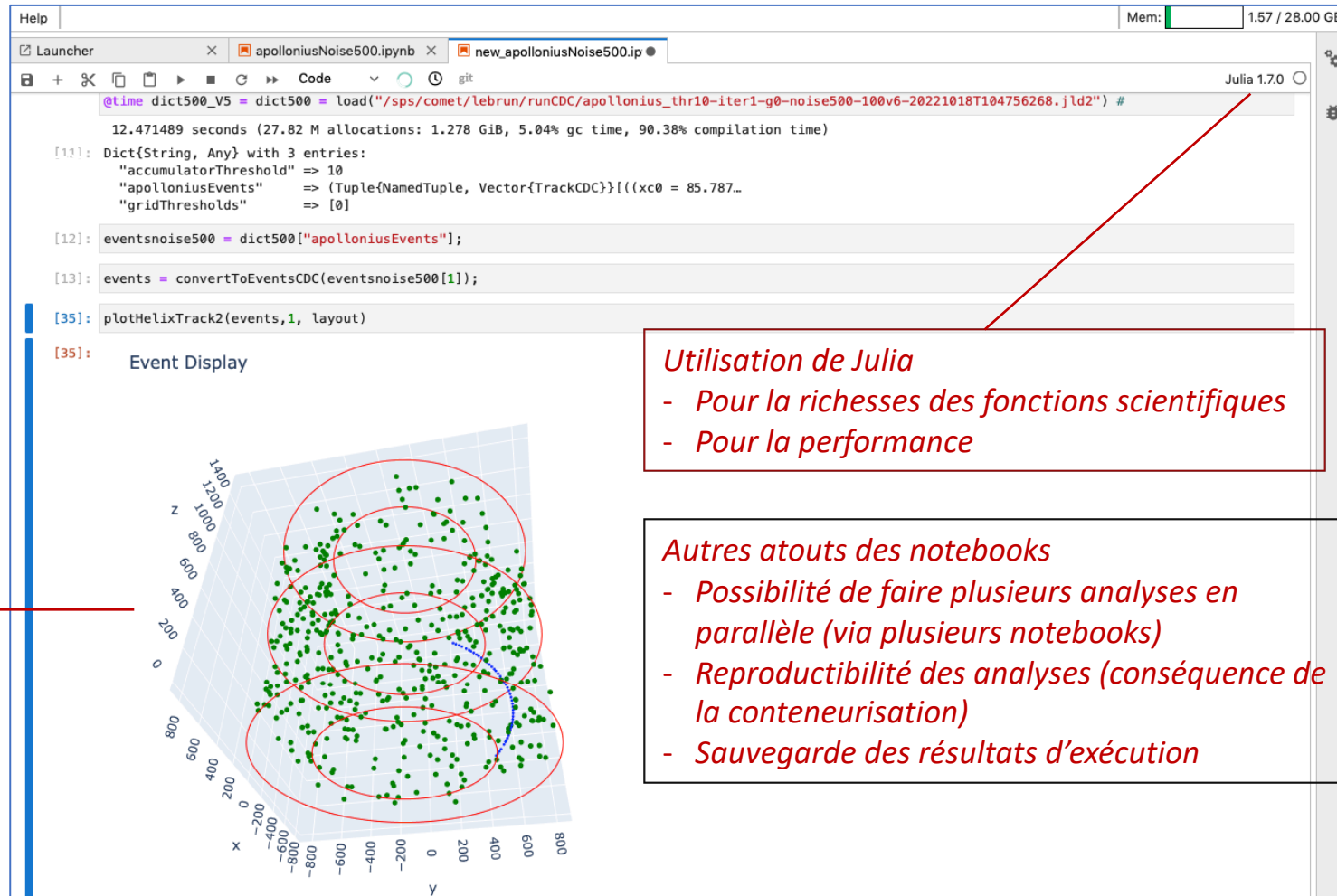
- Moyen
 - Disposer du droit d'accès à cette fonctionnalité (en **bêta-test**). En faire la demande, via un ticket
- L'utilisateur disposera alors
 - D'un serveur de notebooks permettant d'interagir avec SLURM (via le package Python 'dask4in2p3')
 - Pourra spécifier le nombre de jobs, la RAM et le temps de résidence des jobs
 - Pourra spécifier un environnement virtuel de son choix (où sera installé le package 'dask4in2p3')
 - D'un accès au dashboard Dask, fournissant des métriques relativement aux dask-workers
- Documentation
 - Cette fonctionnalité est en **bêta-test**, donc il n'y pas encore de doc officielle, mais ...
 - Il existe ce document [dask4in2p3](#), ainsi que des notebooks d'exemples [demodask4in2p3](#)

Voir exemple
en annexe A2

3 cas d'usage fournis par nos utilisateurs

Cas d'utilisation n° 1 : Développement en langage Julia

Utilisation de la plateforme pour faire du « Track finding ».
(Patrice Lebrun, IP2I Lyon, expérience COMET)



The screenshot shows a Jupyter notebook with the following code and output:

```
@time dict500_V5 = dict500 = load("/sps/comet/lebrun/runCDC/apollonius_thr10-iter1-g0-noise500-100v6-20221018T104756268.jld2") #
12.471489 seconds (27.82 M allocations: 1.278 GiB, 5.04% gc time, 90.38% compilation time)
[11]: Dict{String, Any} with 3 entries:
  "accumulatorThreshold" => 10
  "apolloniusEvents"      => (Tuple{NamedTuple, Vector{TrackCDC}}){((xc0 = 85.787...
  "gridThresholds"       => [0]
[12]: eventsnoise500 = dict500["apolloniusEvents"];
[13]: events = convertToEventsCDC(eventsnoise500[1]);
[35]: plotHelixTrack2(events,1, layout)
[35]: Event Display
```

The plot shows a 3D visualization of particle tracks (green dots) with red circles indicating interaction regions. The axes are labeled x, y, and z, ranging from 0 to 1000.

Interactivité avec le
graphe de plots
(ici via plotlyJS)

Utilisation de Julia

- Pour la richesse des fonctions scientifiques
- Pour la performance

Autres atouts des notebooks

- Possibilité de faire plusieurs analyses en parallèle (via plusieurs notebooks)
- Reproductibilité des analyses (conséquence de la conteneurisation)
- Sauvegarde des résultats d'exécution

Cas d'utilisation n° 2 : Formation au ML sur GPUs

Utilisation de la plateforme pour une formation interactive sur les réseaux de convolution.
(Alexandre Boucaud, APC Paris, Rubin/LSST & Euclid)

Les étudiants utilisent la plateforme afin de prototyper les réseaux et faire leur recherche (du machine-learning pour Rubin/LSST ou Euclid) sur les GPUs en utilisant l'interactivité des notebooks.

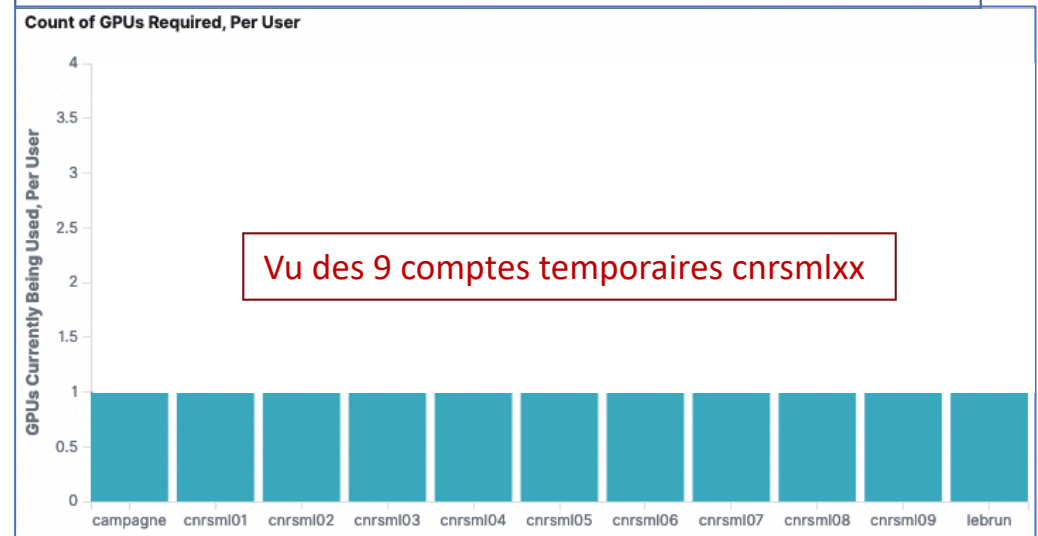
De mon côté, je fais du prototypage, des tests d'architectures, et surtout de la formation, ce pourquoi je pense que cette plateforme est un atout majeur.

Alexandre B.

3 formations réalisées en 2022 (avril, juillet et octobre)

- Mise à disposition de comptes temporaires `cnrsmlxx`, pour les participants
- Prise en compte des retours pour améliorer l'expérience utilisateur (Ex: mise à jour des versions de logiciels)

Nombre de GPUs utilisés lors de la formation d'octobre 2022

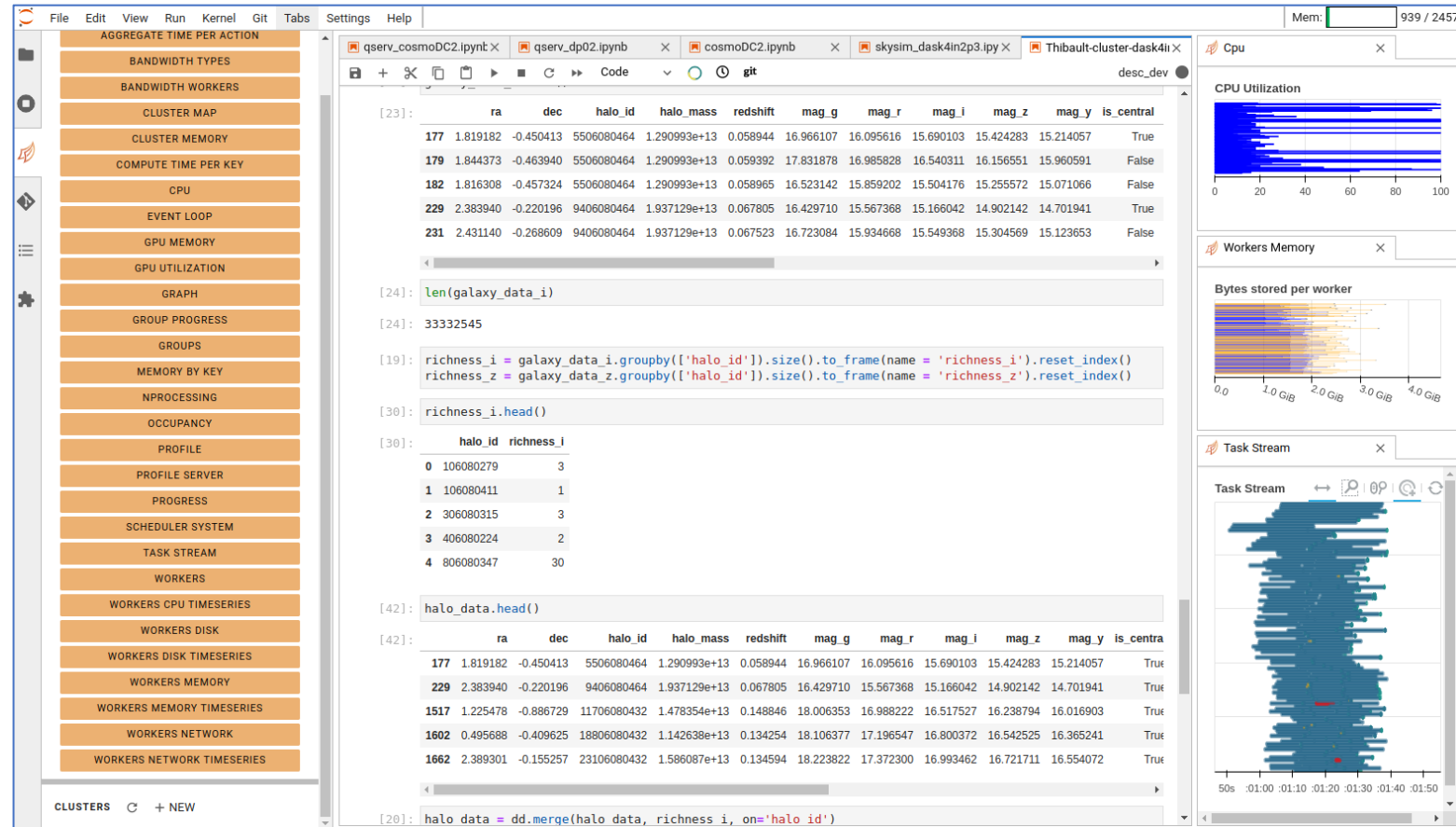


Cas d'utilisation n° 3 : Traitement distribué avec Dask+SLURM

Utilisation de la plateforme pour traiter des données issues d'une simulation cosmologique.
(Dominique Boutigny, LAPP Annecy, Rubin/LSST)

*L'objectif est de filtrer et enrichir un catalogue de galaxies issu d'une simulation cosmologique réaliste (simulation de l'évolution d'un univers dans le cadre d'un modèle cosmologique donné)
29 To de données (1572 fichiers parquet de 20 Go / fichier)*

*Le traitement est distribué sur 100 jobs SLURM avec 5 Go de RAM par job
Calcul exécuté en 7.5 minutes
À la fin on récupère un tableau de 5 millions d'amas de galaxies avec leurs caractéristiques*



```
[23]:
```

	ra	dec	halo_id	halo_mass	redshift	mag_g	mag_r	mag_i	mag_z	mag_y	is_central
177	1.819182	-0.450413	5506080464	1.290993e+13	0.058944	16.966107	16.095616	15.690103	15.424283	15.214057	True
179	1.844373	-0.463940	5506080464	1.290993e+13	0.059392	17.831878	16.985828	16.540311	16.156551	15.960591	False
182	1.816308	-0.457324	5506080464	1.290993e+13	0.058965	16.523142	15.859202	15.504176	15.255572	15.071066	False
229	2.383940	-0.220196	9406080464	1.937129e+13	0.067805	16.429710	15.567368	15.166042	14.902142	14.701941	True
231	2.431140	-0.268609	9406080464	1.937129e+13	0.067523	16.723084	15.934668	15.549368	15.304569	15.123653	False

```
[24]: len(galaxy_data_i)
```

```
[24]: 33332545
```

```
[19]: richness_i = galaxy_data_i.groupby(['halo_id']).size().to_frame(name='richness_i').reset_index()
richness_z = galaxy_data_z.groupby(['halo_id']).size().to_frame(name='richness_z').reset_index()
```

```
[30]: richness_i.head()
```

	halo_id	richness_i
0	106080279	3
1	106080411	1
2	306080315	3
3	406080224	2
4	806080347	30

```
[42]: halo_data.head()
```

	ra	dec	halo_id	halo_mass	redshift	mag_g	mag_r	mag_i	mag_z	mag_y	is_centra
177	1.819182	-0.450413	5506080464	1.290993e+13	0.058944	16.966107	16.095616	15.690103	15.424283	15.214057	True
229	2.383940	-0.220196	9406080464	1.937129e+13	0.067805	16.429710	15.567368	15.166042	14.902142	14.701941	True
1517	1.225478	-0.886729	11706080432	1.478354e+13	0.148846	18.006353	16.988222	16.517527	16.238794	16.016903	True
1602	0.495688	-0.409625	18806080432	1.142638e+13	0.134254	18.106377	17.196547	16.800372	16.542525	16.365241	True
1662	2.389301	-0.155257	23106080432	1.586087e+13	0.134594	18.223822	17.372300	16.993462	16.721711	16.554072	True

```
[20]: halo_data = dd.merge(halo_data, richness_i, on='halo_id')
```

- Matériel

- 1 serveur : VM de 16 GB RAM, 8 CPUs.

Où est déployé JupyterHub

- 18 workers :

Où sont déployés les serveurs de notebooks

- **11** VMs : 8 CPUs et 32 ou 64 GB RAM par VM

- 7 VMs pour le computing
- 4 VMs réservées aux formations

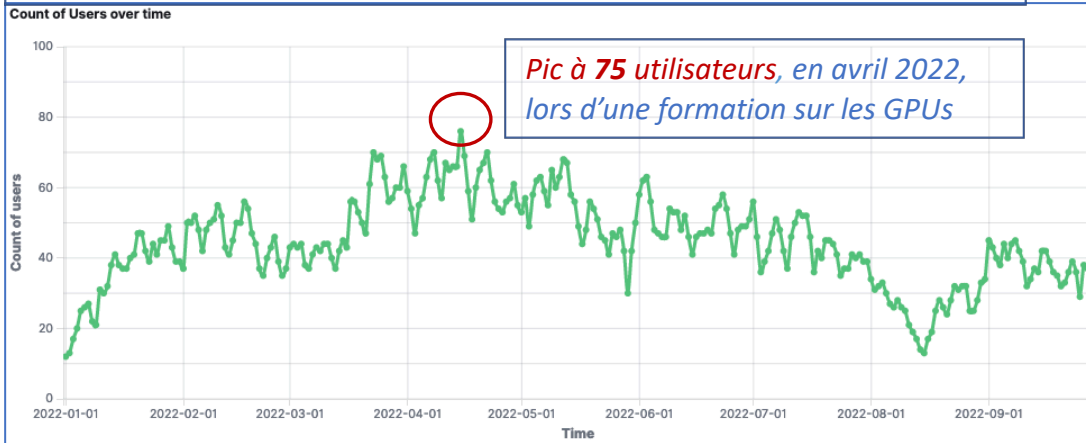
- **7** machines physiques : 16 CPUs et 130 GB RAM par machine

- 4 dédiées au computing sur GPU (modèle K80)
- 3 dédiées au computing sur CPU (pour des utilisateurs ayant des besoins en RAM et/ou E/S)

Les VMs sont fournies par OpenStack. On peut rapidement étendre le service pour les serveurs de notebooks CPU

Quelques chiffres sur l'utilisation de la plateforme

Nombre d'utilisateurs simultanés du 1^{er} janvier 2022 au 30 septembre 2022

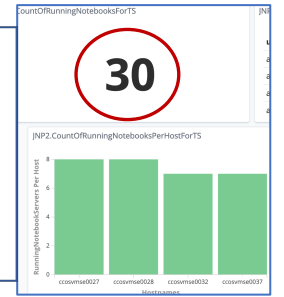


63 groupes distincts
(262 d'utilisateurs),
du 1^{er} janvier 2022 au
30 septembre 2022

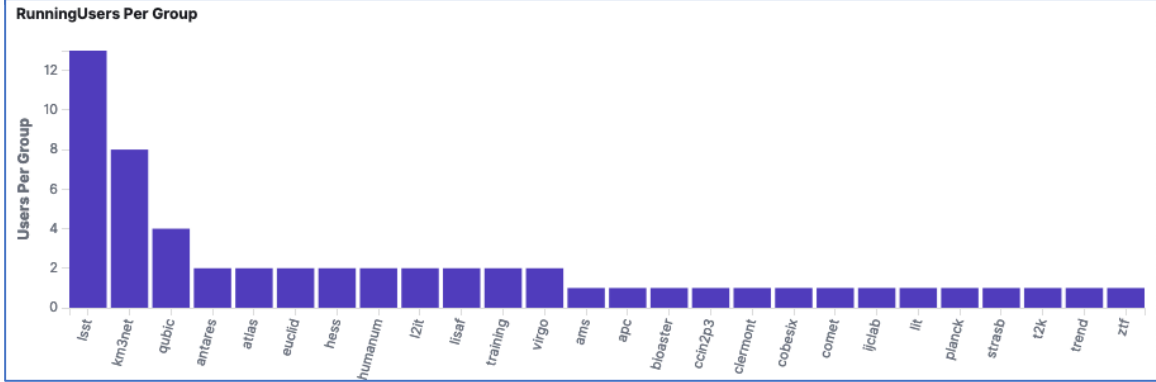
262
Distinct Users

63
Distinct Groups

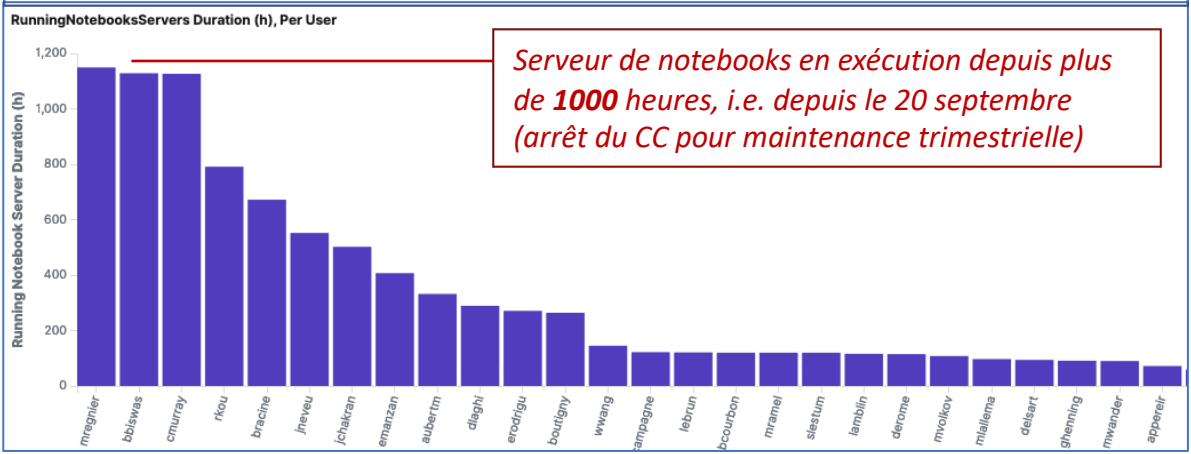
30 participants à l'ANF
'Qualité du logiciel',
en octobre 2021
Répartition sur les 4
machines dédiées aux
formations



Répartition en 26 groupes de 57 utilisateurs (vue instantanée le 14/10/22)



Durée de vie (h) des serveurs de notebooks, par utilisateur (vue instantanée le 04/11/22)



- Améliorer l'offre GPU
 - Améliorer la gestion des GPUs dont la réservation pour les cas d'usage ponctuel (cas des formations)
 - Répondre aux besoins exprimés par les utilisateurs (ex. MLflow)
- Déployer l'offre Dask+SLURM
 - Intégrer les retours des bêta-testeurs, améliorer la documentation, la distribution du package 'dask4in2p3'
 - Prévisionnel de mise en production au T1/2023
- Faire évoluer les versions des logiciels constituant les images Docker
 - Linux, Python, logiciels de ML
 - Objectif d'être en accord avec les autres services de calcul du CC
 - JupyterLab et les extensions | widgets, pour l'interface utilisateur

- Un nouveau service du CC-IN2P3
 - Ouvert à tout utilisateur disposant d'un compte 'calcul' au CC
 - Configuré pour servir des besoins variées
 - Pour de l'analyse de données, pour faire des formations
 - Pour utiliser les ressources du service en CPU | GPU, ou via Dask, des ressources de la ferme de batch SLURM
- URLs
 - Consulter la documentation <https://doc.cc.in2p3.fr>
 - Accéder au service <https://notebook.cc.in2p3.fr/>
 - Contacter le support <https://support.cc.in2p3.fr/>

Merci de votre attention

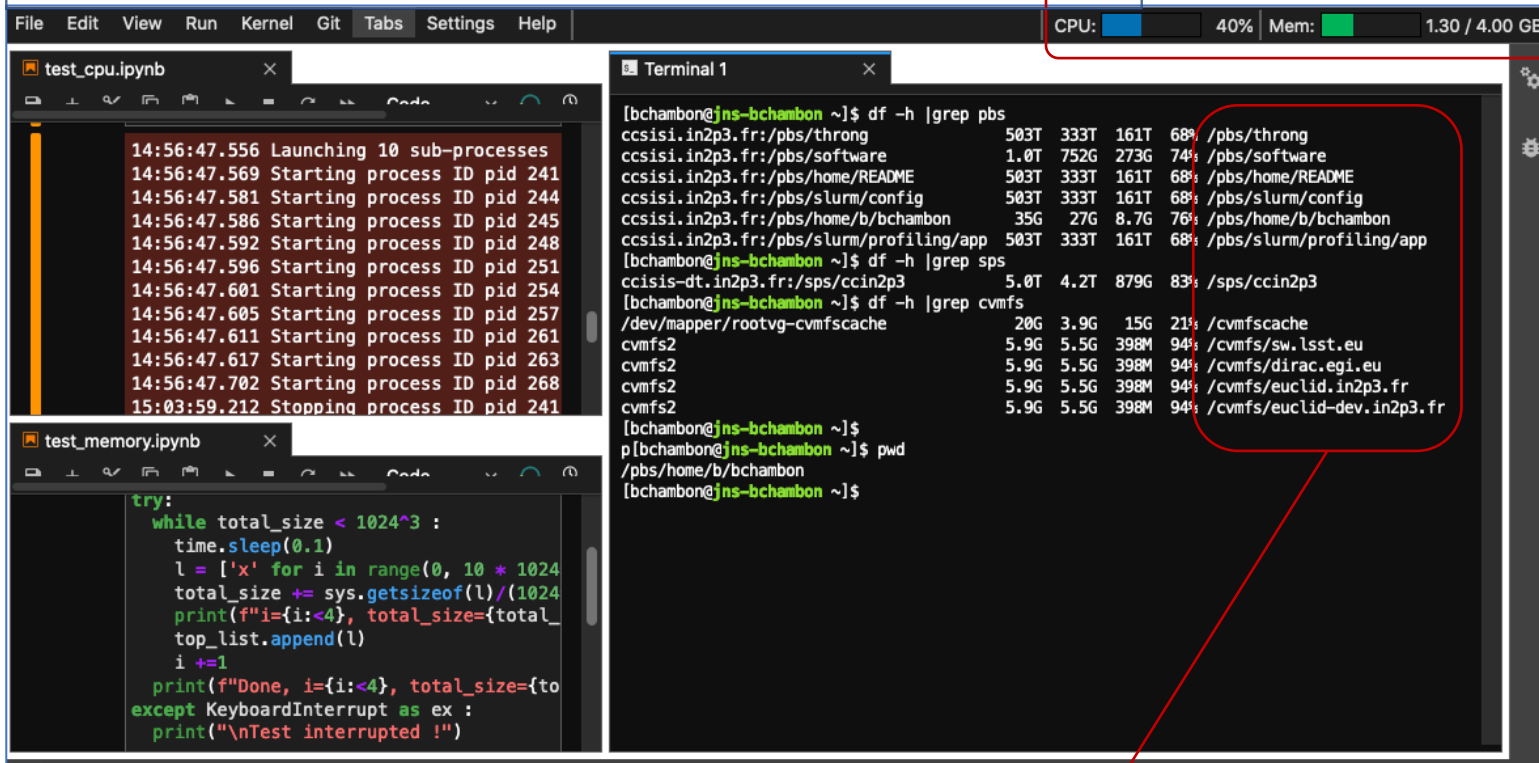
- A1/ Screenshot d'allocation de 2 CPUs et 4 GB de RAM et monitoring de ces ressources.
- A2/ Screenshot d'utilisation de Dask pour traiter 94 fichiers (format CSV), totalisant 25 millions d'entrées.
Recherche des latitudes et longitudes min | max pour la France métropolitaine
- A3/ Quelques indicateurs extraits d'un dashboard Kibana.
- A4/ Quelques métriques extraites d'un dashboard Grafana.
- A5/ Éléments logiciels développés au CC.

A1/ Allocation et monitoring des ressources

Instanciation d'un serveur de notebooks; Métriques CPU et RAM via Grafana

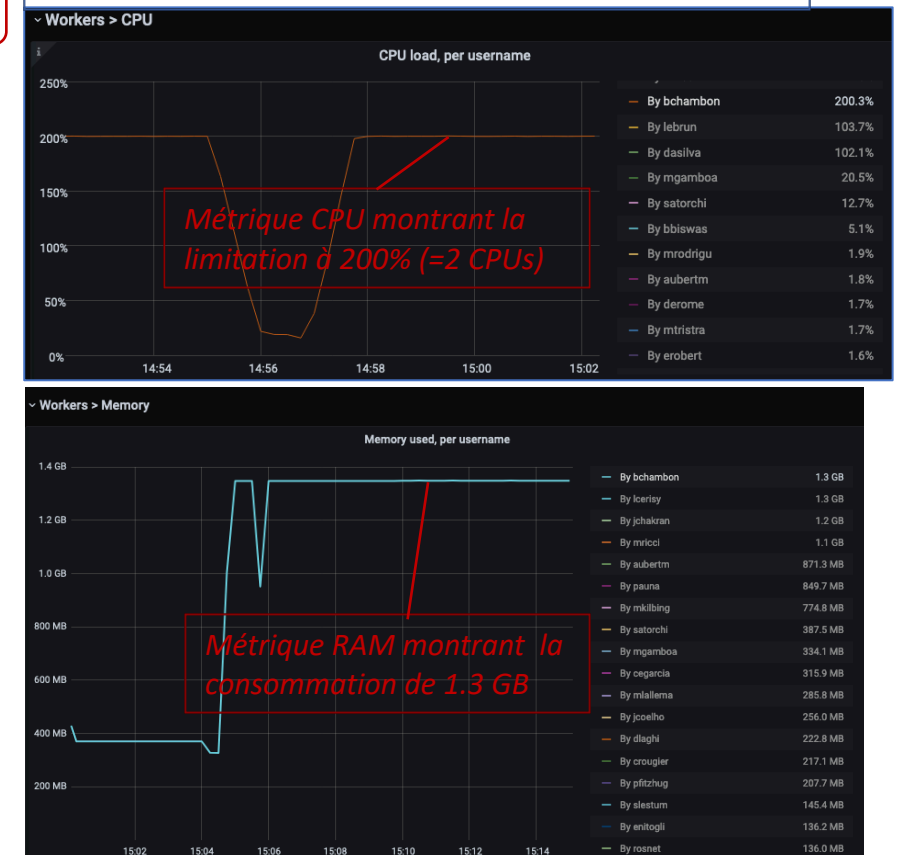
Vue instantanée de l'utilisation du CPU et de la RAM (test de la RAM lors du screenshot)

Un serveur de notebooks instancié pour bchambon, limité à 2 CPUs et 4 GB de RAM



The screenshot shows a JupyterLab interface with two notebooks and a terminal. The top notebook, 'test_cpu.ipynb', shows the execution of 10 sub-processes. The terminal window displays the output of 'df -h | grep pbs' and 'df -h | grep sps', showing disk usage for various paths. The bottom notebook, 'test_memory.ipynb', shows a Python script that generates a large list of 'x' characters to test memory usage. The system status bar at the top indicates CPU usage at 40% and memory usage at 1.30 / 4.00 GB.

Dashboard (Grafana) de monitoring du CPU et de la RAM



Métrique CPU montrant la limitation à 200% (=2 CPUs)

Métrique RAM montrant la consommation de 1.3 GB

Les systèmes de stockage disponible pour le logon 'bchambon'

A2/ Utilisation de la fonctionnalité 'Dask+SLURM'

Recherche des latitudes et longitudes min|max pour la France métropolitaine : 94 fichiers (format CSV), total de 4.2 GB, 25 millions d'entrées

1) Spécification de 4 dask-workers (-o-> 4 jobs SLURM)

1) Setting up the Dask cluster

```
# Importing the class from the module (package.module)
from dask4in2p3.dask4in2p3 import Dask4in2p3

# Creating a dask4in2p3 object
dask4in2p3 = Dask4in2p3()

# Launching a dask-scheduler and one or several dask workers.
# For each dask worker setting :
# partition (--partition option of the sbatch command)
# memory in GB (--mem option of the sbatch command)
# wallclock time (--time option of the sbatch command)
client = dask4in2p3.new_client(dask_worker_jobs=4,
                              dask_worker_memory=3,
                              dask_worker_time='00:15:00',
                              )
```

```
13:54:29,392 INFO Creating and launching the SLURM jobs(s)
13:54:29,470 INFO Waiting for the dask-scheduler SLURM job to be in RUNNING status
13:54:39,670 INFO I've got the dask-scheduler SLURM job in RUNNING status
13:54:39,673 INFO Waiting for the dask-worker SLURM job(s) to be in RUNNING status
13:54:39,718 INFO I've got the expected count (=4) dask-worker SLURM job(s) in RUNNING status
13:54:40,163 INFO A Dask client has been provided
```

2) Le vue des jobs SLURM dans un terminal Unix (4 dask-workers) + dask-scheduler

```
[bchambon@jns-bchambon ~]$ squeue --long
Tue Sep 27 14:36:00 2022
  JOBID PARTITION  NAME  USER  STATE  TIME TIME_LIMI  NODES  NODELIST(REASON)
  15089845  dask dask_wor bchambon  RUNNING  0:19  15:00  1 ccwslurm0407
  15089846  dask dask_wor bchambon  RUNNING  0:19  15:00  1 ccwslurm0406
  15089847  dask dask_wor bchambon  RUNNING  0:19  15:00  1 ccwslurm0405
  15089848  dask dask_wor bchambon  RUNNING  0:19  15:00  1 ccwslurm0404
  15089818  htc_daemo dask_sch bchambon  RUNNING  0:24  90-00:00:00  1 ccwslurm0001
```

3) Définition et soumission des tâches de calcul

```
return result
```

- Let's prepare an array of tasks

```
# Preparing an array of tasks
dask_worker_jobs = len(client.scheduler_info()['workers'])
tasks=[]
for i in range(dask_worker_jobs):
    tasks.append(get_extremas())
```

- Let's compute the task in parallel, gathering results and getting the minimum and maximum from the ones of each slices

```
import dask.dataframe as dd

try:
    logger.info(f"Launching the {len(tasks)} computing tasks")
    futures = client.compute(tasks)

    logger.info(f"Gathering results for the {len(tasks)} task(s). Please wait for the results to be ready")
    results = client.gather(futures) # wait until results are ready

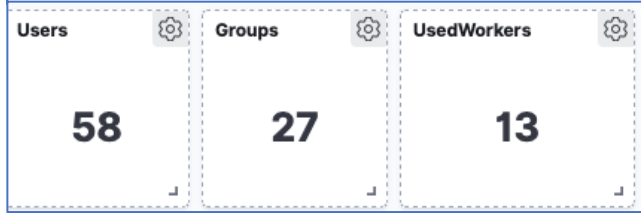
    logger.info(f"Results are available")
```

4) Obtention des résultats en 15s (contre 45s si traitement sur une seule machine)

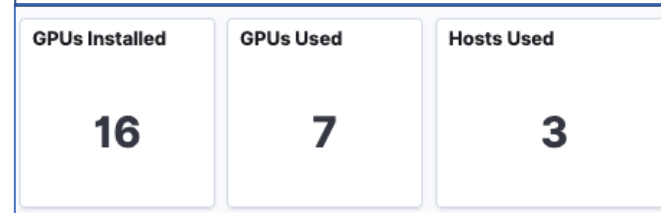
```
14:06:28,496 INFO Launching the 4 computing tasks
14:06:28,517 INFO Gathering results for the 4 task(s). Please wait for the results to be ready
14:06:45,071 INFO Results are available
-----
14:06:45,661 INFO
14:06:45,663 INFO Bray-Dunes 59123 Rue des Goelands +51.082325 +2.524649
14:06:45,665 INFO Coustouges 66260 La Mougue d'Avail +42.346985 +2.618481
14:06:45,667 INFO Lauterbourg 67630 Port du Rhin +48.963112 +8.200513
14:06:45,669 INFO Ouessant 29242 Pern +48.453735 -5.131043
-----
14:06:45,670 INFO
14:06:45,671 INFO It took 15.06 s to process 94 files and 24932730 entries
14:06:45,673 INFO Process durations per slice 15.19, 15.06, 15.45, 14.52,
14:06:45,674 INFO Files counts per slice 23.00, 23.00, 24.00, 24.00,
```

A3/ Quelques indicateurs extraits d'un dashboard Kibana

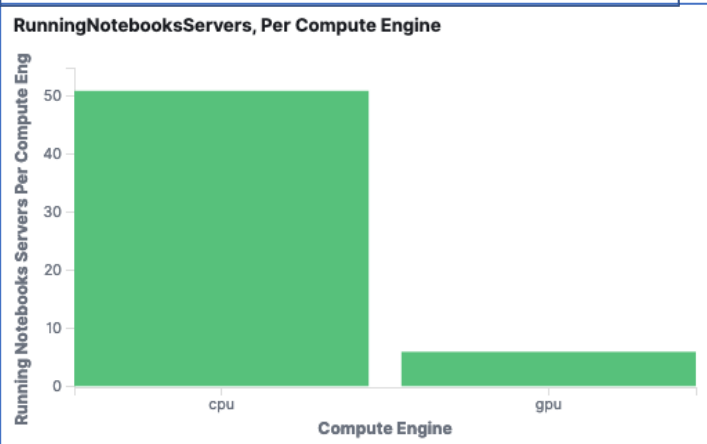
Nombre d'utilisateurs, de groupes et de machines utilisées



Nombre de GPUs disponibles et utilisés.
Nombre de machines GPUs utilisées

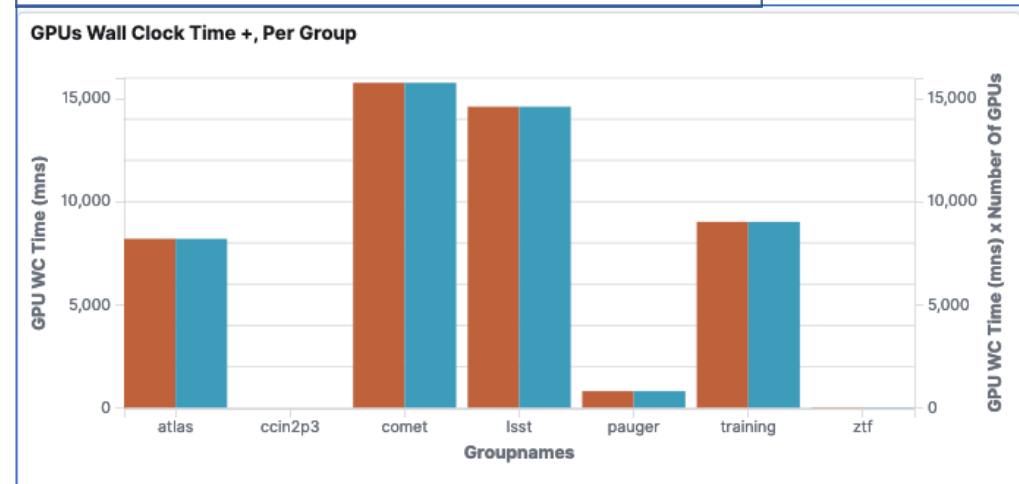


51 + 7 serveurs de notebooks CPU et GPU
(respectivement)



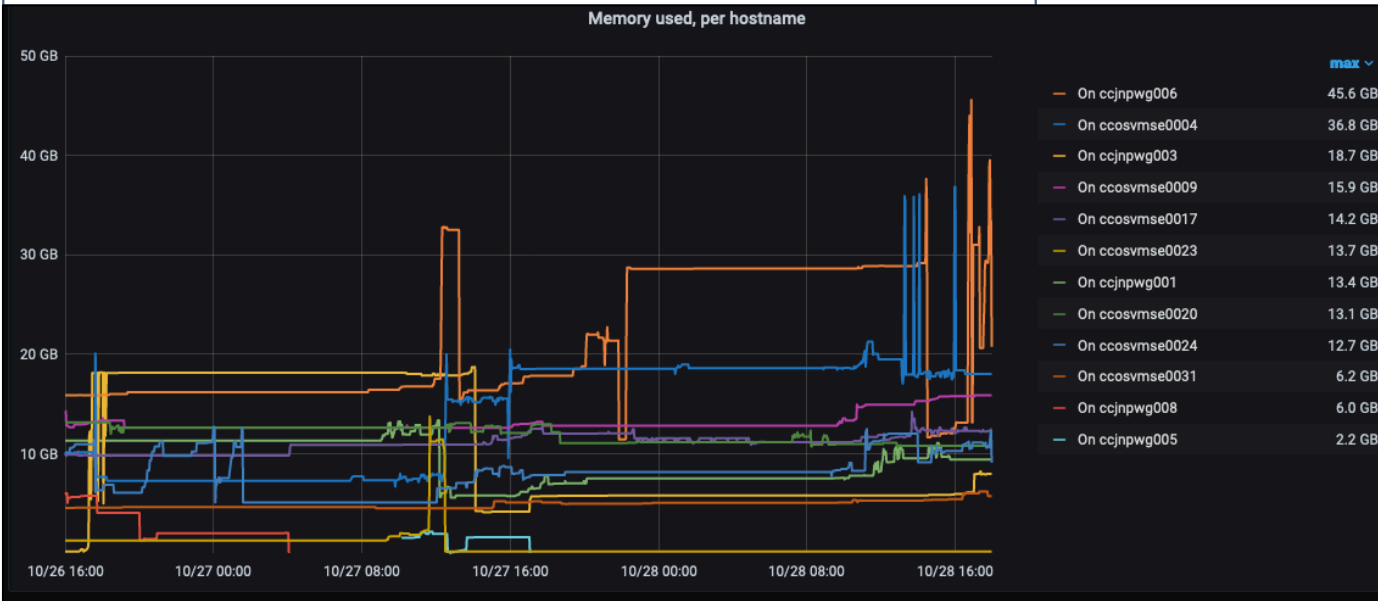
*Vue instantanée
le 14/10/22*

Temps d'utilisation (mn) des GPUs, par groupe



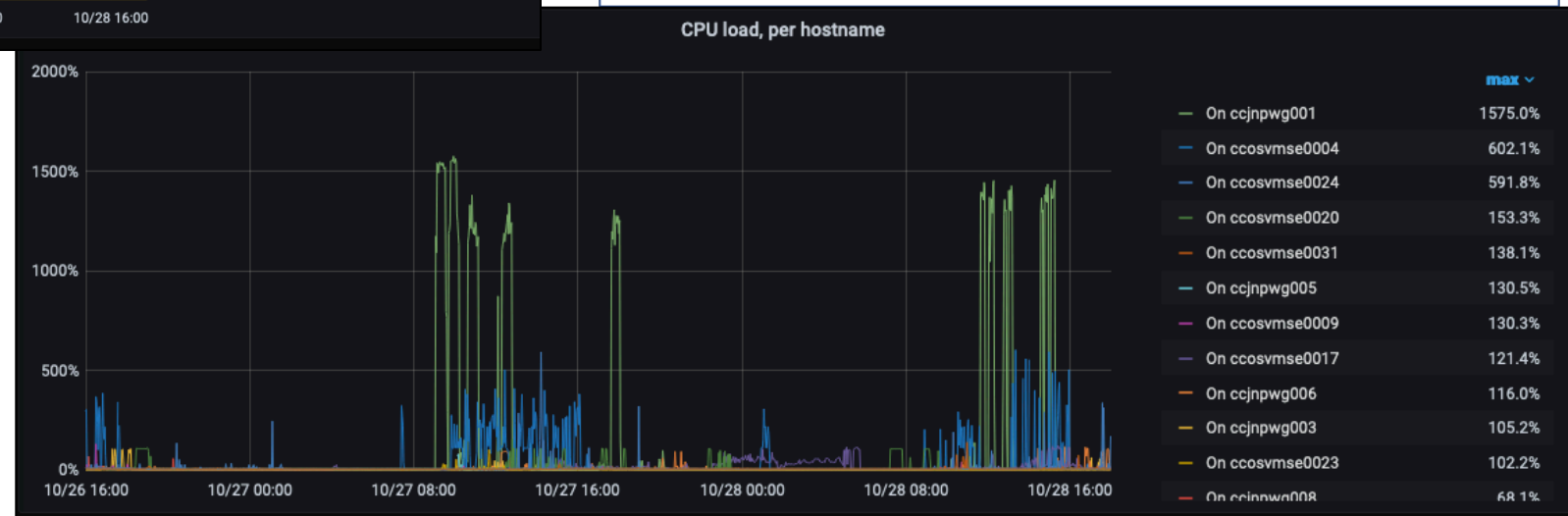
A4/ Quelques métriques extraites d'un dashboard Grafana

Consommation de RAM, par machine, sur les dernières 48 heures



Vue le
28/10/22

Utilisation des CPUs, par machine, sur les dernières 48 heures



- Configuration de JupyterHub : Code Python
 - Répartition des serveurs de notebooks sur les machines., via le mécanisme de placement (orchestrateur Swarm) des containers au regard de critères tels que `usage_type` (computing, training), `compute_engine` (cpu, gpu), `memory_resource` (small, medium, large)
 - Délégation de l'authentification (OAuth) auprès de Keycloak
 - Mise à disposition des espaces de stockage via le binding Docker
 - Gestion des contrôles d'accès, de la limite mémoire, du formulaire des options pour GPU, des éléments relatifs à Dask+SLURM
 - Logging dans Elasticsearch et dashboards via Kibana
- Préparation des images Docker : Configuration (Dockerfile)
 - Basées sur CentOS7 et enrichies de logiciels spécifiques (ex. logiciels de ML pour les GPUs)
- Mise à disposition de kernels prêts à l'emploi, dans `/pbs/software` : Configuration
 - Golang, Julia, R, ROOT/C++ Voir la documentation relativement aux [noyaux disponibles](#)
- Package 'dask4in2p3' : Code Python
 - Intégration de Dask+SLURM dans un serveur de notebooks