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

The Jupyter notebooks platform at CC-IN2P3

FJPPL meeting, Lyon, January 31 - February 1, 2023

Bernard CHAMBON, January 31, 2023

Outline

- Introduction
- Architecture
- Focusing on two features
- Demo
- Infrastructure
- Annex

Introduction

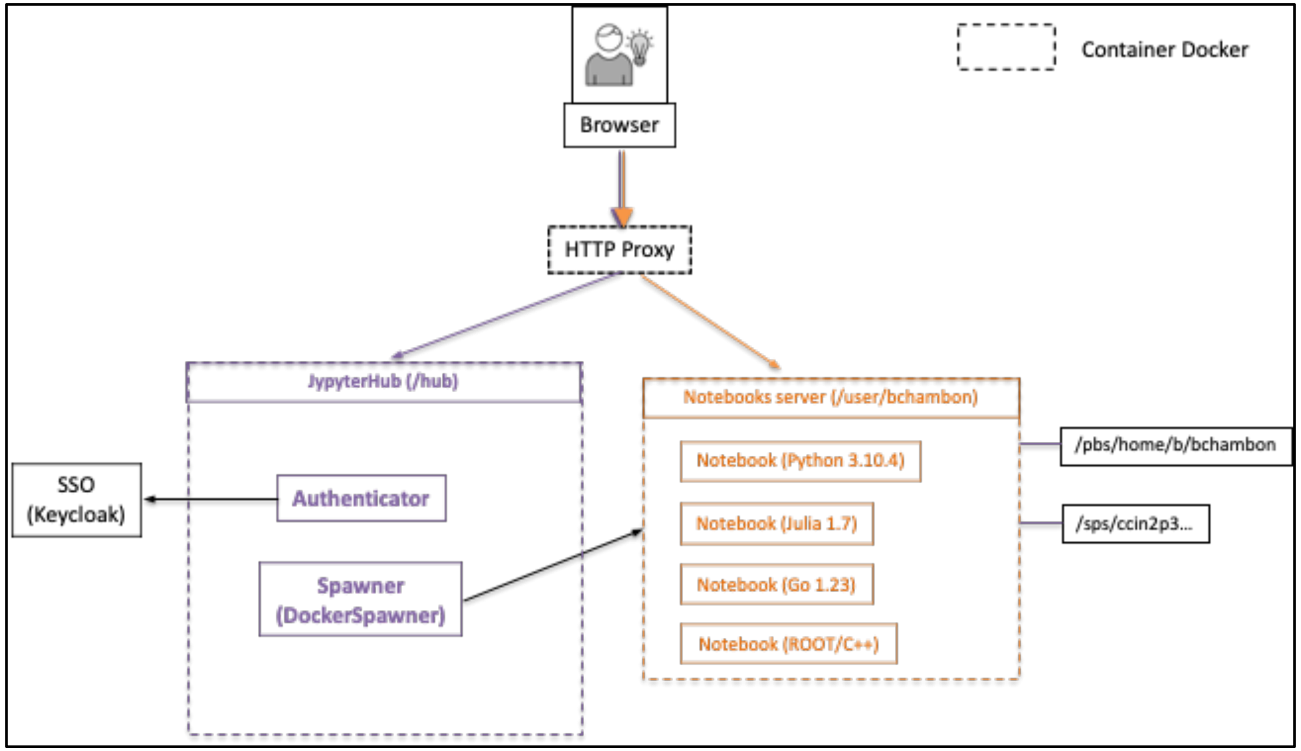
- Objective
  - Provide an analysis service for users, via a Jupyter notebook
  - With access to the same storage systems as those available on the interactive platform (cca.in2p3.fr)
  - Authentication through the SSO of CC-IN2P3

- Some key points of the Jupyter notebooks
  - User friendly
    - Running in a web browser
    - Using a same document for code, documentation, results of execution
    - Providing an UNIX terminal (without ssh-ing)
  - Multiple programming languages, via kernels (Jupyter = **Ju**lia, **Pyt**hon, **R)**
  - Large number of widgets and extensions

Architecture

Architecture : 1/3

- Built around **JupyterHub**
  - Component allowing to plug an external authentication (OAuth), to provide options forms, to spawn Docker images
  - Python config file, allowing advanced configuration



- The service is built on a Docker cluster with Swarm as orchestrator to spawn the notebooks servers on the hosts

Architecture : 2/3

- Access – authentication
  - Access allowed for all users having a ‘computing’ account, but some features are restricted to granted users
  - Authentication using OAuth to SSO Keycloak (certificate or login/password)
  - Getting additional information to provide a complete user’s profile (including all secondary groups)

- Launching the Jupyter notebooks server
  - Docker image prepared at CC-IN2P3, based on CentOS 7.6 (same as batch platform and interactive platform)
  - Container running with IDs $(\mathrm{uid}, \mathrm{gid})$ of the user
  - With the following storage systems :

• HOME area, GROUPS areas according to primary and secondary groups	Specific paths for each user
• SOFTWARE and CVMFS areas	Same path for all users

Architecture : 3/3

- Limits for RAM, CPU and lifetime
  - RAM
    - Default limit of 2 GB (quite small) but higher limit possible per user or per group
    - Several users with 16, 24 or 32 GB, even 64 GB for one user
  - CPU
    - No limit for number of CPUs

RAM, CPU and I/O consumptions are monitored

- Notebook server lifetime
  - No usage time limit, but …
  - IDLE notebooks servers are monitored and stopped after 3 days | 1 day for, respectively, CPU | GPU notebooks servers

Focusing on two features

- GPU
- Dask+SLURM

Features : GPU 1/2

- Objective
  - Allow user to run GPU code via a Jupyter notebook

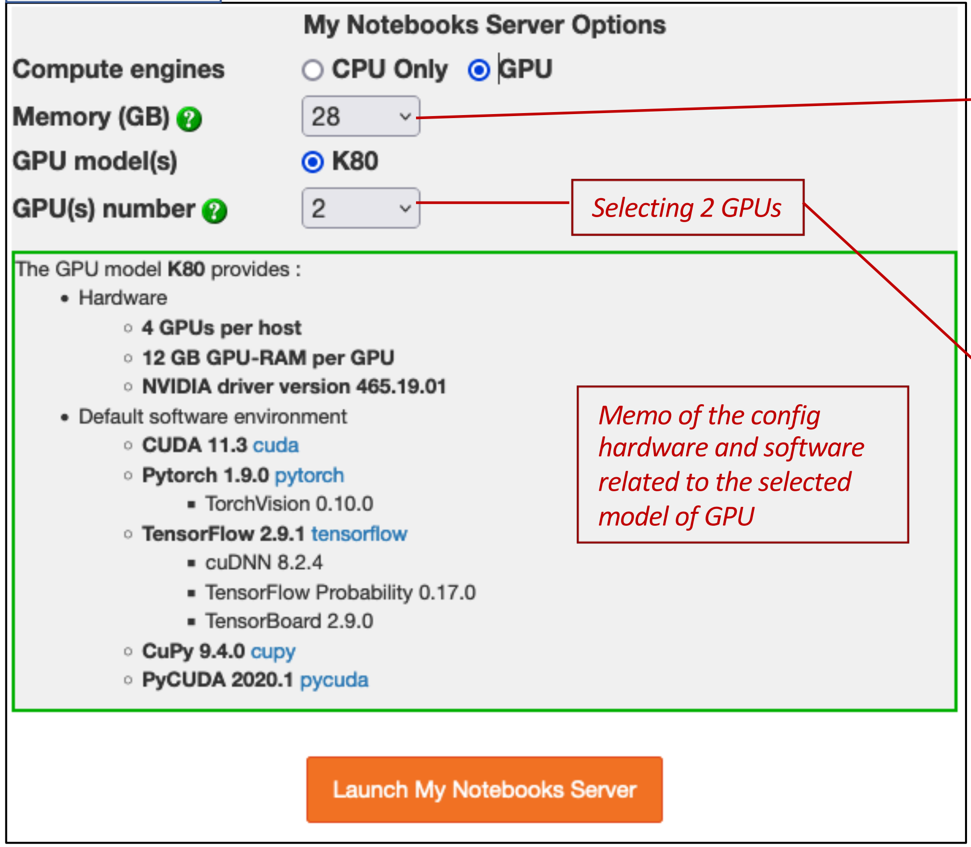
- How to
  - Granted access upon request (possible per user or per group)
  - Option form to select the model of GPU, the number of GPU
  - Also possible to select the amount of RAM of the notebooks server

- User will obtain
  - A running notebooks server with dedicated GPUs
  - Ready-to-use machine learning (ML) frameworks, since already installed in the Docker image
    - Pytorch
    - TensorFlow + TensorBoard + TensorFlow Probability

Features : GPU 2/2

The GPU options form and the resulting notebooks server

Options form



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 :

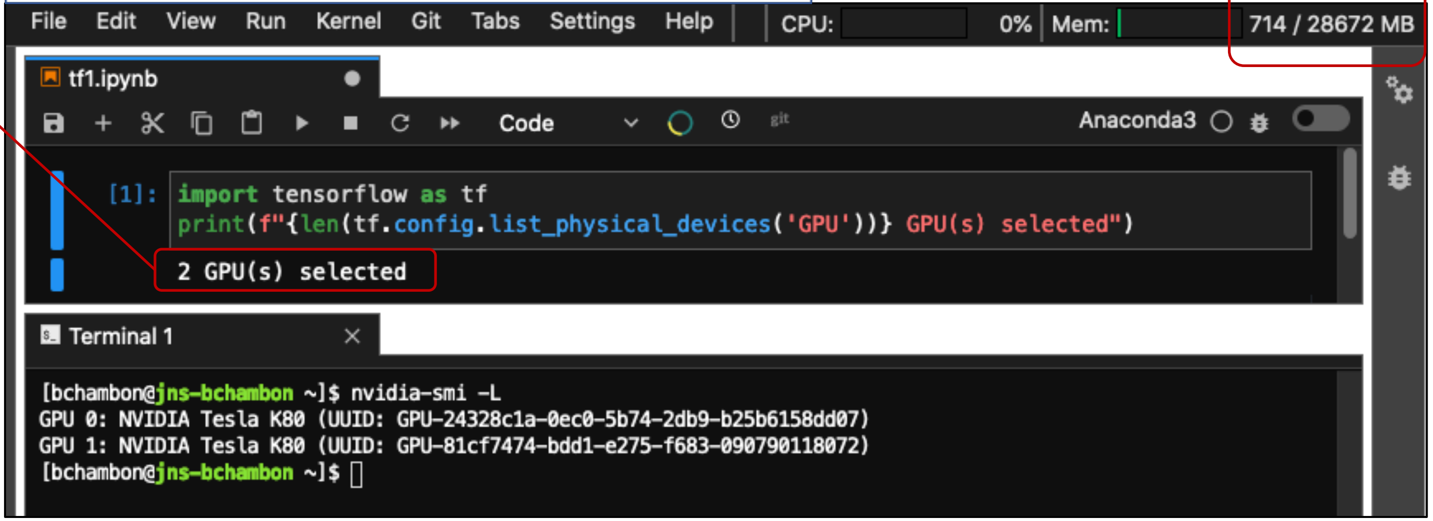
- Hardware
  - **4 GPUs per host**
  - **12 GB GPU-RAM per GPU**
  - **NVIDIA driver version 465.19.01**
- Default software environment
  - **CUDA 11.3** cuda
  - **Pytorch 1.9.0** pytorch
    - TorchVision 0.10.0
  - **TensorFlow 2.9.1** tensorflow
    - cuDNN 8.2.4
    - TensorFlow Probability 0.17.0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

Selecting 2 GPUs

Memo of the config hardware and software related to the selected model of GPU

28 GB of RAM

The GPU notebooks server with 2 GPUs and 28 GB of RAM

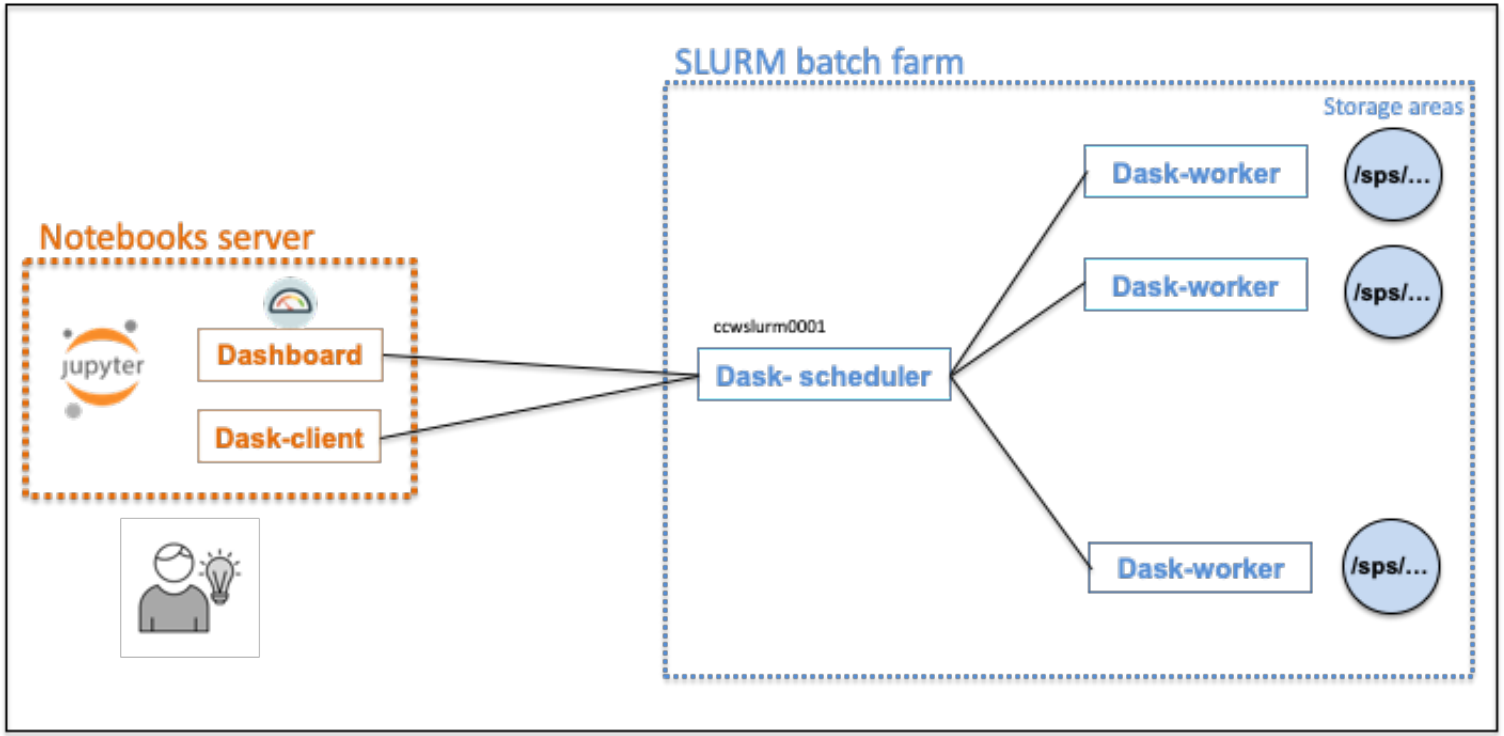


```
[1]: import tensorflow as tf
     print(f"{len(tf.config.list_physical_devices('GPU'))} GPU(s) selected")
     2 GPU(s) selected
```

```
[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 ~]$ 
```

Features : Dash+SLURM 1/2

- Objective
  - Allow user to analyse huge amount of data
  - From notebooks server (for interactivity) and by using resources from SLURM batch farm (for performance)
  - By spreading computing tasks, with Dask, over several hundreds of SLURM jobs
- How to
  - Granted access upon request (possible per user or per group)
  - Architecture



Features : Dask+SLURM 2/2

- User will obtain
  - A running a notebooks server allowing to interact with the SLURM batch farm to
    - Specify the number of jobs (= dask-workers), the duration and the RAM per job (same value for all jobs)
    - Specify the virtual environment to use (= where the package ‘dask4in2p3’ is installed)
  - An integrated dashboard, via dask-labextension, displaying metrics related to the dask-workers

- Under the hood : See annex for more details

  To make all the parts working together, a package ‘**dask4in2p3**’ has been built

  User must install it in the Python virtual environment used on the SLURM batch farm side

- Status & documentation
  - A beta-test feature for now; Production release deployment planned for 2023/Q1
  - Documentation : dask4in2p3
  - Examples : demodask4in2p3

The Jupyter notebooks platform at CC-IN2P3

Demo

Using Dask to process ~100 data files,

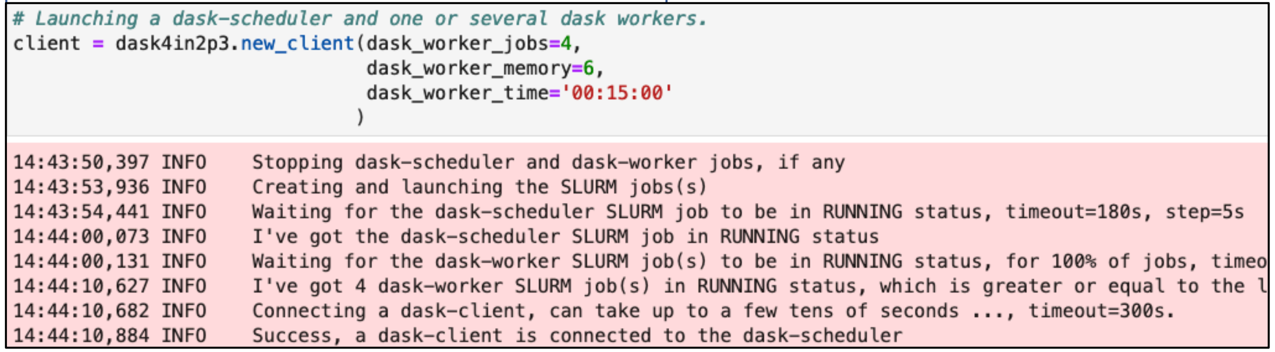
representing all the locations of mainland France,

for a total amount of 25 millions of entries

Screenshot of the demo Dask+SLURM

Searching the min|max latitudes and longitudes of locations from mainland France : 94 files, 25 millions entries

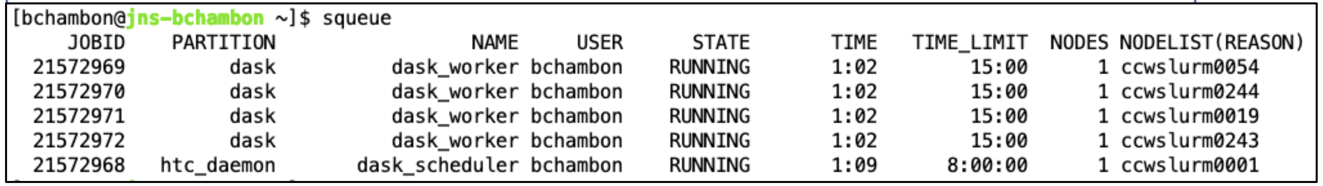
1) Specifying 4 dask-workers (--> 4 SLURM jobs)



```
# Launching a dask-scheduler and one or several dask workers.
client = dask4in2p3.new_client(dask_worker_jobs=4,
                               dask_worker_memory=6,
                               dask_worker_time='00:15:00'
                              )
```

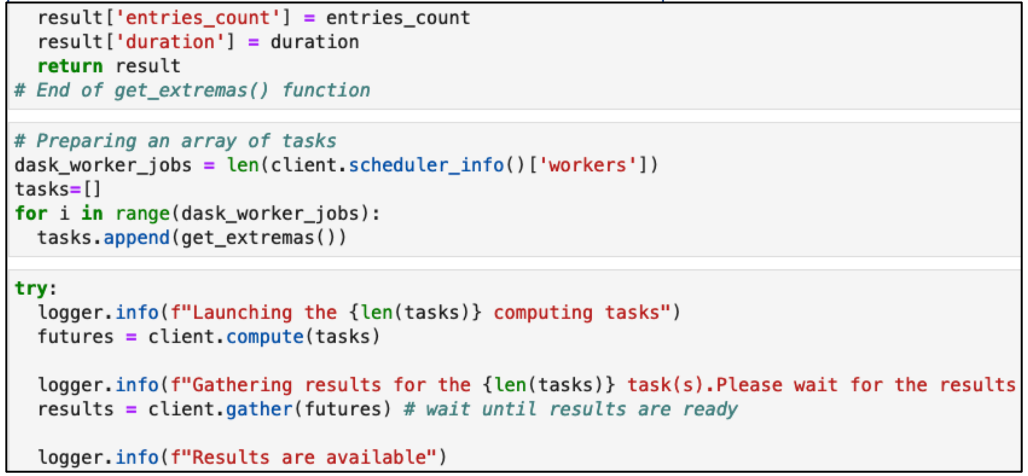
```
14:43:50,397 INFO    Stopping dask-scheduler and dask-worker jobs, if any
14:43:53,936 INFO    Creating and launching the SLURM jobs(s)
14:43:54,441 INFO    Waiting for the dask-scheduler SLURM job to be in RUNNING status, timeout=180s, step=5s
14:44:00,073 INFO    I've got the dask-scheduler SLURM job in RUNNING status
14:44:00,131 INFO    Waiting for the dask-worker SLURM job(s) to be in RUNNING status, for 100% of jobs, timeo
14:44:10,627 INFO    I've got 4 dask-worker SLURM job(s) in RUNNING status, which is greater or equal to the l
14:44:10,682 INFO    Connecting a dask-client, can take up to a few tens of seconds ..., timeout=300s.
14:44:10,884 INFO    Success, a dask-client is connected to the dask-scheduler
```

2) View of the SLURM jobs (4 dask-workers + 1 dask-scheduler)



```
[bchambon@jns-bchambon ~]$ squeue
    JOBID    PARTITION              NAME     USER       STATE       TIME   TIME_LIMIT  NODES NODELIST(REASON)
  21572969         dask        dask_worker bchambon   RUNNING       1:02        15:00      1 ccwslurm0054
  21572970         dask        dask_worker bchambon   RUNNING       1:02        15:00      1 ccwslurm0244
  21572971         dask        dask_worker bchambon   RUNNING       1:02        15:00      1 ccwslurm0019
  21572972         dask        dask_worker bchambon   RUNNING       1:02        15:00      1 ccwslurm0243
  21572968   htc_daemon     dask_scheduler bchambon   RUNNING       1:09      8:00:00      1 ccwslurm0001
```

3) Specifying then running the computing tasks



```
  result['entries_count'] = entries_count
  result['duration'] = duration
  return result
# End of get_extremas() function
```

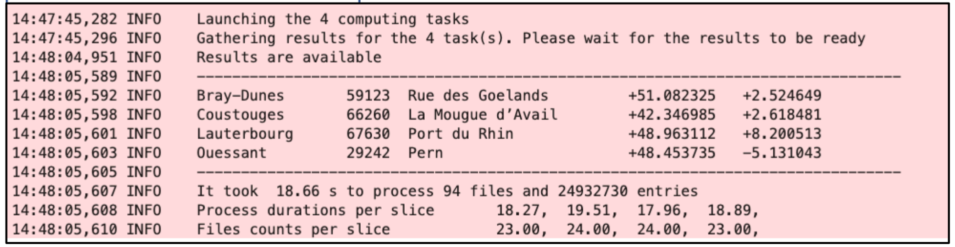
```
# Preparing an array of tasks
dask_worker_jobs = len(client.scheduler_info()['workers'])
tasks=[]
for i in range(dask_worker_jobs):
  tasks.append(get_extremas())
```

```
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
  results = client.gather(futures) # wait until results are ready

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

4) Displaying the results



```
14:47:45,282 INFO    Launching the 4 computing tasks
14:47:45,296 INFO    Gathering results for the 4 task(s). Please wait for the results to be ready
14:48:04,951 INFO    Results are available
14:48:05,589 INFO    ------------------------------------------------------------------------------
14:48:05,592 INFO    Bray-Dunes       59123  Rue des Goelands         +51.082325   +2.524649
14:48:05,598 INFO    Coustouges       66260  La Mougue d'Avail        +42.346985   +2.618481
14:48:05,601 INFO    Lauterbourg      67630  Port du Rhin             +48.963112   +8.200513
14:48:05,603 INFO    Ouessant         29242  Pern                     +48.453735   -5.131043
14:48:05,605 INFO    ------------------------------------------------------------------------------
14:48:05,607 INFO    It took  18.66 s to process 94 files and 24932730 entries
14:48:05,608 INFO    Process durations per slice       18.27,  19.51,  17.96,  18.89,
14:48:05,610 INFO    Files counts per slice            23.00,  24.00,  24.00,  23.00,
```

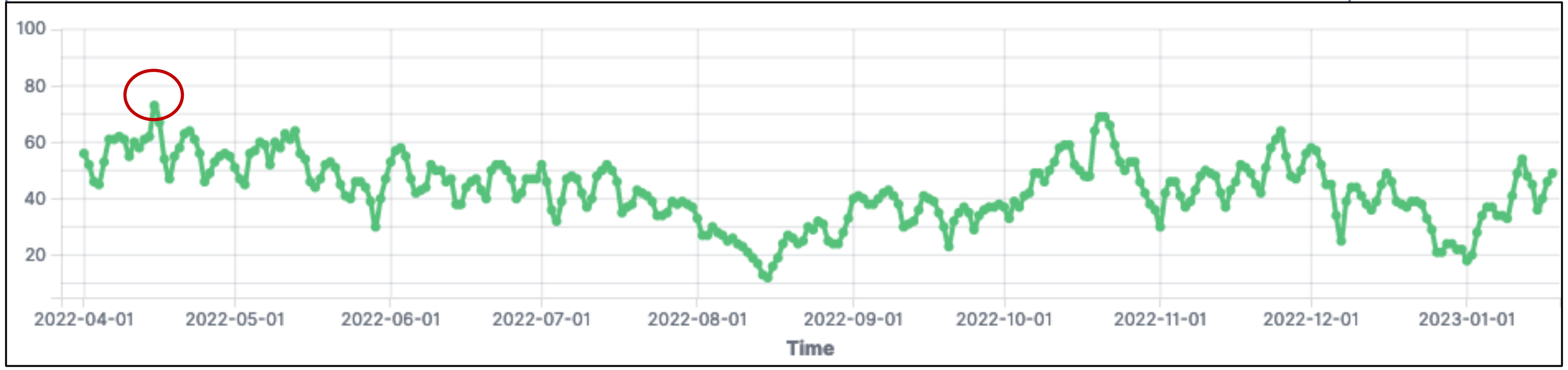
Infrastructure

Infrastructure : 1/2

- Hardware
  - 1 server : VM 16 GB RAM, 8 CPUs. where JupyterHub runs
  - 18 workers : where notebooks servers run
    - **11** VMs : 8 CPUs, 32 or 64 GB RAM, per host
      - 7 VMs for computing
      - 4 VMs dedicated for training
    - **7** bare metal hosts: 16 CPUs, 130 GB RAM, 1 Gbps I/O, per host
      - 4 dedicated for computing on GPU (model K80)
      - 3 dedicated for computing on CPU (for users with high requirements in terms of RAM or I/O)

  - This infrastructure can serve up to ~100 users. Currently ~ 50-60 simultaneous users

Number of users over the last 9 months (up to **73** users during a training session)



Infrastructure : 2/2

- Software development
  - For JupyterHub configuration : mainly Python code but also shell and JavaScript
    - Spreading notebooks servers on hosts, via docker placement (Swarm orchestrator), according to criteria like usage_type (computing, training), compute_engine (cpu, gpu), memory_resource (small, medium, large, huge)
    - Authentication via OAuth to Keycloak
    - Managing access controls, memory limits, placement criteria from config files (.yml config files)
    - Managing the GPU options form
    - Customization of user's Docker container (e.g. binding of storage areas mount points)
  - For Docker images management
    - Built at CC-IN2P3, on the CI of gitlab.in2p3.fr
  - For package 'dask4in2p3' : Python code
  - For monitoring
    - Using jsonlogger, ElasticSearch and Kibana to collect, persist and build dashboard on the platform usage (See example in Annex)
    - Using cAdvisor, Prometheus and Grafana to collect, persist and build dashboard on resources consumptions (See example in Annex)
  - For 'ready-to-use' Jupyter kernels : Golang, Julia, R, ROOT/C++

Outcome

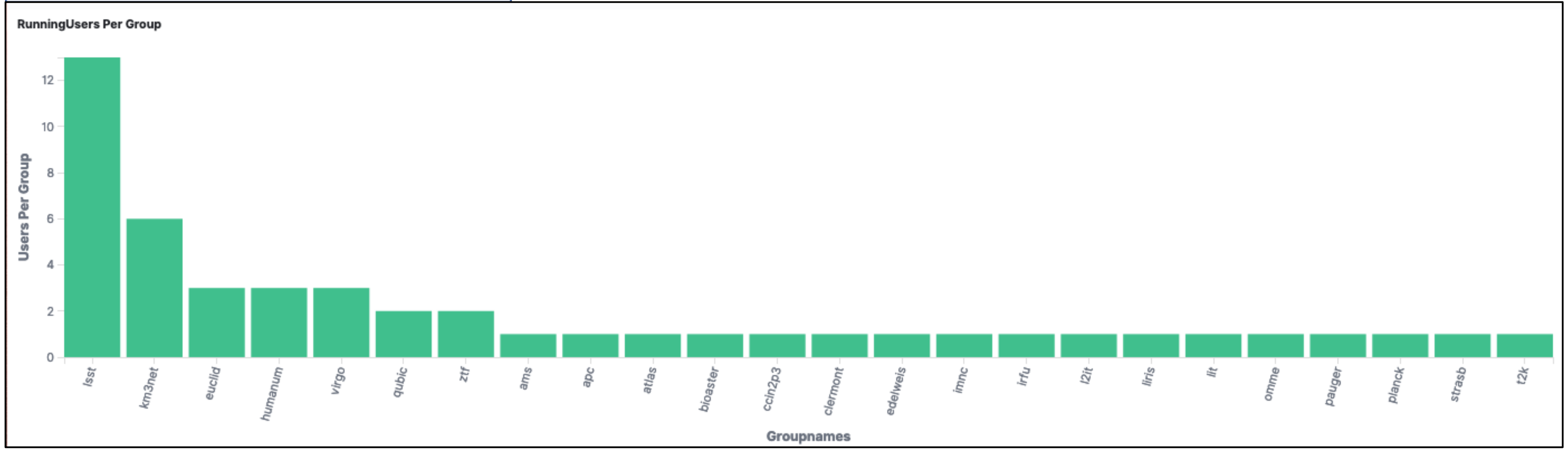
- A new service at CC-IN3P3
  - In production since July 2020 for CPU, since April 2021 for GPU, Dask+SLURM planed for 2023/Q1
  - Available for all users having a ‘computing’ account
  - For data analysis, but also for training sessions
  - Providing both CPU or GPU resources
  - Enabling the CPU resources usage of the SLURM batch farm, by using Dask

- URLs
  - Read the documentation https://doc.cc.in2p3.fr
  - Access to the service https://notebook.cc.in2p3.fr/
  - Ask for support https://support.cc.in2p3.fr/

Annex

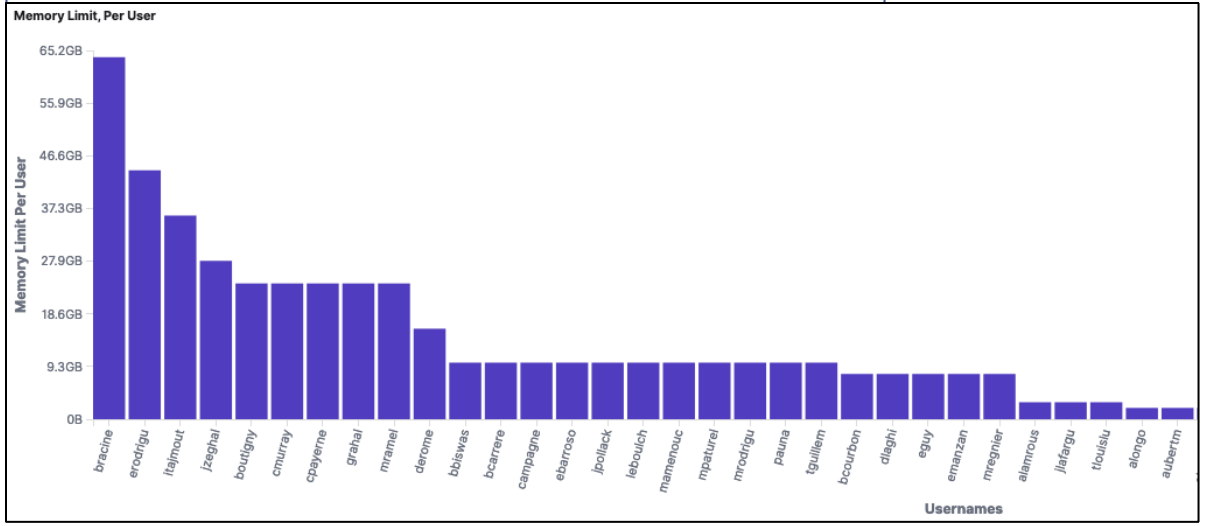
Metrics on the platform usage, from dashboard Kibana

Number of users, per group (49 users, 24 groups)

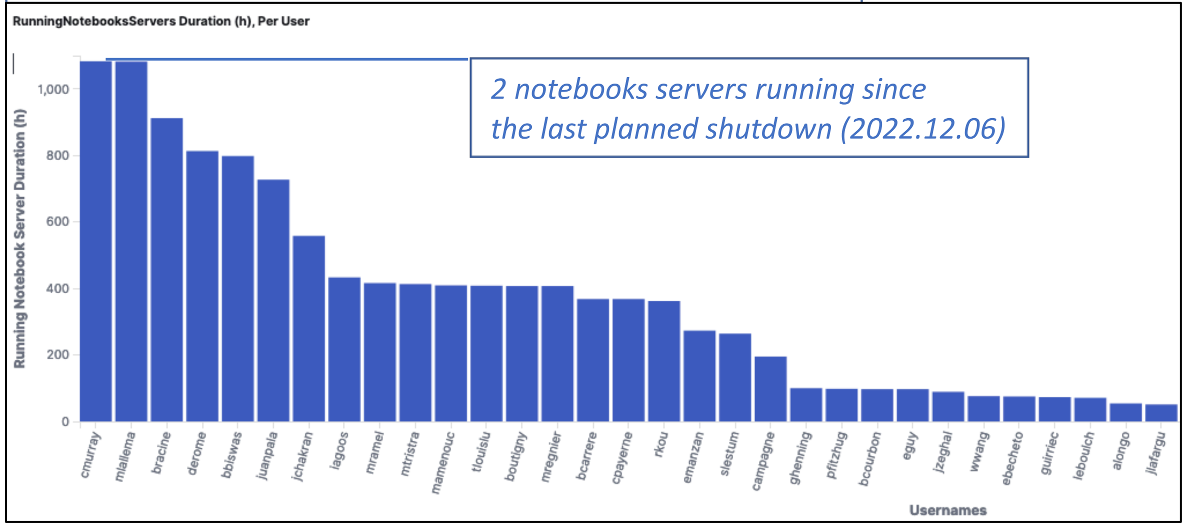


Screenshots on 2023.01.20

Memory limit (GB) of notebooks servers, per user (truncated graph on X axe)



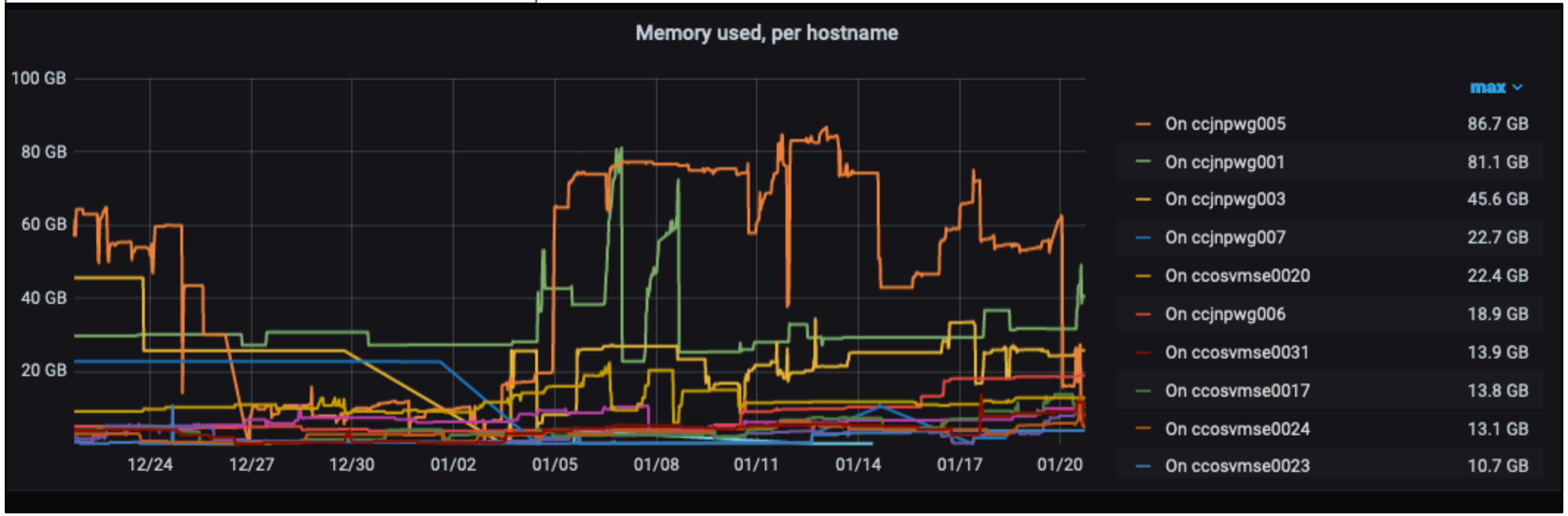
Lifetime (hours) of notebooks servers, per user (truncated graph on X axe)



2 notebooks servers running since the last planned shutdown (2022.12.06)

Metrics on the resources usage, from dashboard Grafana

Max memory used, per host, over the last month

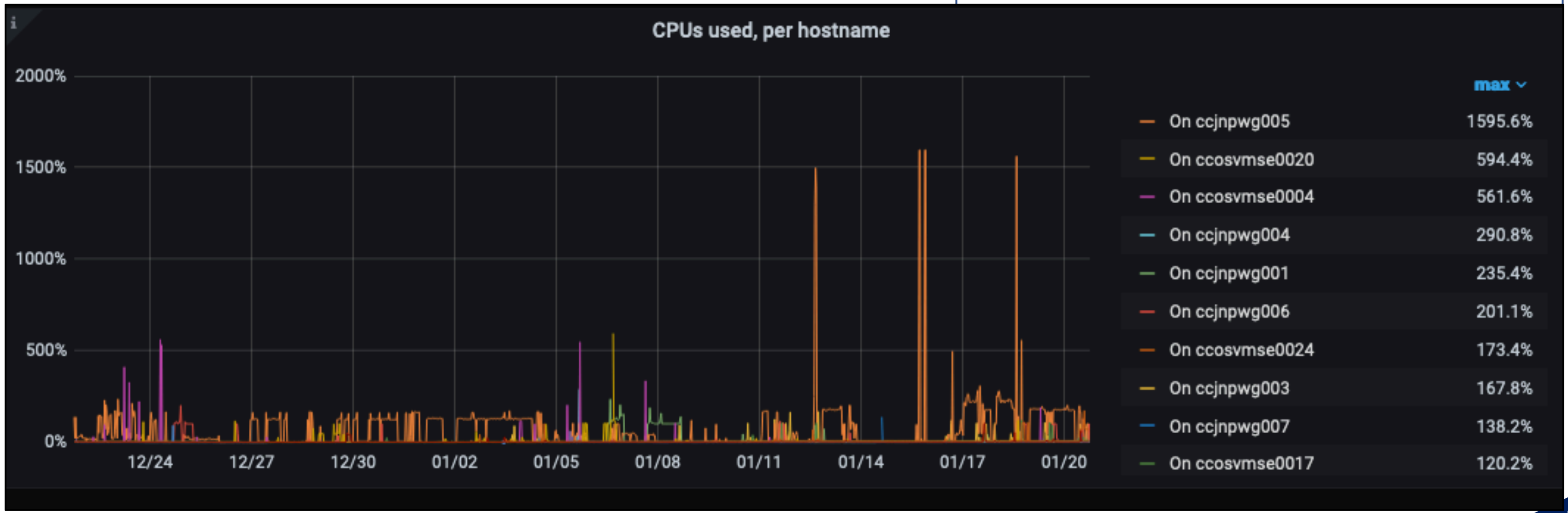


Memory used, per hostname

	max
On ccjnpwg005	86.7 GB
On ccjnpwg001	81.1 GB
On ccjnpwg003	45.6 GB
On ccjnpwg007	22.7 GB
On ccosvmse0020	22.4 GB
On ccjnpwg006	18.9 GB
On ccosvmse0031	13.9 GB
On ccosvmse0017	13.8 GB
On ccosvmse0024	13.1 GB
On ccosvmse0023	10.7 GB

Screenshots on 2023.01.20

Max number of CPUs used, per host, over the last month



CPUs used, per hostname

	max
On ccjnpwg005	1595.6%
On ccosvmse0020	594.4%
On ccosvmse0004	561.6%
On ccjnpwg004	290.8%
On ccjnpwg001	235.4%
On ccjnpwg006	201.1%
On ccosvmse0024	173.4%
On ccjnpwg003	167.8%
On ccjnpwg007	138.2%
On ccosvmse0017	120.2%

About Dask+SLURM : Implementation

- On JupyterHub side
  - Network ports management : two ports per notebooks server
  - Dask certificate management : for authentication between dask-client, dask-scheduler, dask-workers
  - SLURM jobs management when notebooks server ends

- On Jupyter notebooks server side
  - API to interact with SLURM jobs and connect a dask-client
  - Dask certificate management : transporting certificate from notebooks server to SLURM worker-nodes
  - Tracking user's requirements and logging : number of jobs, amount of memory, etc.
  - Many other things …

⇒ package '**dask4in2p3**'

This package must be installed both on the notebooks server side and on the SLURM batch farm side

- On notebooks server side
  - Installed in the Docker image providing the container
- On SLURM batch farm side
  - Must be installed, by the end-user, in a Python virtual environment available in the SLURM batch farm

(A 'ready-to-use' virtual environment is provided, by default, for demo purpose only)

About Dask+SLURM : Memo of the features

- Admin point of view
  - SLURM client availability on JNP worker-nodes
  - Network ACLs (2 ports /notebooks server) between JNP worker-nodes and the SLURM worker ccwslurm0001 (‘htc_daemon’)
  - Authentication between Dask components by using certificates
  - Access control per logon and/or per group, via config file

- User point of view
  - Package ‘**dask4in2p3**’ to be installed by end-user in a virtual env.
  - SLURM jobs submitting with user’s logon and user’s group as SLURM account
  - Jobs stopped on timeout or when the notebooks server stops or on user’s request (‘*close()*’ method)
  - Shared area between SLURM batch farm and notebooks server (at least for ‘scheduler_info.json’)
  - ‘dask-worker-space’ directory set to $/\mathrm{tmp}$
  - Jobs’ stdout+err sent to files, into $\log/$ directory
  - Available parameters (but with default values)
    - virtual environment, SLURM partition, number of jobs, amount of memory, wall clock time
    - percentage of dask-workers jobs being running before connecting a dask-client, improving response time with large amount of jobs