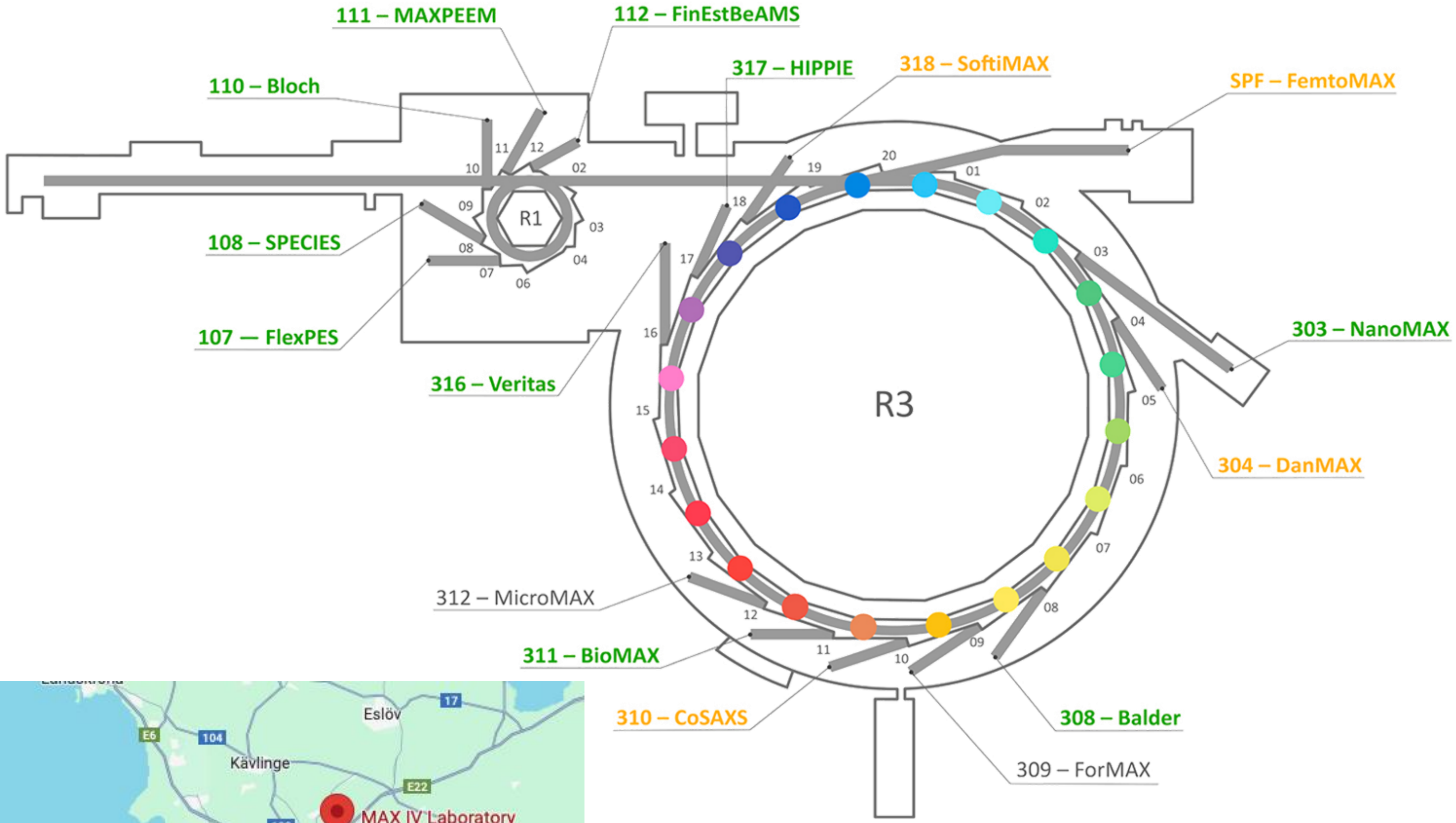
JupyterHub on Kubernetes Deployment at MAX IV

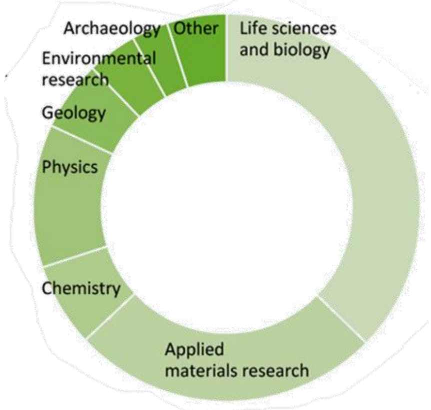
Andrii Salnikov,
Zdenek Matej, Dmitrii Ermakov, Jason Brudvik

Mail To: andrii.salnikov@maxiv.lu.se

MAX IV

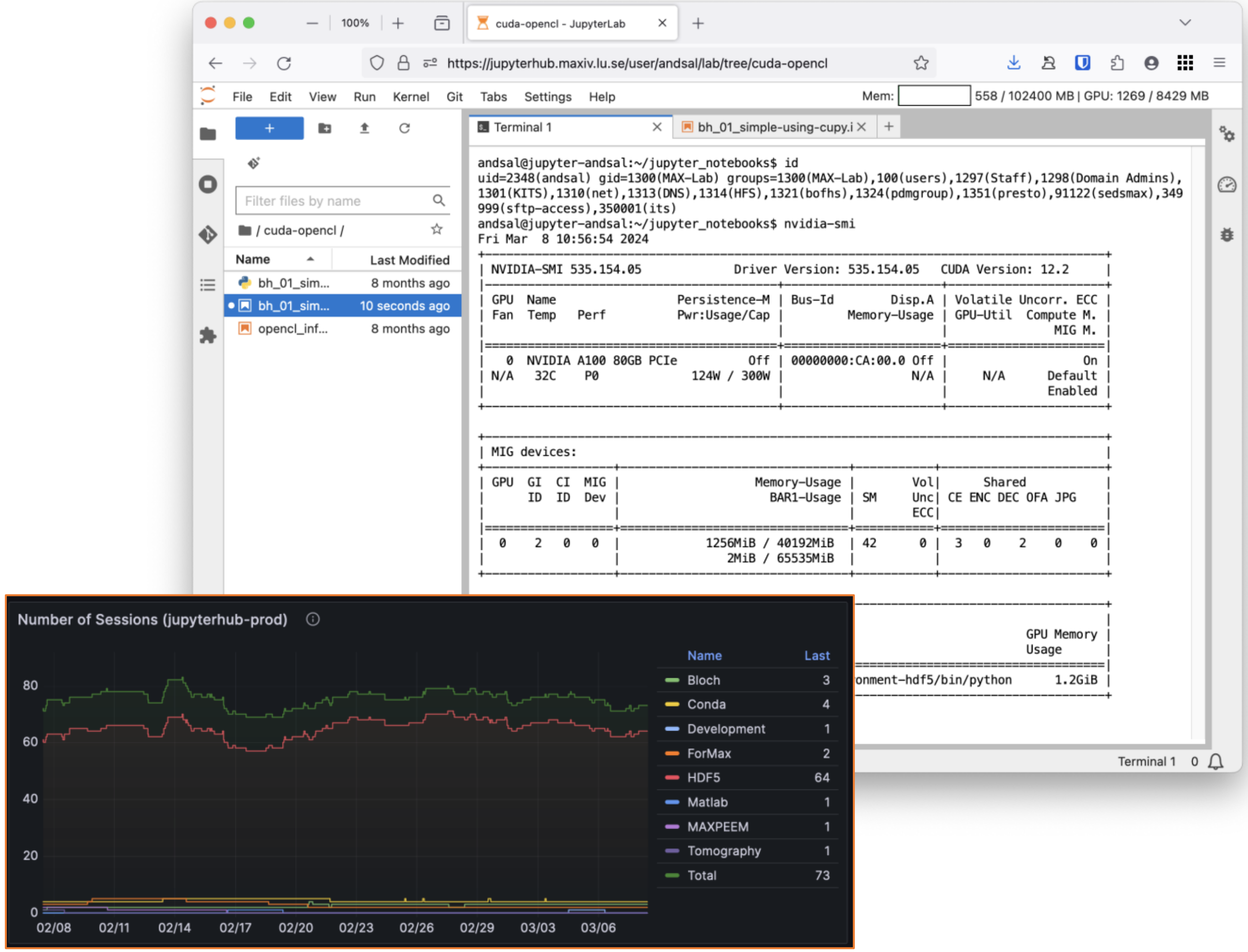


MAX IV Laboratory is a Swedish national synchrotron laboratory that has operated as a user facility since 2016.



MAX IV Interactive data analysis platform

- Container images with JupyterLab frontend
  - Jupyter Docker Stacks based
  - Kernels as part of container, custom Conda environment or as Apptainer image (new)
- Kubernetes cluster
  - as a **resource pool**:
    - Moderate CPU
    - Large RAM
    - V100/A100 GPUs
  - as a **deployment platform:**
    - review/prod/next lifecycle
    - CI testing of notebook images
- Shared service for staff and researchers
  - Remote-desktop style experience
  - Resources overcommit

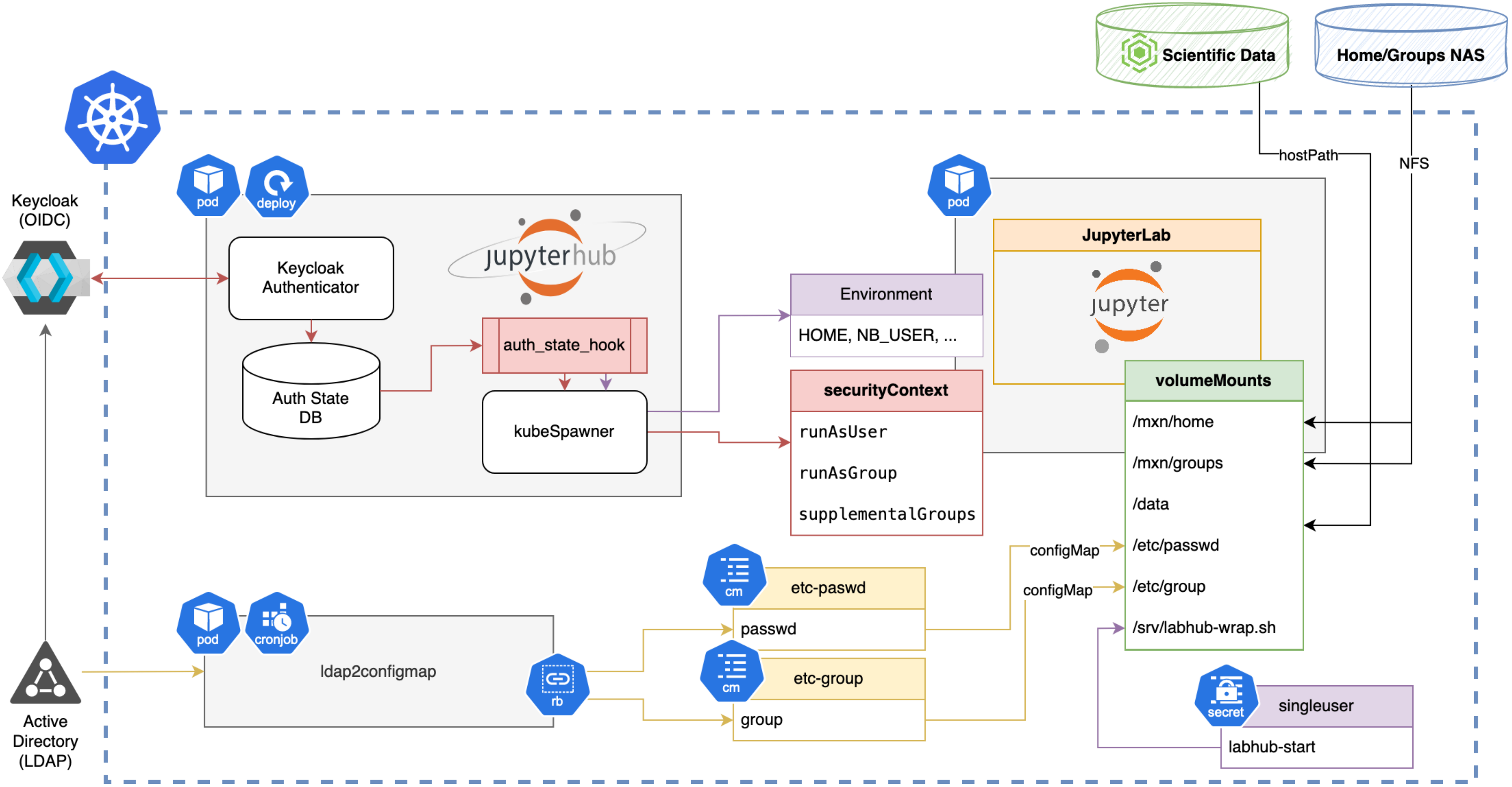


Goals and technical requirements

- **Key objective: fully unprivileged** container environment that operates seamlessly **with existing LDAP user credentials**
- **Functional Requirements:**
  - Integration with **MAX IV storage systems** (home, group, data)
  - Run **any notebook images** without modifications
  - Ensure available **resources visibility**
  - Efficient **sharing of available GPU** resources between users
  - Observability of **usage metrics**
- **Operation Requirements:**
  - Zero to JupyterHub with Kubernetes Helm Chart **without** modifications
  - Just custom hooks and proper `values.yaml`

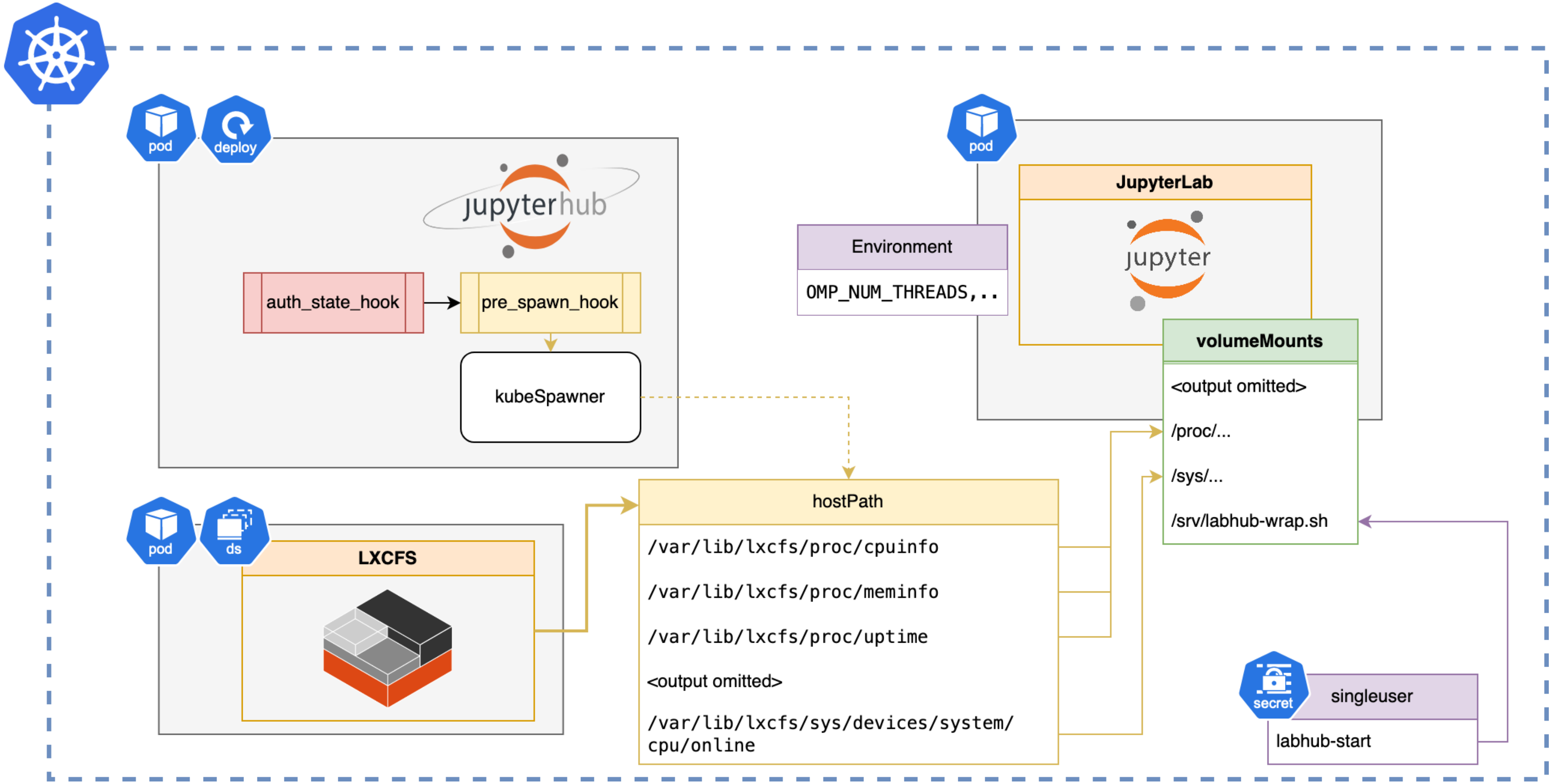
Unprivileged runtime

- UID/GIDs from Token to securityContext
- NSS data sync from LDAP to configMap to mount inside container
- Environment variables to define HOME directory, etc
- Wrapper startup script to bootstrap environment
- Storage mounts are simply defined in the values.



LXCFS: Resources visibility

- [LXCFS](#) is a FUSE filesystem offering overlay files for `cpuinfo`, `meminfo`, `uptime`, etc
- Deployed as DaemonSet on Kubernetes level
- Visible CPU and RAM container limits
- Mounted to `/proc` and `/sys` in `pre_spawn` hook
- Defining additional environment variables in startup scripts



MortalGPU: GPU sharing

Artifact Hub mortalgpu

- Kubernetes **device plugin** for GPU sharing with **memory overcommit**, while maintaining **allocation limit** per GPU workload
  - the approach identical to sharing RAM on Kubernetes
  - represent GPU (or MIG partition) with configurable number of meta-devices (e.g. `320` of `mortalgpu/v100`)

```
deviceSharing:
  - resourceName: mortalgpu/a100-shared
    metagpusPerGpu: 400
    uuid: []
    migid:
     - 2
```

```
Capacity:
  cpu:                         96
  mortalgpu/a100-10g:          200
  mortalgpu/a100-20g:          200
  mortalgpu/a100-shared:       400
  memory:                      1056011536Ki
  pods:                        220
```

Scheduling

```
metadata:
  annotations:
    gpu-mem-limit.mortalgpu/a100-shared: "80"
```

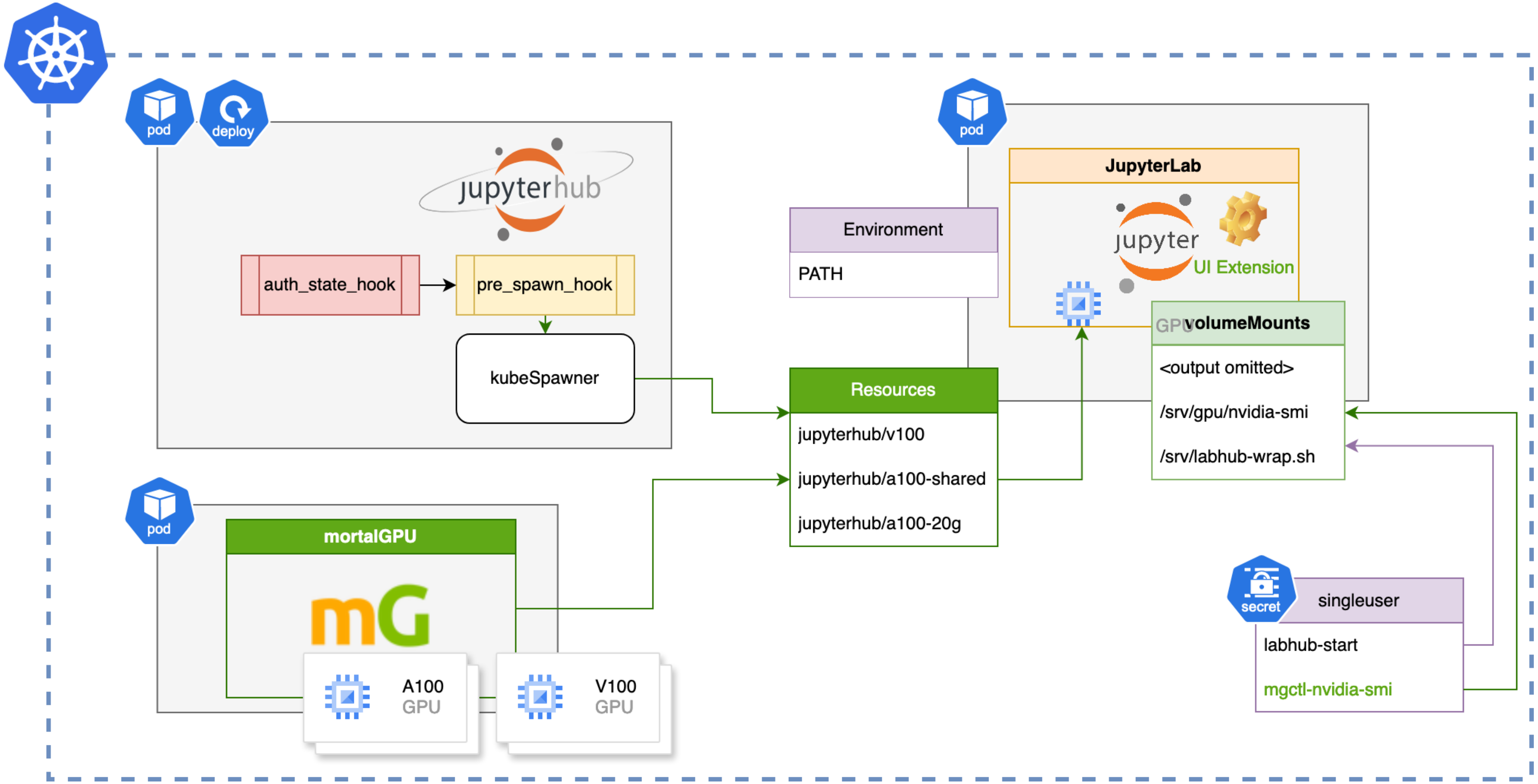
Overcommit Limit

```
resources:
  limits:
    cpu: "8"
    mortalgpu/a100-shared: "40"
    memory: "107374182400"
  requests:
    cpu: 100m
    mortalgpu/a100-shared: "40"
    memory: 524288k
```

Developed at MAX IV based on abandoned **MetaGPU** Device Plugin

JupyterHub with MortalGPU

- Kubernetes DaemonSet
- GPU RAM resource requests and limits, **defined the same way as RAM**
- Multiple MortalGPU resources available (different GPUs and partitions)
- Wrapper over `mgctl` to provide `nvidia-smi` output for container processes only



Artifact Hub mortalgpu

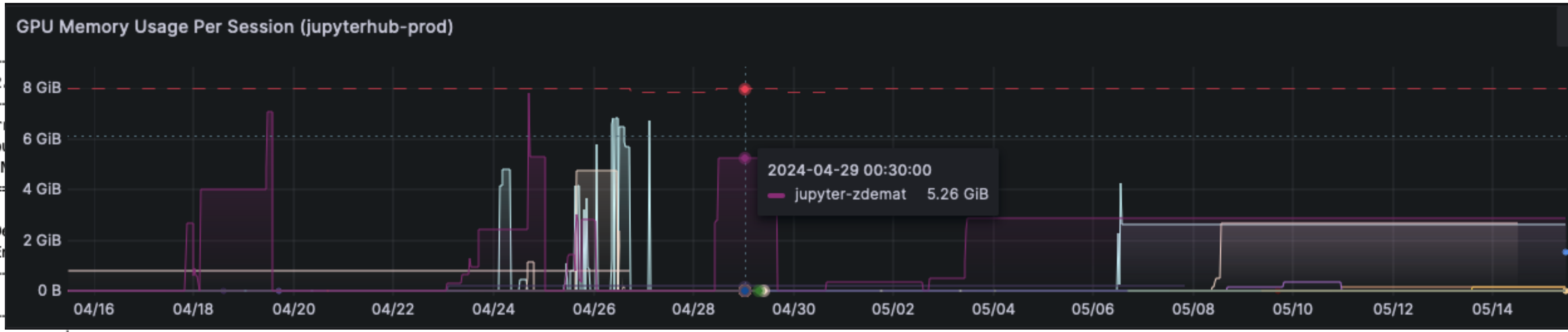
MortalGPU: Observability

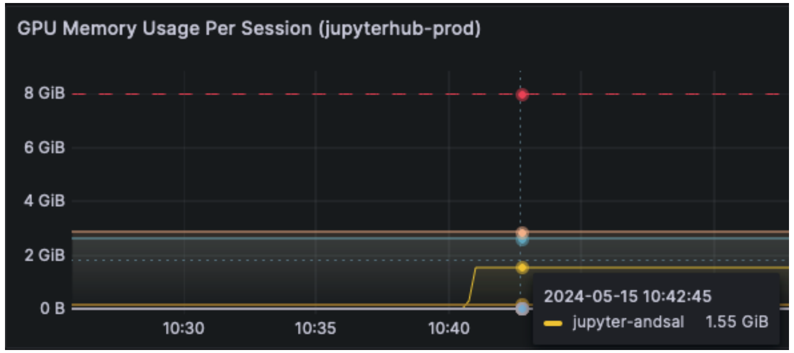
- Kubernetes-awareness
  - MortalGPU monitoring loop
    - mapping of GPU processes to containers running in Kuberentes Pods
    - container-scoped resource usage
  - Memory enforcer
    - OOM-style killing the processes based on container-scoped GPU RAM usage metrics
- Observability tools
  - MortalGPU gRPC interface to monitoring data
    - Container can read own data
  - `mgctl` tool mounted inside containers to access GPU allocation metrics
  - Prometheus exporter
    - GPU Device level metrics
    - Process level metrics (with Kubernetes metadata: Pod, Container)

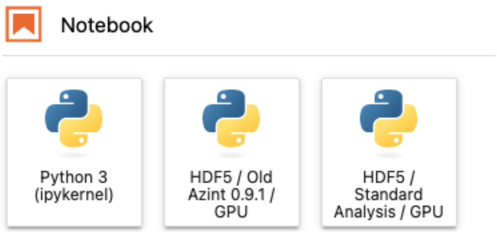
JupyterHub GPU Observability

```
andsal@jupyter-andsal:~/jupyter_notebooks$ nvidia-smi
Wed May 15 10:41:05 2024
+-----------------------------------------------------------------------------------------
| NVIDIA-SMI 535.154.05             Driver Version: 535.154.05   CUDA Version: 12
|-----------------------------------------+----------------------+-----------------------
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncor
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compu
|                                         |                      |               M
|=========================================+======================+=======================
|   0  NVIDIA A100 80GB PCIe          Off | 00000000:CA:00.0 Off |
| N/A   31C    P0              62W / 300W |                  N/A |      N/A      De
|                                         |                      |               En
+-----------------------------------------+----------------------+-----------------------

+-----------------------------------------------------------------------------------------
| MIG devices:                                                                          |
+------------------+--------------------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |                   Memory-Usage |        Vol|         Shared        |
|      ID  ID  Dev |                     BAR1-Usage | SM     Unc| CE ENC DEC OFA JPG    |
|                  |                                |        ECC|                       |
|==================+================================+===========+=======================|
|  0    2   0   0  |            4495MiB / 40192MiB  | 42      0 |  3   0    2    0    0 |
|                  |              6MiB / 65535MiB   |           |                       |
+------------------+--------------------------------+-----------+-----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|    0    2    0    2691248      C   ...maxiv-jup-conda-env-hdf5/bin/python      1.5GiB |
+---------------------------------------------------------------------------------------+

andsal@jupyter-andsal:~/jupyter_notebooks$ mgctl get processes
POD            NS              DEVICE  NODE                 GPU  MEMORY                                PID      CMD                                                 REQ/LIMITS
jupyter-andsal jupyterhub-prod 0       w-jupyterhub-a100-wn0 0%   1663041536/421443664/8428873280      2691248  /opt/conda/envs/maxiv-jup-conda-env-hdf5/bin/python 4/80
1                                                                 1663041536MB                                                                                   800/4/80
andsal@jupyter-andsal:~/jupyter_notebooks$ mgctl get processes -o json
```

[{"container_id":"f1aac41081c10e4d3ecd26ae7133d527fde2e53aae8f2cc65f1bb7b8936cdb9b","container_name":"notebook","pod_id":"jupyter-andsal","pod_namespace":"jupyterhub-prod","meta
gpu_requests":4,"metagpu_limits":80,"resource_name":"jupyterhub/a100-shared","node_name":"w-jupyterhub-a100-wn0","container_devices":[{"device":{"uuid":"MIG-72bdef0f-5d81-52f6-b
e55-d662a1eadbad","devuuid":"GPU-c4dafcee-8f9e-b2fd-3eba-14a9ed7a0c29","migid":2,"allocated_shares":152,"shares":400,"memory_total":42144366592,"memory_free":37430820864,"memory
_used":4713545728,"memory_share_size":105360916,"resource_name":"jupyterhub/a100-shared","node_name":"w-jupyterhub-a100-wn0"},"allocated_shares":4}],"device_processes":[{"mguuid
":"MIG-72bdef0f-5d81-52f6-be55-d662a1eadbad","devuuid":"GPU-c4dafcee-8f9e-b2fd-3eba-14a9ed7a0c29","pid":2691248,"memory":1663041536,"cmdline":"/opt/conda/envs/maxiv-jup-conda-en
v-hdf5/bin/python","container_id":"f1aac41081c10e4d3ecd26ae7133d527fde2e53aae8f2cc65f1bb7b8936cdb9b","gpu_instance_id":2,"relative_gpu_usage_percent":0,"compute_instance_id":0,"
mig_device":-1}]}]

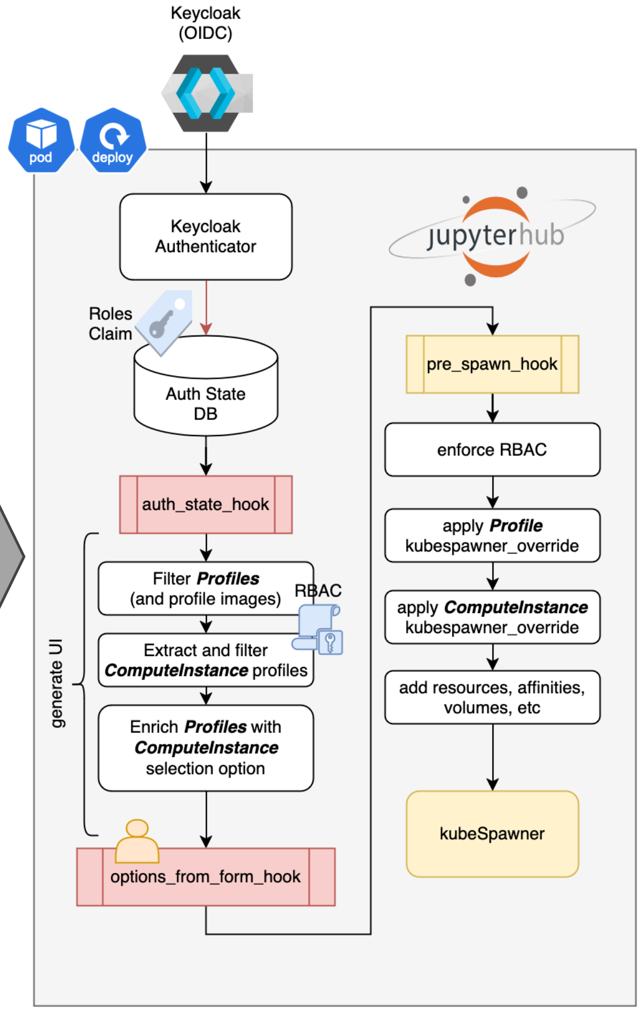


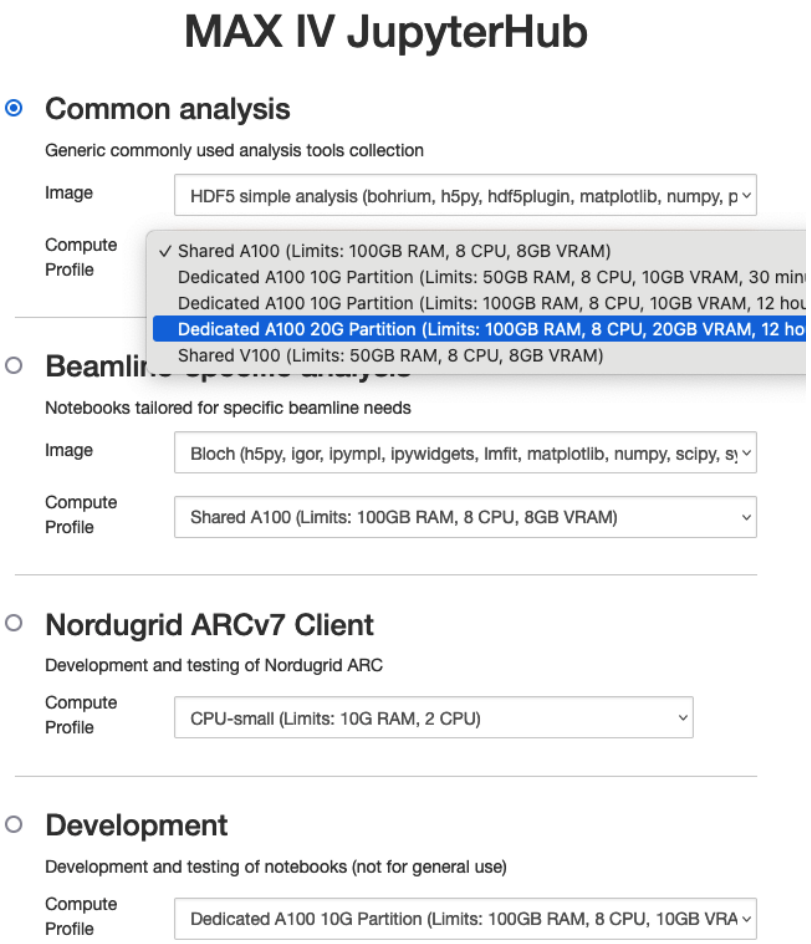




Compute Instance profiles and RBAC

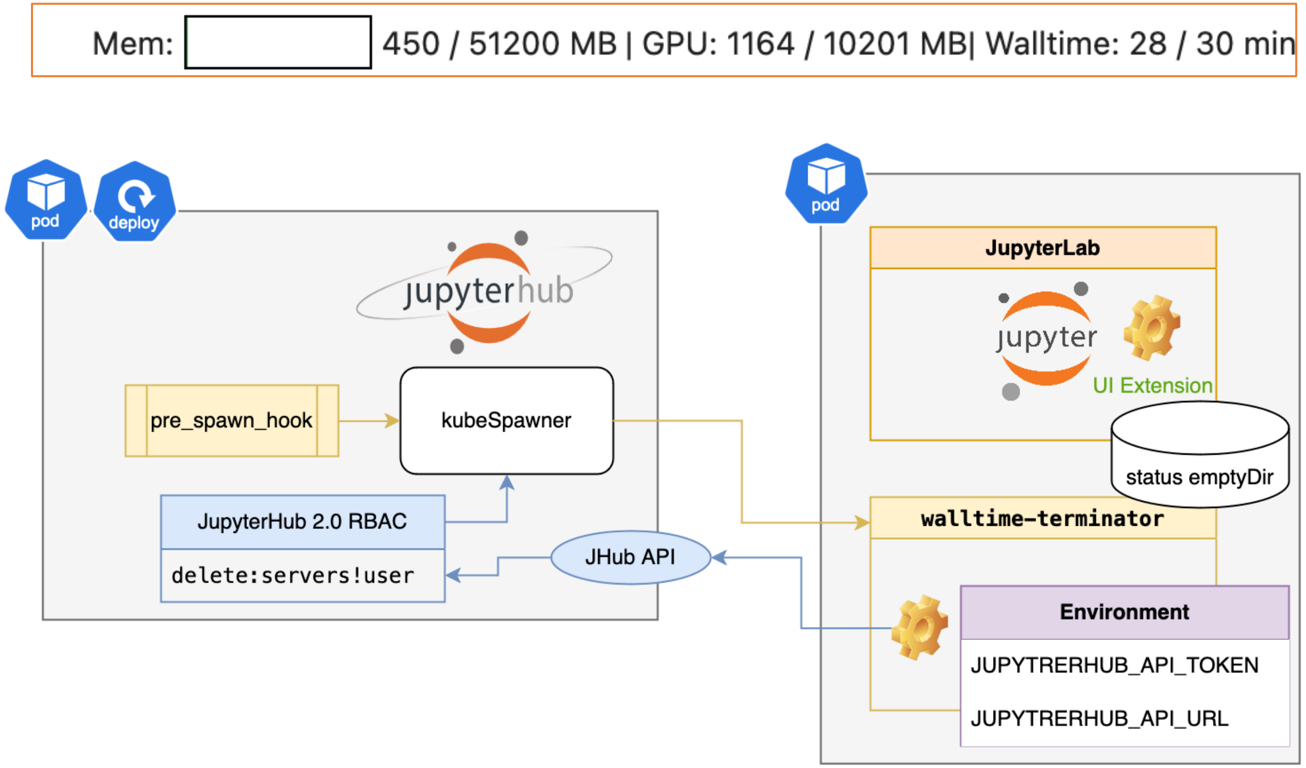
```yaml
profileList:
  # Compute instance resource profiles
  - compute_instances:
      a100_shared:
        display_name: "Shared A100 (Limits: 100GB RAM, 8 CPU, 8GB
        cpu_guarantee: 0.1
        mem_guarantee: 500M
        cpu_limit: 8
        mem_limit: 100G
        gpu_resource_name: jupyterhub/a100-shared
        gpu_guarantee: 4
        gpu_limit: 80
        node_type: a100
      a100_20g:
        display_name: "Dedicated A100 20G Partition (Limits: 100G
        required_role: a100-20g
      # ...
  - display_name: "Beamline-specific analysis"
    description: "Notebooks tailored for specific beamline needs"
    slug: "beamline"
    allowed_compute_instances:
      - a100_shared
      - a100_10g_quick
      - a100_10g
      - a100_20g
      - v100
    profile_options:
      image:
        display_name: Image
        choices:
          bloch:
            display_name: "Bloch (h5py, igor, ipympl, ipywidgets,
            kubespawner_override:
              image: harbor.maxiv.lu.se/jupyterhub/bloch-notebook
```





Extra containers = extra features

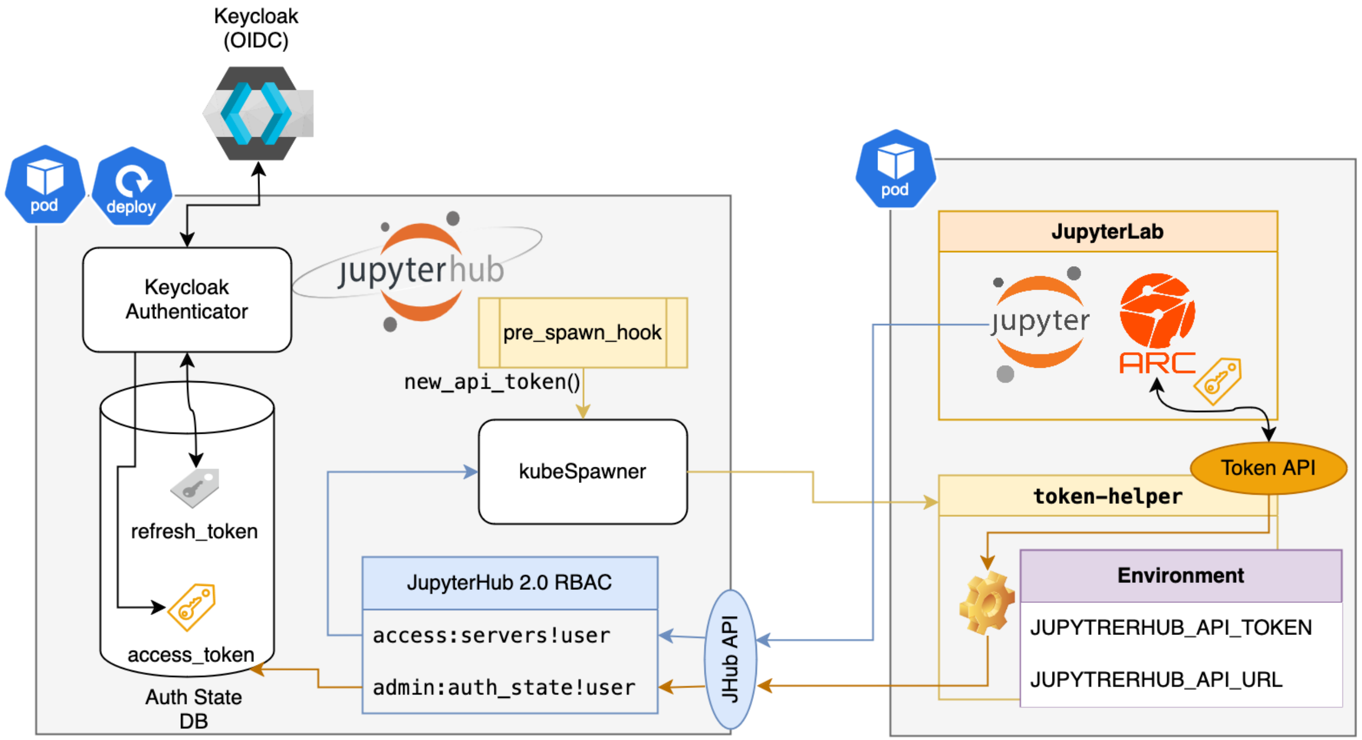
Walltime enforcement



- KubeSpawner is capable of running additional containers in the user Pod
- Isolated walltime countdown container terminating user server via JupyterHub API
  - Using the JupyterHub RBAC feature
- Developed UI extension to show values to end-user

Extra containers = extra features

”OIDC-agent” for Nordugrid ARC



- KeyCloak Authenticator to refresh access tokens
- Isolated token-helper container with privileges to read `auth_state`
  - Using the Jupyterhub RBAC feature
- API to provide only Access Tokens to JupyterLab container
  - wrapper to use in ARC CLI transparently

Conclusions

- Extensibility of both JupyterHub and Kubernetes *allows* to build data analysis platforms matching the organization needs in functionality and security.
- LXCFS on the Kubernetes *brings* allocated resources visibility to both interactive and batch workloads.
- MortalGPU flexible and observable GPUs sharing *enriches* the interactive shared environments with CUDA availability.
- Compute Instance profiles and RBAC *extends* the usage patterns of the shared platform, improving the end-user experience.
- Additional containers in the user server Pod *open a way* to securely add features beyond the usual JupyterHub capabilities.

Thank you for attention!

Source code and
deployment configuration
can be found on
gitlab.com

Mail To: andrii.salnikov@maxiv.lu.se