CS3 2024, CERN

WARNING! KUBERNETES TERMINOLOGY

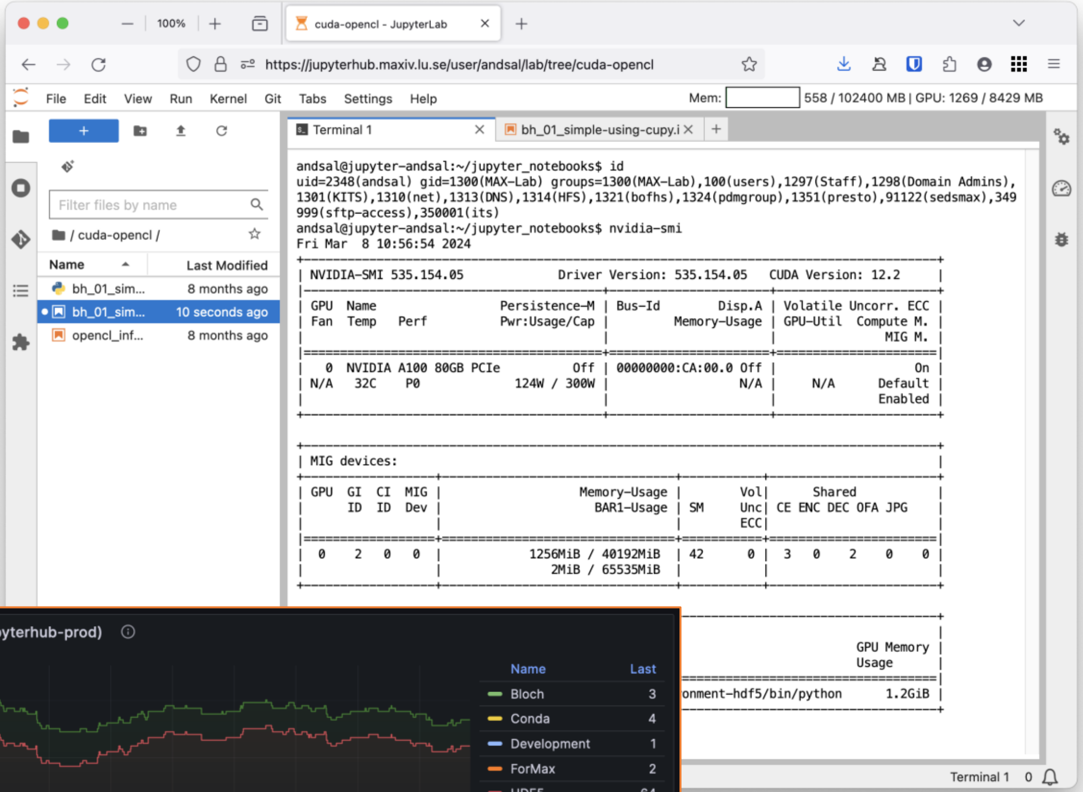
JupyterHub on Kubernetes as a platform for developing secure shared environment for data analysis at MAX IV

Andrii Salnikov,
Zdenek Matej, Dmitrii Ermakov, Jason Brudvik

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

Interactive data analysis environment

- Container images with pre-defined and custom kernels
- 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
  - as a **runtime environment**
- Shared service for staff and researchers
  - Remote-desktop style experience
  - Resources overcommit

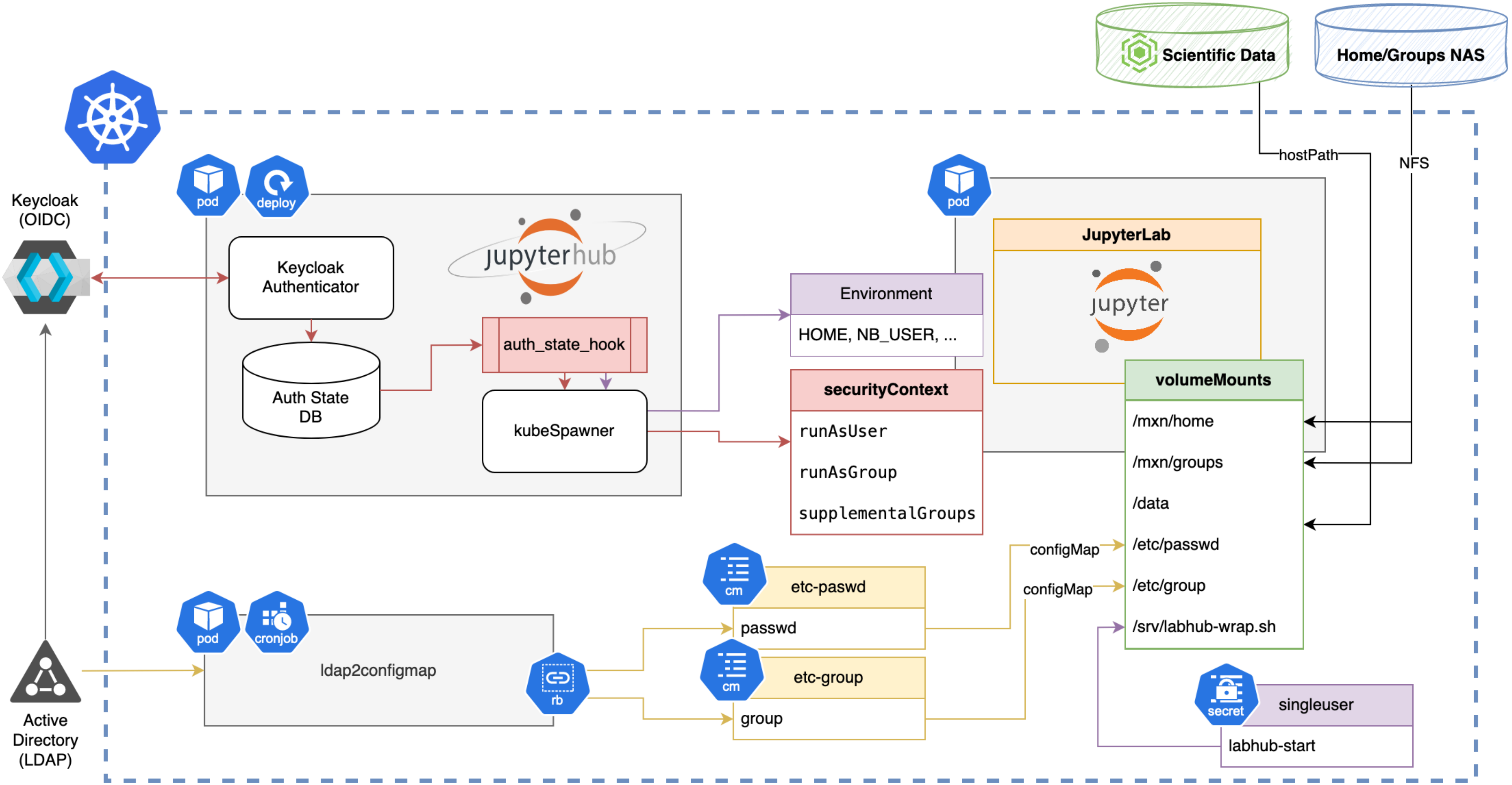


Goals and technical requirements

- **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`

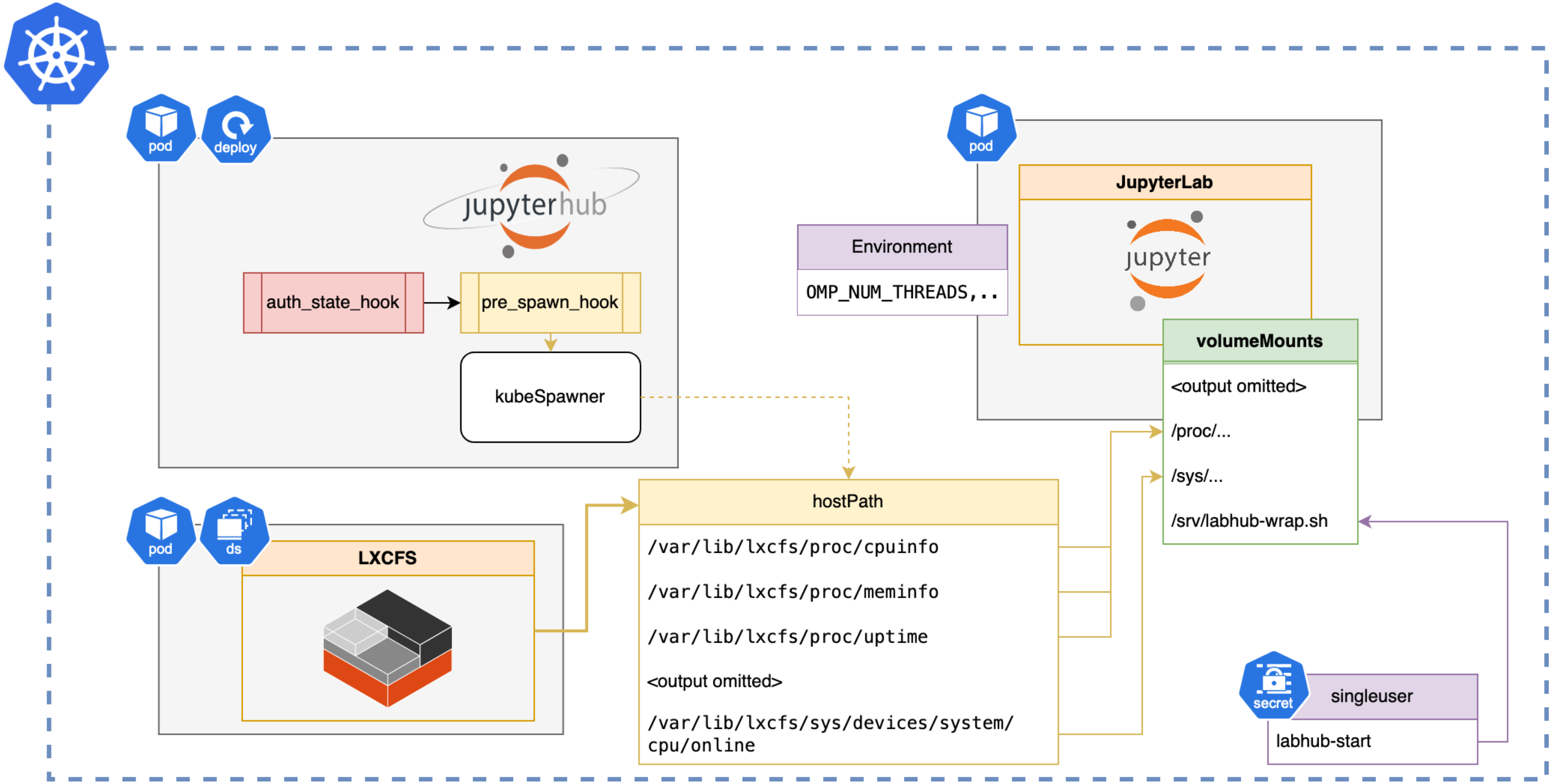
Existing LDAP credentials

- 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



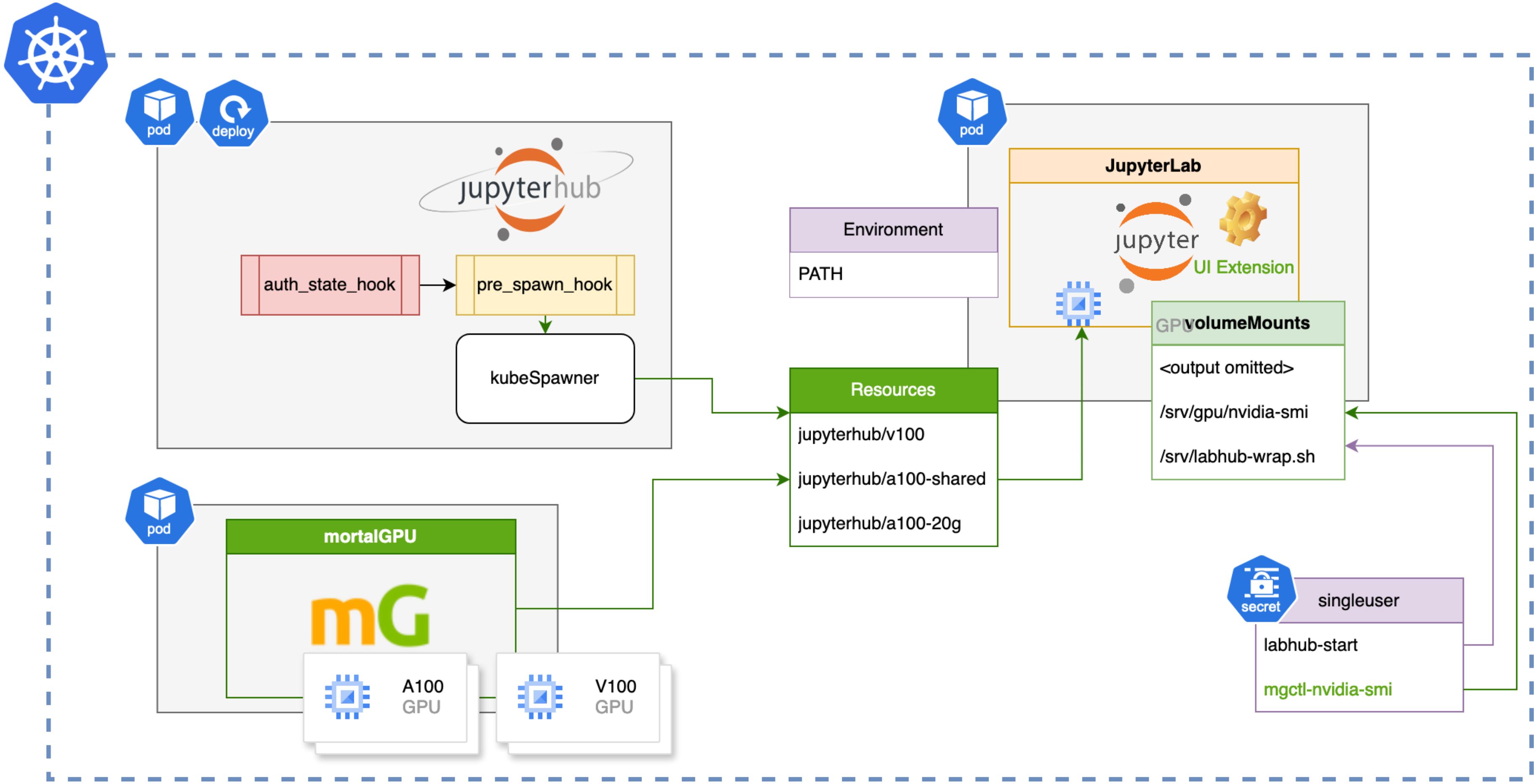
GPU sharing: MortalGPU development

- Kubernetes **device plugin** for GPU **memory overcommit**, while maintaining allocation **l**imit per GPU workload - the approach used for sharing RAM on Kubernetes.
- Fork of MetaGPU with development focus on **interactive workloads** run by mortals (with operations support by mortal admins)
- Provides:
  - Device Plugin: represent GPU (or MIG partition) with configurable number of meta-devices (e.g. `320` of `mortalgpu/v100`)
  - Memory enforcement based on the usage monitoring data
  - Kubernetes-aware observability in general and container-scoped resource usage in particular:
    - `mgctl` tool and Prometheus exporter

Artifact Hub | mortalgpu

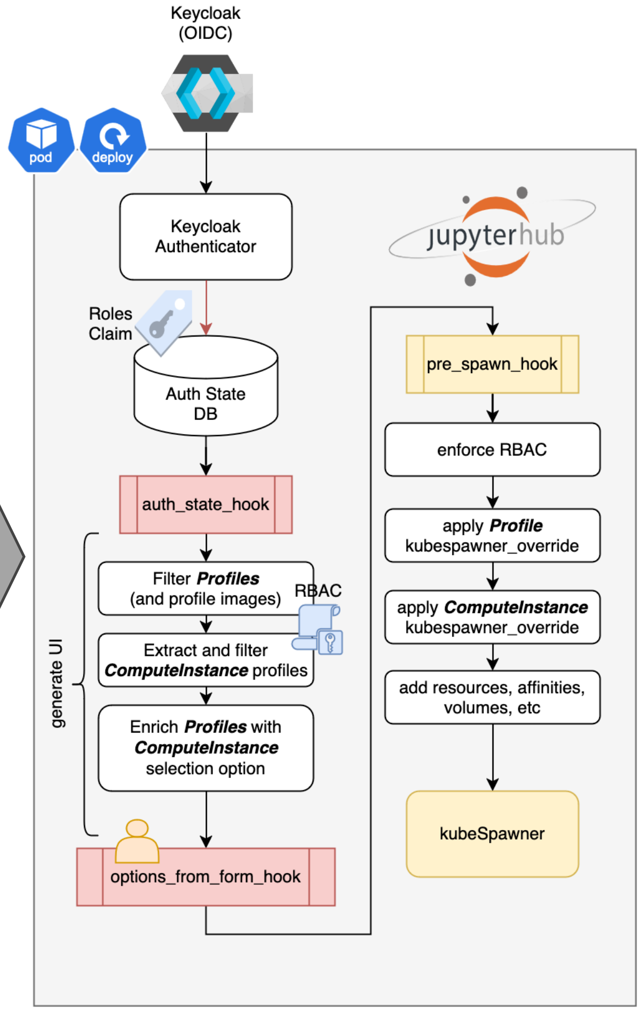
Jupyterhub with MortalGPU

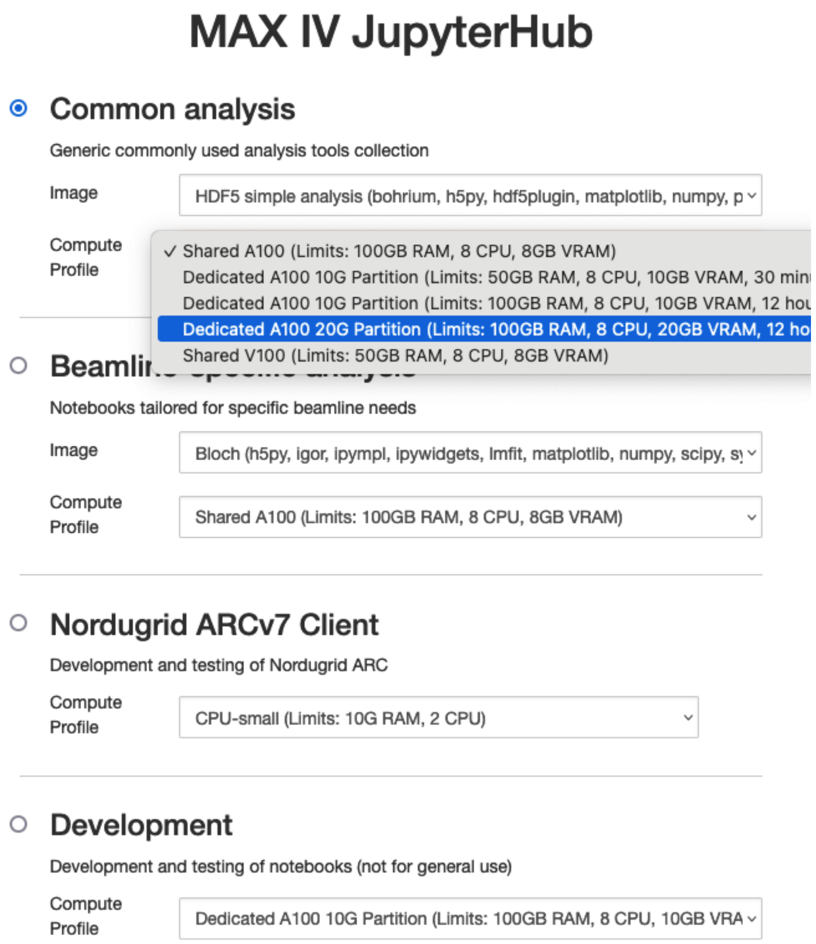
- Kubernetes DaemonSet
- GPU RAM resource requests and limits, **defined the same way as RAM**
- Multiple MortalGPU resources available (different GPUs and partitions)
- Wrapper over `mgctl` to provide `nvidia-smi` output for container processes only



Compute Instance profiles and RBAC

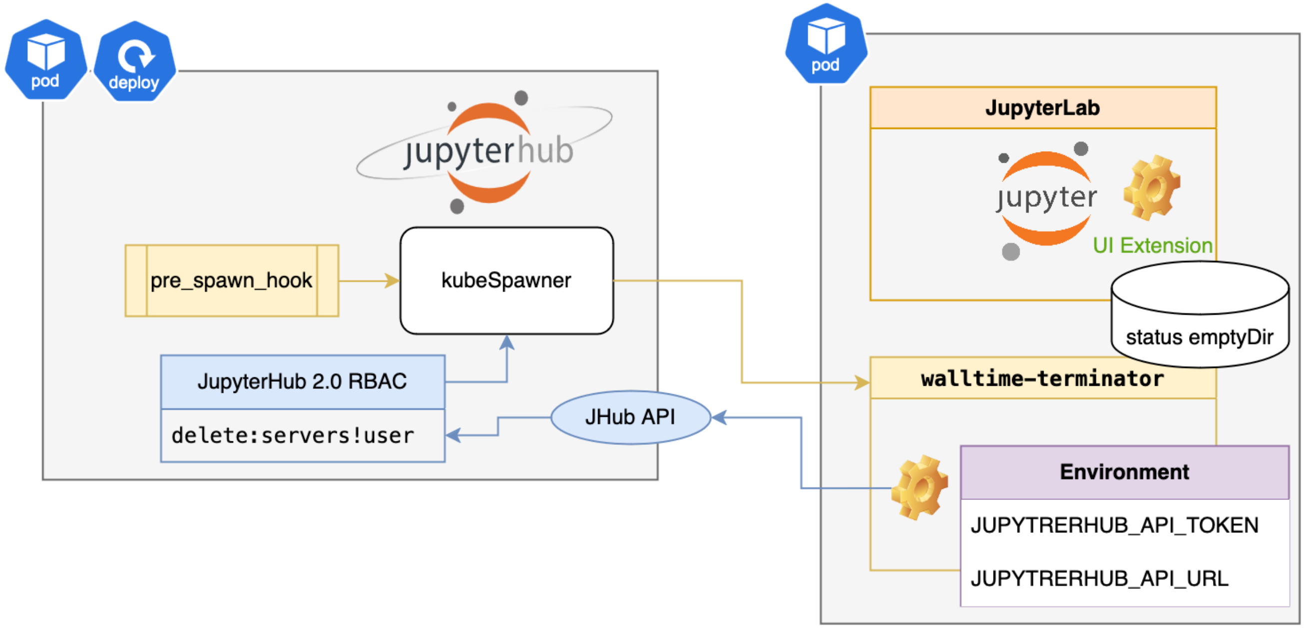
```yaml
profileList:
  # Compute instance resource profiles
  - compute_instances:
      a100_shared:
        display_name: "Shared A100 (Limits: 100GB RAM, 8 CPU, 8GB
        cpu_guarantee: 0.1
        mem_guarantee: 500M
        cpu_limit: 8
        mem_limit: 100G
        gpu_resource_name: jupyterhub/a100-shared
        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

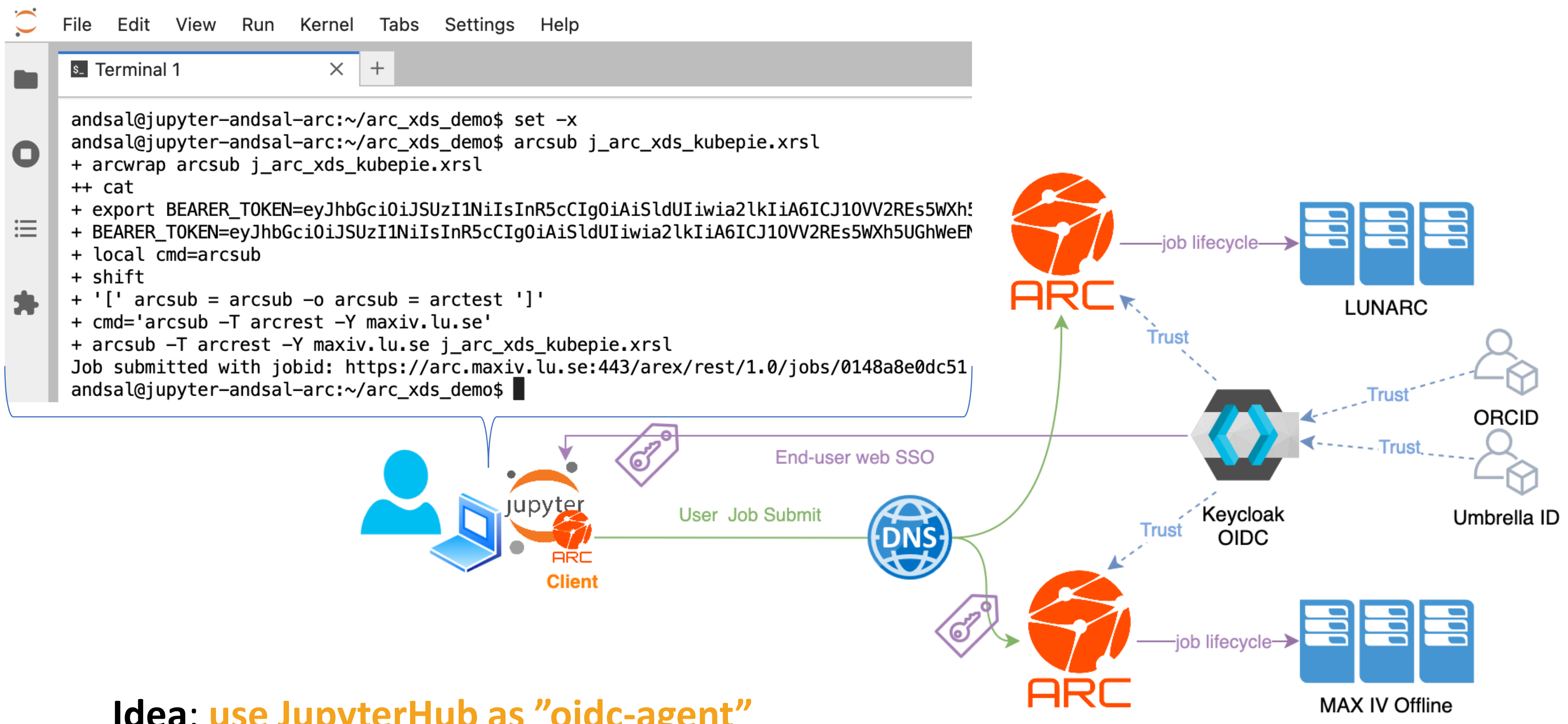
Walltime enforcement



- 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

Use-case: Nordugrid ARC Client

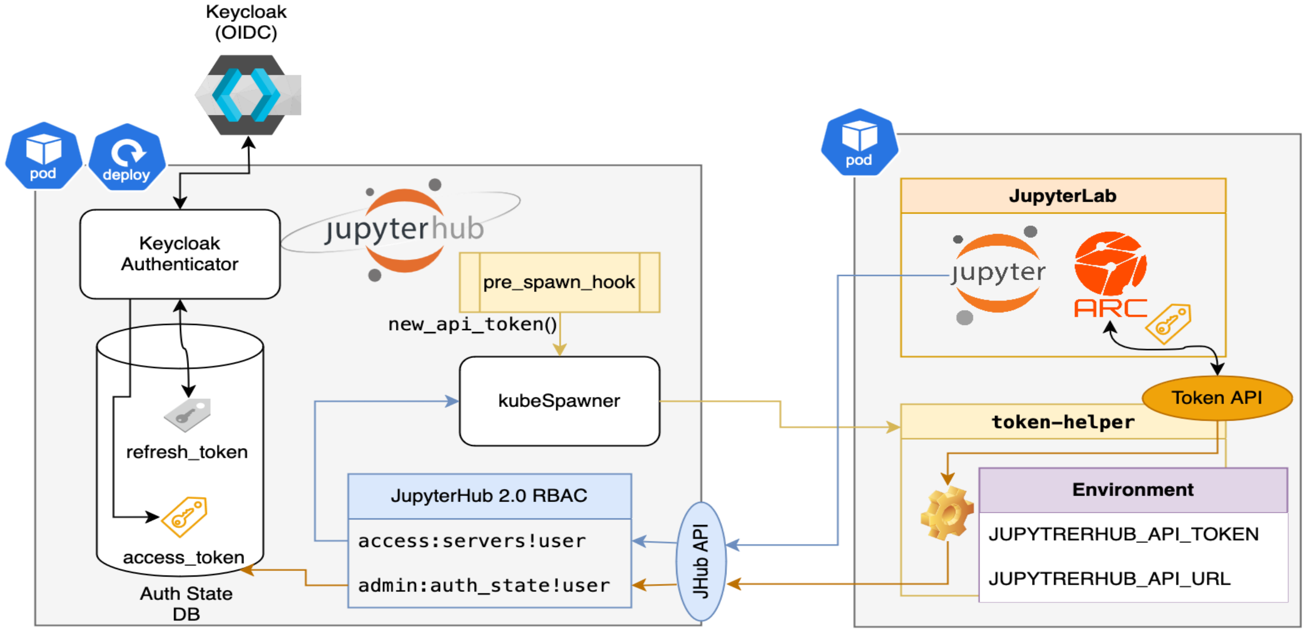
- PoC: Small grid for transparent HPC usage
- ARC with OAuth2 JWT tokens auth:
  - Map to self at MAX IV resources
  - Map to pool on external resources
- Additional challenge: existing data sharing to external sites with JWT auth, following user permissions



Idea: use JupyterHub as "oidc-agent"

Extra containers = extra features

"OIDC-agent" for 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

KubePie: sharing existing data over https

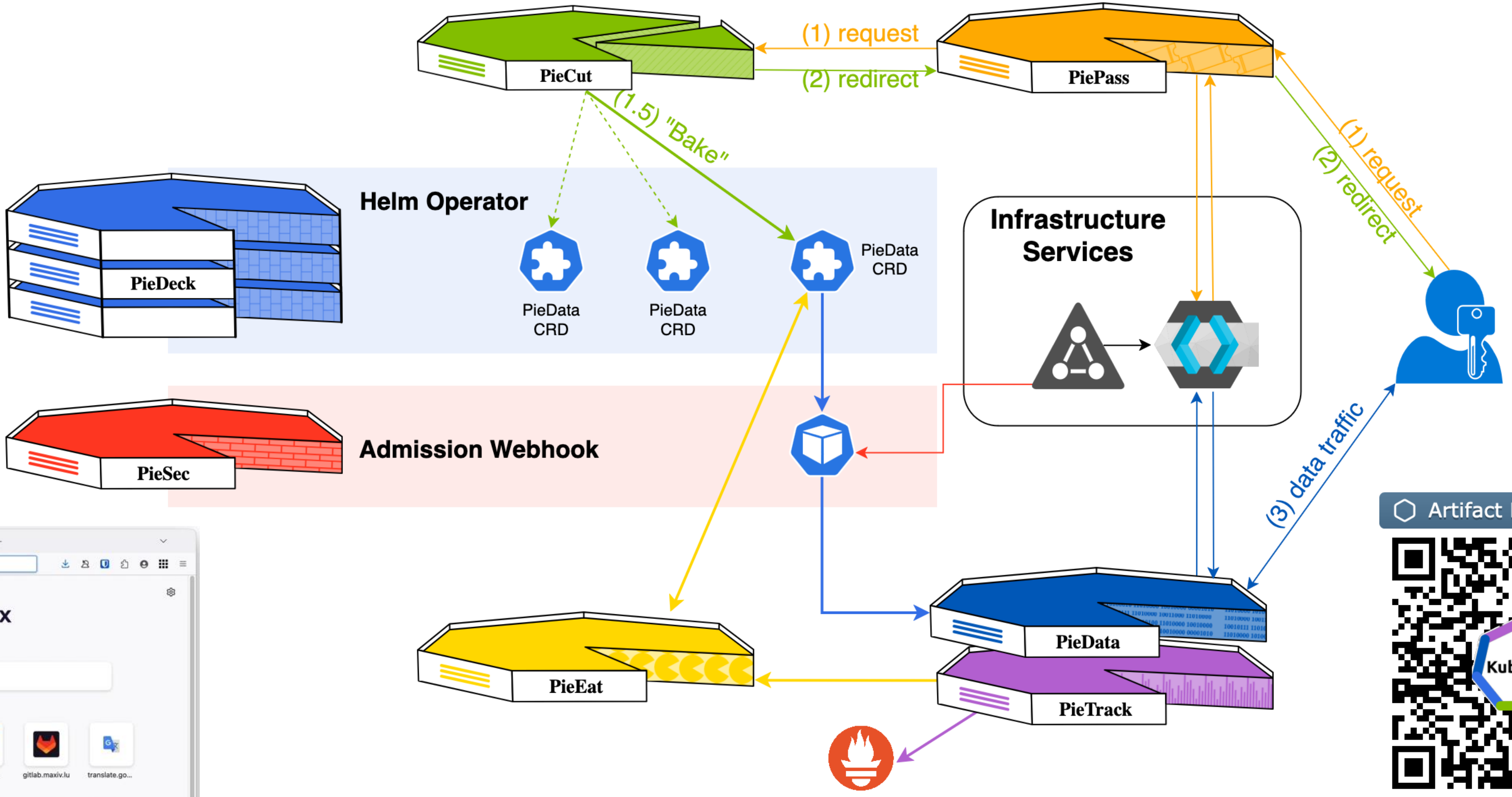
- **Idea**: *”own”* web server for each user with the correct UID/GIDs
  - *Sounds crazy? But we do run such Pods for each user in JupyterHub!*

CHALLENGE ACCEPTED

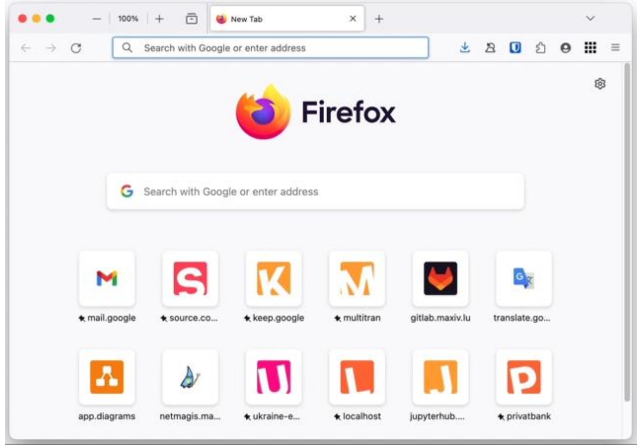
- **KubePie** is harnessing Kubernetes' scalability and deployment capabilities by running, managing and securing web servers for every user
- **KubePie** is strictly relying on *OpenID Connect* flow or *OAuth2 bearer tokens* when it comes to the user identification
  - OAuth2 used in ARC PoC for data transfers
  - Claims-based user-mapping during Pod instantiation (admission)
  - Other auth credentials accessible via OIDC:
    - WebDAV with S3-like credentials is implemented as an example

Artifact Hub kubepie

KubePie: Baking process



KubePie@MAX IV is running on the Data Acquisition Kubernetes Cluster



Artifact Hub kubepie

KubePie

Conclusions

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

Thank you for attention!

Ask me about:

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

EUROPEAN OPEN SCIENCE CLOUD

We are working towards establishing similar deployment for providing EOSC service as Open Data analysis platform

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