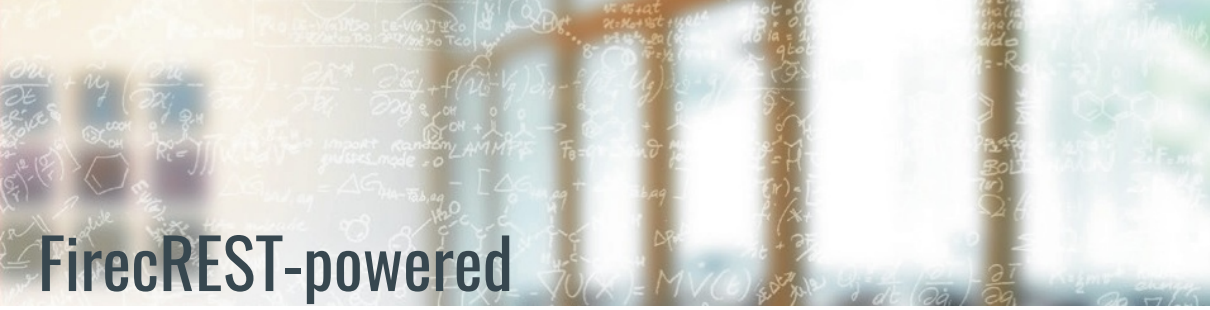


FirecREST-powered JupyterHub for HPC environments

JupyterHub with FirecRESTSpawner at CSCS

Rafael Sarmiento & Juan Dorsch
ETH Zürich / CSCS
SKACH Winter Meeting 2026

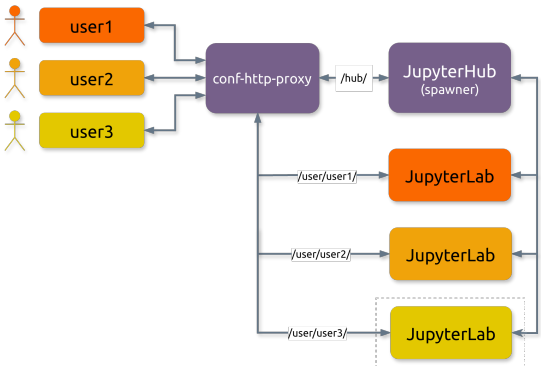
ETH zürich
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Outline

- JupyterHub architecture
- Previous deployment with BatchSpawner at CSCS
- FirecRESTSpawner
- The `fc4jhub` Helm chart
- Automatic testing and security
- Collaboration between teams

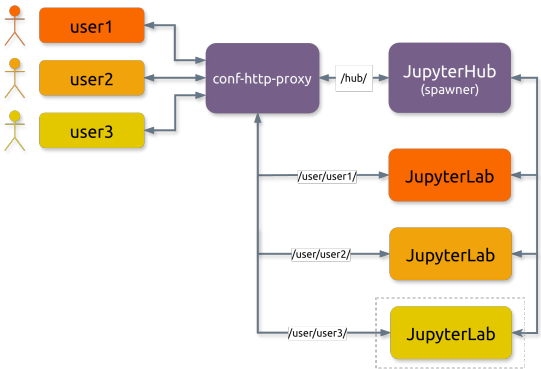
- JupyterHub is a multi-user server for Jupyter notebooks, designed for shared computing environments
- It allows multiple users to access Jupyter notebooks through a central web interface
- Administrators can manage authentication, resource allocation, and user environments
- It supports deployment on local servers, cloud platforms, or Kubernetes clusters
- Ideal for educational institutions, research groups, and data science teams

JupyterHub's Architecture



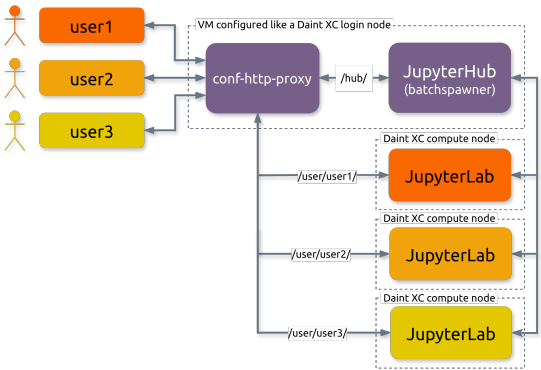
- The proxy routes all incoming traffic to the correct component (hub or user servers)
- The hub is the central controller that manages authentication, user sessions, and permissions. Launches and monitors user notebook servers through a spawner. Maintains overall state of the system (who is logged in, which servers are running, ...)
- The single-user notebook servers are dedicated JupyterLab instances for each authenticated user. They run with the user's identity and environment

JupyterHub Spawner



- The spawner launches and manages single-user JupyterLab instances
- Abstract interface that supports different backends (LocalProcessSpawner, BatchSpawner, KubeSpawner, ...)
- Has three main functions: start, stop and poll
- Reports server state (running, stopped, failed) back to the Hub
- Handles isolation by running each user's server in its own process, container, or pod depending on the spawner type

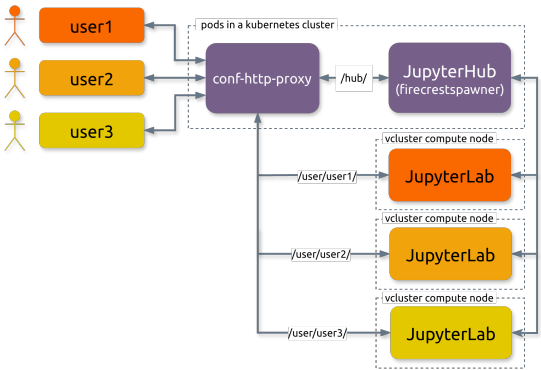
JHub + Workload Manager with BatchSpawner



- Integrates JupyterHub with HPC batch schedulers (Slurm, PBS, Torque, ...)
- Submits notebook servers as batch jobs
- The spawner monitors job status, waiting for the scheduler to start the job and provides a node and port before proxying the notebook
- Supports customizable job scripts
- Deployed on a VM that's setup as a login node where JupyterHub must run as root

https://github.com/jupyterhub/batchspawner

JHub + Workload Manager with FirecRESTSpawner



- Uses the FirecREST REST API to interact with HPC resources
- Provides a cloud-like interface to HPC systems
- Manages the notebook's life cycle on compute resources through requests to different FirecREST endpoints
- Improves security by eliminating the need for JupyterHub to manage user SSH keys or direct scheduler credentials.
- Supports custom job parameters
- Enables quicker deployments since no VM login nodes are required to run JupyterHub

https://github.com/eth-cscs/firecrestspawner

The `fc4jhub` Helm chart

- A Helm chart to deploy JupyterHub with FirecRESTSpawner on Kubernetes
- Hosted at https://eth-cscs.github.io/firecrestspawner
- Written with Alps in mind but general enough to be deployed in other sites
- Support for Vault
- Uses the Reloader controller to ensure that changes in Vault or in JupyterHub's configuration trigger a restart of the hub's pod
- Annotations to enable automatic scraping by Metricbeat
- Supports custom job parameters via a Slurm job template
- Service account for polling to avoid issues with expired KeyCloak SSO sessions
- Allows restarting the hub without affecting users running JupyterLab servers

https://github.com/eth-cscs/firecrestspawner/tree/main/chart

Deployment

Kubernetes cluster

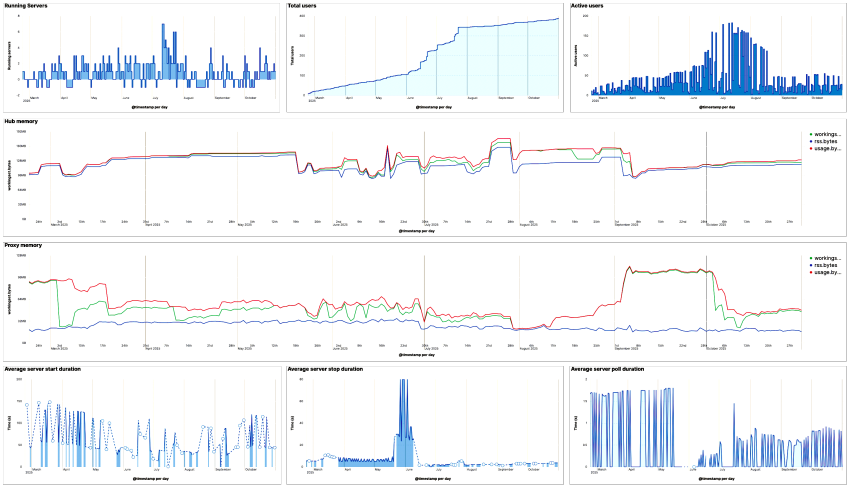
- Create secret that will be used to access Vault (optional)
- Get KeyCloak client from the Dev portal
- The creation of the ArgoCD repo happens automatically
- Deploy the `fc4jhub` chart with `values.yaml` configured for the particular vcluster
- Set up egress

vcluster

- Create and deploy uenvs with JupyterLab and a programming environment
- Create and deploy container image

- Allow connections from the vcluster's compute nodes to JupyterHub running on Kubernetes
- Request a url for the service direct at the cscs.ch domain

Monitoring JupyterHub's metrics



Automatic testing with JupyterHub's REST API

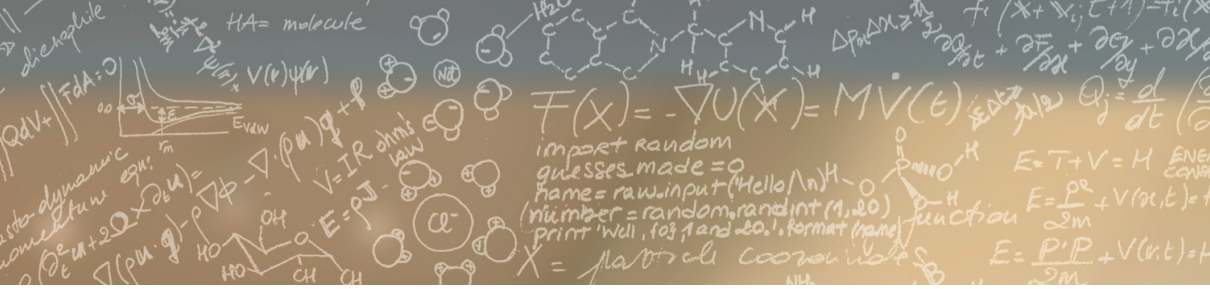

```python
def start_server(username, options):
    """Launch JupyterLab with the given custom options"""
    response = requests.post(
      f"{JUPYTERHUB_URL}/hub/api/users/{username}/server",
      headers=HEADERS,
      json=options)
    ...

def check_server_status(username, timeout=60):
    """Wait until the server is up or times out"""
    response = requests.get(
      f"{JUPYTERHUB_URL}/hub/api/users/{username}",
      headers=HEADERS)
    ...

def stop_server(username):
    """Stop the JupyterHub server for the given user"""
    response = requests.delete(
      f"{JUPYTERHUB_URL}/hub/api/users/{username}/server",
      headers=HEADERS)
    ...
```

Ongoing work on security

- Unit tests to ensure core functionality behaves as expected (Reused from PyFirecREST's)
- Multi-stage Dockerfile to reduce the attack surface and image size by separating build and runtime environments
- Static code analysis with **CodeQL** to detect vulnerabilities in the codebase
- Automatically track and update vulnerable or outdated dependencies with **Dependabot**
- Evaluation of Kubernetes manifests for best-practice security and configuration issues with **kube-score**
- Scanning container images (`ghcr.io/eth-cscs/f7t4jhub`) with **Trivy** for known vulnerabilities and misconfigurations
- Static analysis of Python source code with **Bandit** for common security issues



Thank you for your attention!