

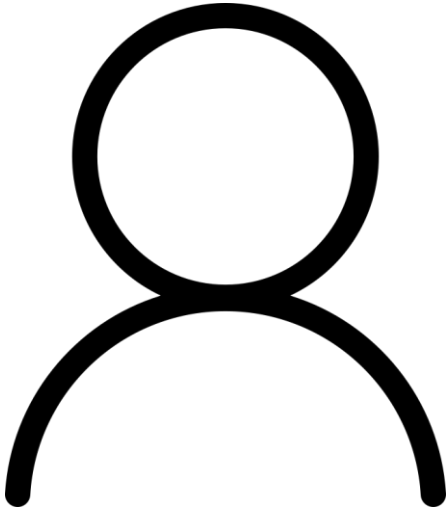


JUPYTERHUB OUTPOST

START SERVICES ON MULTIPLE REMOTE RESOURCES

MAY 15, 2024 | TIM KREUZER

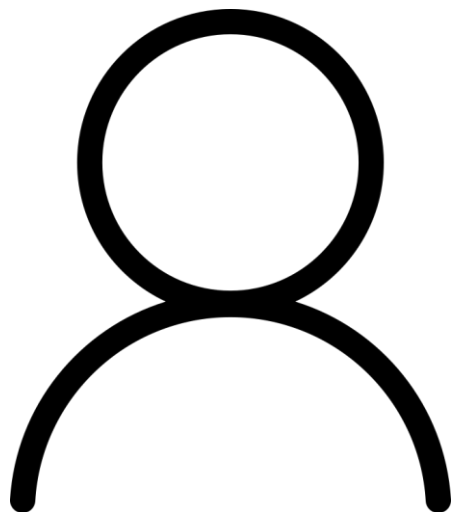
MOTIVATION



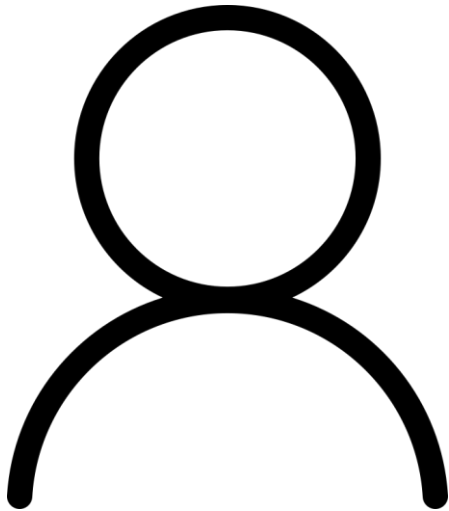
- 3.400+ accounts at HPC-systems in Juelich
- Each user is part of various HPC-projects
- Each HPC-project has access to different partitions on different systems

- Users are experts in their field, but not everyone is an expert in computer science
- Easy, unified access to all systems

MOTIVATION



MOTIVATION

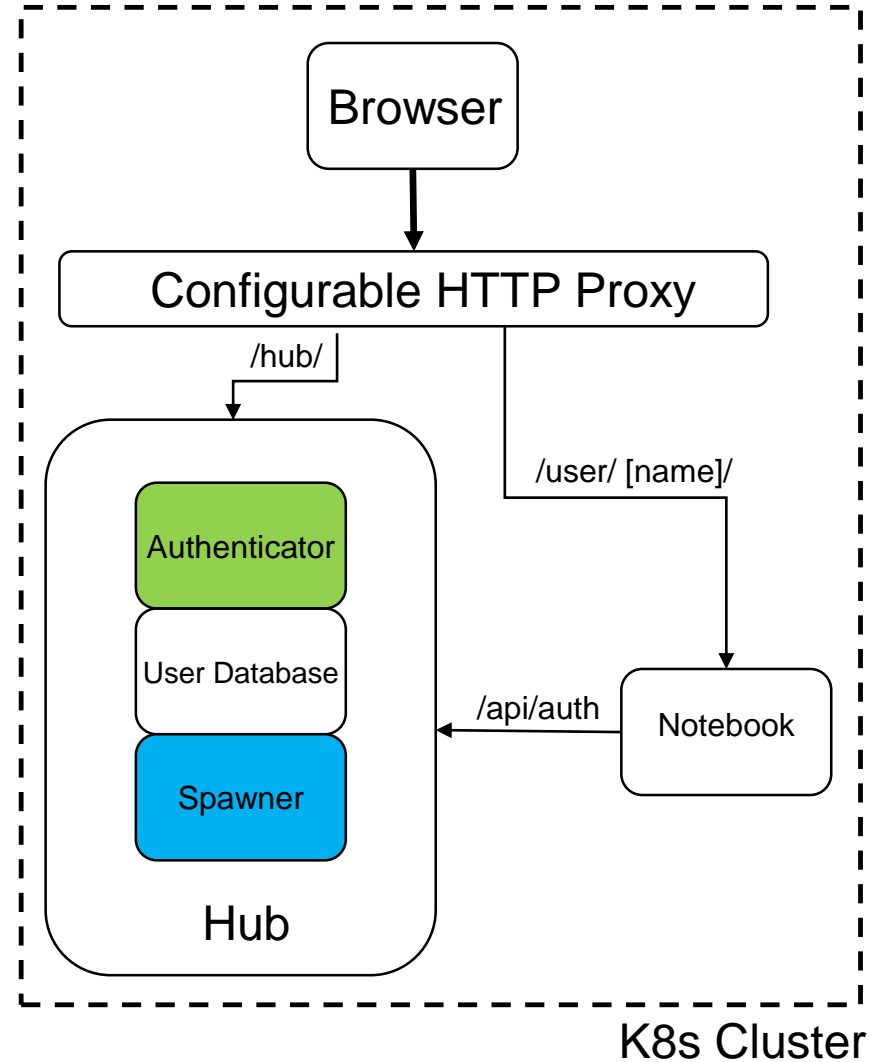


REQUIREMENTS

- One central JupyterHub should be able to start a notebook server on each system
- Add more systems in the future with different resource types (k8s, slurm, torque, etc.)
- Each system has to keep full control over its resources
- Compatible with vanilla JupyterHub – no patches or changes required

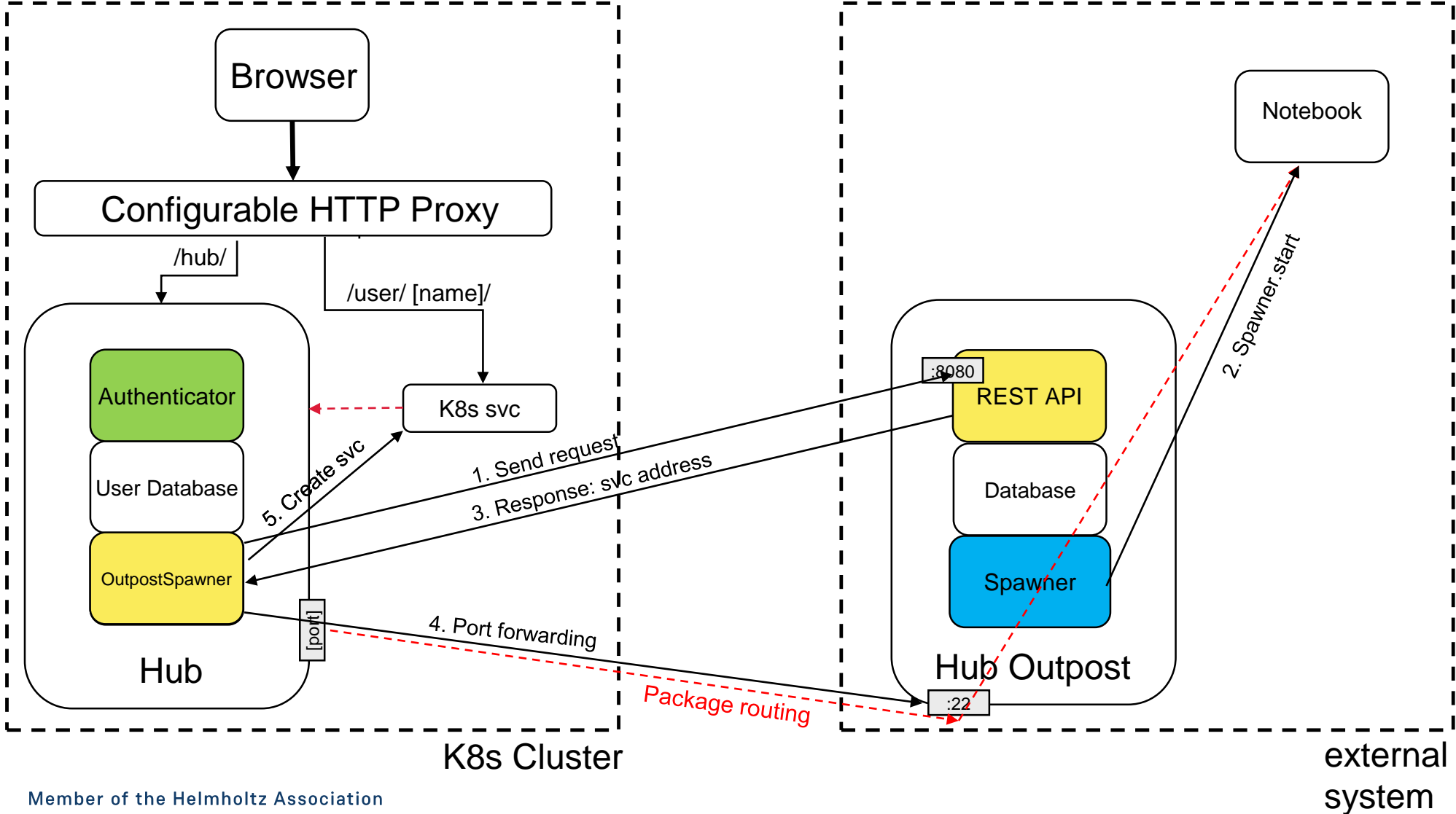
ARCHITECTURE

JupyterHub



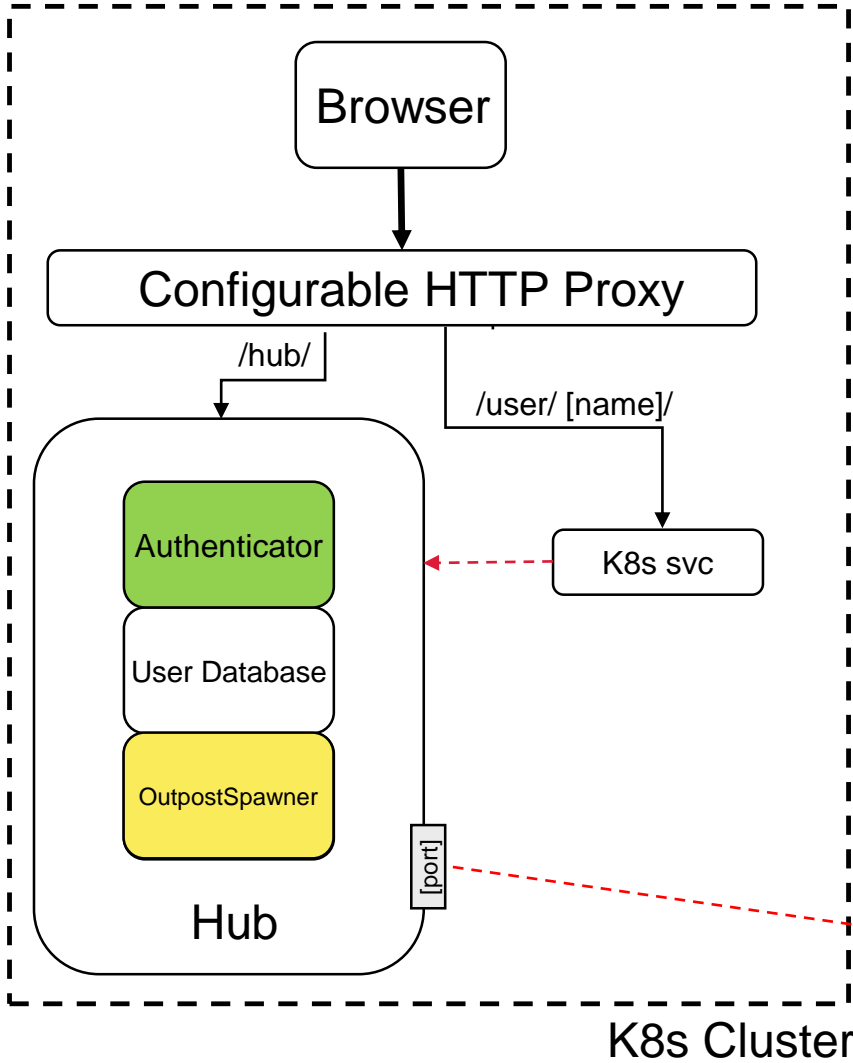
ARCHITECTURE

JupyterHub

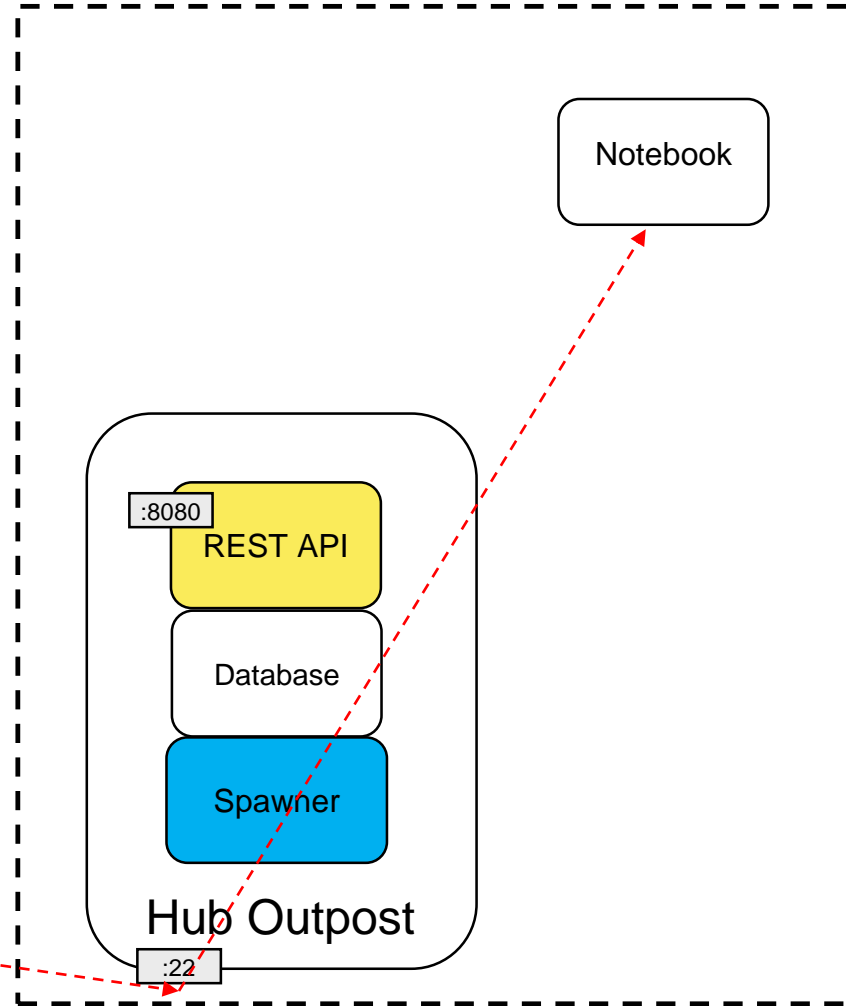


ARCHITECTURE

JupyterHub



K8s Cluster

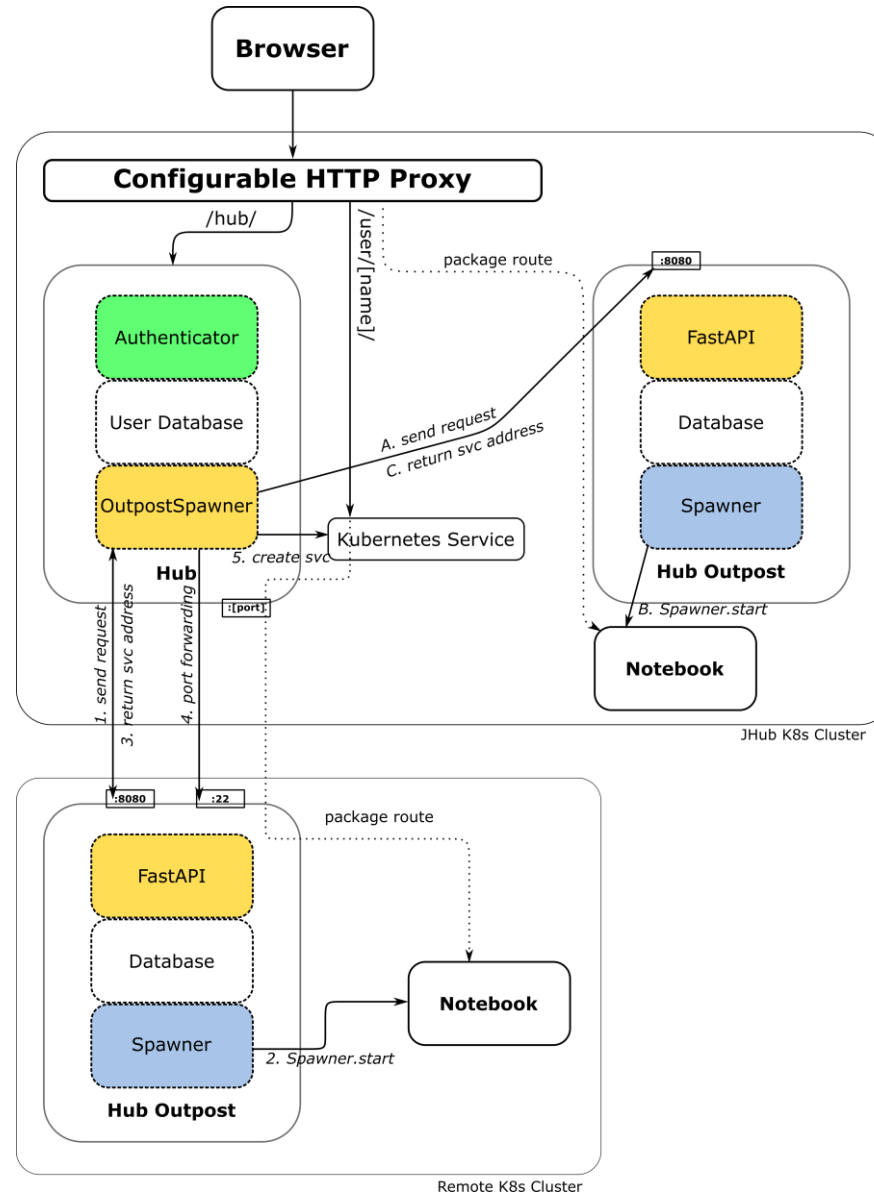


external system

ARCHITECTURE

Start Jupyter-server on two systems

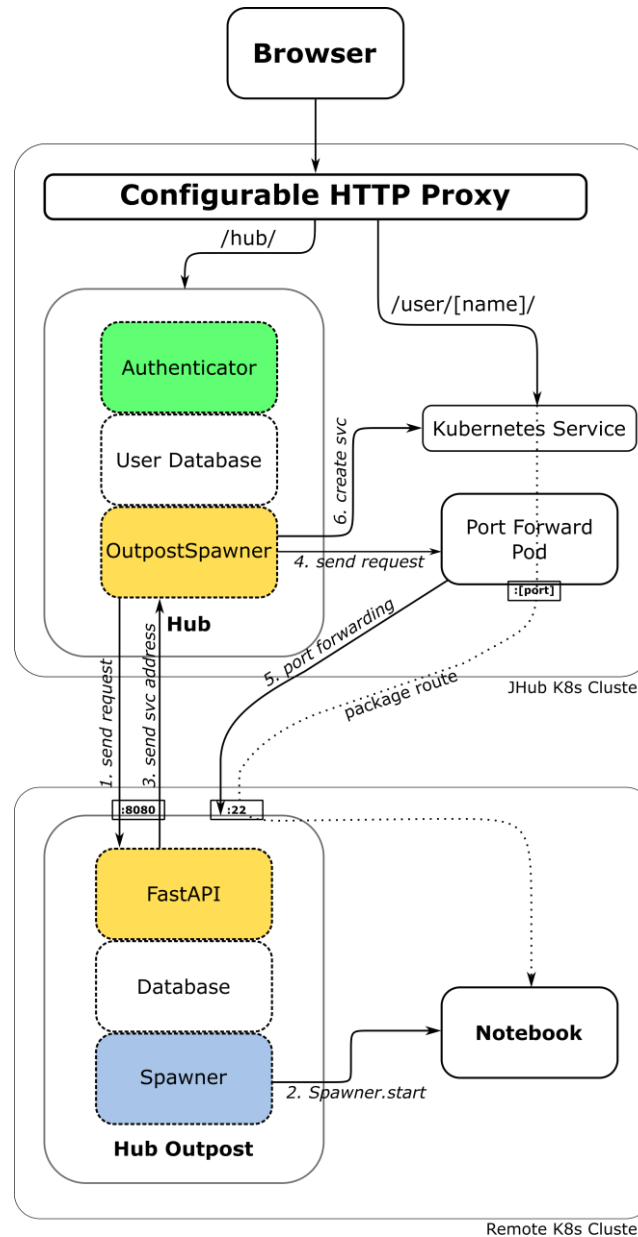
- Simple Setup to allow start on two different K8s clusters
- Remote cluster can be managed by different administrator
- Remote cluster has full control of their resources via flavors



ARCHITECTURE

External Tunneling

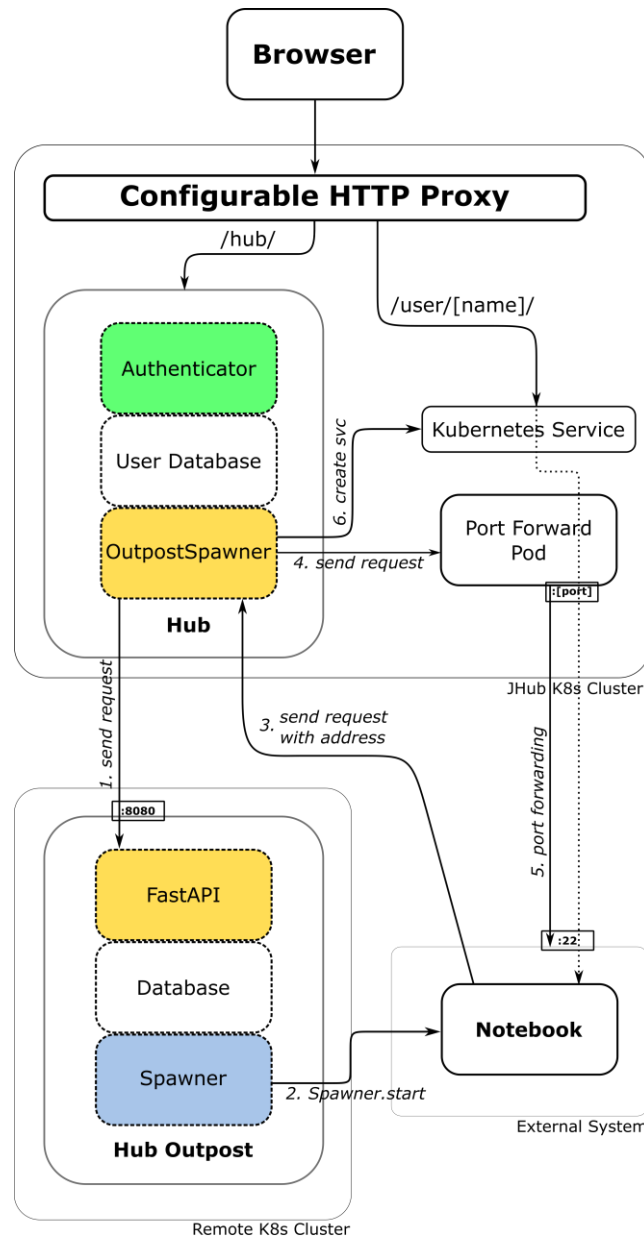
- External tunneling pod responsible for ssh port-forwarding
- Create your own solution, or use an existing one
- Users keep access to their running servers if JupyterHub is unavailable



ARCHITECTURE

Delayed Tunneling

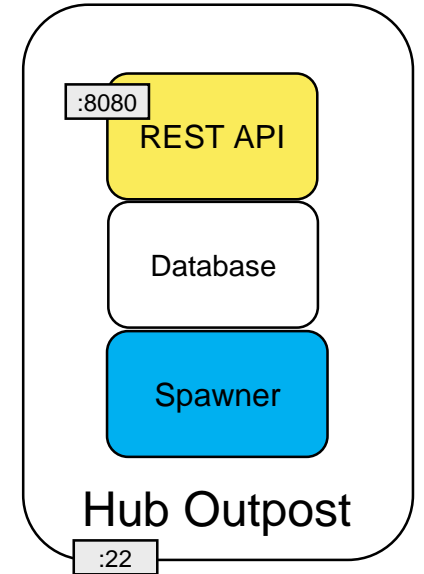
- Jupyter server may run on external system
- Perfect for hpc systems, if the svc address is not known during start time
- Start process of jupyter server has to send a request with its address to the Outpostspawner



OUTPOST

Key features

- Start jupyter server on remote systems, using classical JupyterHub spawner
- A central JupyterHub may use multiple Outposts, each with its own administrator and configuration
- Each Outpost may be used by multiple JupyterHubs
 - Each JupyterHub uses its own credentials, so they don't interfere with each other
- Override Spawner configuration for each jupyter server possible
 - e.g. user selects docker image which should be used, if allowed by Outpost
- Flavors – Configure how many resources should be used by a hub or user
 - Hub-based flavors: Different hubs may use different amount of resources
 - User-based flavors: Allow resource usage for each individual user



OUTPOST

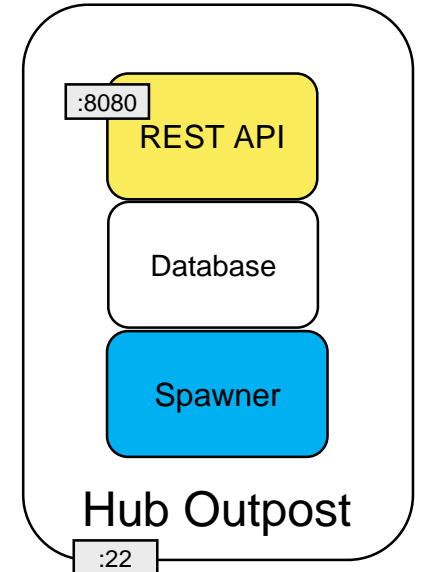
Key features

- Start jupyter server
- A central JupyterHub administrator
- Each Outpost
 - Each Jupyter server
- Override Spawner
 - e.g. user-specific
- Flavors – Configurable
 - Hub-based
 - User-based

Configured at each Outpost

```
from kubespawner import KubeSpawner
c.JupyterHubOutpost.spawner_class = KubeSpawner
c.KubeSpawner.image = "jupyter/minimal-notebook:notebook-7.0.3"
c.KubeSpawner.... = ... # use any Spawner with any configuration you like

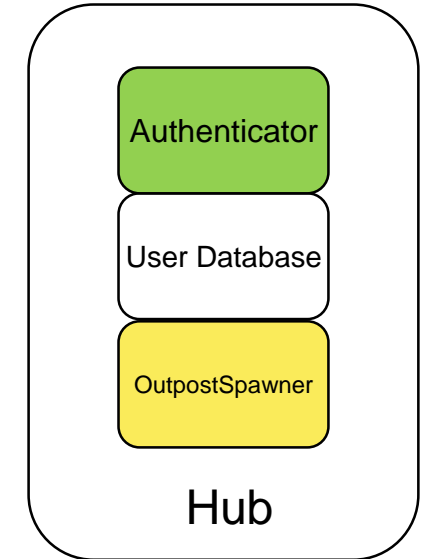
flavors = {
    "m1": {
        "max": 5,
        "display_name": "2GB RAM, 1 VCPU, 120 hours",
        "description": "Service will run for maximum 5 days with 2GB RAM",
        "runtime": { "days": 5 }
    }
}
c.JupyterHubOutpost.flavors = flavors # may also be a function
# c.JupyterHubOutpost.user_flavors = ... # create a function to allow \
different flavors for each user
# c.JupyterHubOutpost.flavors_undefined_max = 10 # used when no flavor \
is used, limits the number of servers to 10 overall
...
```



OUTPOSTSPAWNER

Key features

- Remote jupyter server management via POST/GET/DELETE requests
 - Send all required information to the Outpost, which will then use a classical spawner to start the jupyter server.
- Additional API Endpoints
 - List all running servers. May be used by Outpost to cleanup servers no longer running
 - SpawnEvents – Outpost (or start process of jupyter-server) may send information about current spawn status
 - SetupTunnel – May be used by jupyter-server start process, to create ssh tunnel to running server
 - TunnelRestart – Inform hub that tunnels to a certain node must be recreated
 - Flavor Update – Outpost may update the flavors for its own system



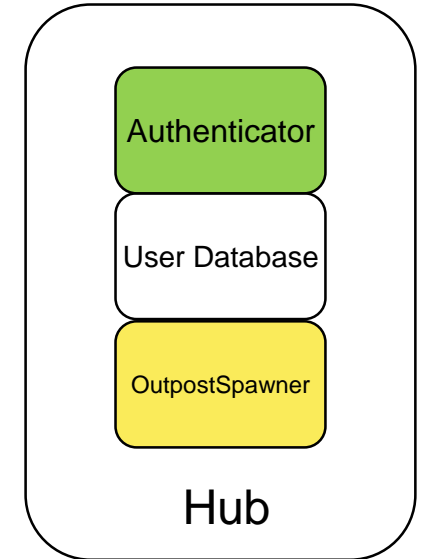
OUTPOSTSPAWNER

Key features

- Remote jupyter
 - Send all requests to jupyter server
- Additional API endpoints
 - List all running notebooks
 - SpawnEvent to spawn stateful notebooks
 - SetupTunnels
 - TunnelResources
 - Flavor Upgrades

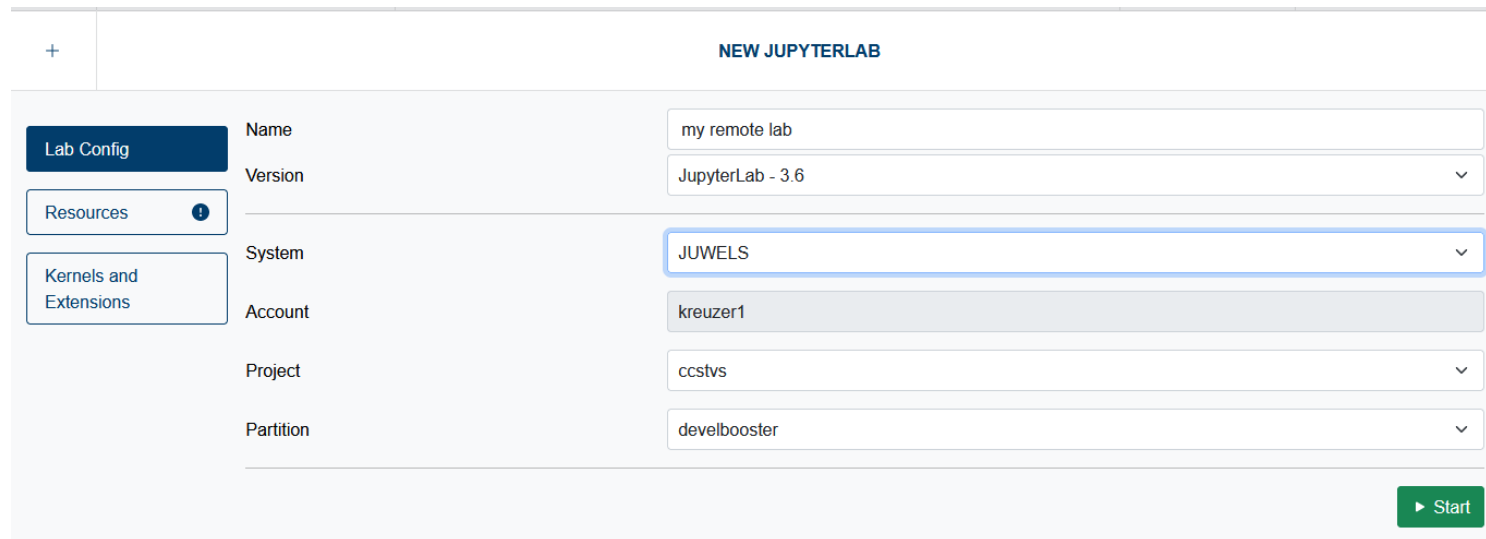
Configured at central JupyterHub (e.g. z2jh)

```
c.JupyterHub.custom_scopes = { ... } # define scopes for API-Endpoints
from outpostspawner import OutpostSpawner
c.JupyterHub.spawner_class = OutpostSpawner
c.OutpostSpawner.options_form = """
Choose a system:
<select name="system">
  <option value="Local">Local</option>
  <option value="Remote-A">Remote A</option>
  <option value="Remote-B">Remote B</option>
</select>
"""
def my_request_url(spawner, user_options): ... # define your own logic
def my_request_headers(spawner, user_options): ... # for each system
c.OutpostSpawner.request_url = my_request_url
c.OutpostSpawner.request_headers = my_request_headers
... # full example: https://github.com/jupyterhub/team-compass/issues/722
```



SUMMARY

- The JupyterHub Outpost allows us to reach multiple systems with one central Hub
- Remote systems keep full control over their resources
- Using Kubernetes ingress + https enables a secure connection between Hub and Outposts
- Using `c.JupyterHub.internal_ssl` feature allows secure connection between Hub and Jupyter server
- Already in use at <https://jupyter.jsc.fz-juelich.de> and a few more project specific JupyterHubs



The screenshot shows the 'NEW JUPYTERLAB' configuration page. On the left, there are three tabs: 'Lab Config' (selected), 'Resources', and 'Kernels and Extensions'. The main form contains the following fields:

| Field | Value |
|-----------|------------------|
| Name | my remote lab |
| Version | JupyterLab - 3.6 |
| System | JUWELS |
| Account | kreuzer1 |
| Project | ccstvs |
| Partition | develbooster |

A green 'Start' button is located at the bottom right of the form.

SUMMARY

- The JupyterLab Outpost allows us to reach multiple systems with one central Hub
- Remote systems
- Using Kubernetes
- Using containers
- Already in use

The screenshot displays the configuration page for a JupyterLab Outpost. On the left, a sidebar contains navigation tabs: 'Lab Config' (selected), 'Resources', and 'Kernels and Extensions'. Below these are fields for 'Name', 'Version', 'System', 'Account', 'Project', and 'Partition'. The main area is divided into two sections. The top section, 'Lab Config', includes fields for 'Name' (my remote lab), 'Version' (Custom Docker image (starting a Jupyter server)), 'Image' (jupyter/minimal-notebook:notebook-7.0.3), 'Mount user data' (checked), and 'User data path' (/mnt/userdata). The bottom section, 'System', shows 'System' (JSC-Cloud) and 'Flavor' (8GB RAM, 2VCPUs, 10 hours). Below this is the 'Available Flavors' section, which includes a legend: green for Free, blue for Used, and red for Limit exceeded. Three flavor bars are shown: 2GB RAM, 1VCPU, 5 days (3 used, infinity free); 4GB RAM, 1VCPUs, 2 days (10 used, infinity free); and 8GB RAM, 2VCPUs, 10 hours (5 used, infinity free). A 'Start' button is located at the bottom right.

| Flavor | Used | Free |
|---------------------------|------|------|
| 2GB RAM, 1VCPU, 5 days | 3 | ∞ |
| 4GB RAM, 1VCPUs, 2 days | 10 | ∞ |
| 8GB RAM, 2VCPUs, 10 hours | 5 | ∞ |