Unchaining JupyterHub

Running notebooks on resources without inbound connectivity

Oliver Freyermuth, Katrin Kohl, Peter Wienemann

University of Bonn
{freyermuth,kohl,wienemann}@physik.uni-bonn.de

27th May, 2021

Physikalisches Institut
UNIVERSITÄT BONN

Why JupyterHub?

- JupyterHub is a web 'hub' providing access to notebooks
- Notebooks can use various kernels (Python 2/3, R, Julia, ROOT / C++,...)
- Interactive graphics, terminals, X11 via XPRA / noVNC,...
- Collaborative work possible (shared filesystems, git...)

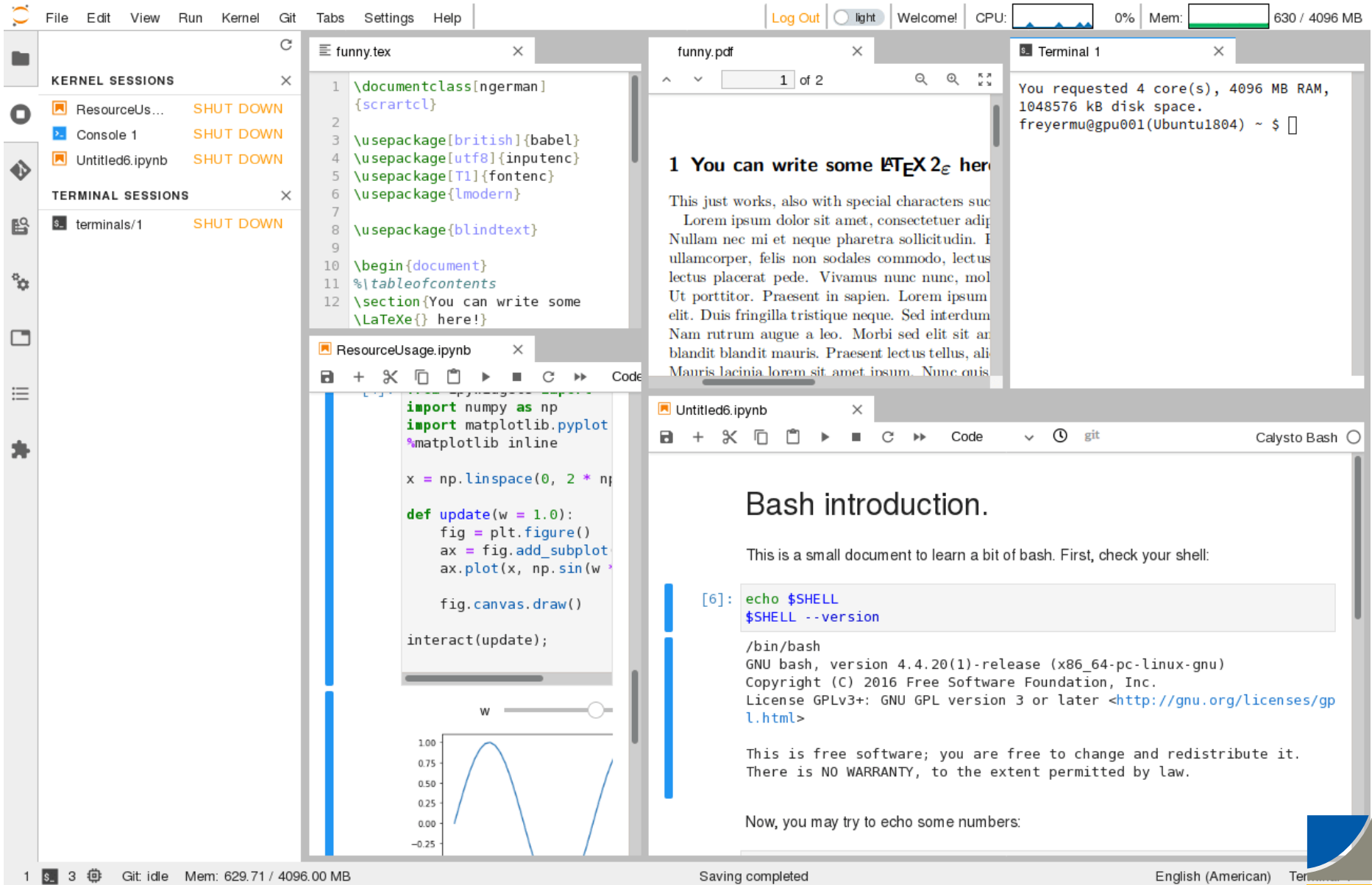
In summary...

JupyterHub allows interactive work from a browser, without installing software on end user device.

Use cases

- Rapid prototyping / 'Trying things out'
- Teaching (algorithms, methods)
- Sharing of small analyses (self-documenting)
- Remote work (with notebooks / remote desktop in browser)

An example workspace



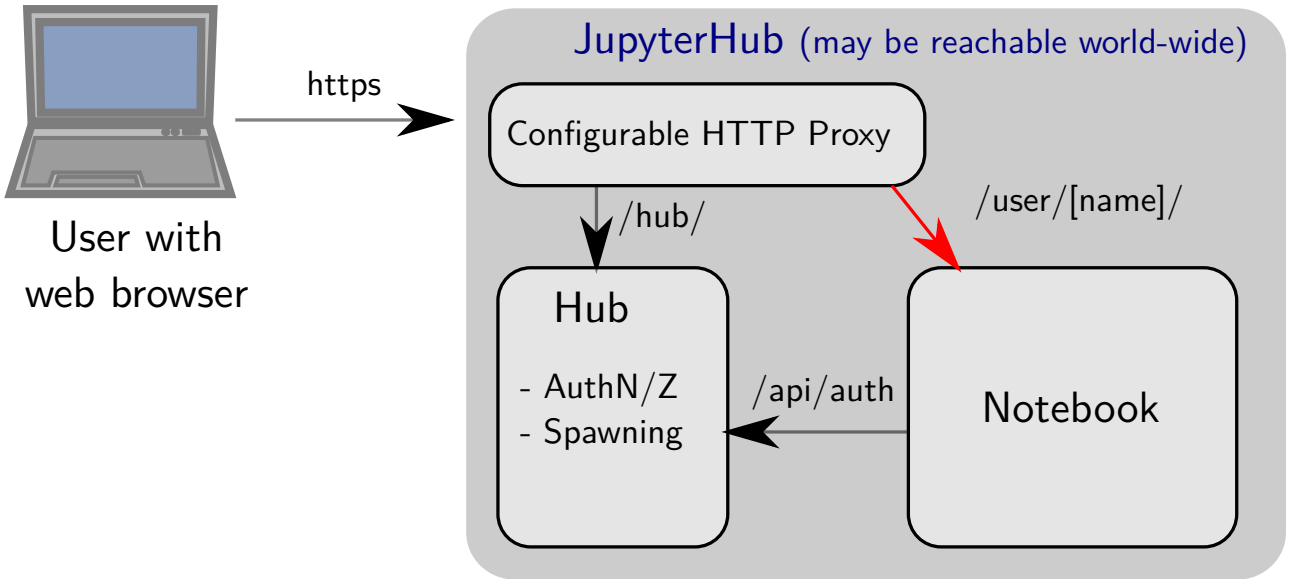
Operational hurdles

- Commonly operated on dedicated cloud infrastructure (e.g. Kubernetes) ⇒ Typically runs in different environment than other scientific use cases
- Combines a plethora of versions and packaging systems (pip, conda, npm, yarn, . . .) →  Upgrade headache
- Very active development with breaking changes
- In many cases problematic security concepts (e.g. Hub server needs direct access to execute nodes)
- Operationally, a Hub is 'chained' to the resource admins
  (note this also prevents safe use of distributed / federated resources)

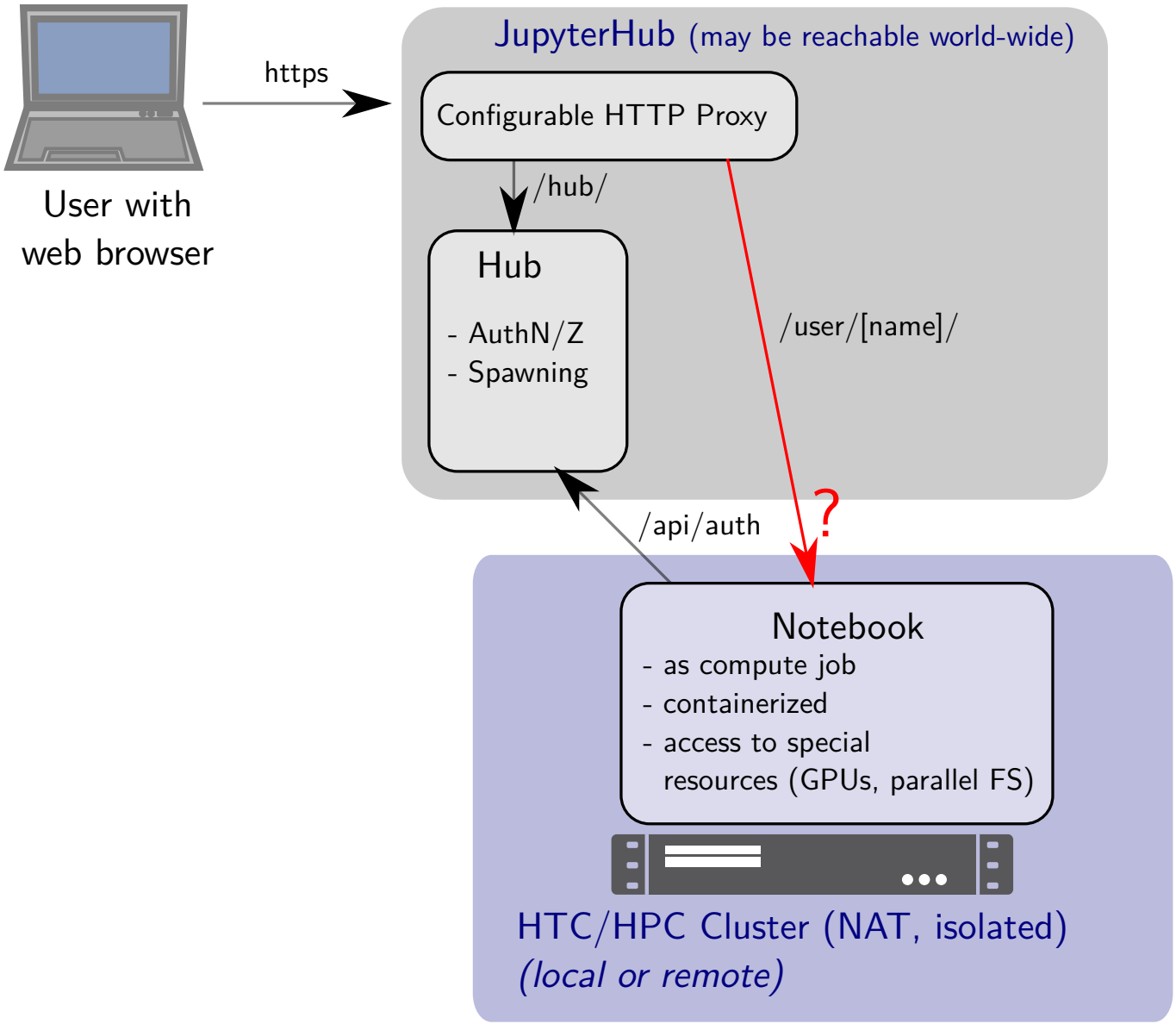
Can we overcome some of these?

Let us investigate JupyterHub networking!

Networking with JupyterHub



Networking with JupyterHub



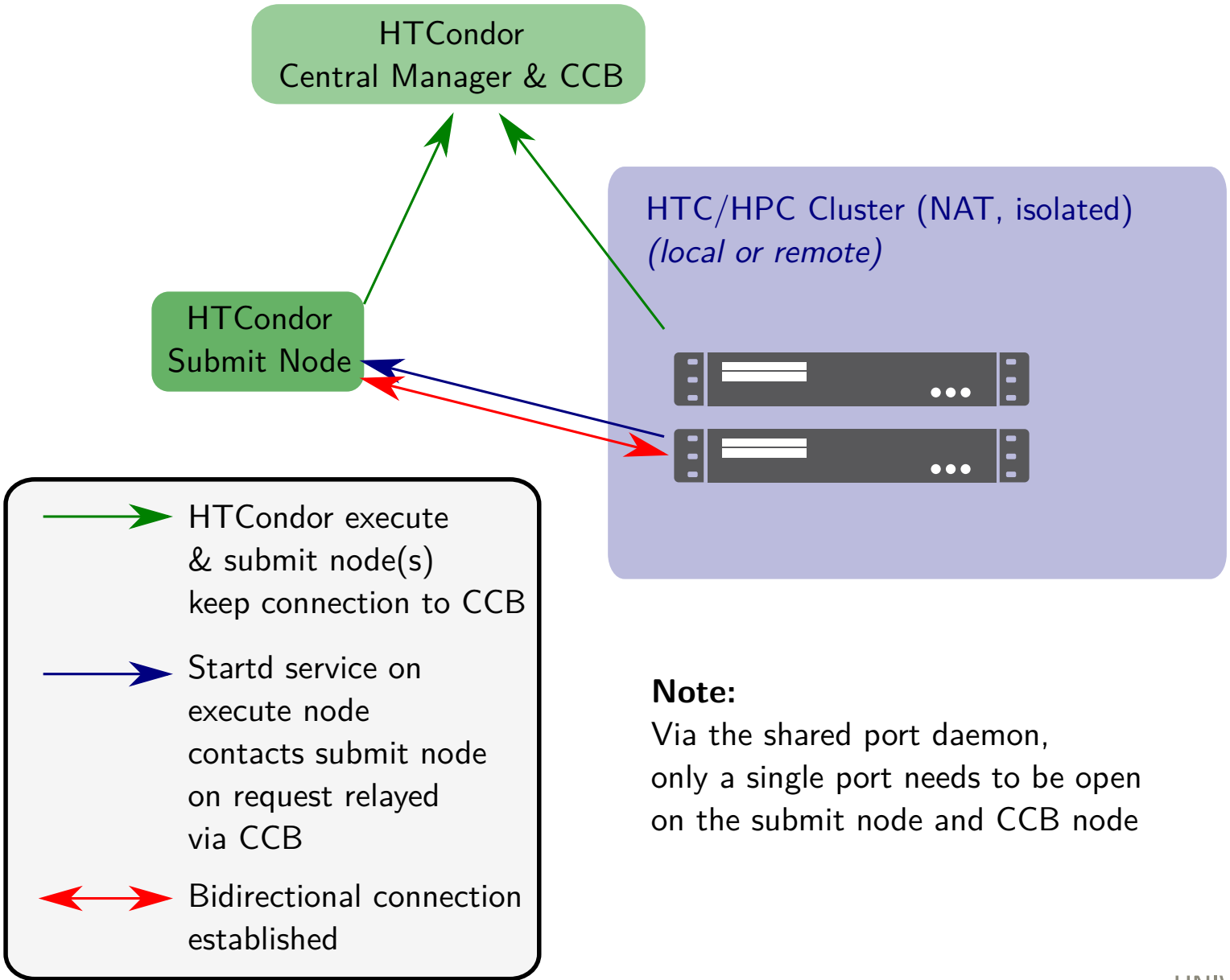
Networking with JupyterHub

- The inbound connection to the notebook will use a random port, defined by the spawned notebook
- The (potentially world-reachable) Hub needs direct access to the execute node
- Additionally, no / reduced firewalling on the execute node possible (random ports)

Can we overcome this issue?

Can HTCondor help out?

Networking with HTCondor (simplified)



Note:
Via the shared port daemon,
only a single port needs to be open
on the submit node and CCB node

Networking with HTCondor (simplified)

- CCB (HTCondor Connection Brokering) allows submit node to connect to execute node by leveraging a reverse connection
- This works both for daemon communication and command line tools
- It overcomes the common case of isolated execute nodes
- Notably, it also works for `condor_ssh_to_job`
- Regular HTCondor AuthN/Z applies first
- For SSH, a temporary pair of keys is used
- That means we can SSH into any worker node which has outbound connectivity, even without inbound connectivity

Can we forward the port of the notebook via an SSH tunnel?

Manual testing: Yes!
But: Batch spawner needs to be extended.

JupyterHub Batch spawner

Concept

1. A job is submitted to the batch system ('spawning')
2. JupyterHub monitors the state of the job
3. Payload starts (single-user notebook): random listen port (TCP)
4. Payload contacts JupyterHub server (fixed API port), communicates the random port on the execute node
5. Classically: JupyterHub tells 'configurable HTTP proxy' to proxy the user *directly* to the random port on the execute node

JupyterHub batch spawner needs to be extended

1. Add a generic, optional 'connect to job' functionality
2. In case of HTCondor, leverage `condor_ssh_to_job` to forward the port to `localhost` on the Hub

JupyterHub Batch spawner

Our generic implementation

1. Payload has communicated random port (startup finished)
2. If required for the 'connect to job' command:
   1. JupyterHub selects an unused, local random port
   2. Remote and local port passed to the 'connect to job' command

   This allows to forward from the remote port to an unused, randomized local port
3. 'connect to job' command is called as background command
4. Aborted if 'connect to job' exits during startup
5. Job killed if connection is lost during session

For `CondorSpawner`

- use `condor_ssh_to_job` with `-oExitOnForwardFailure=yes`
- override notebook `hostname` with `localhost`

JupyterHub Batch spawner

How to use the implementation?

- Full implementation in this pull request (awaiting review): https://github.com/jupyterhub/batchspawner/pull/200
- For maximum profit, an HTCondor setup with CCB and shared port configuration is needed
- For other batch systems: start from generic implementation added to the Batch spawner
  - 'connect to job'-command is a template string with job id, remote and local port, hostname
  - We're not aware of built-in functionality as in HTCondor
  - Requirement: some command to establish the connection (like `ssh`, e.g. via a bastion host)

In production since a month.

A few more details on the components we use...

Components of our setup

- Deployment and configuration with Foreman / Puppet for cluster, desktops, servers and services (→HEPiX Autumn 2019)
- Desktops are submit nodes, allow interactive jobs with X11
- All jobs executed in containers
- Infrastructure using a mix of CentOS 7 and 8
- Desktops with Ubuntu 18.04 → Debian 11
- CephFS as cluster file system (can optionally be used in JupyterHub) (→HEPiX Autumn 2019)

For JupyterHub...

- Puppetized VM setting up the Hub web service
- Regular containers extended with a VirtualEnv & Lab extensions, based on Anaconda, activated via Lmod
- Plan to build environments via automated workflows (CI/CD)
- Distributed via CVMFS

Components of our setup

Authentication

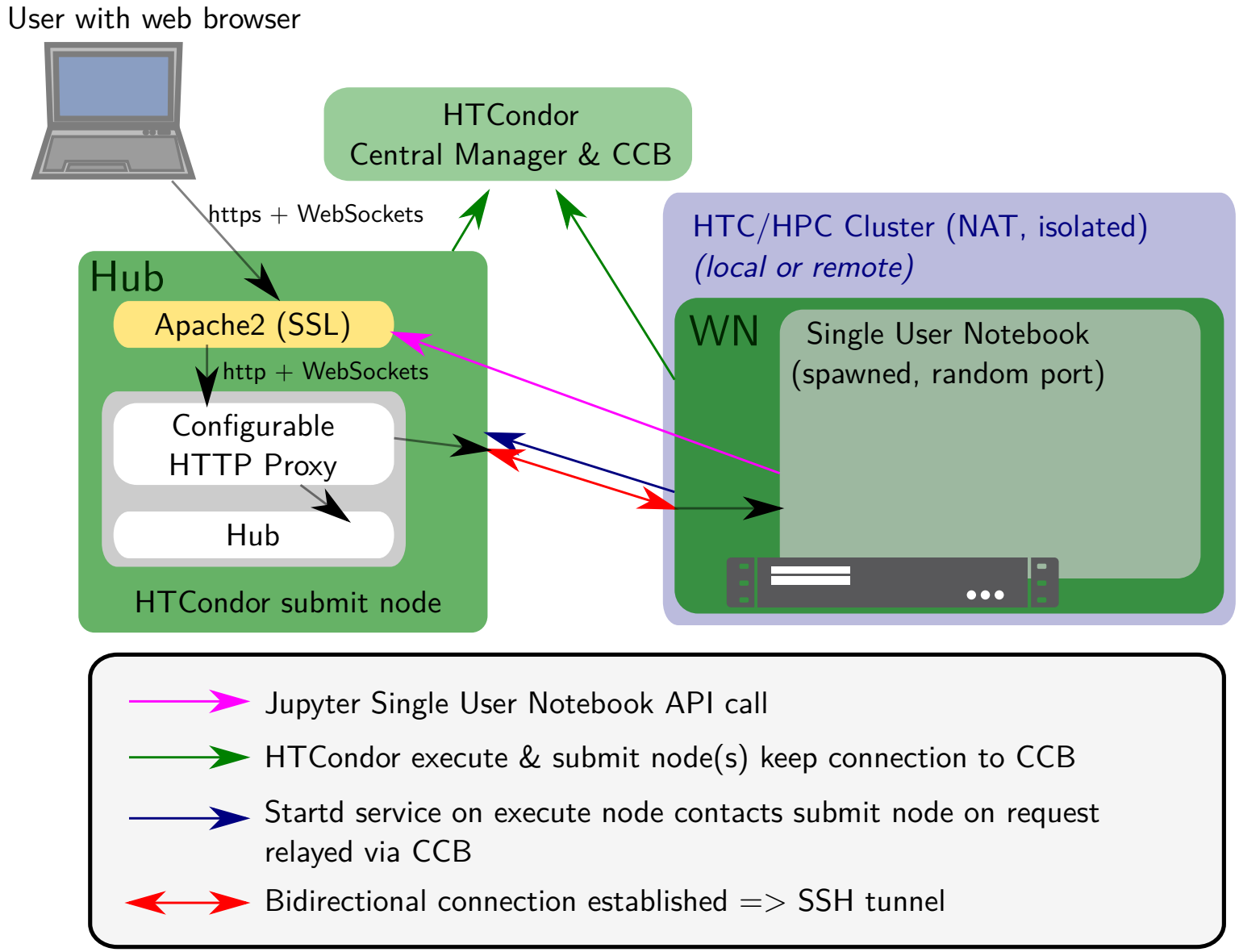
- Login to the hub creates a Kerberos TGT (via PAM)
- Kerberos used for job submission (and inter-daemon communication with HTCondor)
- However: not a requirement (tokens on the horizon)

File system decoupled

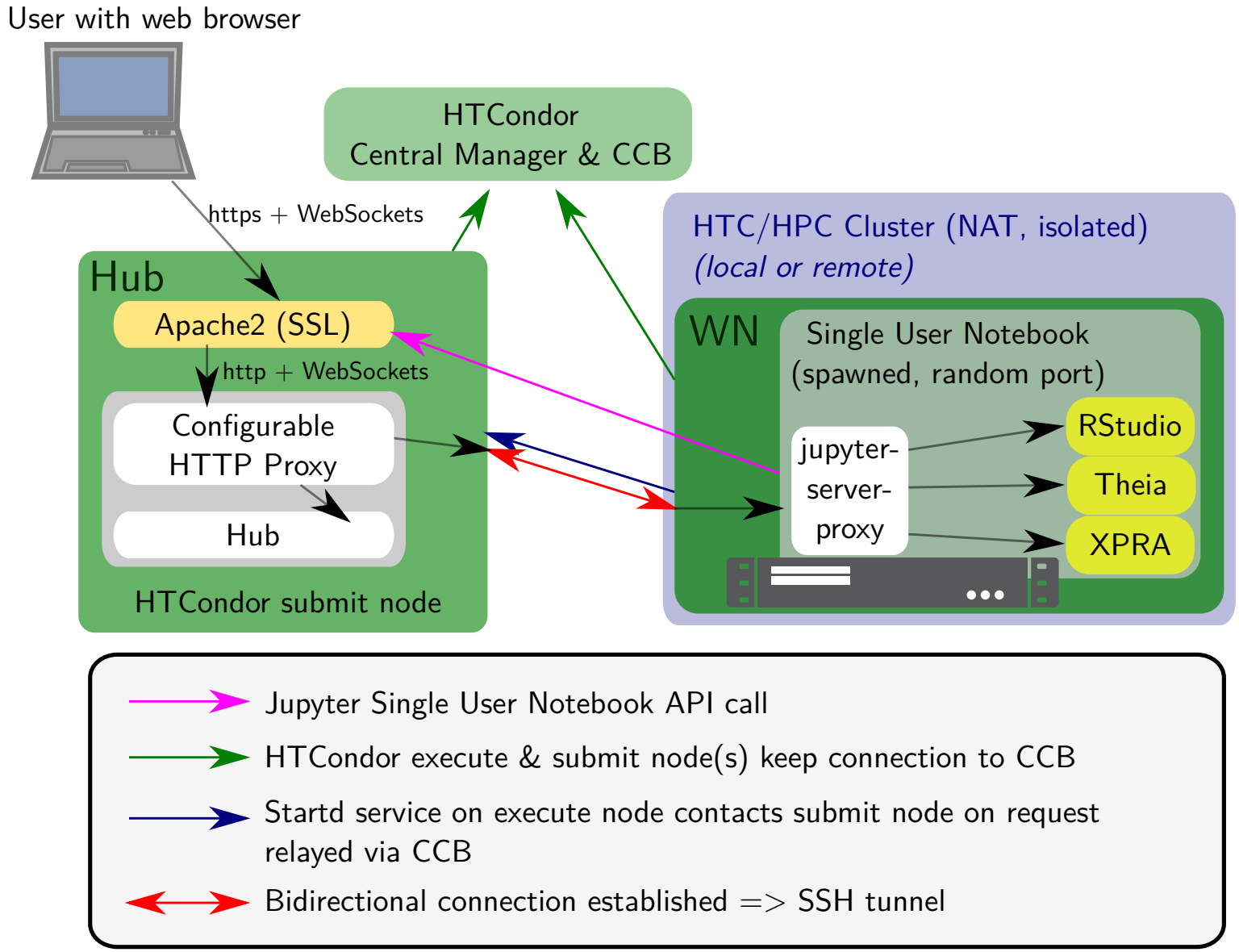
- Users have kerberized home directories on NFS, mounted on the Hub, but not on cluster nodes
- HTCondor file transfer used to transfer a ~/jupyter directory into the job and back when job exits:

```
when_to_transfer_output = ON_EXIT_OR_EVICT
+SpoolOnEvict = False
```

Overall schematic



Overall schematic



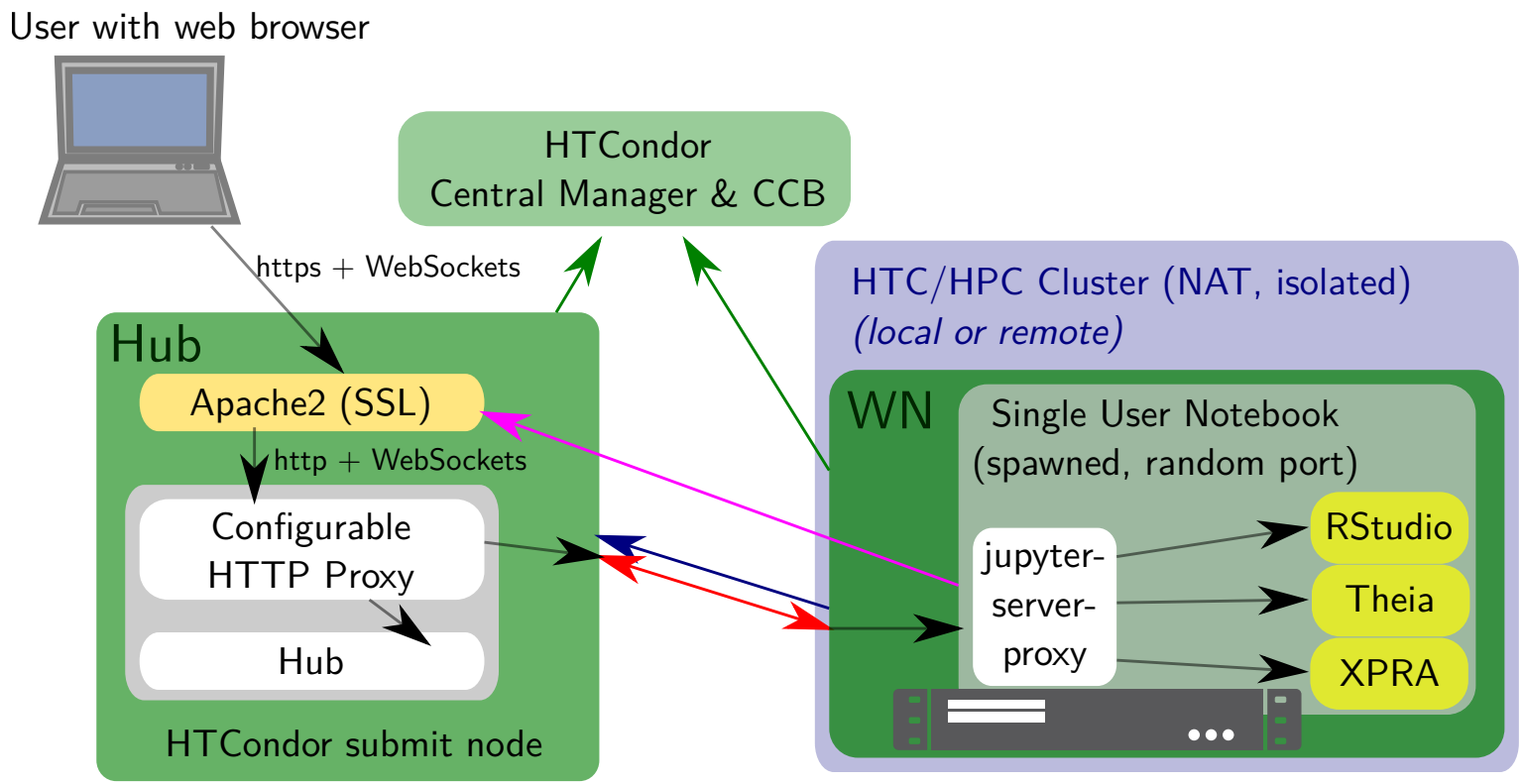
Other Web Services

Adding a proxy to the notebook

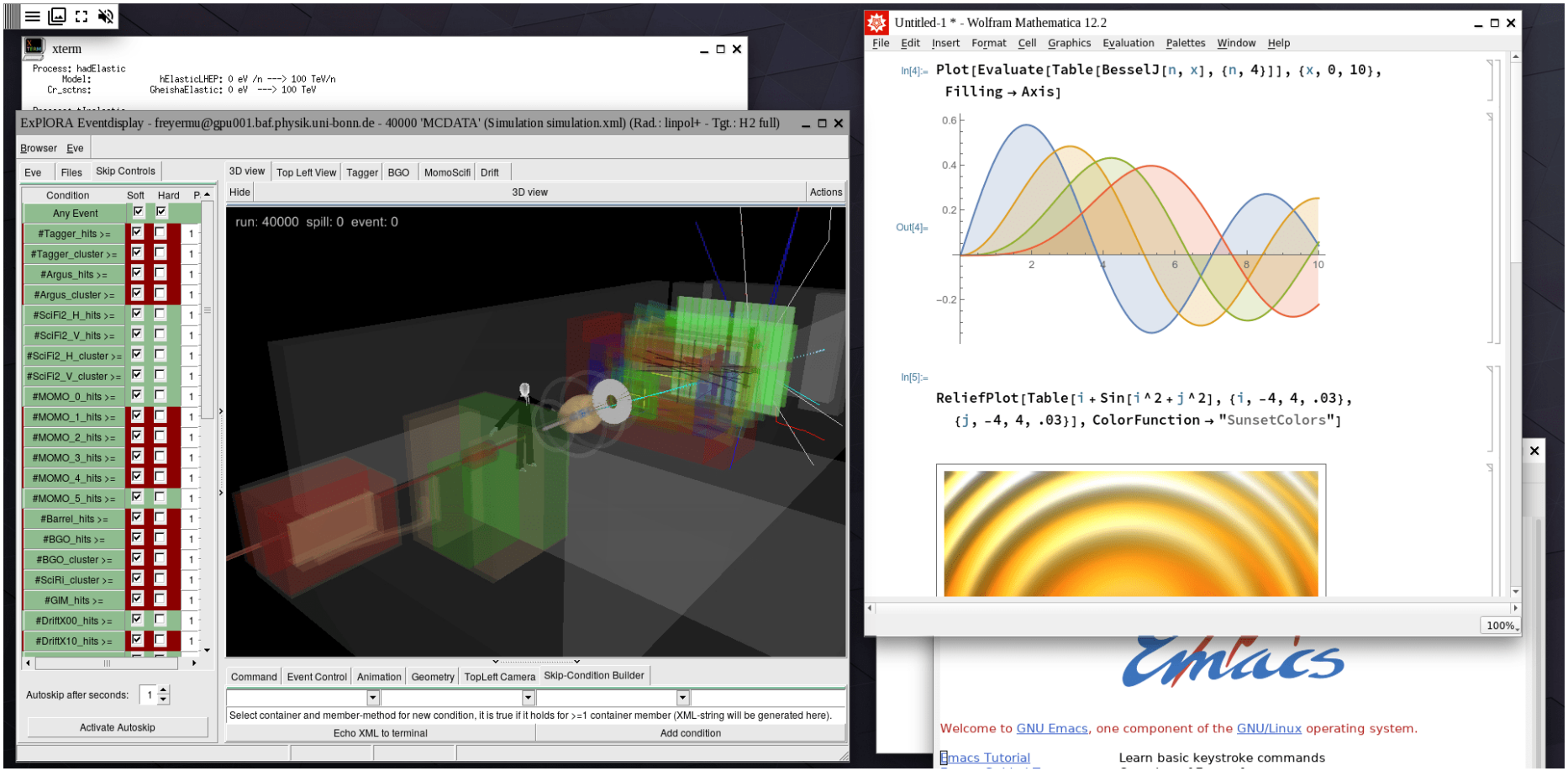
- jupyter-server-proxy extension adds another proxy layer (HTTP / WebSockets) inside single-user notebooks
- Single point of entry to notebook remains one port (i.e. our SSH tunnel)
- Proxying is done after authentication
- Allows to access tools external to JupyterLab, for example:
  - X11 desktop (e.g. via XPRA) via jupyter-xprahtml5-proxy
  - Tools with HTML5 frontends (RStudio, Theia,...)

Note: Secure authentication should happen on shared nodes!

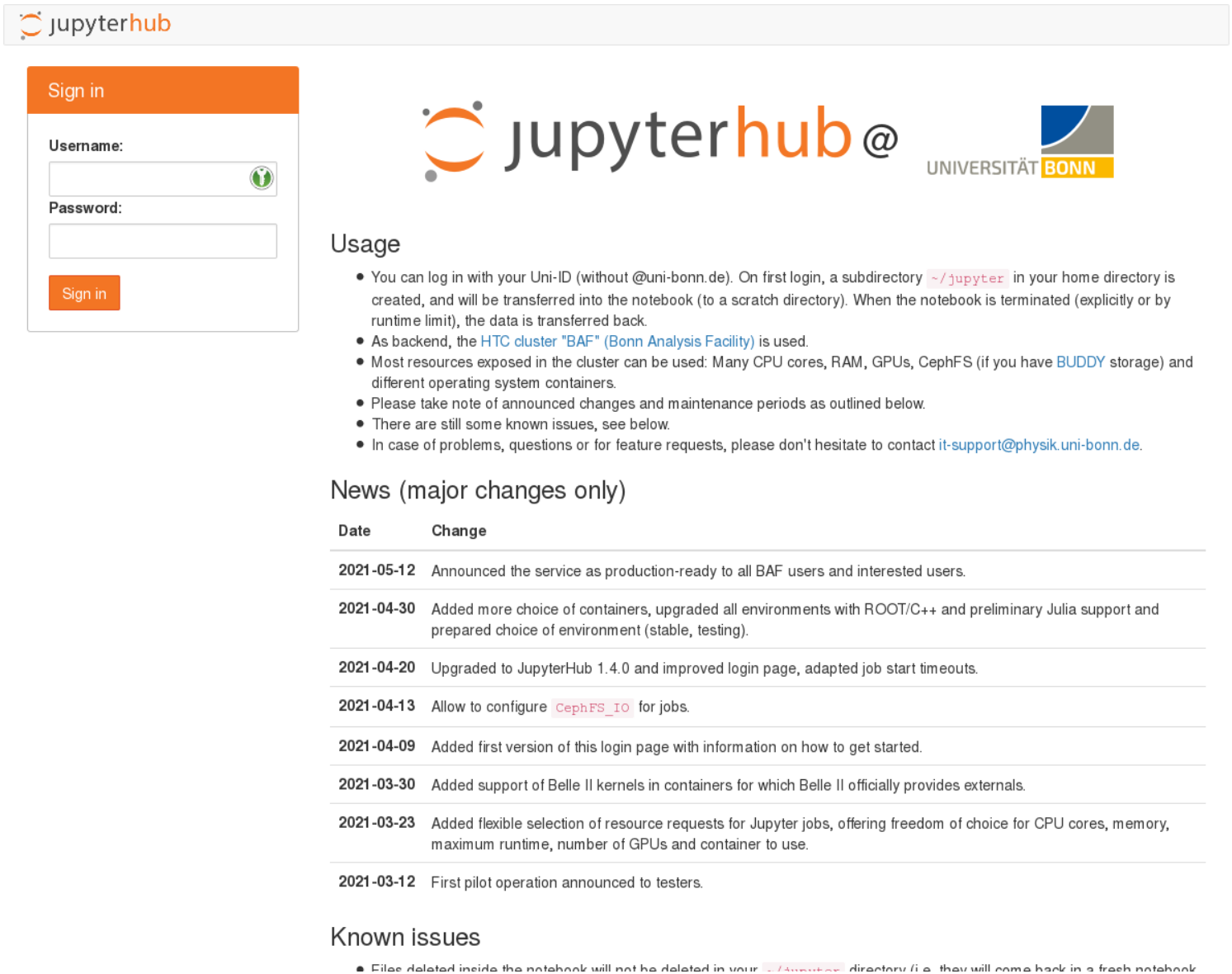
Overall schematic



Some impressions: X11 applications in your browser

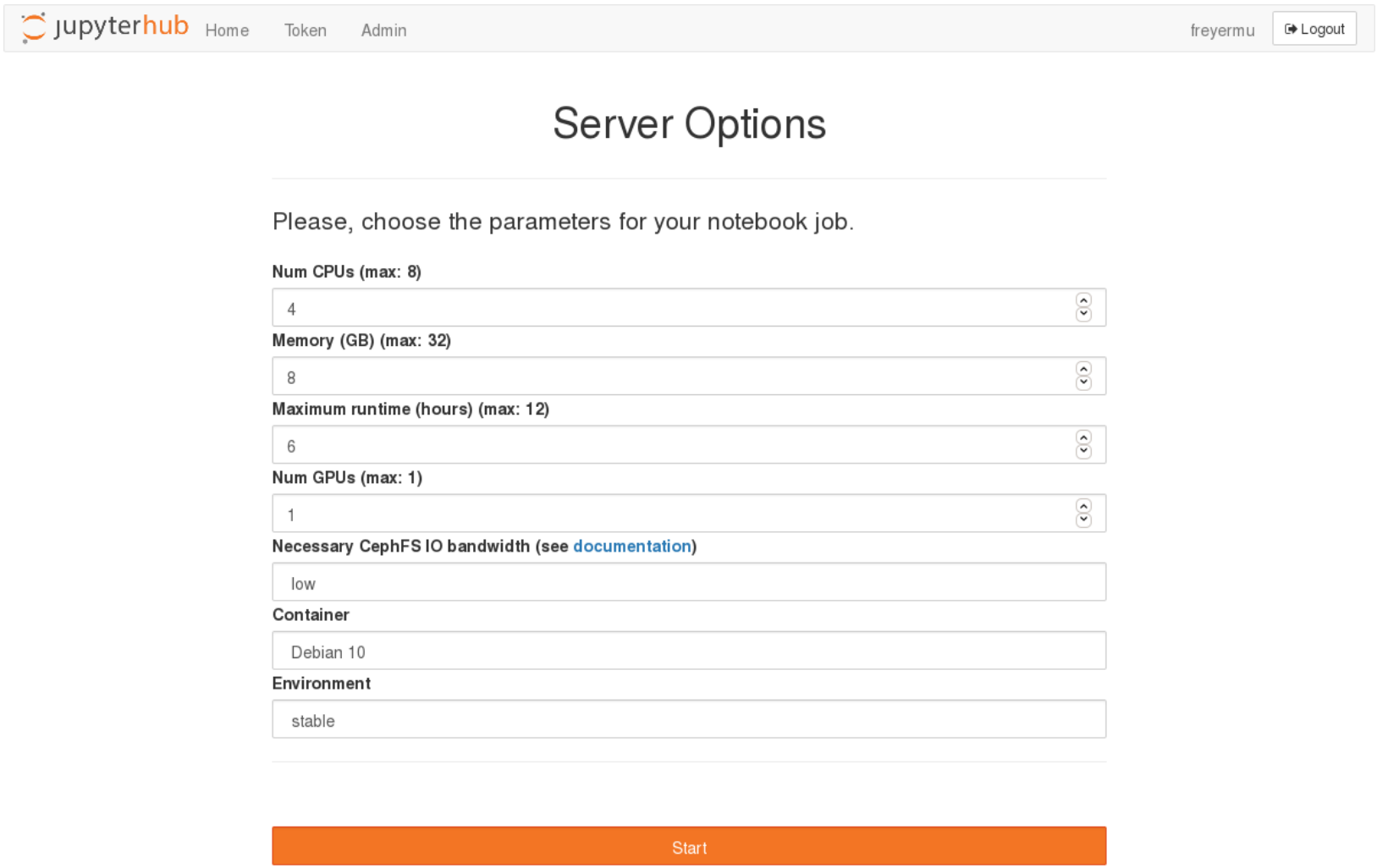


Some impressions: Customized login page



see also: JupyterHub issue 3414

Some impressions: Customized FormSpawner



see also: unibonn/ubnjupyterspawner (GitHub)

Scaling out

Resource Federations

- Overlay batch systems can be used with this implementation
- JupyterHub Unchained: Resources can be used without privileges and without dropping the firewalls
- Allows for use in a federated research platform

Components for scaling out

- HTCondor (flexible scheduling, file transfer functionality)
- CVMFS for software stack and container images
- Containerization (possibility to use user namespaces)
- COBaID/TARDIS to spawn resources for an overlay batch system

$\Rightarrow$ For more details, see HEPiX Spring 2021!
(likely also HTCondor Europe 2021)

Summary & Outlook

Summary

- JupyterHub Batch spawner extended to remove need for inbound connectivity
- Highly portable notebook environment (containerized, can spawn on almost any HPC / HTC resource)
- Now collecting experience in pilot operation phase

Outlook

- Use CI/CD to build notebook environment
- Extend functionality (e.g. offer HTMap)
- Test scaling out to other resources
- Publication in a peer-reviewed paper in preparation

Thank you

for your attention!