

# 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`

16<sup>th</sup> March, 2021

# 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 locally.

## 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

The screenshot displays a JupyterLab interface with the following components:

- Top Bar:** Includes menu items (File, Edit, View, Run, Kernel, Git, Tabs, Settings, Help), status indicators (Log Out, Light, Welcome!), and resource usage (CPU: 0%, Mem: 630 / 4096 MB).
- Left Sidebar:**
  - KERNEL SESSIONS:** Lists 'ResourceUs...', 'Console 1', and 'Untitled6.ipynb', all with a 'SHUT DOWN' button.
  - TERMINAL SESSIONS:** Shows 'terminals/1' with a 'SHUT DOWN' button.
- Main Workspace:**
  - funny.tex:** A LaTeX document with a preamble including packages like `\documentclass[ngerman]{scrartcl}`, `\usepackage[british]{babel}`, `\usepackage[utf8]{inputenc}`, `\usepackage[T1]{fontenc}`, `\usepackage{lmodern}`, and `\usepackage{blindtext}`. The body contains a table of contents and a section titled 'You can write some LaTeX here!'.
  - Untitled6.ipynb:** A Python notebook with the following code:
 

```
import numpy as np
import matplotlib.pyplot
%matplotlib inline

x = np.linspace(0, 2 * np.pi, 100)

def update(w = 1.0):
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    ax.plot(x, np.sin(x))

    fig.canvas.draw()

interact(update);
```

 Below the code is a plot of a sine wave.
  - Terminal 1:** Displays the output of a command: 'You requested 4 core(s), 4096 MB RAM, 1048576 kB disk space. freyermu@gpu001(Ubuntu1804) ~ \$'.
  - Bash introduction:** A document showing the output of `echo $SHELL` and `$SHELL --version`, displaying GNU bash version 4.4.20(1)-release.

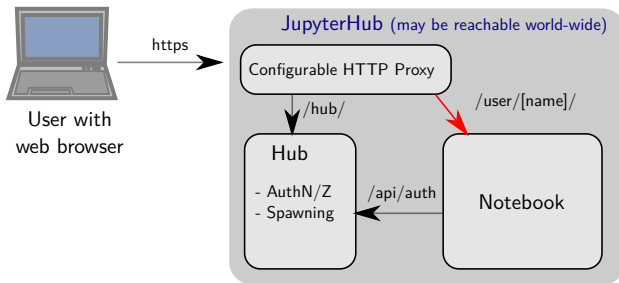
# Operational hurdles

- Commonly operated on dedicated cloud infrastructure (e.g. Kubernetes)  $\Rightarrow$  Typically runs in different environment than other scientific use cases
- Combines a plethora of versions and packaging systems (pip, conda, npm, yarn, ...)  $\rightarrow$  🍌 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*)

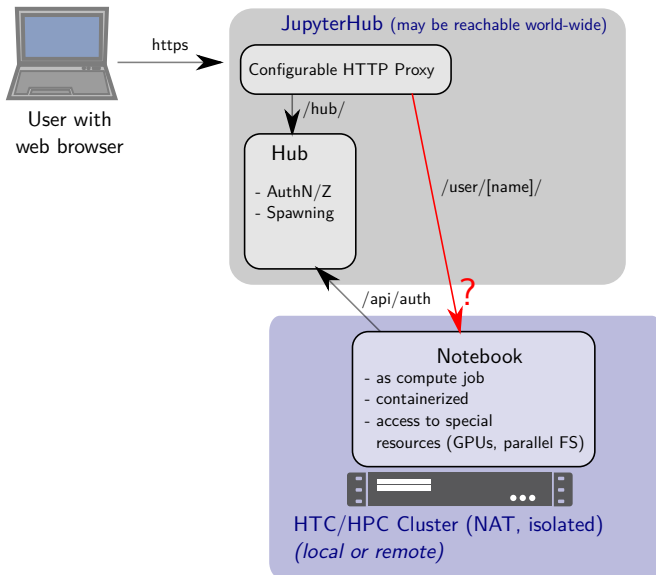
## Can we overcome some of these?

Let us investigate JupyterHub networking!

# Networking with JupyterHub



# Networking with JupyterHub



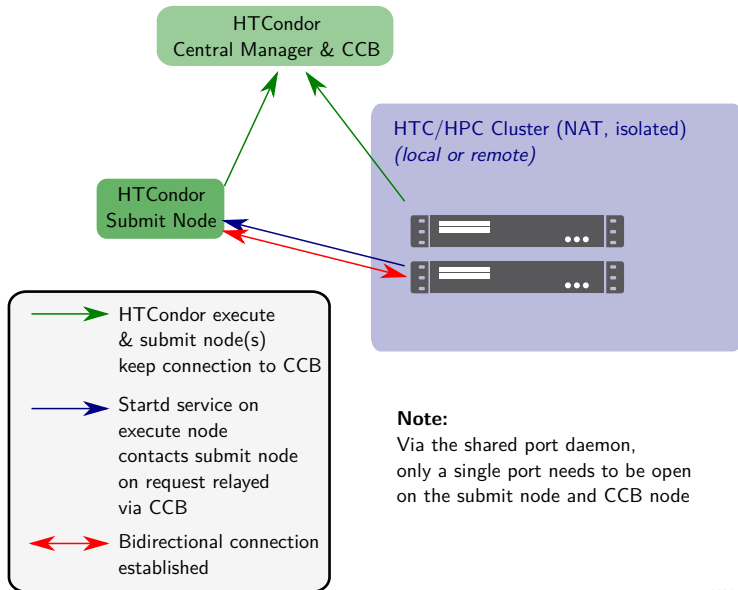
# Networking with JupyterHub

- 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?

How do workload management systems work with NATed execute nodes...?

# 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
- 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
  - 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)

## We are still in the pilot operation phase.

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)

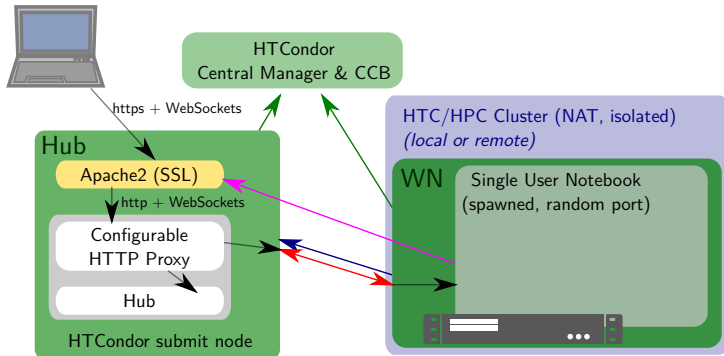
## File system decoupled





- 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

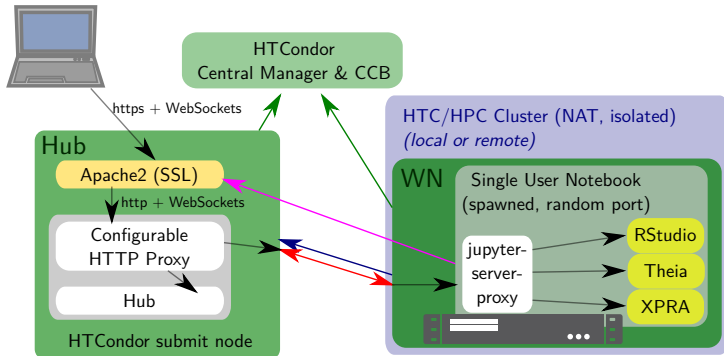
User with web browser



-  Jupyter Single User Notebook API call
-  HTCondor execute & submit node(s) keep connection to CCB
-  Startd service on execute node contacts submit node on request relayed via CCB
-  Bidirectional connection established => SSH tunnel

# Overall schematic

User with web browser



# Other Web Services

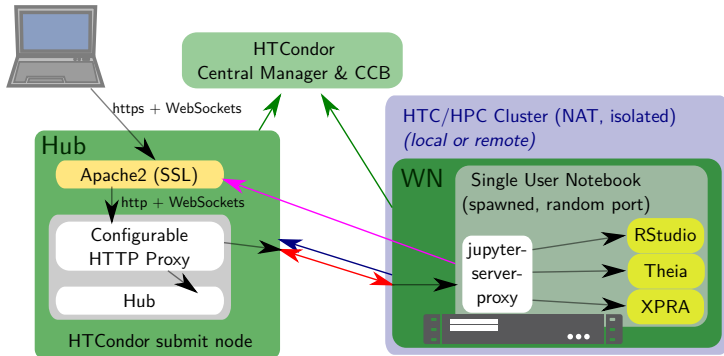
## Adding a proxy to the notebook

- [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, ...)

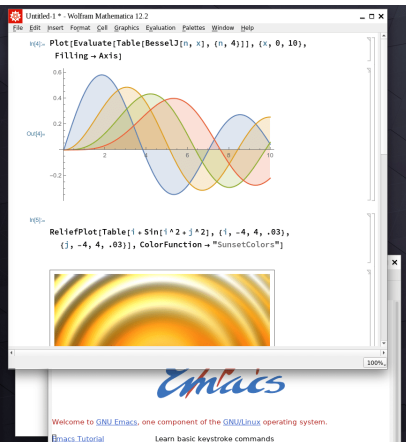
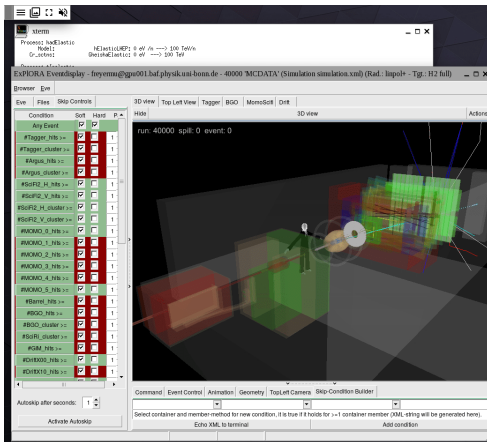
*Note: Secure authentication should happen on shared nodes!*

# Overall schematic

User with web browser



# X11 applications in your browser



# 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

⇒ For more details, stay for the [next talk](#)!

# 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 experiences in pilot operation phase

## Outlook

- Use CI/CD to build notebook environment
- Extend functionality (e.g. offer [HTMap](#))
- Test scaling out to other resources

Thank you  
for your attention!

