HTCondor and Interactive use

A success story

Oliver Freyermuth, Michael Hübner

University of Bonn
freyermuth@physik.uni-bonn.de,michael.huebner@uni-bonn.de

12th July, 2024

Physics Institute at University of Bonn

- over 280 members in 28 working groups, plus users from related Physics institutes and with HTC workloads
- Biggest particle accelerator run by a German university (‘ELSA’, 164.4 m circumference) with two experiments ($\approx$ 50 people)
- Groups from:
  - particle physics: ATLAS, Belle II, COMPASS/AMBER, Alice, LHCb, . . .
  - hadron physics
  - detector development
  - photonics
  - theory groups
  - economics

Extremely diverse requirements on software environments & job resources.

since 2017: **HTCondor** with **interactive-first** concept

‘Interactive First’

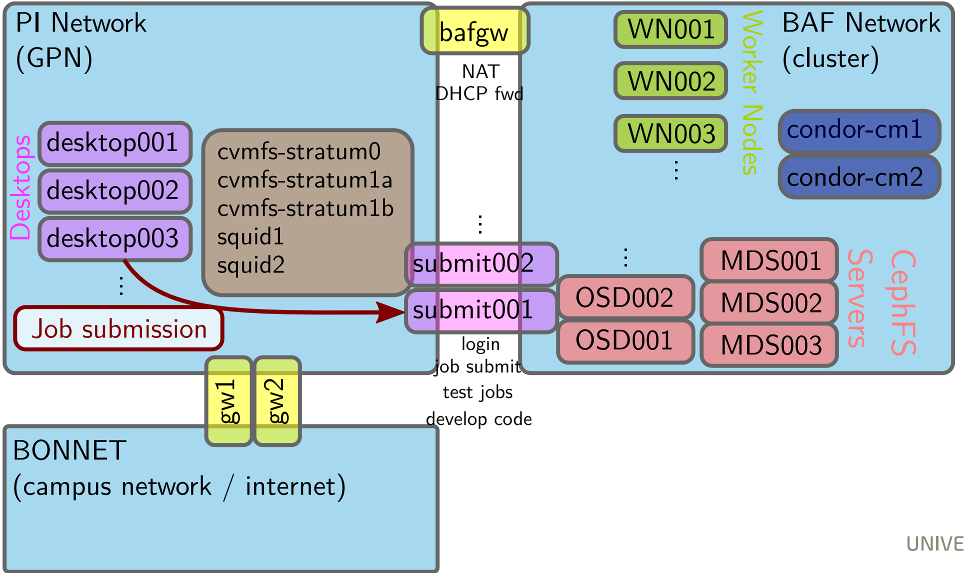
- Scientific software tends to require more and more dependencies (user-defined software stacks)
  ⇒ often via containers, Python environments, CVMFS trees,. . .
- Users do not want to hassle with the setup on their desktop, on which they prefer to use a modern OS
  ⇒ Decent versions of IDEs, graphics editors, browsers etc.
- Goals:
  - Offer a way to `SSH` or ‘browse’ into the required environment
  - This environment should be the same as the batch environment
  - The admins (we) must be happy to operate it

Solutions

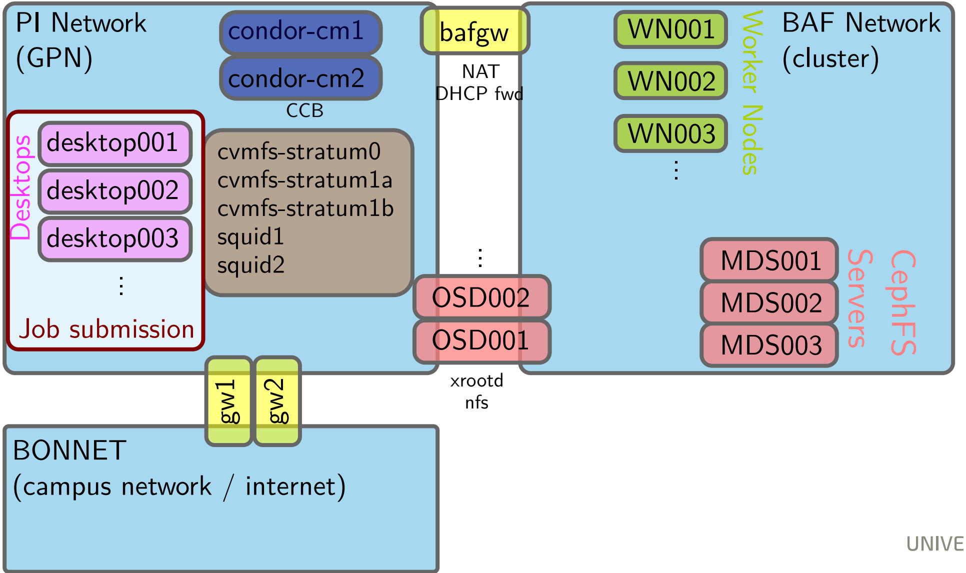
Two parts of this talk:

- `SSH` into containers on Batch resources
- JupyterHub on Batch resources

Classical Cluster Setup (until 2017)



Our setup: ‘Submit Locally, Run Globally’



Key changes in our new setup (since 2017)

- All desktops, worker nodes, condor central managers fully puppetized, for HTCondor: HEP-Puppet/htcondor (→HEPiX Autumn 2019)
  can set up queue super-users, block users from submission, set up for Apptainer,. . .
- **No login / submission nodes**:
  'use your desktop' or a 'rack-mounted desktop' from remote / mobile systems
- Condor central managers in desktop network
- Desktops running Debian 11 (ongoing migration to Debian 12)
- Cluster nodes running RockyLinux 8
- Full containerization (all user jobs run in containers)
- Containerization decouples OS upgrades from user jobs
- Cluster file system (CephFS) directly accessible from Desktop machines via NFS for access to results (→HEPiX Autumn 2019)
- Different connectivity for worker nodes: Partially InfiniBand FDR (56 Gbit/s), partially via 10 Gbit/s ethernet, partially (different location) via 1 Gbit/s ethernet

HTCondor Configuration

- Authentication via Kerberos / LDAP
  - Issues with ticket lifetime don't hit us heavily — automatic prolongation by `sssd` (up to one week)
- Node health script
  - prevent blackholing
  - critical for interactive use (responsiveness)
  - considering a `HEALTHY_FOR_INTERACTIVE_USE` flag
- Automated reboots: Draining with backfilling (uptime over 30 days, security updates . . .)
  - Fraction of 'interactivve' resources always available
  - Ensure responsiveness

Choice of Container Runtime

Requirements

- Aiming for unprivileged lightweight runtime
- Needs working HTCondor support including interactive jobs
- Allow image distribution via CernVM FS

CernVM FS

- Read-only file system with aggressive caching and deduplication
- Ideal for many small files and high duplication factor
- Perfect match for unpacked containers
- ‘Unpacked’ is (mostly) a requirement for rootless operation

⇒ Settled on Apptainer for now, but wishing for support for off-the-shelf solutions such as `Podman` / `runc`.

Apptainer (fork of Singularity)

APPTAINER

- Supports privileged and unprivileged operation
- Linux Foundation project, optimized for HPC use: https://apptainer.org/
- Process and file isolation, optional network isolation (no kernel isolation)
- Commonly used in HEP community

However, compared to competing solutions. . .

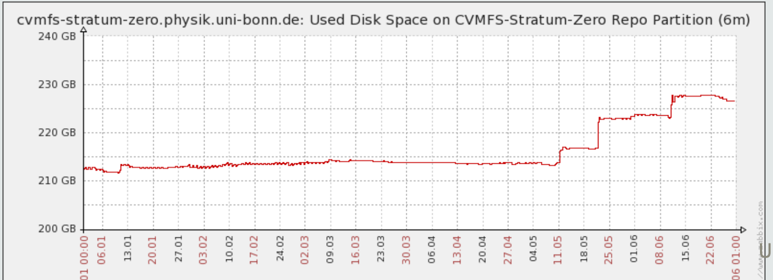
- Non-negligible rate of CVEs and breakage from new / changed functionality
- Not admin-friendly (e. g. breaking CentOS 7 support by default before its EoL)
- No **distro** packaging for Debian / RHEL, hence no LTS packaging
- Bundling of dependencies makes it hard / impossible for distros to keep packages secure: https://blogs.gentoo.org/mgorny/2021/02/23/why-not-rely-on-app-developer-to-handle-security/
- Reimplements existing standards (e. g. no build from `Dockerfile`)

⇒ **Use it, but avoid a lock-in as far as possible.**

Container Build Workflow

- All containers based on official DockerHub base images
- Ubuntu 20.04, Debian 11 / 12, RockyLinux 8 / 9
- Rebuilt at least daily with Apptainer recipe (site-specifics)
- Deployed to our own CVMFS, kept there for at least 30 days after build
- Unpacked images also work with other runtimes (only site-specifics in Singularity / Apptainer recipes slightly builder-dependent)

CVMFS usage over half a year, Containers (daily) & Software



Container Site-Specifics

- Compatibility with HEP experiments' requirements (HEP_OSlibs, ALRB)
- User data directory in environment variable, quota check tool
- DBUS hacks for X11 applications in containers
- More X11 hacks:
  https://htcondor-wiki.cs.wisc.edu/index.cgi/wiki?p=SingularityCondor
- HTCondor resource requests (login message, environment)
- lmod environment modules integration:

```
module load mathematica/14.0.0
```

- Source user-defined `.bashrc`, potentially OS-specific, from shared file system
- Necessary hacks for CUDA / GPU support
- OpenMPI without HTCondor inside containers (via HTChirp)
- Allow users to relay mail
- Timezone setup
- Add packages requested by users

HTCondor Integration

- All jobs forced into Singularity / Apptainer:

```
SINGULARITY_JOB = true
```

- Users can select from pre-build containers (‘choose your OS’)

```
CHOSEN_IMAGE = "$(ROCKY9_DEFAULT_IMAGE)"
CHOSEN_IMAGE = ifThenElse(TARGET.ContainerOS is "Debian11",
↪   "$(DEBIAN11_DEFAULT_IMAGE)", $(CHOSEN_IMAGE))
CHOSEN_IMAGE = ifThenElse(TARGET.ContainerOS is "Debian12",
↪   "$(DEBIAN12_DEFAULT_IMAGE)", $(CHOSEN_IMAGE))
CHOSEN_IMAGE = ifThenElse(TARGET.ContainerOS is "Rocky8",
↪   "$(ROCKY8_DEFAULT_IMAGE)", $(CHOSEN_IMAGE))
SINGULARITY_IMAGE_EXPR = $(CHOSEN_IMAGE)
```

- Paths to most recent image per OS and available OSes provided by `include command : someScript.sh`

‘Choose your OS’

- Users add to their Job ClassAd:

```
+ContainerOS = "Rocky9"
```

- Their jobs run in a container
- Same for interactive jobs (‘login-node experience’!)
- Small fractions of worker nodes exclusively for interactive jobs
  But: Interactive jobs can go to any slot!
- Resource-request specific tuning via `/etc/profile` possible:

```
REQUEST_CPUS=$(awk '/^RequestCpus/{print $3}' ${_CONDOR_JOB_AD})
export NUMEXPR_NUM_THREADS=${REQUEST_CPUS}
export MKL_NUM_THREADS=${REQUEST_CPUS}
export OMP_NUM_THREADS=${REQUEST_CPUS}
export CUBACORES=${REQUEST_CPUS}
export JULIA_NUM_THREADS=${REQUEST_CPUS}
```

⇒ Part of HTCondor 8.9.4 and later! *(see #7296)*, already extended with more flags.

Remaining issues in 9.0. . . (we should upgrade!)

- Difference between batch and interactive (`source /etc/profile` needed in batch)
  (may be worked around with a job wrapper launching a login shell)
- Interactive jobs are not yet contained within cgroups.
- Need some obscure extra bind mounts:

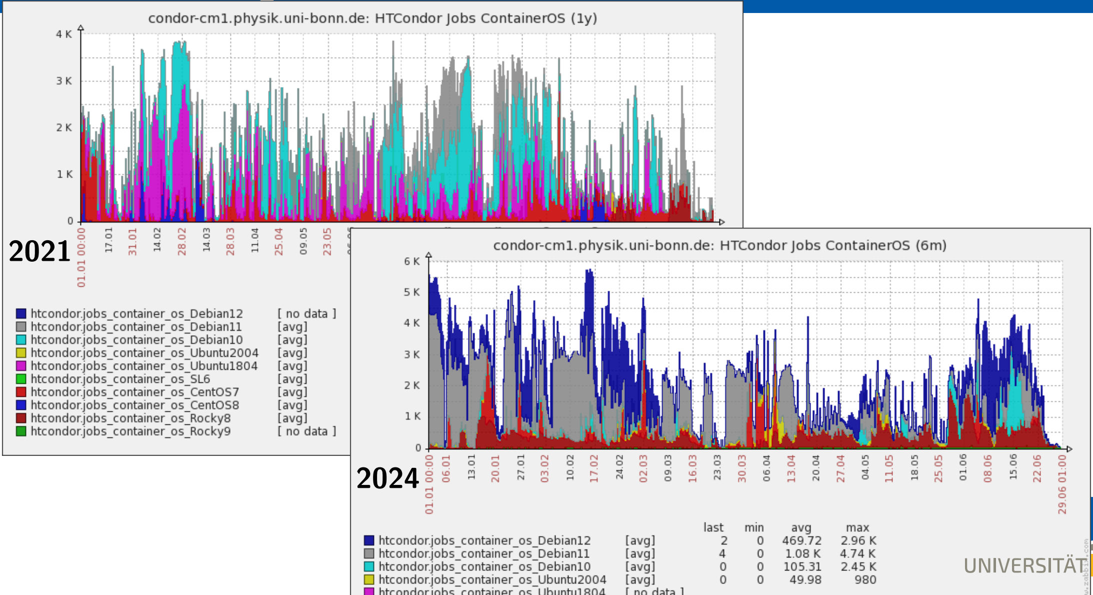
```
SINGULARITY_BIND_EXPR =
↪  "/pool,/usr/libexec/condor/,/cephfs,/cvmfs,/dev/infiniband"
```

⇒ Need to include `EXECUTE` directory (`/pool`) and `/usr/libexec/condor` here!

However. . .

- We have been running with this for over seven years now.
- Users are delighted by the choices, and `ssh -X` effectively works!
- We must upgrade to a newer HTCondor release which will bring improvements.

Container Usage



2021

condor-cm1.physik.uni-bonn.de: HTCondor Jobs ContainerOS (1y)

■ htcondor.jobs_container_os_Debian12	[no data]
■ htcondor.jobs_container_os_Debian11	[avg]
■ htcondor.jobs_container_os_Debian10	[avg]
■ htcondor.jobs_container_os_Ubuntu2004	[avg]
■ htcondor.jobs_container_os_Ubuntu1804	[avg]
■ htcondor.jobs_container_os_SL6	[avg]
■ htcondor.jobs_container_os_CentOS7	[avg]
■ htcondor.jobs_container_os_CentOS8	[avg]
■ htcondor.jobs_container_os_Rocky8	[avg]
■ htcondor.jobs_container_os_Rocky9	[no data]

2024

condor-cm1.physik.uni-bonn.de: HTCondor Jobs ContainerOS (6m)

		last	min	avg	max
■ htcondor.jobs_container_os_Debian12	[avg]	2	0	469.72	2.96 K
■ htcondor.jobs_container_os_Debian11	[avg]	4	0	1.08 K	4.74 K
■ htcondor.jobs_container_os_Debian10	[avg]	0	0	105.31	2.45 K
■ htcondor.jobs_container_os_Ubuntu2004	[avg]	0	0	49.98	980
■ htcondor.jobs_container_os_Ubuntu1804	[no data]				

Container Usage: Well accepted!

Instead of `ssh` to a login node, users run:

```
freyermu@exp199:~$ condor_submit -interactive -append '+ContainerOS="Rocky9"'
Submitting job(s).
1 job(s) submitted to cluster 15.
/usr/bin/xauth:  file /pool/condor/dir_489494/.Xauthority does not exist
Welcome to sloti_2_1@wn003.baf.physik.uni-bonn.de!
Your condor job is running with pid(s) 489575.
You requested 2 core(s), 2000 MB RAM, 1024000 kB disk space.
freyermu@wn003(Rocky9) ~ $
```

Well accepted by users!

Things fresh users tend to stumble upon

- Sometimes, new users try to run CentOS 7 code on RockyLinux 8 or similar. . . (legacy, ‘inherited’ job scripts or instructions)
- Workflow without shared file system between Access Point and Execution Point:
  - If offered, does invite to mis-use as ‘home directory’
  - Need to understand file transfer possibilities
  - Access to resources (Git etc.) possible in interactive jobs
- No ‘good’ way to run an IDE in the same environment (but similar for login nodes). ⇒ Remote editing via `condor_ssh_to_job`, i. e. ‘edit locally, execute globally’? There are some nitty-gritty details here, wrapper-script for VS Code upcoming!

Expectation

VS Code remote editing will hopefully fix some of those woes (and create others?), also there is JupyterHub. . .

Why JupyterHub?

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

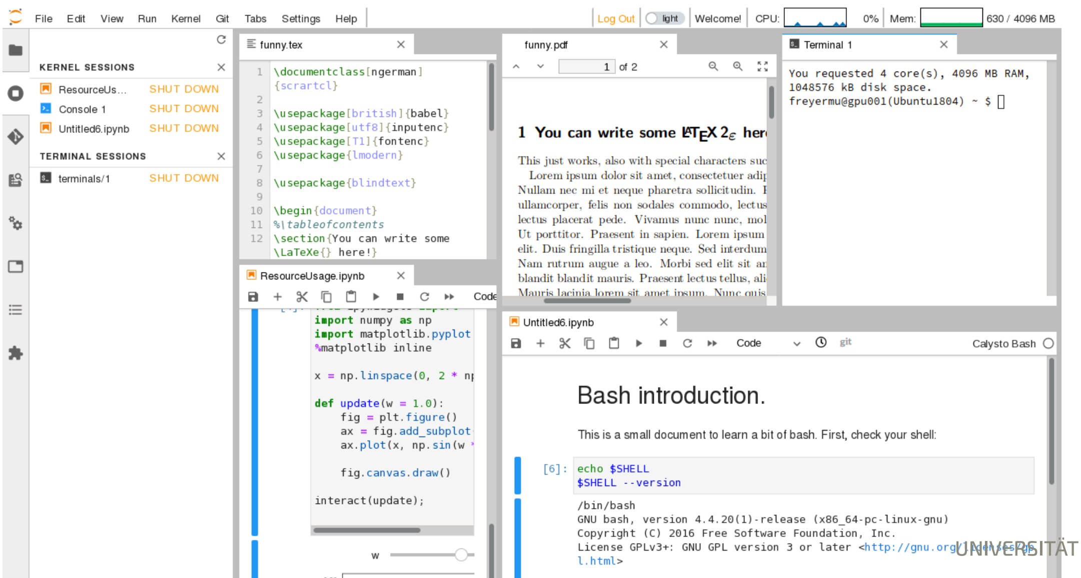
In summary. . .

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

Usual 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



```
\documentclass[ngerman]
{scrartcl}

\usepackage[british]{babel}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{lmodern}

\usepackage{blindtext}

\begin{document}
%\tableofcontents
\section{You can write some
\LaTeXe{} here!}
```

1 You can write some LaTeX 2_ε her

This just works, also with special characters suc
Lorem ipsum dolor sit amet, consectetuer adip
Nullam nec mi et neque pharetra sollicitudin. P
ullamcorper, felis non sodales commodo, lectus
lectus placerat pede. Vivamus nunc nunc, mol
Ut porttitor. Praesent in sapien. Lorem ipsum
elit. Duis fringilla tristique neque. Sed interdum
Nam rutrum augue a leo. Morbi sed elit sit an
blandit blandit mauris. Praesent lectus tellus, ali
Mauris lacinia lorem sit amet ipsum. Nunc quis

```
You requested 4 core(s), 4096 MB RAM,
1048576 kB disk space.
freyermu@gpu001(Ubuntu1804) ~ $
```

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

x = np.linspace(0, 2 * np

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

    fig.canvas.draw()

interact(update);
```

Bash introduction.

This is a small document to learn a bit of bash. First, check your shell:

```
[6]: echo $SHELL
     $SHELL --version
```

```
/bin/bash
GNU bash, version 4.4.20(1)-release (x86_64-pc-linux-gnu)
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gp
l.html>
```

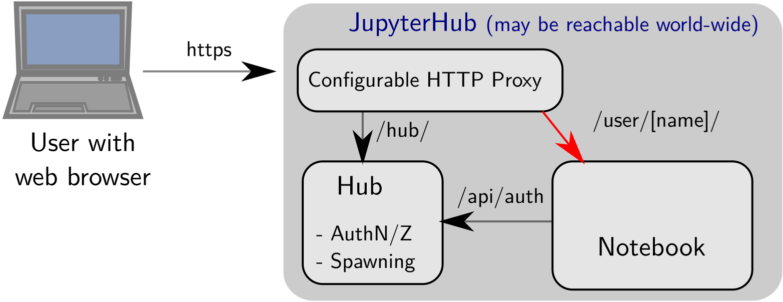
Operational hurdles

- 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)

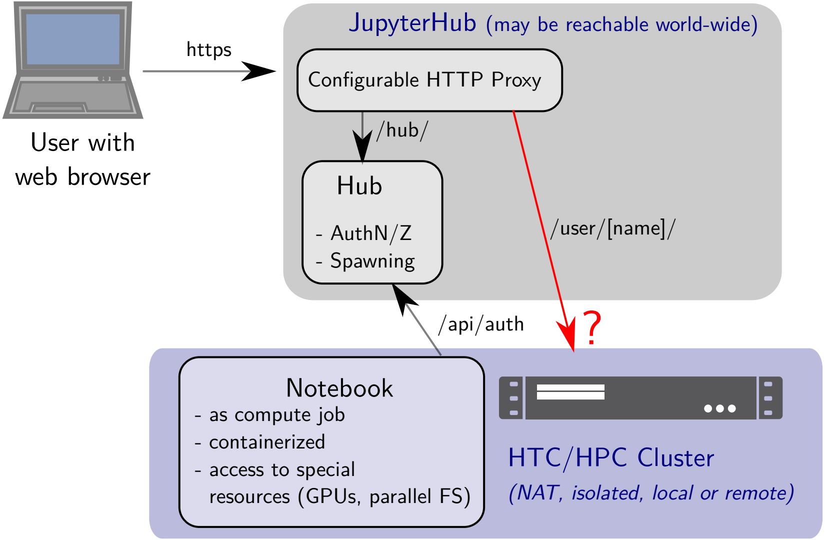
Need to overcome networking issue for use in a split desktop / cluster network

Let us investigate JupyterHub networking!
(if we find a workaround, this will also allow to scale out!)

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)

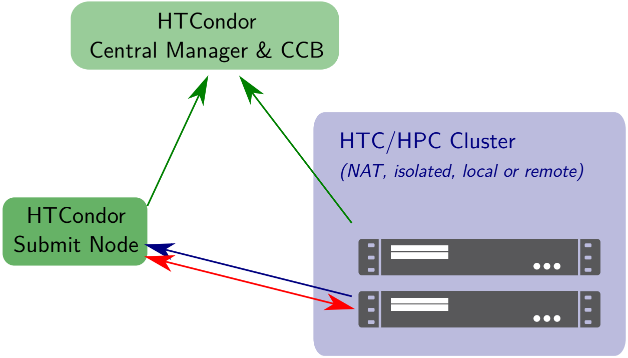
If somebody takes over your web service...

...the attacker may have direct access to your cluster network!

Can we overcome this issue?

Can HTCondor help out?

Networking with HTCondor (simplified)



→ 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

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?

Tested and confirmed.

For JupyterHub integration: Batch spawner needs to be extended.

(mote details in the Backup slides)

JupyterHub in Production

How to use the implementation?

- Full implementation in this pull request (awaiting review): `https://github.com/jupyterhub/batchspawner/pull/200`
- For maximum profit, HTCondor setup with CCB and shared port configuration required
- Can also be adapted for other batch systems and environments

JupyterHub in production since 2021...

- 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

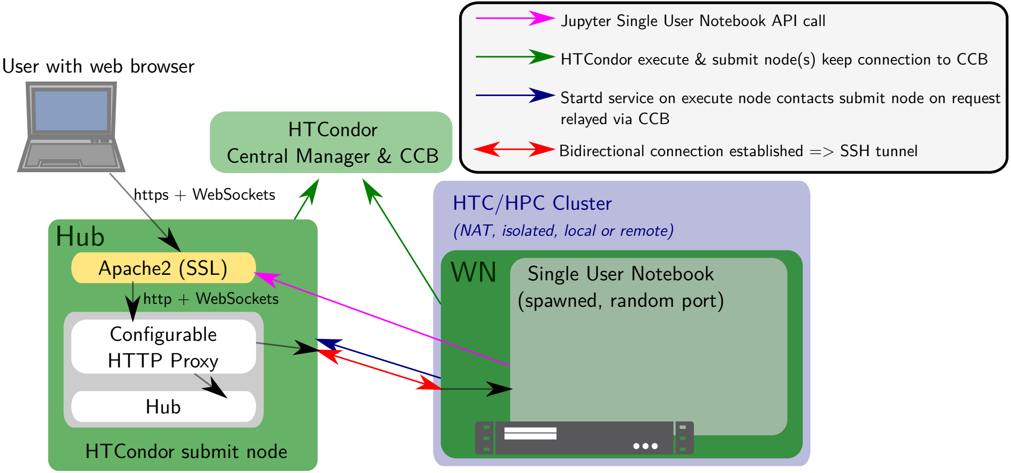
- Login to the hub creates a Kerberos TGT (via PAM)
- Kerberos used for job submission (and inter-daemon communication with HTCondor)

File system?

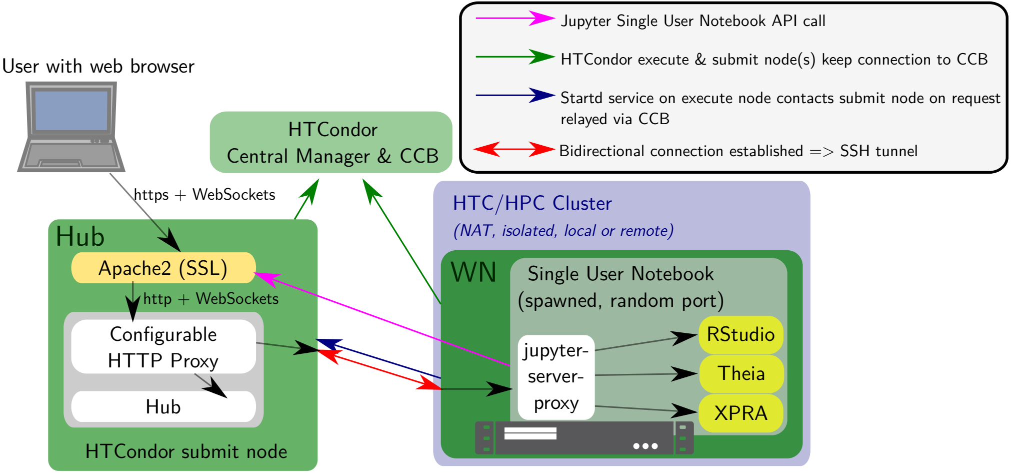
Currently, use HTCondor file transfer: transfer a ~/jupyter directory into the job and back when job exits:

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

Overall schematic


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
User with web browser
https + WebSockets
HTCondor
Central Manager & CCB
Hub
Apache2 (SSL)
http + WebSockets
Configurable
HTTP Proxy
Hub
HTCondor submit node
HTC/HPC Cluster
(NAT, isolated, local or remote)
WN
Single User Notebook
(spawned, random port)

Overall schematic



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

User with web browser

https + WebSockets

HTCondor Central Manager & CCB

Hub

Apache2 (SSL)

http + WebSockets

Configurable HTTP Proxy

Hub

HTCondor submit node

HTC/HPC Cluster

(NAT, isolated, local or remote)

WN

Single User Notebook (spawned, random port)

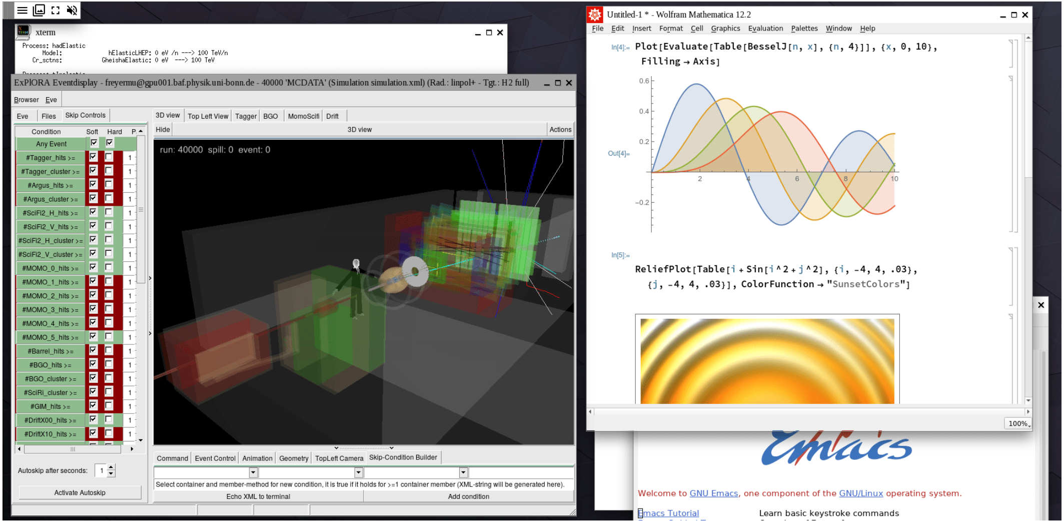
jupyter-server-proxy

RStudio

Theia

XPRA

Some impressions: X11 applications in your browser



xterm

```
Process: hadElastic
      Model:              hElasticLHEP: 0 eV /n ---> 100 TeV/n
    Cr_sctns:            GheishaElastic: 0 eV  ---> 100 TeV
```

ExPlORA Eventdisplay - freyermu@gpu001.baf.physik.uni-bonn.de - 40000 'MCDATA' (Simulation simulation.xml) (Rad.: linpol+ - Tgt.: H2 full)

run: 40000 spill: 0 event: 0

Select container and member-method for new condition, it is true if it holds for >=1 container member (XML-string will be generated here).

Autoskip after seconds: 1

Untitled-1 * - Wolfram Mathematica 12.2

```
In[4]:= Plot[Evaluate[Table[BesselJ[n, x], {n, 4}]], {x, 0, 10},
         Filling → Axis]
Out[4]=
```

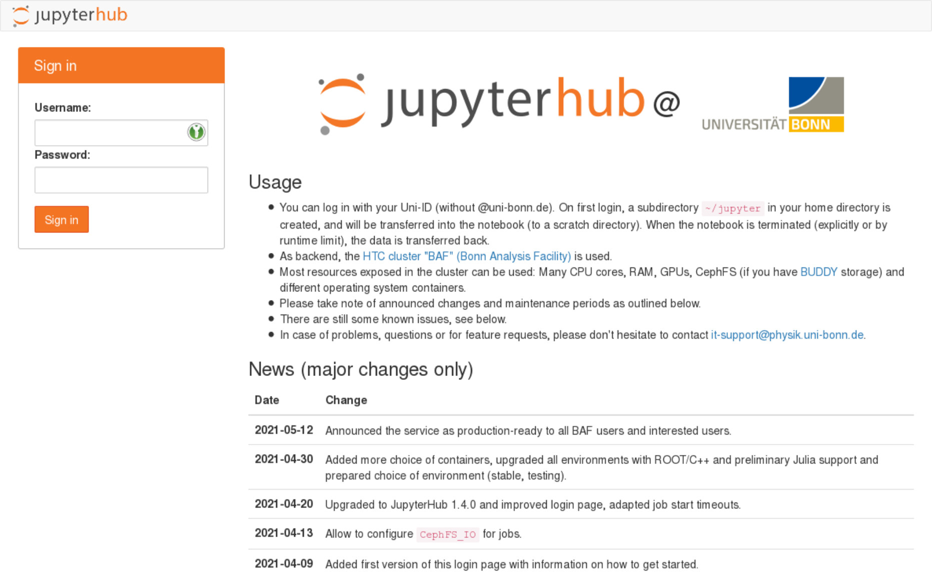
```
In[5]:=
        ReliefPlot[Table[i + Sin[i^2 + j^2], {i, -4, 4, .03},
          {j, -4, 4, .03}], ColorFunction → "SunsetColors"]
```

Welcome to GNU Emacs, one component of the GNU/Linux operating system.

Emacs Tutorial   Learn basic keystroke commands

UNIVERSITÄT BONN

Some impressions: Customized login page



Sign in

Username:

Password:

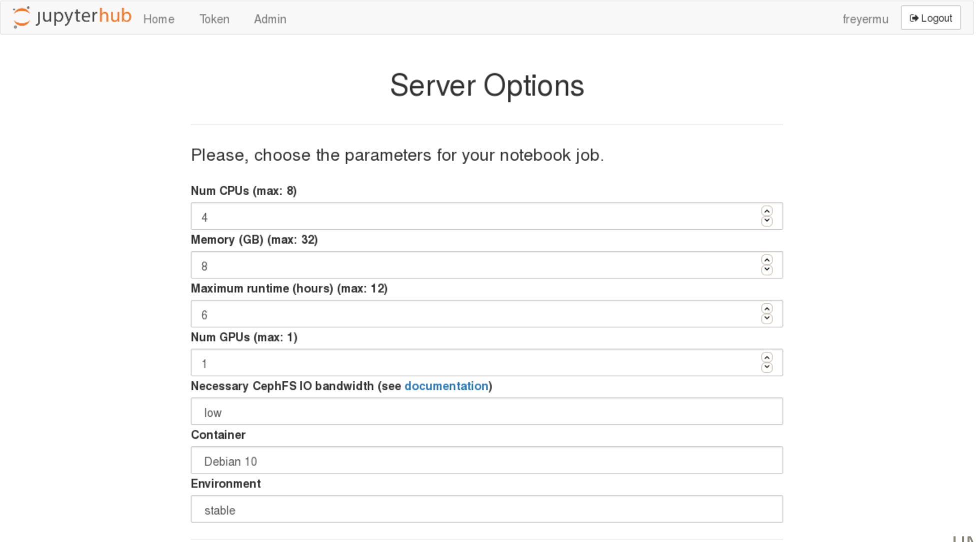
Usage

- You can log in with your Uni-ID (without @uni-bonn.de). On first login, a subdirectory `~/jupyter` in your home directory is created, and will be transferred into the notebook (to a scratch directory). When the notebook is terminated (explicitly or by runtime limit), the data is transferred back.
- As backend, the HTC cluster "BAF" (Bonn Analysis Facility) is used.
- Most resources exposed in the cluster can be used: Many CPU cores, RAM, GPUs, CephFS (if you have BUDDY storage) and different operating system containers.
- Please take note of announced changes and maintenance periods as outlined below.
- There are still some known issues, see below.
- In case of problems, questions or for feature requests, please don't hesitate to contact it-support@physik.uni-bonn.de.

News (major changes only)

Date	Change
2021-05-12	Announced the service as production-ready to all BAF users and interested users.
2021-04-30	Added more choice of containers, upgraded all environments with ROOT/C++ and preliminary Julia support and prepared choice of environment (stable, testing).
2021-04-20	Upgraded to JupyterHub 1.4.0 and improved login page, adapted job start timeouts.
2021-04-13	Allow to configure `CephFS_IO` for jobs.
2021-04-09	Added first version of this login page with information on how to get started.

Some impressions: Customized FormSpawner



Server Options

Please, choose the parameters for your notebook job.

Num CPUs (max: 8): 4

Memory (GB) (max: 32): 8

Maximum runtime (hours) (max: 12): 6

Num GPUs (max: 1): 1

Necessary CephFS IO bandwidth (see documentation): low

Container: Debian 10

Environment: stable

Experiences and Outlook for JupyterHub

Experience

- Hub environment maintenance (naturally) remains a hassle
- Scientific use at our site remains low (less than 500 notebook sessions in 3 years)
  - It seems notebooks are often used for plotting, which are then run locally.
  - Note We have a centrally operated Hub for teaching purposes.
- Significant interest for use in research platform projects:
  - 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

Oliver Freyermuth, Katrin Kohl, Peter Wienemann

Unleashing JupyterHub:
Exploiting resources without inbound network connectivity using HTCondor
Computing and Software for Big Science 5, 24 (2021)

Conclusions

- 'Interactive first' approach (with containers) works very well for us!
- Getting rid of login nodes solved a lot of issues and headaches
- Containers with different software environments well-accepted and heavily used
- JupyterHub can scale to federated infrastructures (thanks to HTCondor)

Thank you!

All of this also works in a federated environment / with opportunistic resources

Tools to auto-scale worker nodes which register into an overlay batch system:

- COBalD — the Opportunistic Balancing Daemon
- TARDIS — The Transparent Adaptive Resource Dynamic Integration System

developed by KIT, used e. g. to run WLCG jobs, in a federated research platform. . .

Thank you

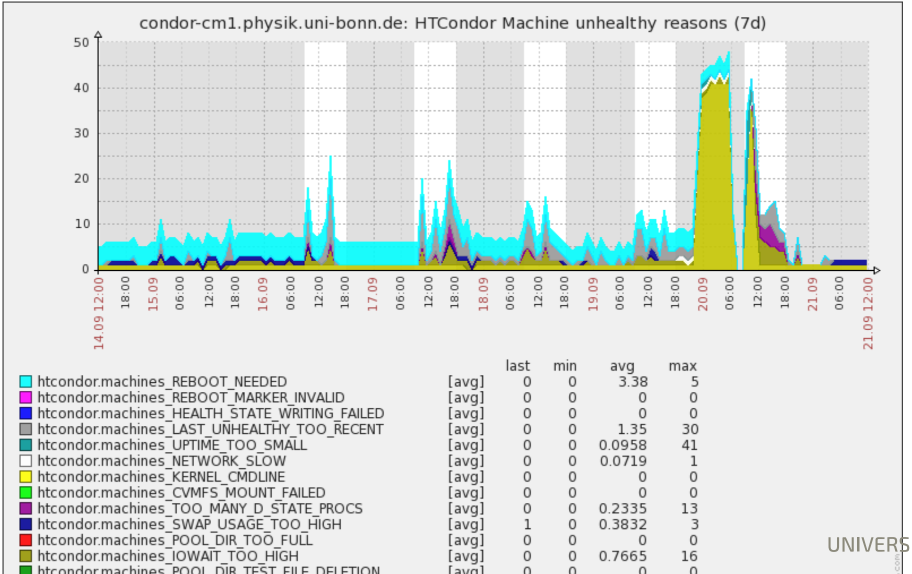
for your attention!

☕

Health Checking

- Node health script (critical for interactive use and to prevent blackholing):
  - run via `STARTD_CRON`
  - can pick up admin-enforced state via Puppet *(e.g. for maintenance)*
  - picks up state from 'reboot-needed' cronjob
  - Captures common node overload issues:
    - Heavy I/O on local disks (`iowait`)
    - Heavy swapping (HTCondor cannot limit swap usage yet, in our version)
  - Critical to ensure jobs (especially interactive jobs) are routed to responsive nodes
  - Considering a `HEALTHY_FOR_INTERACTIVE_USE` flag

Node health checking

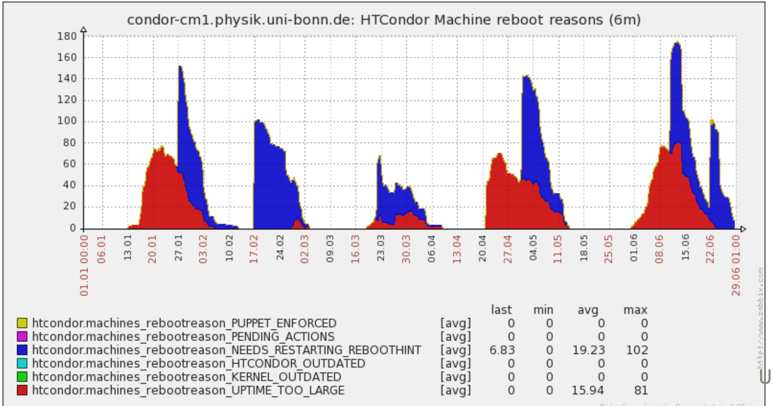


condor-cm1.physik.uni-bonn.de: HTCondor Machine unhealthy reasons (7d)

		last	min	avg	max
htcondor.machines_REBOOT_NEEDED	[avg]	0	0	3.38	5
htcondor.machines_REBOOT_MARKER_INVALID	[avg]	0	0	0	0
htcondor.machines_HEALTH_STATE_WRITING_FAILED	[avg]	0	0	0	0
htcondor.machines_LAST_UNHEALTHY_TOO_RECENT	[avg]	0	0	1.35	30
htcondor.machines_UPTIME_TOO_SMALL	[avg]	0	0	0.0958	41
htcondor.machines_NETWORK_SLOW	[avg]	0	0	0.0719	1
htcondor.machines_KERNEL_CMDLINE	[avg]	0	0	0	0
htcondor.machines_CVMFS_MOUNT_FAILED	[avg]	0	0	0	0
htcondor.machines_TOO_MANY_D_STATE_PROCS	[avg]	0	0	0.2335	13
htcondor.machines_SWAP_USAGE_TOO_HIGH	[avg]	1	0	0.3832	3
htcondor.machines_POOL_DIR_TOO_FULL	[avg]	0	0	0	0
htcondor.machines_IOWAIT_TOO_HIGH	[avg]	0	0	0.7665	16
htcondor.machines_POOL_DIR_TEST_FILE_DELETION	[avg]	0	0	0	0

Node reboot handling

- Detection mainly via `needs-restarting -r`
- Start of drain smeared out over 10 days
- While nodes are draining, still accept jobs running shorter than the longest job



		last	min	avg	max
🟨 htcondor.machines_rebootreason_PUPPET_ENFORCED	[avg]	0	0	0	0
🟪 htcondor.machines_rebootreason_PENDING_ACTIONS	[avg]	0	0	0	0
🟦 htcondor.machines_rebootreason_NEEDS_RESTARTING_REBOOTHINT	[avg]	6.83	0	19.23	102
▨ htcondor.machines_rebootreason_HTCONDOR_OUTDATED	[avg]	0	0	0	0
🟩 htcondor.machines_rebootreason_KERNEL_OUTDATED	[avg]	0	0	0	0
🟥 htcondor.machines_rebootreason_UPTIME_TOO_LARGE	[avg]	0	0	15.94	81

Behind the scenes: Entering containers via `nsenter`

- Enter the namespaces the container runtime has created
  ⇒ Essentially, 'attach' to the container!
- Compatible with *any* container runtime using namespaces (with potential quirks)
- Other container runtimes one could think of:
  - Charliecloud (`https://hpc.github.io/charliecloud/`)
    - Even more lightweight (no PID / network namespaces)
      PID namespace could be handled by HTCondor
    - Code is short and easily auditable
    - Fully unprivileged container build, `Dockerfile` support, extensive test suite
  - Podman / runc (`https://podman.io/`)
    - Included since RHEL 7.6 with official support, distro packages also for Debian
    - Can be used with `alias docker=podman`
    - Can run rootless
    - CRIU integration (freeze, live-migrate)
    - Still requires bind-mount target directories to exist for rootless (GitHub issue 1671)

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`

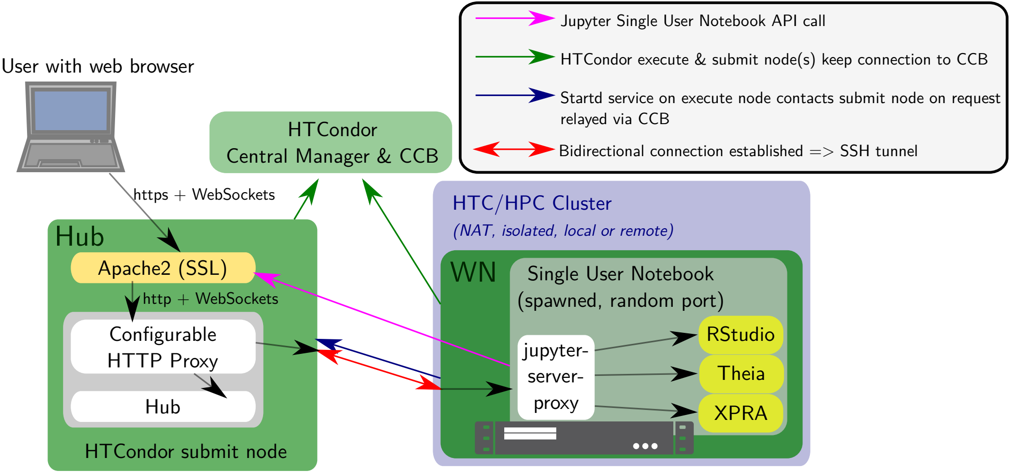
Other Web Services

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


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
User with web browser
https + WebSockets
HTCondor Central Manager & CCB
Hub
Apache2 (SSL)
http + WebSockets
Configurable HTTP Proxy
Hub
HTCondor submit node
HTC/HPC Cluster
(NAT, isolated, local or remote)
WN
Single User Notebook (spawned, random port)
jupyter-server-proxy
RStudio
Theia
XPRA