



INTERACTIVE HPC WITH JUPYTERLAB

EDIH (European-Data-Innovation-Hub) -Workshop, Part 2

2023-10-23 | JENS HENRIK GÖBBERT (J.GOEBBERT@FZ-JUELICH.DE)

CUSTOM JUPYTER KERNEL

JUPYTER KERNEL

How to create your own Jupyter Kernel

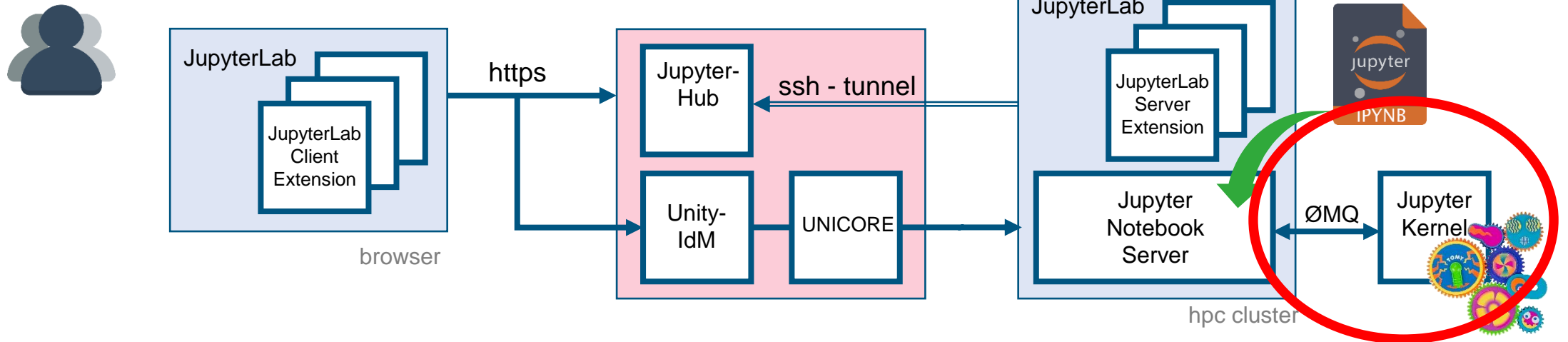
Jupyter Kernel

A “kernel” refers to the separate process

with

Ju

-
-
-
-



You can easily **create your own kernel** which for example runs your specialized virtual Python environment.

<https://github.com/jupyter/jupyter/wiki/Jupyter-kernels>

JUPYTER KERNEL

How to create your own Jupyter Kernel

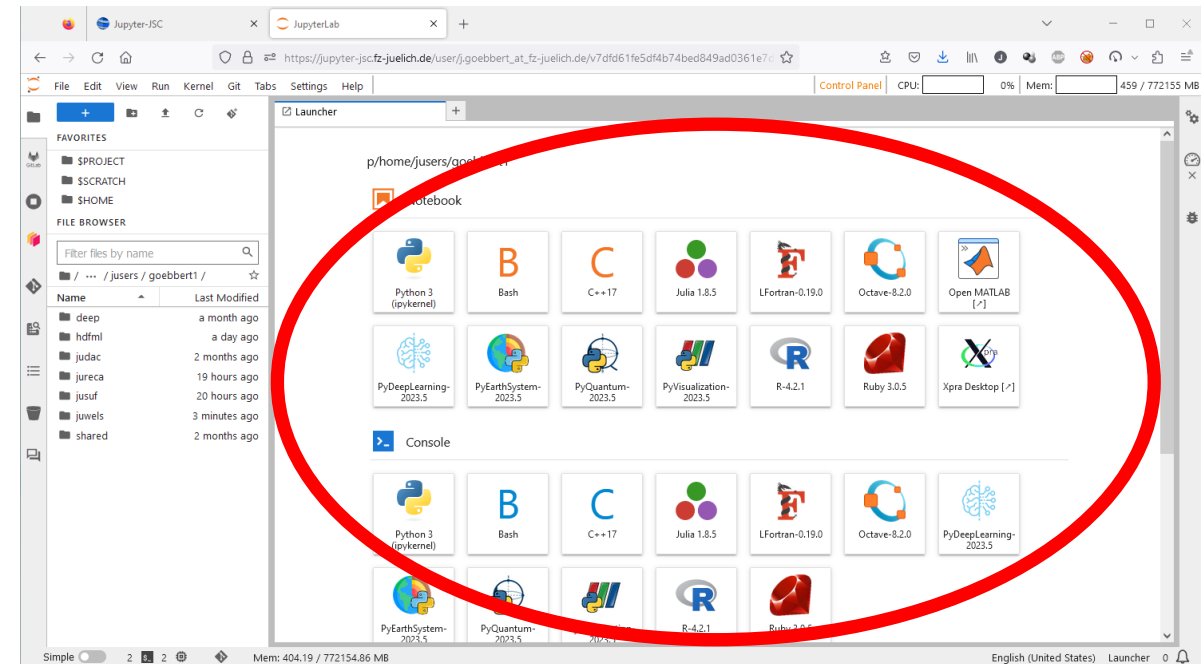
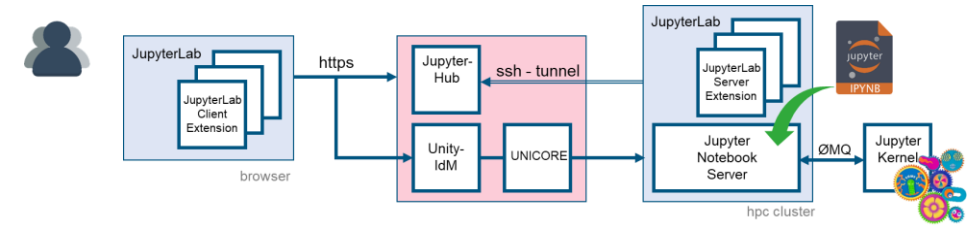
Jupyter Kernel

A “kernel” refers to the separate process which executes code cells within a Jupyter notebook.

Jupyter Kernel

- run code in different programming languages **and environments**.
- can be connected to a notebook (one at a time).
- communicates via ZeroMQ with the JupyterLab.
- Multiple **preinstalled** Jupyter Kernels can be found on our clusters
 - Python, R, Julia, Bash, C++, Ruby, JavaScript
 - Specialized kernels for visualization, quantum computing

You can easily **create your own kernel** which for example runs your specialized virtual Python environment.



<https://github.com/jupyter/jupyter/wiki/Jupyter-kernels>

JUPYTER KERNEL

How to create your own Jupyter Kernel

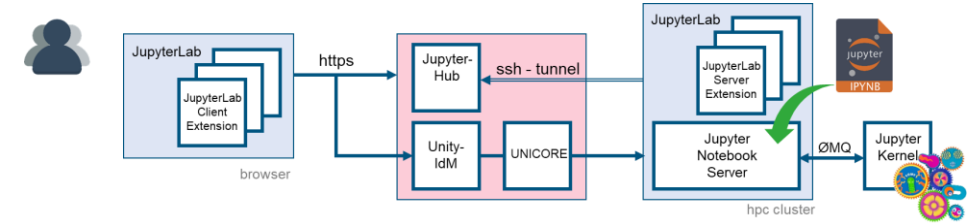
Jupyter Kernel

A “kernel” refers to the separate process which executes code cells within a Jupyter notebook.

Jupyter Kernel

- run code in different programming languages **and environments**.
- can be connected to a notebook (one at a time).
- communicates via ZeroMQ with the JupyterLab.
- Multiple **preinstalled** Jupyter Kernels can be found on our clusters
 - Python, R, Julia, Bash, C++, Ruby, JavaScript
 - Specialized kernels for visualization, quantum computing

You can easily **create your own kernel** which for example runs your specialized virtual Python environment.



Building your own Jupyter kernel is a three step process

1. Create/Pimp new **virtual Python environment**
`venv`
2. Create/Edit **launch script** for the Jupyter kernel
`kernel.sh`
3. Create/Edit Jupyter **kernel configuration**
`kernel.json`

<https://github.com/jupyter/jupyter/wiki/Jupyter-kernels>

SHORT DIGRESSION:

Lmod (Lua-based Modules) for managing environment modules

What is the problem Lmod solves?

- On a “normal” workstation software is provided in general on system level once. It is not required that any distinct shell can change fundamental settings.
- HPC systems need to support **multiple versions of software packages**
 - Compilers (e.g. gcc, icc, clang), libraries (e.g. MPI, HDF5), software (e.g. Python)
→ Lmod calls each of these “a **module**”

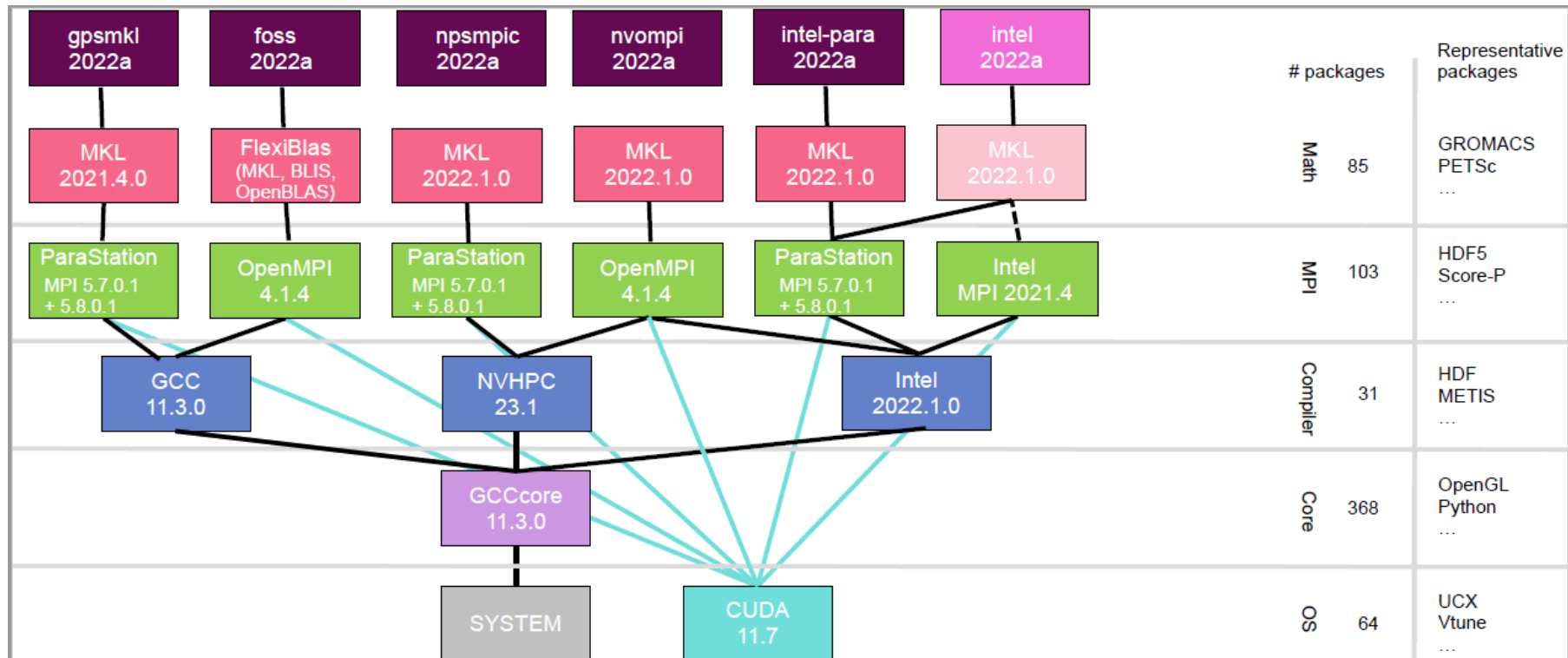
How does Lmod allow to switch between modules?

- Switching between modules is done by
 - Changing **environment variables** (most prominent PATH and LD_LIBRARY_PATH)
 - Ensuring that **dependencies** to other modules are fulfilled.
→ unload/load modules which conflict/required

SHORT DIGRESSION:

Lmod (Lua-based Modules) for managing environment modules

The module dependencies are organized in a dependency tree (one tree per stage)



Toolchain dependency tree used at Jülich Supercomputing Centre

SHORT DIGRESSION:

Virtual Python Environment

- **Isolation:**

- Self-contained and isolated environment for Python projects
- Allows to install and manage different versions of Python, libraries, and packages without interfering with other installations / configurations.

- **Reproducibility:**

- Recreate the environment in which your code was developed and tested, even on a different machine.

- **Consistency:**

- Ensures that same versions of Python and packages are used.
- Reduces the likelihood of compatibility issues and makes it easier to collaborate on a project.

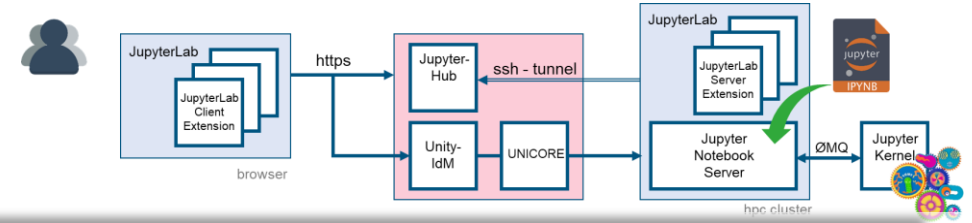
- **Flexibility:**

- Allows to easily switch between different versions of Python and packages.



JUPYTER KERNEL

How to create your own Jupyter Kernel

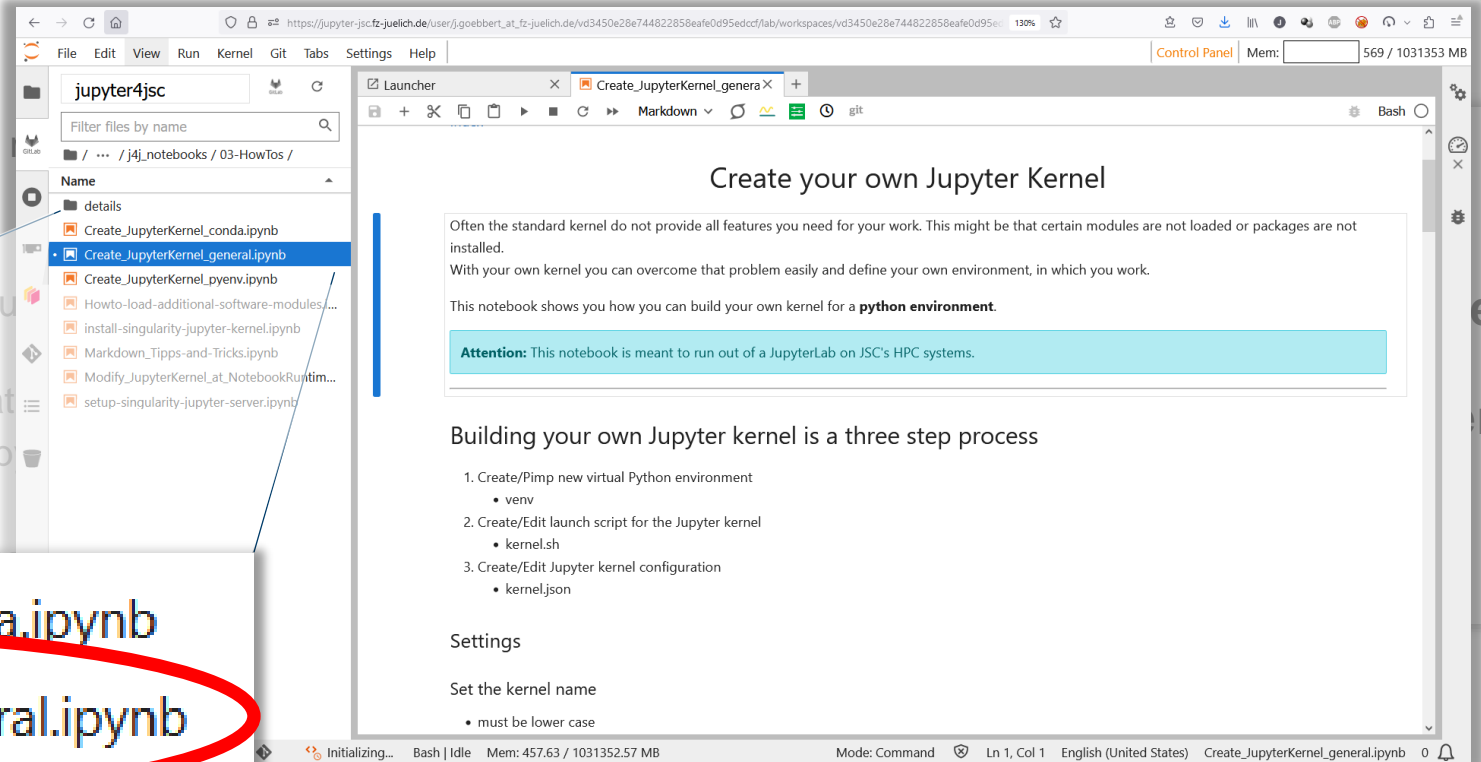


Jupyter Kernel

A “kernel” refers to the separate process which executes code cells within a Jupyter Notebook.

Jupyter Kernel

- run code in different programming languages and environments.
- can be connected to a notebook (one at a time)
- communicates via ZeroMQ with the Jupyter Notebook Server



- Create_JupyterKernel_conda.ipynb
- Create_JupyterKernel_general.ipynb
- Create_JupyterKernel_pyenv.ipynb

<https://github.com/jupyter/jupyter/wiki/Jupyter-kernels>

https://gitlab.version.fz-juelich.de/jupyter4jsc/j4j_notebooks/-/blob/master/001-Jupyter/Create_JupyterKernel_general.ipynb

JUPYTER KERNEL

1. Create/Pimp new virtual Python environment (1)

1. Login to JupyterLab and open terminal

2. Load required modules

```
Lnode:> module purge
Lnode:> module load Stages/2023
Lnode:> module load GCCcore/.11.3.0
Lnode:> module load Python/3.10.4
```

3. Load extra modules you need for your kernel

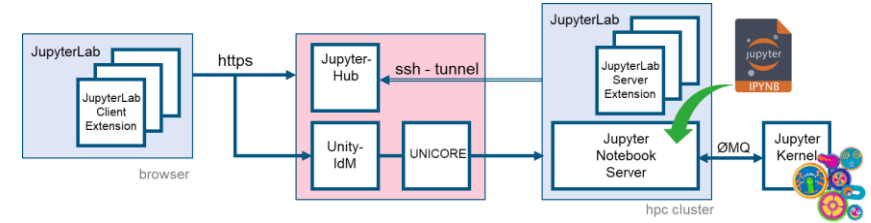
```
Lnode:> module load <module you need>
```

1. Create a virtual environment named <venv_name> at a path of your choice:

```
Lnode:> python -m venv --system-site-packages <your_path>/<venv_name>
```

2. Activate your environment

```
Lnode:> source <your_path>/<venv_name>/bin/activate
```



**Building your own Jupyter kernel
is a three step process**

1. Create/Pimp new virtual Python environment
venv

2. Create/Edit launch script for the Jupyter kernel
kernel.sh

3. Create/Edit Jupyter kernel configuration
kernel.json

JUPYTER KERNEL

1. Create/Pimp new virtual Python environment (2)

1. Ensure python packages installed in the virtual environment are always preferred

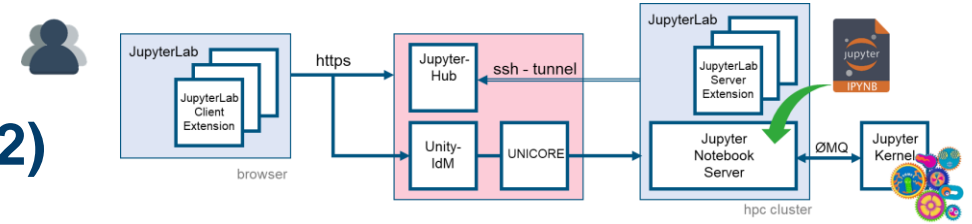
```
(<venv_name>) Lnode:> export PYTHONPATH=\n${VIRTUAL_ENV}/lib/python3.10/site-packages:${PYTHONPATH}
```

2. Install Python libraries required for communication with Jupyter

```
(<venv_name>) Lnode:>\n    pip install --ignore-installed ipykernel
```

3. Install whatever else you need in your Python virtual environment (using pip)

```
(<venv_name>) Lnode:>\n    pip install <python-package you need>
```



Building your own Jupyter kernel is a three step process

1. Create/Pimp new virtual Python environment
venv
2. Create/Edit **launch script** for the Jupyter kernel
kernel.sh
3. Create/Edit Jupyter **kernel configuration**
kernel.json

JUPYTER KERNEL

2. Create/Edit launch script for the Jupyter kernel (1)

1. Create launch script, which loads your Python virtual environment and starts the ipykernel process inside:

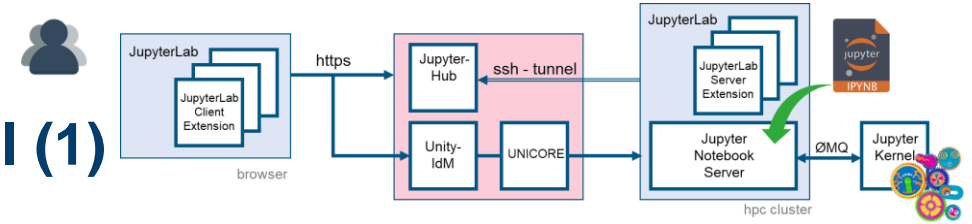
```
(<venv_name>) Lnode:> touch ${VIRTUAL_ENV}/kernel.sh
```

2. Make launch script executable

```
(<venv_name>) Lnode:> chmod +x ${VIRTUAL_ENV}/kernel.sh
```

3. Edit the launch script for your new Jupyter kernel

```
(<venv_name>) Lnode:> vi ${VIRTUAL_ENV}/kernel.sh
```

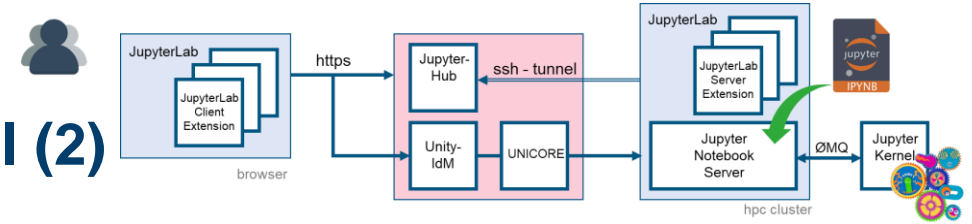


Building your own Jupyter kernel is a three step process

1. Create/Pimp new **virtual Python environment**
venv
2. Create/Edit **launch script** for the Jupyter kernel
kernel.sh
3. Create/Edit Jupyter **kernel configuration**
kernel.json

JUPYTER KERNEL

2. Create/Edit launch script for the Jupyter kernel (2)



```
#!/bin/bash
```

```
# Load required modules
```

```
module purge
```

```
module load Stages/2023
```

```
module load GCCcore/.11.3.0
```

```
module load Python/3.10.4
```

```
# Load extra modules you need for your kernel
```

```
#module load <module you need>
```

```
# Activate your Python virtual environment
```

```
source <your_path>/<venv_name>/bin/activate
```

```
# Ensure python packages installed in the virtual environment are always preferred
```

```
export PYTHONPATH=${VIRTUAL_ENV}/lib/python3.10/site-packages:${PYTHONPATH}
```

```
exec python -m ipykernel $@
```

**Building your own Jupyter kernel
is a three step process**

1. Create/Pimp new virtual Python environment

venv

2. Create/Edit launch script for the Jupyter kernel

kernel.sh

3. Create/Edit Jupyter kernel configuration

kernel.json

JUPYTER KERNEL

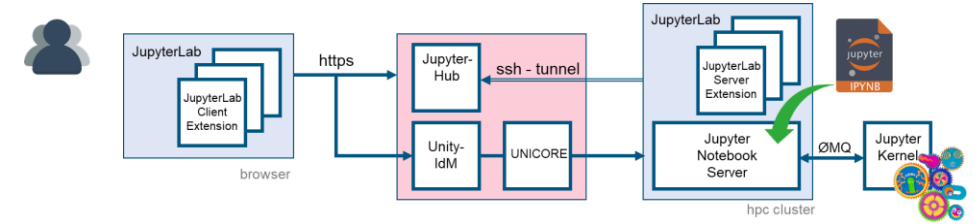
3. Create/Edit Jupyter kernel configuration (1)

1. Create your Jupyter kernel configuration files

```
(<venv_name>) Lnode:>
python -m ipykernel install --user --name=<my-kernel-name>
```

2. Update your kernel file to use the launch script

```
(<venv_name>) Lnode:>
vi ~/.local/share/jupyter/kernels/<my-kernel-name>/kernel.json
{
  "argv": [
    "<your_path>/<venv_name>/kernel.sh",
    "-m",
    "ipykernel_launcher",
    "-f",
    "{connection_file}"
  ],
  "display_name": "<my-kernel-name>",
  "language": "python"
}
```



Building your own Jupyter kernel is a three step process

1. Create/Pimp new **virtual Python environment**
venv
2. Create/Edit **launch script** for the Jupyter kernel
kernel.sh
3. Create/Edit Jupyter **kernel configuration**
kernel.json

JUPYTER KERNEL

Run your Jupyter kernel configuration

Run your Jupyter Kernel

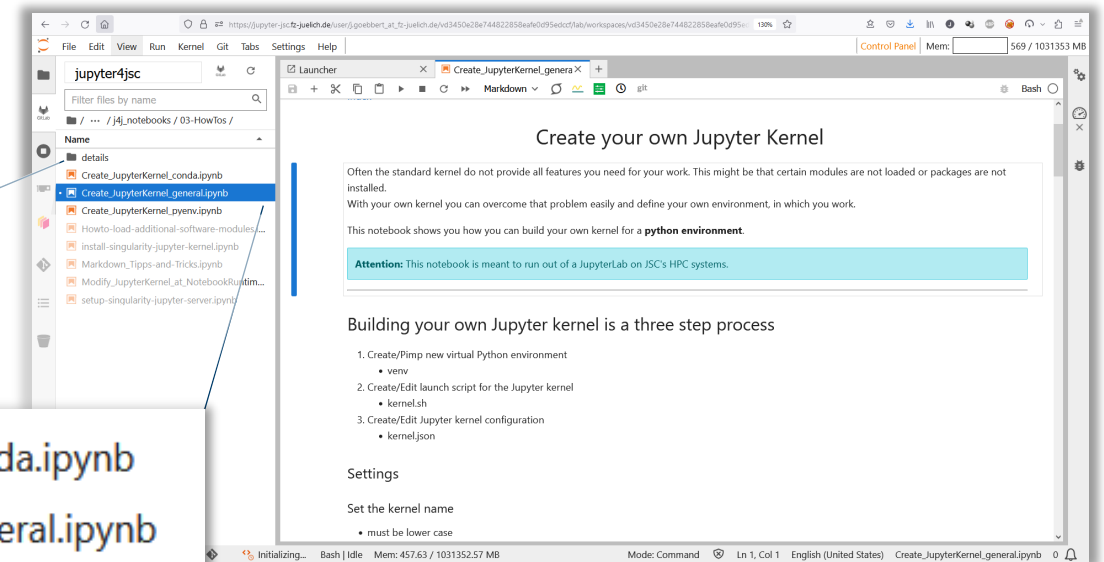
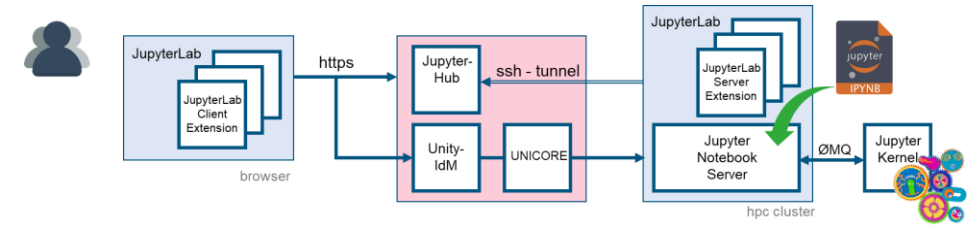
1. <https://jupyter-jsc.fz-juelich.de>
2. Choose system where your Jupyter kernel is installed in `~/ .local/share/jupyter/kernels`
3. Select your kernel in the launch pad or click the kernel name.

One of the many alternatives: Conda

Base your Jupyter Kernel on a Conda environment.

https://gitlab.version.fz-juelich.de/jupyter4jsc/j4j_notebooks/-/blob/master/001-Jupyter/Create_JupyterKernel_conda.ipynb

Create_JupyterKernel_conda.ipynb
Create_JupyterKernel_general.ipynb
Create_JupyterKernel_pyenv.ipynb



Jupyter kernel are **NOT limited** to Python at all!

The kernel-endpoint just needs to talk the Jupyter's kernel protocol (in general over ZeroMQ).

E.g.

- IRkernel for R (<https://github.com/IRkernel/IRkernel>)
- IJulia.jl (<https://github.com/JuliaLang/JuliaLang>)

SLURM WRAPPED KERNELS WITH SLURM-PROVISIONER

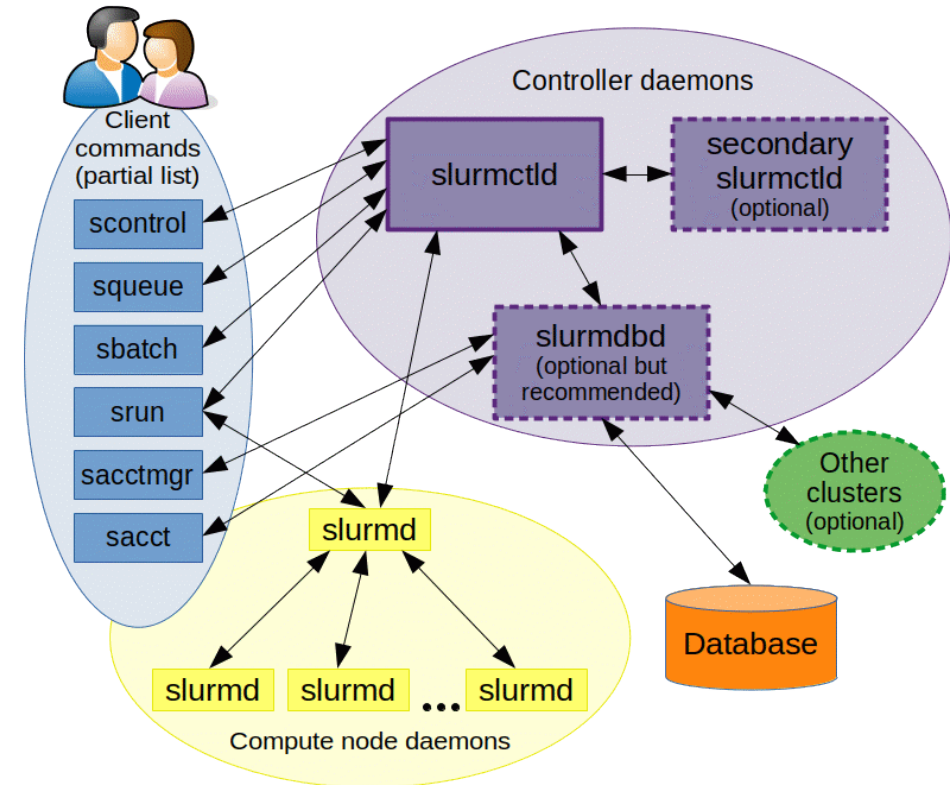
SHORT DIGRESSION:

Simple Linux Utility for Resource Management (SLURM)

Slurm is an

- open source,
- fault-tolerant, and
- highly scalable cluster management and
- job scheduling system

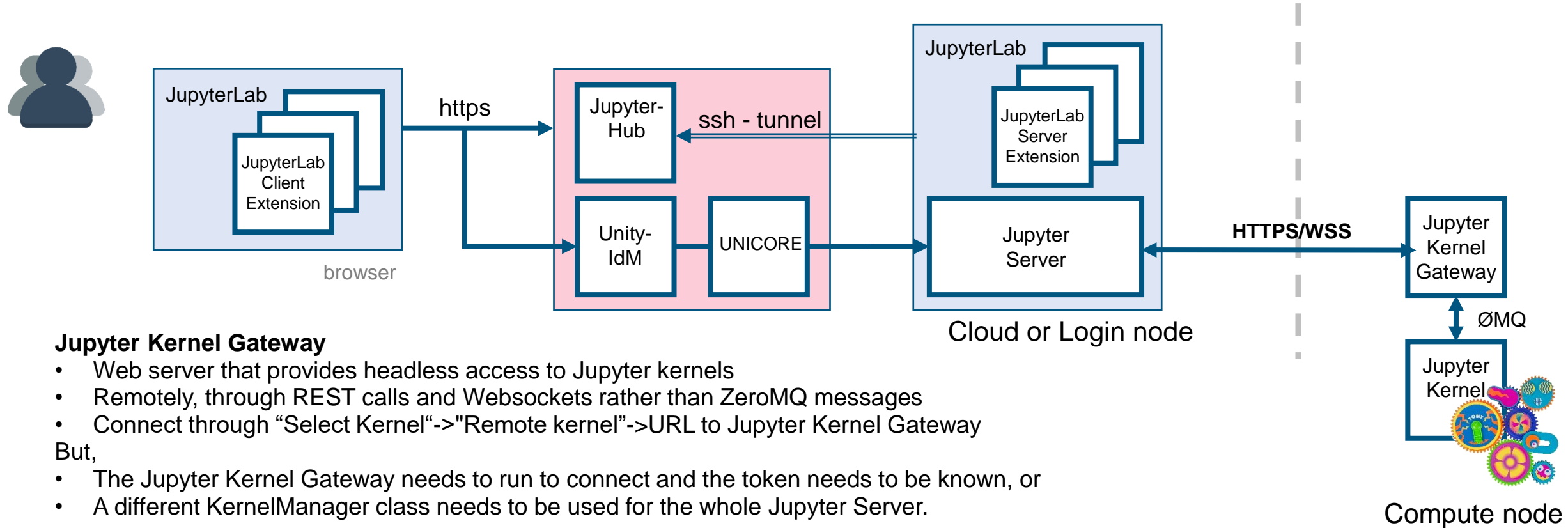
for large and small Linux clusters.



Source: <https://slurm.schedmd.com/overview.html>

REMOTE JUPYTER KERNELS

Running multiple Jupyter kernels separate on the HPC system



Jupyter Kernel Gateway

- Web server that provides headless access to Jupyter kernels
- Remotely, through REST calls and Websockets rather than ZeroMQ messages
- Connect through "Select Kernel" -> "Remote kernel" -> URL to Jupyter Kernel Gateway

But,

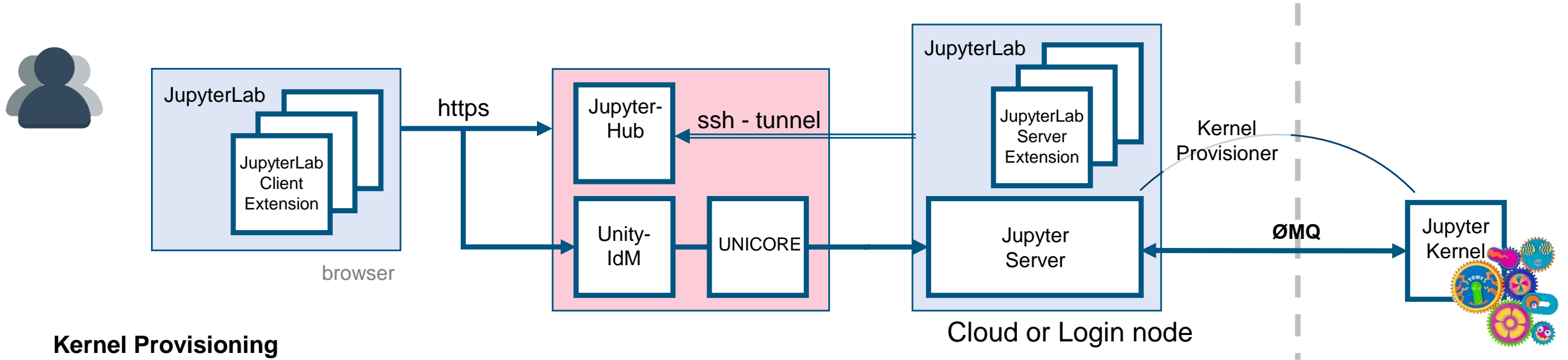
- The Jupyter Kernel Gateway needs to run to connect and the token needs to be known, or
- A different KernelManager class needs to be used for the whole Jupyter Server.

Both ways are non-intuitive and limit the user – especially as integration with the scheduler SLURM is missing.

Jupyter Enterprise Gateway is significantly richer in functionality, but a service users can connect to and primarily made for a cloud.

REMOTE JUPYTER KERNELS

Running multiple Jupyter kernels separate on the HPC system



Kernel Provisioning

Kernel Provisioning enables the ability for third parties to **manage the lifecycle of a kernel's runtime environment**.

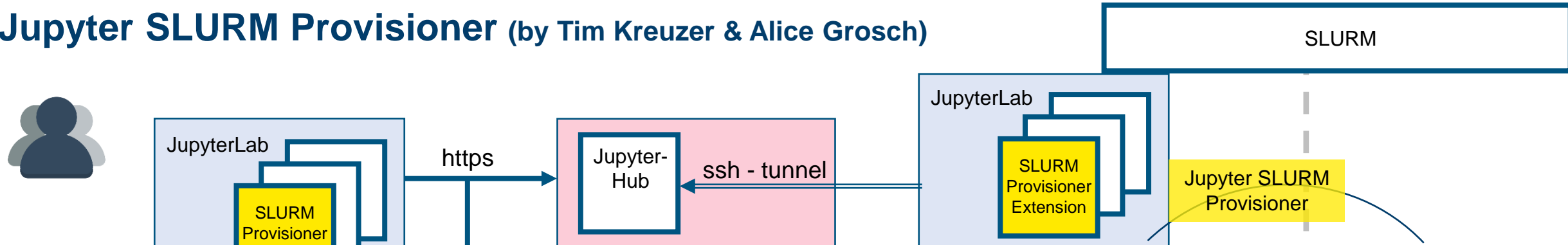
By implementing and configuring a *kernel provisioner*, third parties have the ability to **provision kernels for different environments**, typically managed by resource managers like Kubernetes, Hadoop YARN, Slurm, etc.

The kernel provisioner optionally extends the current **metadata stanza within the kernel.json** to include the specification of the kernel provisioner name, along with an optional config stanza

```
[..]
"metadata": {
  "kernel_provisioner": {
    "provisioner_name": "slurm-provisioner",
    "config": {
      "kernel_argv": "Python",
      "project": "zam",
      "partition": "batch",
      "nodes": 1,
      "runtime": 3600,
    }
  }
},
```

REMOTE JUPYTER KERNELS

Jupyter SLURM Provisioner (by Tim Kreuzer & Alice Grosch)



Slurm wrapped kernels allow you to run kernels on compute nodes while your Jupyter Server runs on a login node.

This has the advantage that when your allocation on the compute node(s) ends, **only the kernel is stopped**, but your JupyterLab server keeps running. You will only have to restart the kernel, not your entire JupyterLab instance.

The screenshot shows the JupyterLab interface for selecting a kernel. The 'Select allocation for slurm wrapper' dropdown is set to 'New'. The 'Select kernel for slurm wrapper' dropdown is set to 'Custom Python 3 (ipykernel)'. The 'Select project for slurm wrapper' dropdown is set to 'ccstvs'. The 'Select partition for slurm wrapper' dropdown is set to 'develgpu'. The 'Nodes [1-2]' dropdown is set to '1'. The 'GPUs [1-4]' dropdown is set to '4'. The 'Runtime (min) [10-120]' dropdown is set to '30'. The 'Cancel', 'Save', and '(Re)Start' buttons are visible at the bottom.



The screenshot shows a JupyterLab terminal window. The top part shows the command `hostname` being executed, resulting in `jsf102`. The bottom part shows the command `hostname` being executed in a terminal window, resulting in `jsf102.jusuf`. A red arrow points from the `jsf102` output in the top terminal to the `jsf102.jusuf` output in the bottom terminal. A terminal window titled `kreuzer1@jsf102:~/slurm-prov` is also visible. A terminal window titled `[goebbert1@jsf101 jusuf]$ jupyter kernelspec provisioners` shows the output: `Available kernel provisioners:`, `local-provisioner`, `jupyter_client.provisioning:LocalProvisioner`, `slurm-provisioner`, and `jupyter_slurm_provisioner:SlurmProvisioner`.

<https://github.com/FZJ-JSC/jupyter-slurm-provisioner>
<https://github.com/FZJ-JSC/jupyter-slurm-provisioner-extension>

REMOTE JUPYTER KERNELS

Jupyter

The screenshot displays the JupyterLab web interface in a browser window. The address bar shows the URL: `https://jupyter-jsc-fz-juelich.de/user/j.goebbert_at_fz-juelich.de/be5f53dfad7b4b4f91b2f5ac9c6342d8/lab/workspaces/be5f53dfad7b4b4f91b2f5ac9c6342d8`. The interface includes a top menu bar (File, Edit, View, Run, Kernel, Git, Tabs, Settings, Help), a left sidebar with navigation icons, and a main content area titled "Launcher".

The "Current Configuration" sidebar on the left indicates "Nothing configured yet. Click configure and choose a partition." with a "Configure" button. Below it, "Kernel Allocations" shows "There are no allocations available."

The main "Launcher" view displays the path `p/home/jusers/goebbert1/jusuf` and a "Notebook" section with a grid of kernel icons:

- Python 3 (ipykernel)
- Bash
- C++17
- covid19dynstat_jusuf
- covid19dynstat_juwels
- esm2022_kernel
- goebbert1_kerneljsonmerge
- goebbert1_pyferrret
- goebbert1_workshop2
- JavaScript (Node.js)
- Julia 1.7.1
- my_env
- Octave-6.4.0
- PyDeepLearning-1.1
- PyQuantum-3.0
- PyVisualization-1.0
- R-4.1.2
- Ruby 3.0.1
- Slurm Wrapper
- unseen_kernel
- Xpra Desktop [↗]

A "Try the Welcome Tour." notification is visible in the bottom right corner. The bottom status bar shows "Mem: 195.93 / 257467.88 MB" and "English (United States) Launcher".

RECORDED WITH
SCREENCAST
Simple

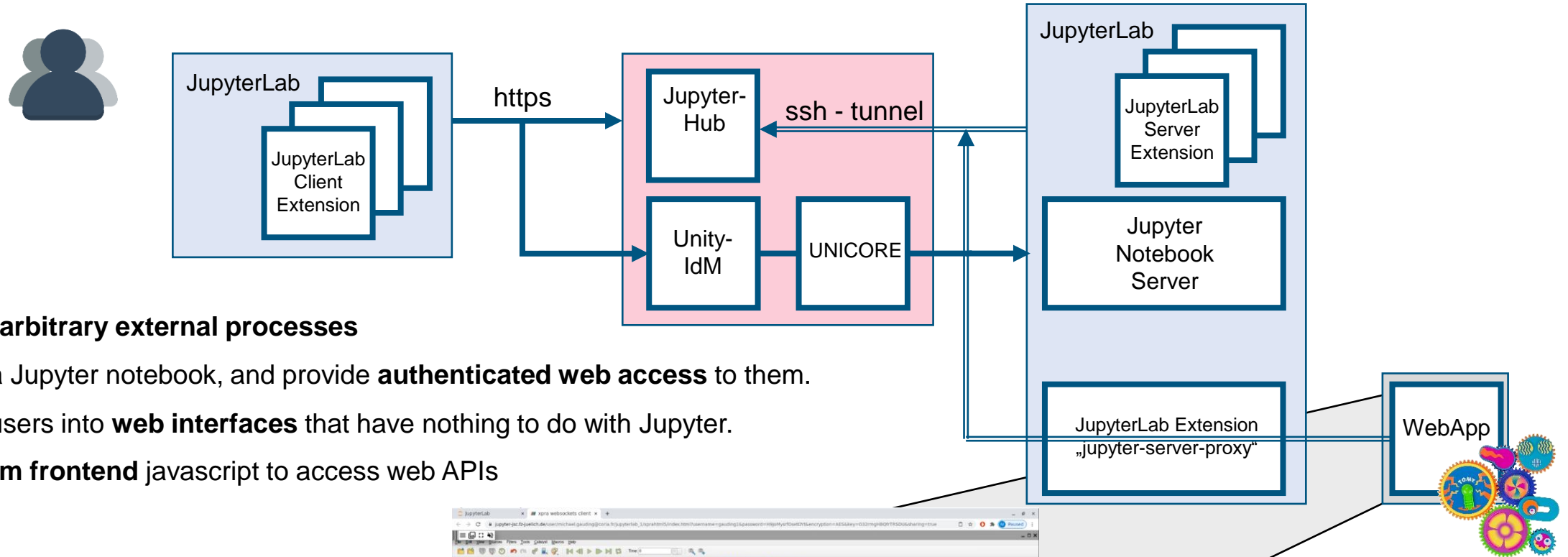
<https://github.com/FZJ-JSC/jupyter-slurm-provisioner>
<https://github.com/FZJ-JSC/jupyter-slurm-provisioner-extension>

slurm-provisioner jupyter_slurm_provisioner:SlurmProvisioner
Forschungszentrum

JUPYTER SERVER PROXY

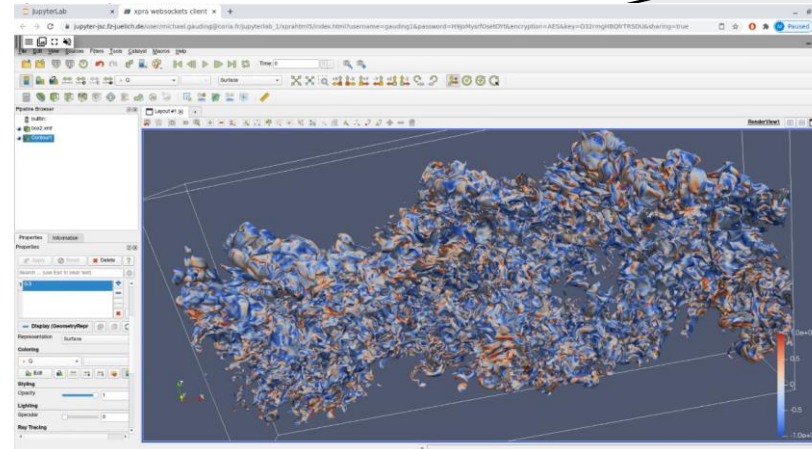
JUPYTERLAB – WEBSERVICE PROXY

Extension: jupyter-server-proxy



Allows to run **arbitrary external processes**

- alongside a Jupyter notebook, and provide **authenticated web access** to them.
- launching users into **web interfaces** that have nothing to do with Jupyter.
- **access from frontend javascript** to access web APIs

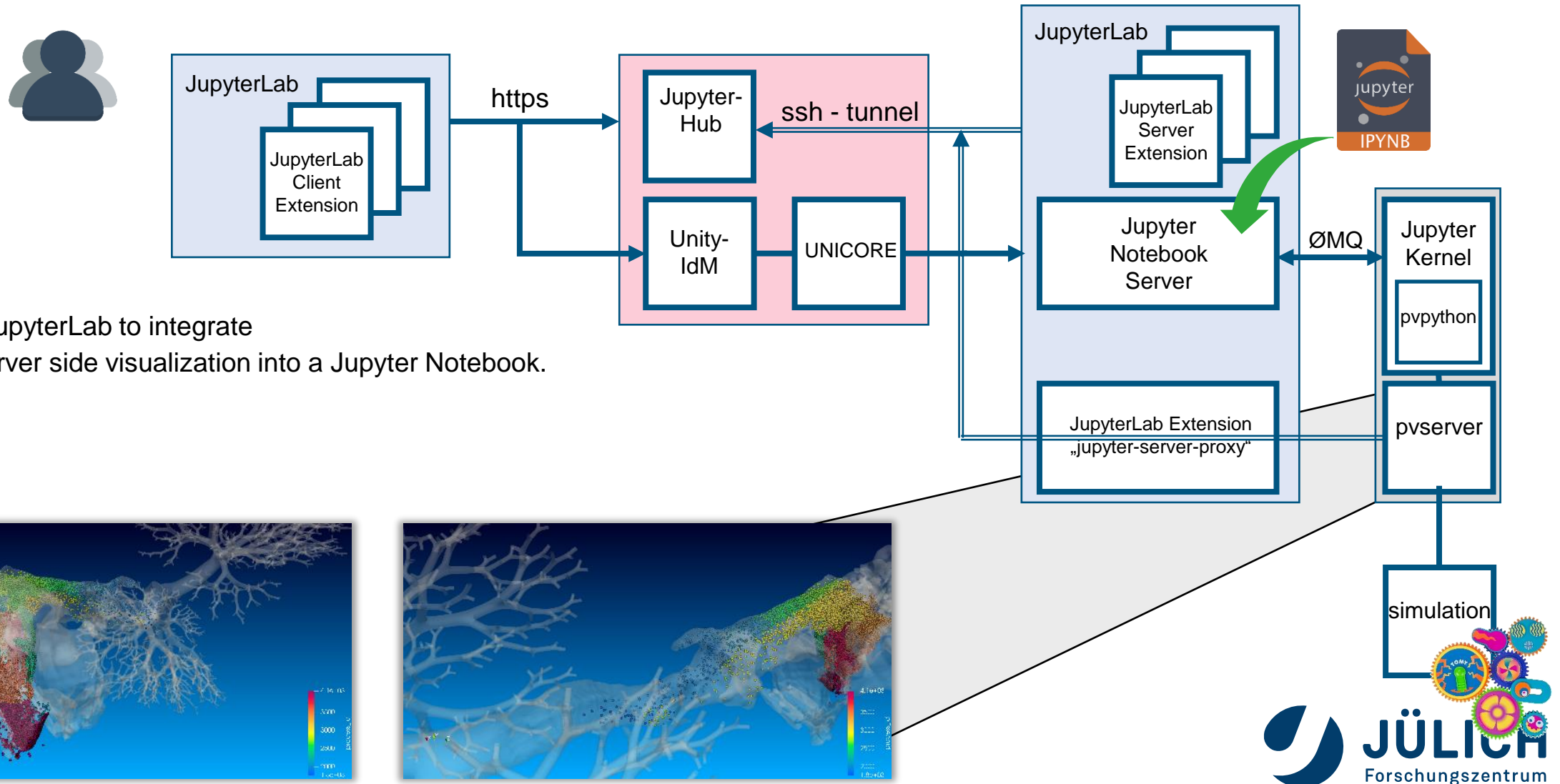


Turbulent mixing with variable density,
subset of 1939x600x3584 grid points, Michael Gauding, CORIA

<https://github.com/jupyterhub/jupyter-server-proxy>

JUPYTERLAB – WEBSERVICE PROXY

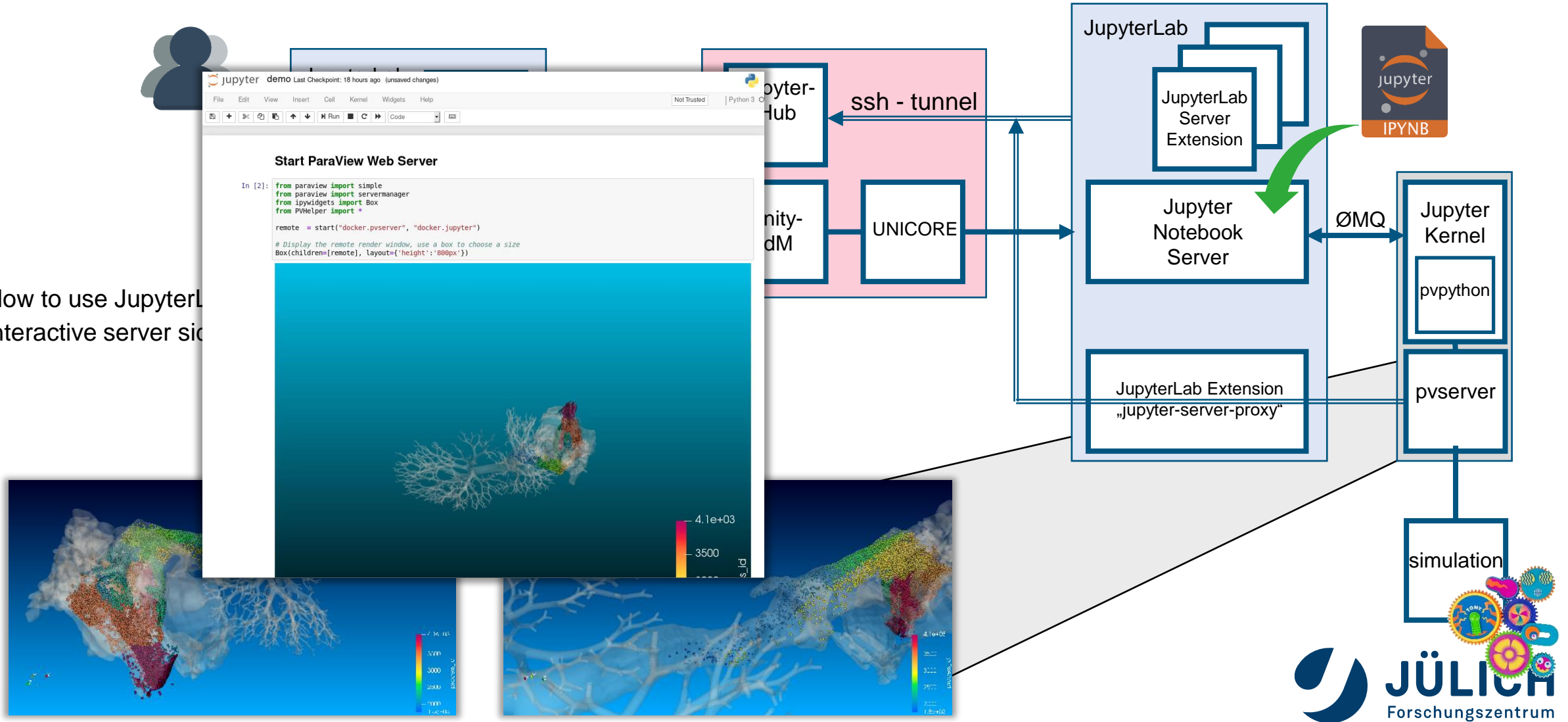
Extension: jupyter-server-proxy



JUPYTERLAB – WEBSERVICE PROXY

Extension: jupyter-server-proxy

How to use JupyterLab
interactive server side



PORT TUNNELING – WEBSERVICE PROXY

Extension: jupyter-server-proxy

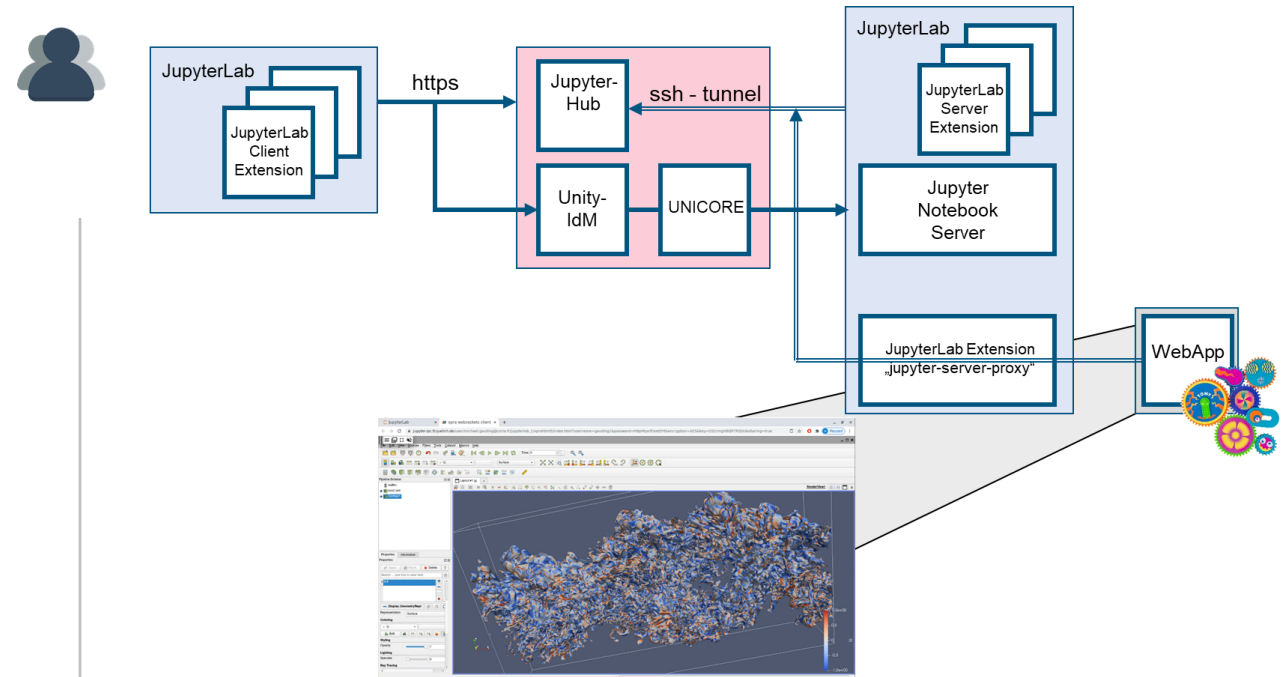
Accessing Arbitrary Ports or Hosts from the Browser

If you have a web-server running on the server listening on <port>, you can access it through the notebook at **<notebook-base>/proxy/<port>**

The URL will be rewritten to remove the above prefix.

You can disable URL rewriting by using **<notebook-base>/proxy/absolute/<port>** so your server will receive the full URL in the request.

This works for all ports listening on the local machine.



Example:

`https://jupyter-jsc.fz-juelich.de/user/j.goebbert@fz-juelich.de/juwels_login/proxy/<port>`

`https://jupyter-jsc.fz-juelich.de/user/j.goebbert@fz-juelich.de/juwels_login/proxy/<host>:<port>`

Upcoming: Support proxying to a server process via a Unix socket (#337)

<https://jupyter-server-proxy.readthedocs.io/en/latest/arbitrary-ports-hosts.html>

PORT TUNNELING – WEBSERVICE PROXY

Extension: jupyter-server-proxy

Accessing Arbitrary Ports or Hosts from the Browser

If you have a web-server running on the server

listening on <port>

<notebook-base>

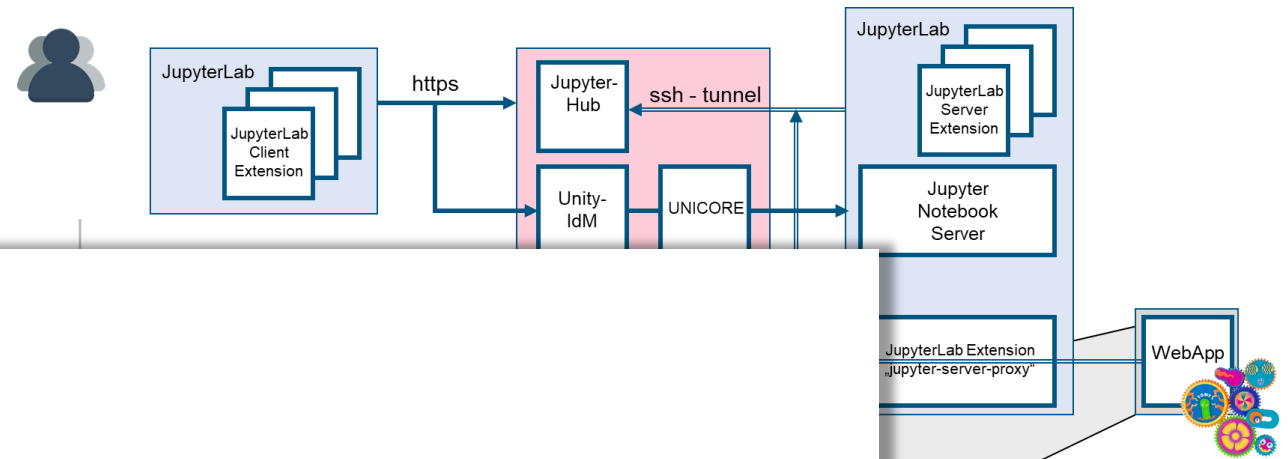
The URL will be re

You can disable U

<notebook-base>

so your server will

This works for all p



LET'S TEST IF THAT WORKS

Example:

https://jupyter-jsc.fz-juelich.de/user/j.goebbert@fz-juelich.de/juwels_login/proxy/<port>

https://jupyter-jsc.fz-juelich.de/user/j.goebbert@fz-juelich.de/juwels_login/proxy/<host>:<port>

Upcoming: Support proxying to a server process via a Unix socket (#337)

<https://jupyter-server-proxy.readthedocs.io/en/latest/arbitrary-ports-hosts.html>

JUPYTER SERVER PROXY ON THE EXAMPLE OF REMOTE DESKTOP BASED ON XPRA

JUPYTERLAB – REMOTE DESKTOP

Run your X11-Applications in the browser

Jupyter-JSC gives you easy access to a remote desktop

1. <https://jupyter-jsc.fz-juelich.de>
2. Click on “Xpra”

Xpra - X Persistent Remote Applications

is a tool which runs X clients on a remote host and directs their display to the local machine.

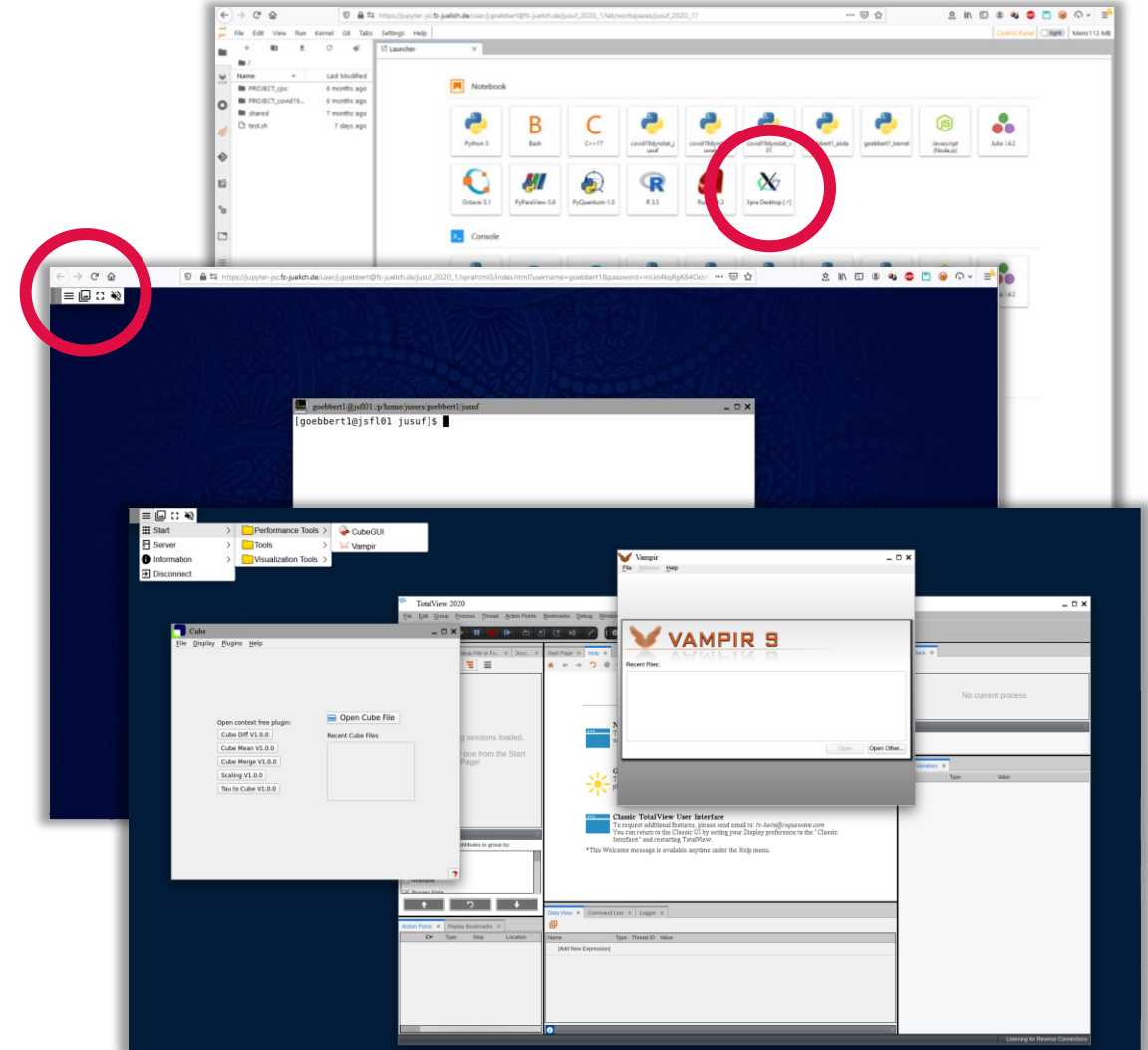
- Runs in a browser
- allows dis-/reconnection without disrupting the forwarded application
- <https://xpra.org>

The remote desktop will run on the same node as your JupyterLab does (this includes compute nodes).

It gets killed, when you stop your JupyterLab session.

Hint:

- CTRL + C -> CTRL + Insert
- CTRL + V -> SHIFT + Insert



JUPYTERLAB – REMOTE DESKTOP

Run your X11-Applications in the browser

Jupyter-JSC gives you easy access to a remote desktop

1. <https://jupyter-jsc.fz-juelich.de>
2. Click on “Xpra”

Xpra - X Persistent Remote Applications

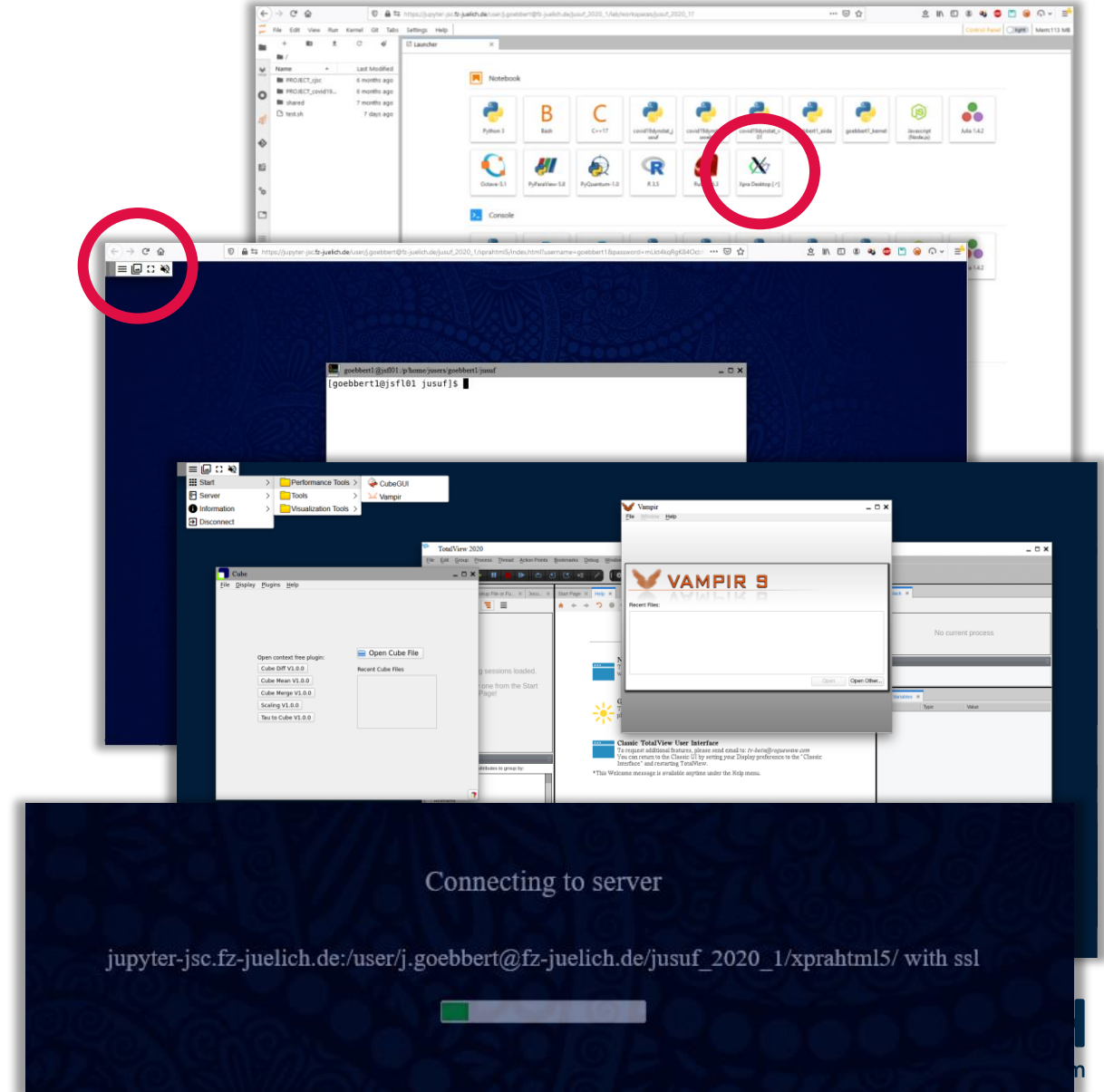
is a tool which runs X clients on a remote host and directs their display to the local machine.

- Runs in a browser
- allows dis-/reconnection without disrupting the forwarded application
- <https://xpra.org>

If the connection got lost at some point,
just hit the “reload” button of your browser.

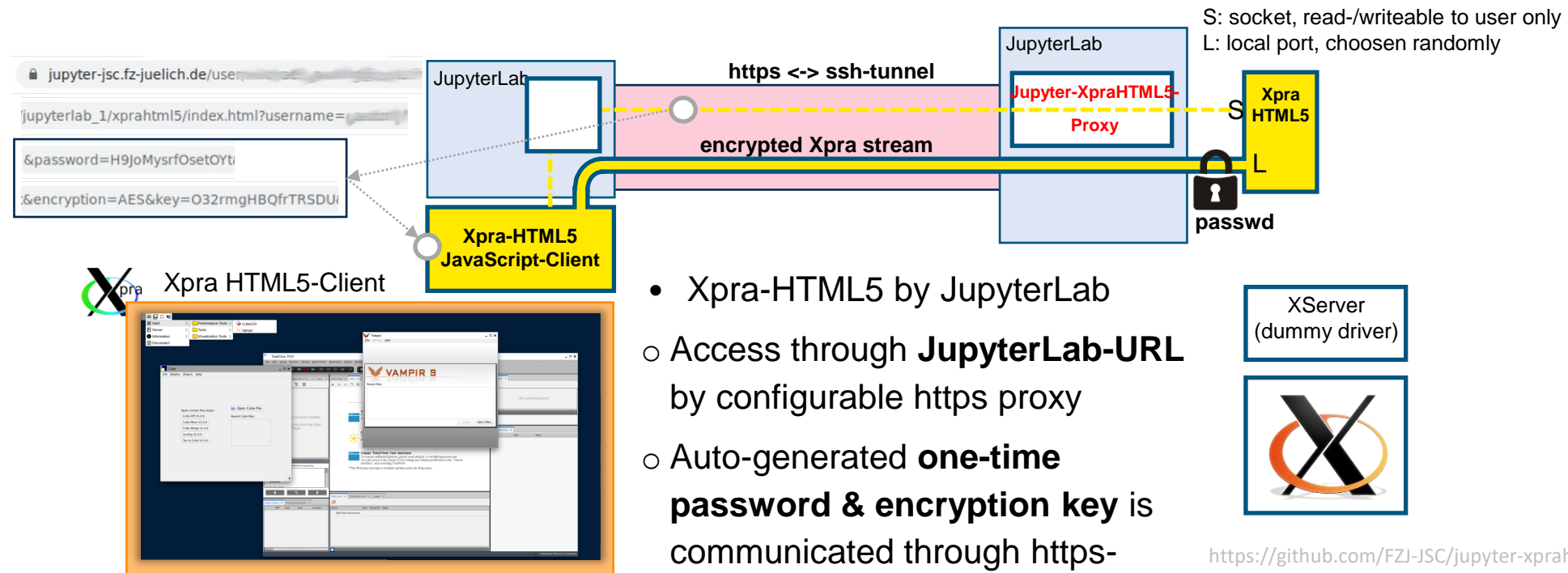
Hint:

- CTRL + C -> CTRL + Insert
- CTRL + V -> SHIFT + Insert



JUPYTERLAB – REMOTE DESKTOP

Run your X11-Applications in the browser



- Xpra-HTML5 by JupyterLab
- Access through **JupyterLab-URL** by configurable https proxy
- Auto-generated **one-time password & encryption key** is communicated through https-proxy

<https://github.com/FZJ-JSC/jupyter-xprahtml5-proxy>

Plugin for Jupyter-Server-Proxy: jupyter-xprahtml5-proxy
<https://github.com/FZJ-JSC/jupyter-xprahtml5-proxy>

jupyter-xprahtml-proxy – behind the scenes

- ## Python entry_point in setup.py

2. Launcher entry gets created

based on the returned values of `setup_xprahtml5()`

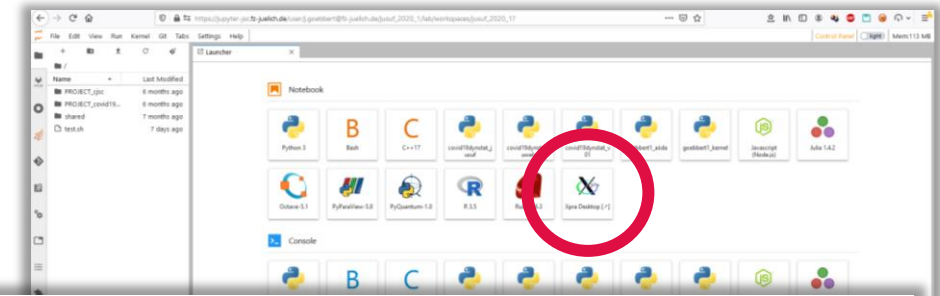
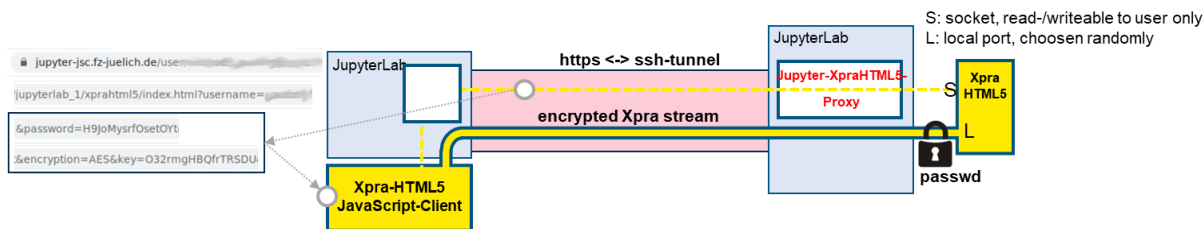
```
cmd = [
    get_xpra_executable('xpra'),
    'start',
    '--html=on',
    '--bind-tcp=0.0.0.0:{port}',
    # '--socket-dir="" + socket_path + "/"', # fixme: socket_dir not recognized
    # '--server-idle-timeout=86400', # stop server after 24h with no client connection
    # '--exit-with-client=yes', # stop Xpra when the browser disconnects
    '--start=xterm -fa "DejaVu Sans Mono" -fs 14',
    # '--start-child=xterm', '--exit-with-children',
    '--tcp-auth=file:filename=' + fpath_passwd,
    '--tcp-encryption=AES',
    '--tcp-encryption-keyfile=' + fpath_aeskey,
    '--clipboard-direction=both',
    '--keyboard-sync=no', # prevent keys from repeating unexpectedly on high latency
    '--no-mdns', # do not advertise the xpra session on the local network
    '--no-bell',
    '--no-speaker',
    '--no-printing',
    '--no-microphone',
    '--no-notifications',
    '--no-systemd-run', # do not delegated start-cmd to the system wide proxy server instance
    # '--dpi=96', # only needed if Xserver does not support dynamic dpi change
    '--sharing', # this allows to open the desktop in multiple browsers at the same time
    '--no-daemon', # mandatory
```

JUPYTERLAB – REMOTE DESKTOP

jupyter-xprahtml5-proxy – behind the scenes

1. `--tcp-auth=file:filename`

2. `--tcp-encryption=AES`
`--tcp-encryption-keyfile`



```
cmd = [  
    get_xpra_executable('xpra'),  
    'start',  
    '--html=on',  
    '--bind-tcp=0.0.0.0:{port}',  
    # '--socket-dir=' + socket_path + '/', # fixme: socket_dir not recognized  
    # '--server-idle-timeout=86400', # stop server after 24h with no client connection  
    # '--exit-with-client=yes', # stop Xpra when the browser disconnects  
    '--start=xterm -fa "DejaVu Sans Mono" -fs 14',  
    # '--start-child=xterm', '--exit-with-children',  
    '--tcp-auth=file:filename' + fpath_passwd,  
    '--tcp-encryption=AES',  
    '--tcp-encryption-keyfile=' + fpath_aeskey,  
    '--clipboard-direction=both',  
    '--keyboard-sync=no', # prevent keys from repeating unexpectedly on high latency  
    '--no-mdns', # do not advertise the xpra session on the local network  
    '--no-bell',  
    '--no-speaker',  
    '--no-printing',  
    '--no-microphone',  
    '--no-notifications',  
    '--no-systemd-run', # do not delegated start-cmd to the system wide proxy server instance  
    # '--dpi=96', # only needed if Xserver does not support dynamic dpi change  
    '--sharing', # this allows to open the desktop in multiple browsers at the same time  
    '--no-daemon', # mandatory
```

JUPYTER SERVER PROXY TWO MORE EXAMPLES

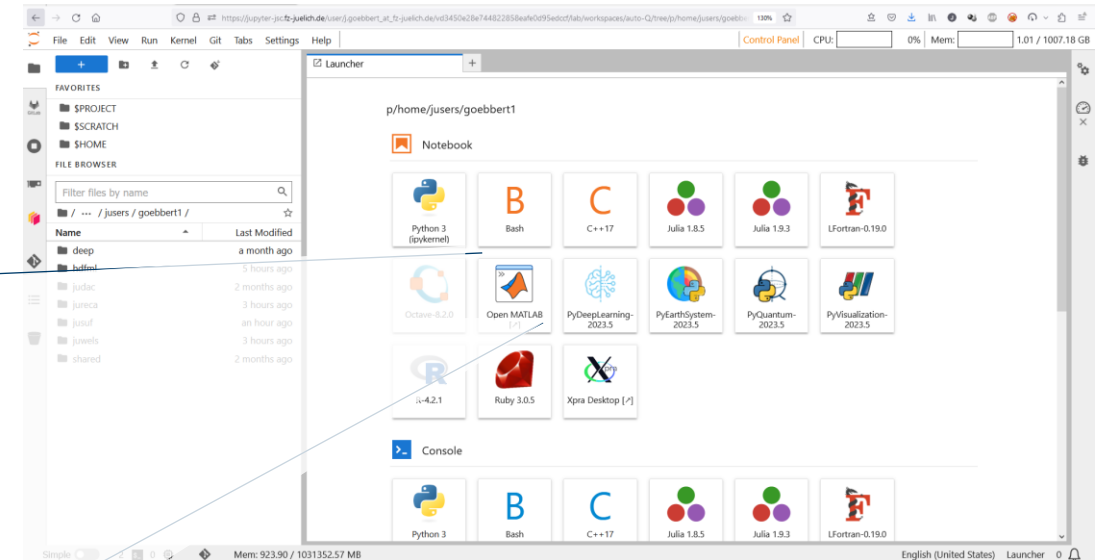
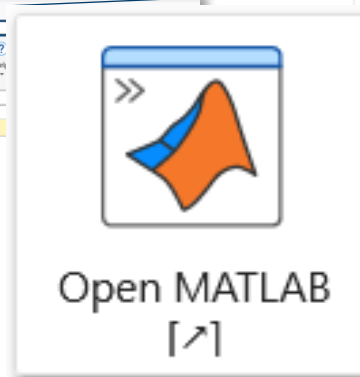
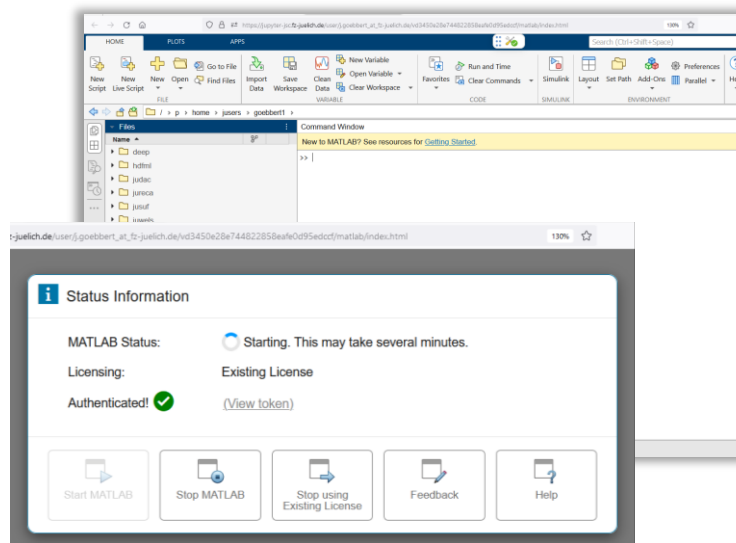
JUPYTERLAB – MATLAB

Web-based GUI for MATLAB

MATLAB – Web-based GUI

Based on an existing connection to the HPC system, MATLAB can be accessed in the browser.

- From here- you can connect directly to the cluster [2]
- Integrates MATLAB the HPC resources into the workflow (partool) [3].



[1] <https://www.fz-juelich.de/en/ias/jsc/services/user-support/software-tools/matlab>

[2] <https://de.mathworks.com/help/parallel-computing/remotecclusteraccess.html>

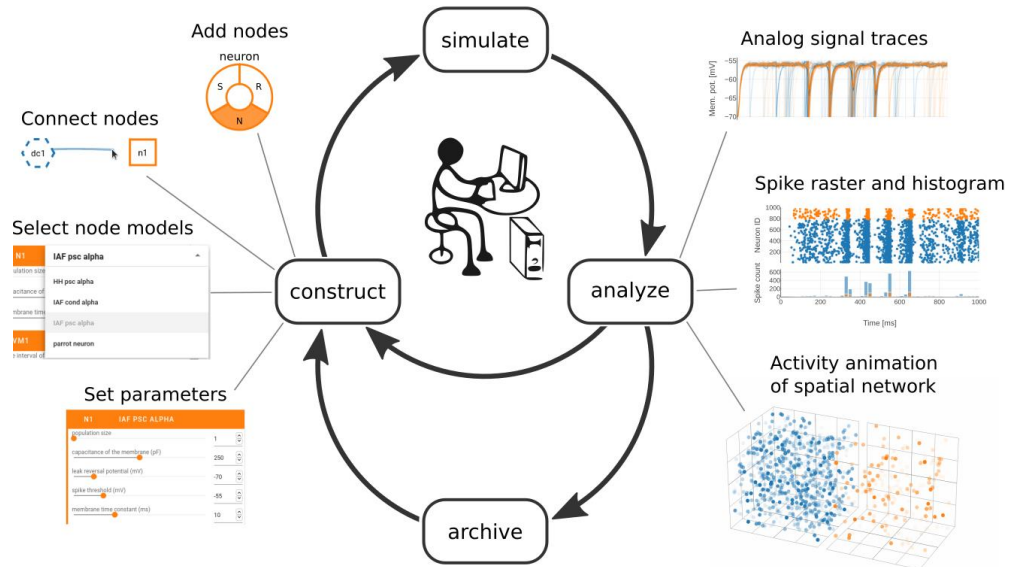
[3] <https://de.mathworks.com/products/parallel-computing.html>

JUPYTERLAB – NEST DESKTOP

Web-based GUI for Neuroscientists using NEST

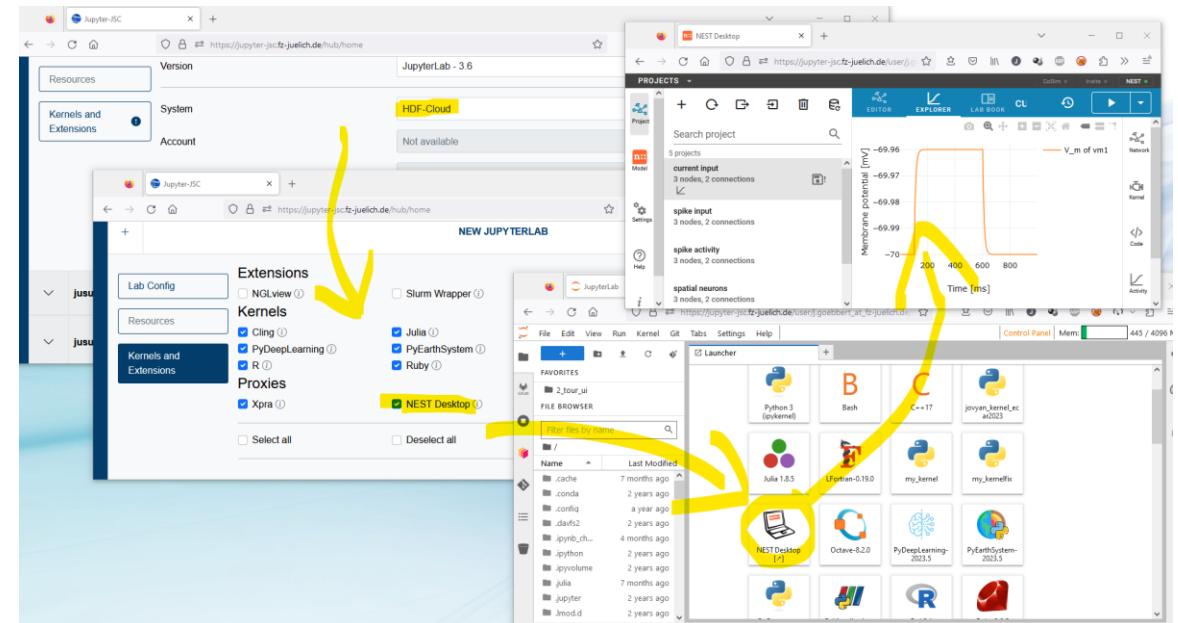
NEST-Desktop

NEST Desktop is a web-based GUI application for NEST Simulator, an advanced simulation tool for the computational neuroscience.



Jupyter-JSC gives you easy access to a NEST-Desktop

With Jupyter-JSC using Jupyter-Server-Proxy authenticated & authorized users get secure access to the WebUI of NEST-Desktop running NEST-simulations on HPC.



Plugin for Jupyter-Server-Proxy: jupyter-xprahtml5-proxy
<https://github.com/jhgoebbert/jupyter-nestdesktop-proxy>

[1] <https://nest-desktop.readthedocs.io>
[2] <https://www.nest-simulator.org>

JUPYTERLAB – NEST DESKTOP

Web-based

NEST-Desktop
NEST Desktop
Simulator
neuroscience

Connect node

Select node model

Set parameters

The screenshot shows the Jupyter-JSC web interface. At the top, there's a navigation bar with the Jülich logo, 'JÜLICH SUPERCOMPUTING CENTRE', and links for 'Start', 'Links', 'JSC Status', and 'Documentation'. The main content area features a large modal window titled 'JUPYTER NOTEBOOKS' with the heading 'Share your Workflows'. The modal text reads: 'We are pleased to bring "Supercomputing in your browser". Jupyter-JSC allows you to create and share documents that contain live code, equations, visualizations and explanatory text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, machine learning and much more. Read more.' Below the modal, there's a horizontal bar with 'Login' and 'Register' buttons. At the bottom, a status bar shows the number of users for various compute resources: Jupyter-JSC (75), JUWELS (30), JURECA (16), JUSUF (3), DEEP (2), HDFML (0), and HDF-Cloud (6). The footer includes 'RECORDED WITH SCREENCAST MATIC', 'Legal Notice | Privacy Policy | Terms of Service | Support', and the 'HELMHOLTZ RESEARCH FOR GRAND CHALLENGES' logo.

JUPYTER-JSC
Supercomputing in Your Browser

Jupyter-JSC starts and provides access to your Jupyter Notebook servers running on JSC compute resources. These can be JUWELS, JURECA, JUSUF, HDFML or DEEP's login or compute nodes or even the HDF cloud - depending on the computing resources available to you.

Please use your JSC account to log in or register if you have not already done so. It's also possible to log in via Helmholtz AAI. ?

Login Register

Jupyter-JSC 75 JUWELS 30 JURECA 16 JUSUF 3 DEEP 2 HDFML 0 HDF-Cloud 6

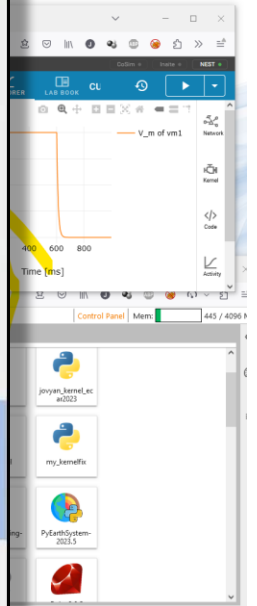
RECORDED WITH SCREENCAST MATIC

Legal Notice | Privacy Policy | Terms of Service | Support

HELMHOLTZ
RESEARCH FOR GRAND CHALLENGES

top

the WebUI



- [1] <https://nest-desktop.readthedocs.io>
- [2] <https://github.com/jhgoebbert/jupyter-nestdesktop-proxy>
- [3] <https://www.nest-simulator.org>

