



# JUPYTERLAB - SUPERCOMPUTING IN YOUR BROWSER

Training course "Introduction to the usage and programming of supercomputer resources in Jülich"

2023-11-21 | JENS H. GÖBBERT

(J.GOEBBERT@FZ-JUELICH.DE)

TIM KREUZER

(T.KREUZER@FZ-JUELICH.DE)

ALICE GROSCH

(A.GROSCH@FZ-JUELICH.DE)

# MOTIVATION

your thinking, your reasoning, your insides, your ideas

“It is all about using and building a machinery **interface between** computational researchers and data, supercomputers, laptops, cloud **and** your thinking, your reasoning, your insides, your ideas about a problem.”

Fernando Perez, Berkely Institute for Data Science  
Founder of Project Jupyter

# JUPYTER NOTEBOOK

creating reproducible computational narratives

Markdown Cells

Code Cells

### Fourier transform

Fourier transforms are one of the universal tools in computational physics, which appear over and over again in different contexts. SciPy provides functions for accessing the classic `FFTPACK` library from NetLib, which is an efficient and well tested FFT library written in FORTRAN. The SciPy API has a few additional convenience functions, but overall the API is closely related to the original FORTRAN library.

To use the `fftpack` module in a python program, include it using:

```
[41]: from numpy.fft import fftfreq
      from scipy.fftpack import *
```

To demonstrate how to do a fast Fourier transform with SciPy, let's look at the FFT of the solution to the damped oscillator:

$$\frac{d^2x}{dt^2} + 2\zeta\omega_0 \frac{dx}{dt} + \omega_0^2 x = 0$$

where  $x$  is the position of the oscillator,  $\omega_0$  is the frequency, and  $\zeta$  is the damping ratio. To write this second-order ODE on standard form we introduce  $p = \frac{dx}{dt}$ :

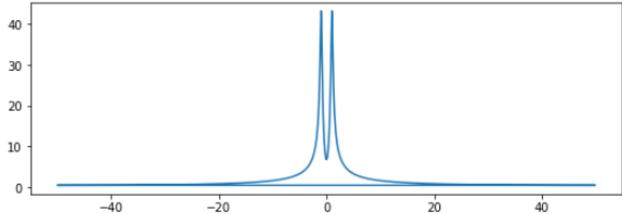
```
[42]: N = len(t)
      dt = t[1]-t[0]
      dt
```

```
[42]: 0.01001001001001001
```

```
[43]: # calculate the fast fourier transform
      # y2 is the solution to the under-damped oscillator from the previous section
      F = fft(y2[:,0])

      # calculate the frequencies for the components in F
      w = fftfreq(N, dt)
```

```
[44]: fig, ax = plt.subplots(figsize=(9,3))
      ax.plot(w, abs(F));
```



Output

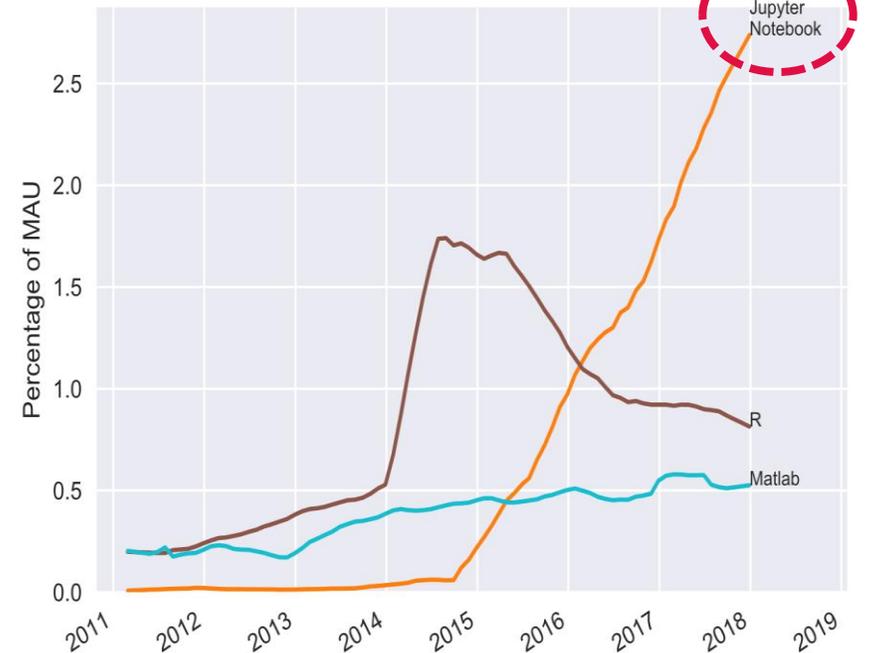
Output

# MOTIVATION

## Rise of Jupyter's popularity

- In 2007, Fernando Pérez and Brian Granger announced „**IPython**: a system for interactive scientific computing“ [1]
- In 2014, Fernando Pérez announced a spin-off project from IPython called **Project Jupyter**.
  - IPython continued to exist as a Python shell and a kernel for Jupyter, while the Jupyter notebook moved under the Jupyter name.
- In 2015, GitHub and the Jupyter Project announced native rendering of Jupyter notebooks file format (.ipynb files) on the **GitHub**
- In 2017, the **first JupyterCon** was organized by O'Reilly in New York City. Fernando Pérez opened the conference with an inspiring talk. [2]
- In 2018, **JupyterLab** was announced as the next-generation web-based interface for Project Jupyter.
- In 2019, JupyterLab 1.0 ...  
In 2020, JupyterLab 2.0 ...  
In 2021, JupyterLab 3.0 ...  
In 2023, JupyterLab 4.0 expected in March 2023.

## Counting how many Monthly Active Users (MAU) on GitHub are using Jupyter Notebooks

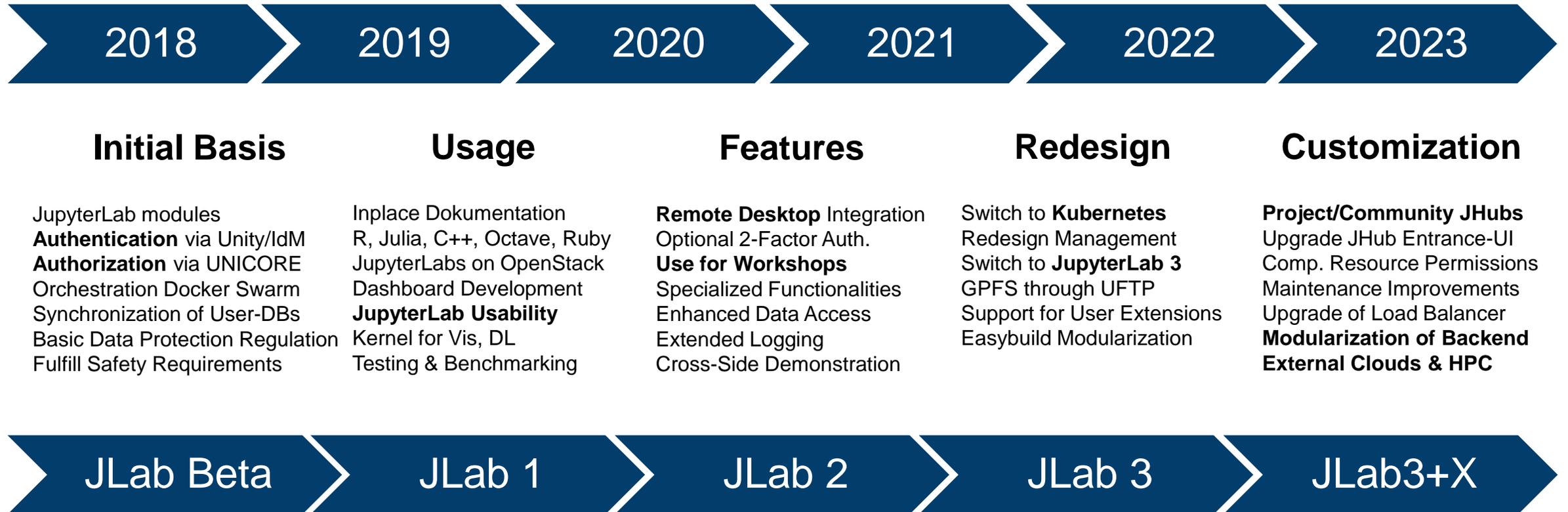


<https://www.benfrederickson.com/ranking-programming-languages-by-github-users/>  
<https://github.com/benfred/github-analysis>

[1] Pérez F, Granger BE (2007) IPython: a system for interactive scientific computing. Comput Sci Eng 9(3):21–29

[2] Pérez F, Project Jupyter: From interactive Python to open science -> <https://www.youtube.com/watch?v=xuNj5paMuow>

# HISTORY OF JUPYTERLAB AT JSC



# HISTORY OF JUPYTERLAB AT JSC

2021

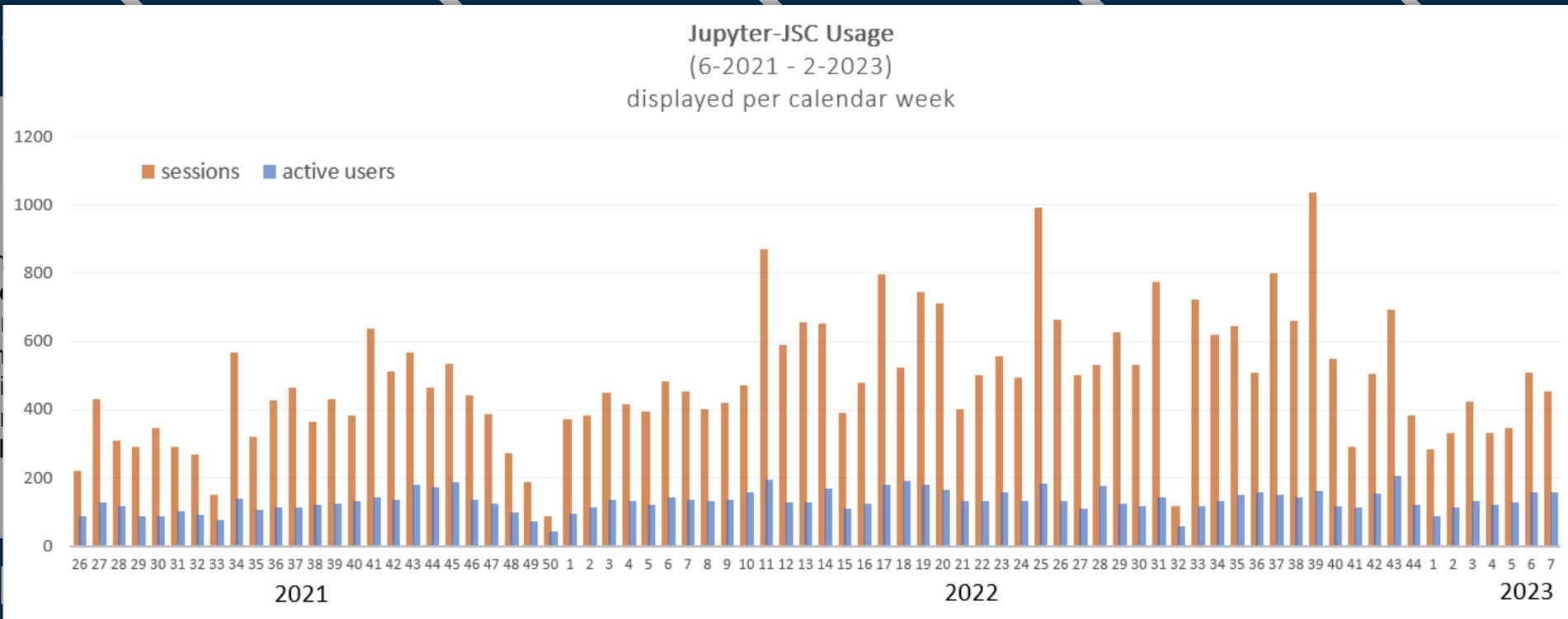
2023

Initi

ization

JupyterLab m  
**Authenticati**  
**Authorizati**  
 Orchestration  
 Synchronizati  
 Basic Data P  
 Fulfill Safety I

Community JHubs  
 Entrance-UI  
 e Permissions  
 improvements  
 d Balancer  
**of Backend**



JLa

B+X

# TERMINOLOGY

# TERMINOLOGY

## What is JupyterLab

### JupyterLab

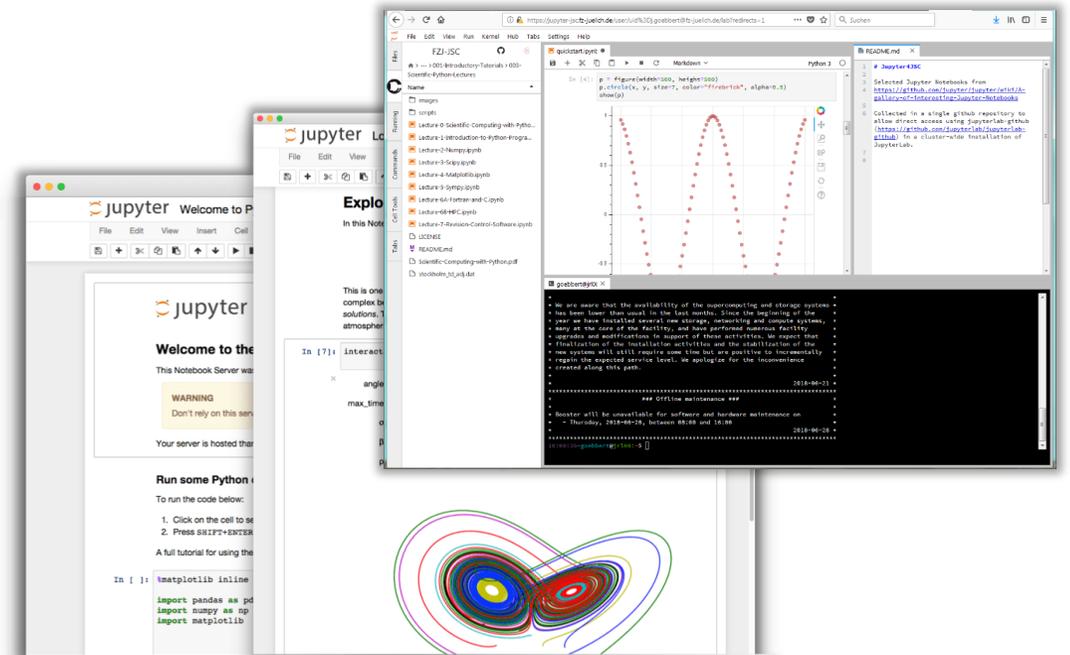
- **Interactive** working environment in the web browser
- For the creation of **reproducible** computer-aided narratives
- Very **popular** with researchers from all fields
- Jupyter = Julia + Python + R

### Multi-purpose working environment

- Language agnostic
- Supports execution environments (“*kernels*”)
  - For dozens of languages: Python, R, Julia, C++, ...
- Extensible software design („*extensions*“)
  - many server/client plug-ins available
  - Eg. in-browser-terminal and file-browsing

### Document-Centered Computing (“*notebooks*”)

- Combines code execution, rich text, math, plots and rich media.
- All-in-one document called Jupyter Notebook



<https://jupyterlab.readthedocs.io>

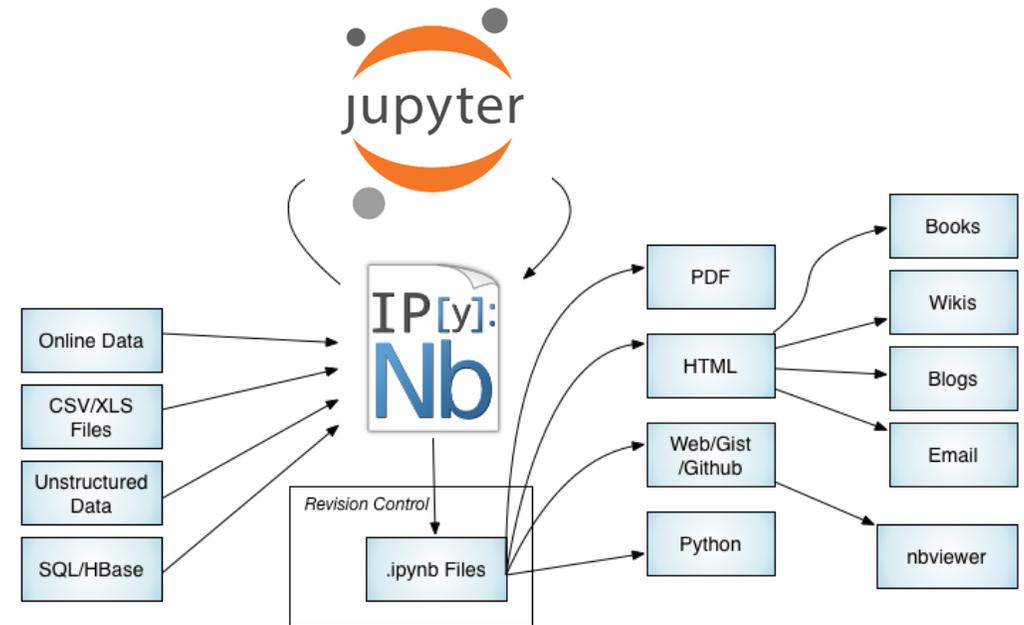
# TERMINOLOGY

## What is a Jupyter Notebook?

### Jupyter Notebook

A notebook document (file extension .ipynb) is a document that can be rendered in a web browser

- It is a file, which stores your work in JSON format
- Based on a set of open standards for interactive computing
- Allows development of custom applications with embedded interactive computing.
- Can be extended by third parties
- Directly convertible to PDF, HTML, LaTeX ...
- Supported by many applications such as GitHub, GitLab, etc..



<https://jupyter-notebook.readthedocs.io/>

<https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks>

# TERMINOLOGY

## What is a 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://jupyter-notebook.readthedocs.io/>  
<https://github.com/jupyter/jupyter/wiki/Jupyter-kernels>  
<https://zeromq.org>

# TERMINOLOGY

## What is a JupyterLab Extension?

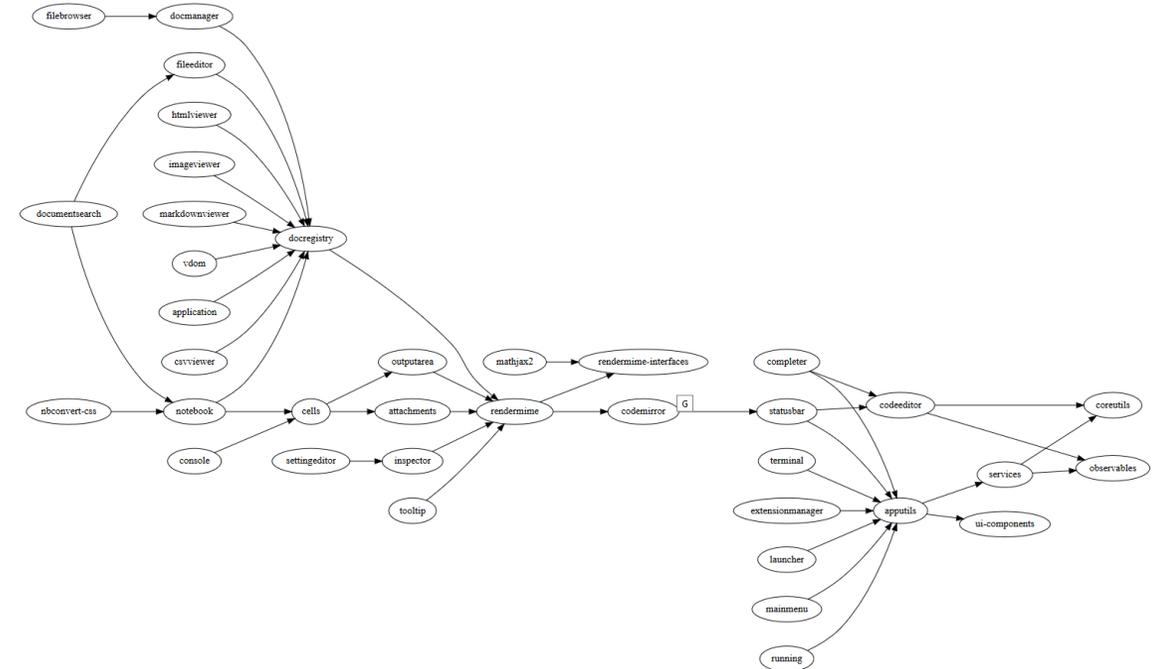
### JupyterLab Extension

JupyterLab extensions can customize or enhance any part of JupyterLab.

### JupyterLab Extensions

- provide new file viewers, editors, themes
  - provide renderers for rich outputs in notebooks
  - add items to the menu or command palette
  - add keyboard shortcuts
  - add settings in the settings system.
- 
- Extensions can even provide an API for other extensions to use and can depend on other extensions.

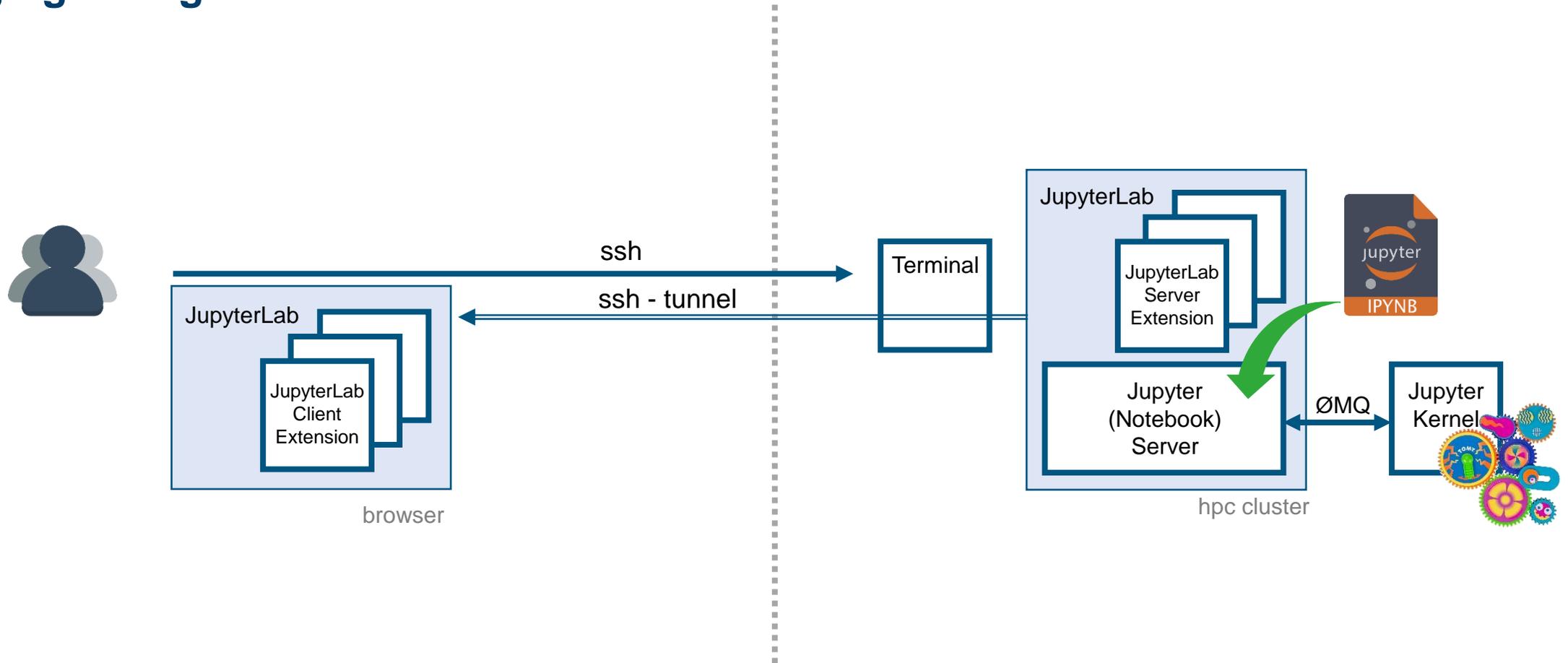
The whole JupyterLab itself is simply a **collection of extensions** that are no more powerful or privileged than any custom extension.



<https://jupyterlab.readthedocs.io/en/stable/user/extensions.html>  
<https://github.com/topics/jupyterlab-extension>

# TERMINOLOGY

## Bringing all together



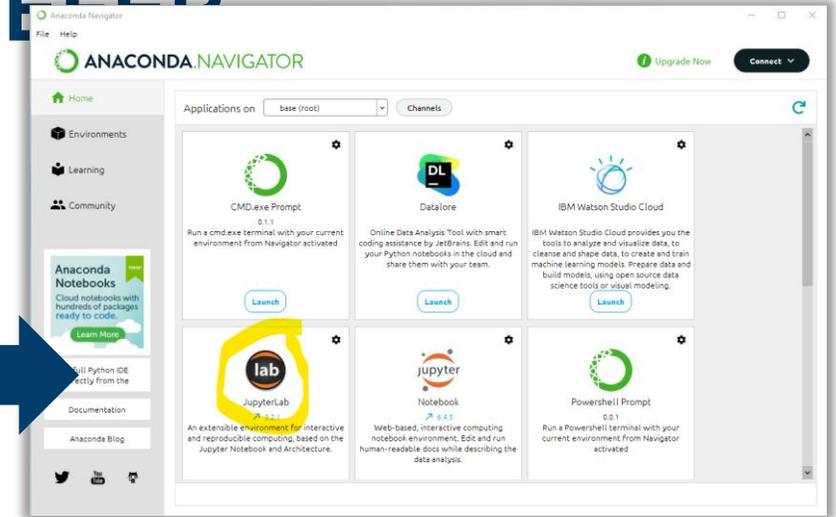
# INSTALLATION

# JUPYTERLAB - WHEREVER YOU PREFER

Local, Remote, Browser-only

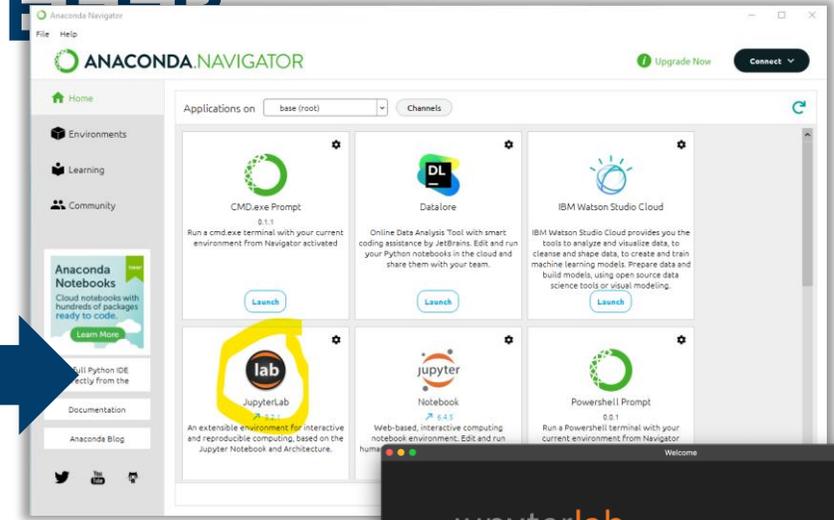
## Local installation:

- **JupyterLab** installed using conda, mamba, pip, pipenv or docker.  
→ [https://jupyterlab.readthedocs.io/en/stable/getting\\_started/installation.html](https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html)



# JUPYTERLAB - WHEREVER YOU PREFER

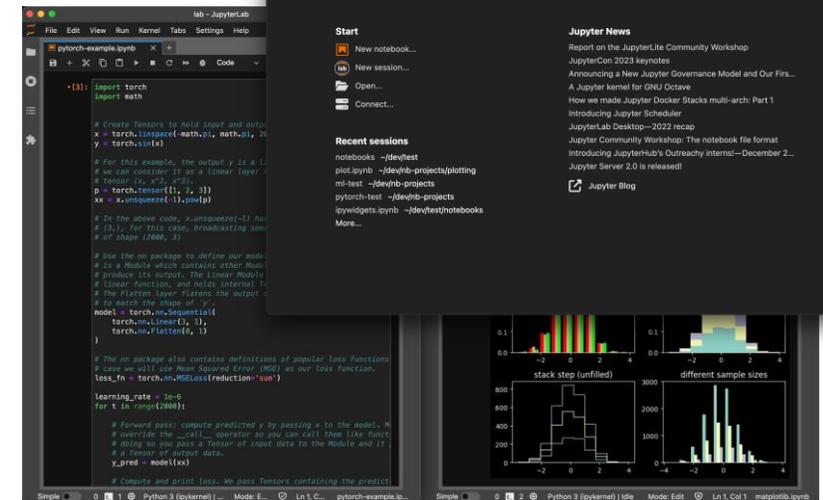
Local, Remote, Browser-only



## Local installation:

- **JupyterLab** installed using conda, mamba, pip, pipenv or docker.  
→ [https://jupyterlab.readthedocs.io/en/stable/getting\\_started/installation.html](https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html)
- **JupyterLab** installed as normal desktop application = **JupyterLab Desktop**  
→ <https://github.com/jupyterlab/jupyterlab-desktop/releases>

**JupyterLab Desktop** is the cross-platform desktop application for JupyterLab. It is probably the quickest and easiest way to get started with Jupyter notebooks on your personal computer, with the flexibility for advanced use cases.  
(Windows, macOS, Debian/Ubuntu, RedHat/Fedora)



# JUPYTERLAB - WHEREVER YOU PREFER

## Local, Remote, Browser-only

### Local installation:

- **JupyterLab** installed using conda, mamba, pip, pipenv or docker.  
→ [https://jupyterlab.readthedocs.io/en/stable/getting\\_started/installation.html](https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html)
- **JupyterLab** installed as normal desktop application = **JupyterLab Desktop**  
→ <https://github.com/jupyterlab/jupyterlab-desktop/releases>

### Remote (cluster) installation:

- **JupyterLab** installed on a remote server and accessed through the browser
  - in \$HOME (e.g. using pip or miniconda)
  - system-wide (e.g. with Easybuild, Spark) by the admins.



**Tunnel the new JupyterLab to your local machine**

**Linux or Mac:**  
If your operating system is Linux or Mac user:

```
ssh -N -L <LOCAL_PORT>:<JLAB_NODE>:<JLAB_PORT> <USERID>@<LOGIN_NODE>.fz-juelich.de  
# example: ssh -N -L 8888:jwels04:8888 goebbert1@jwels01.fz-juelich.de  
  
# if you want to tunnel to jwels04 only, then you should set JLAB_NODE to "localhost"
```

**Attention:**

- LOGIN\_NODE - Hostname of login node from the view of your local machine
- JLAB\_NODE - Hostname of the node running JupyterLab from the view of LOGIN\_NODE
- LOCAL\_PORT - port on your local machine
- JLAB\_PORT - port on the node running JupyterLab

**Windows:** In case your operating system is Windows, the setup of the tunnel depends on your ssh client. Here a short overview on how-to setup a tunnel with **PuTTY** is given.

It is assumed that PuTTY is already configured in a way that a general ssh connection to JUWELS is possible. That means that host name, user name and the private ssh key (using PuTTY's Pageant) are correctly set. You already made a first connection to JUWELS using PuTTY.

To establish the ssh tunnel start PuTTY and enter the "SSH->tunnels" tab in the PuTTY configuration window before connecting to JUWELS. You have to enter the source port (eg. <LOCAL\_PORT> = 8888) and the destination (eg. jwels01.fz-juelich.de:8888) and then press add. After pressing add, the tunnel should appear in the list of forwarded ports and you can establish the tunnel by pressing the open button.

# JUPYTERLAB - WHEREVER YOU PREFER

## Local, Remote, Browser-only

### Local installation:

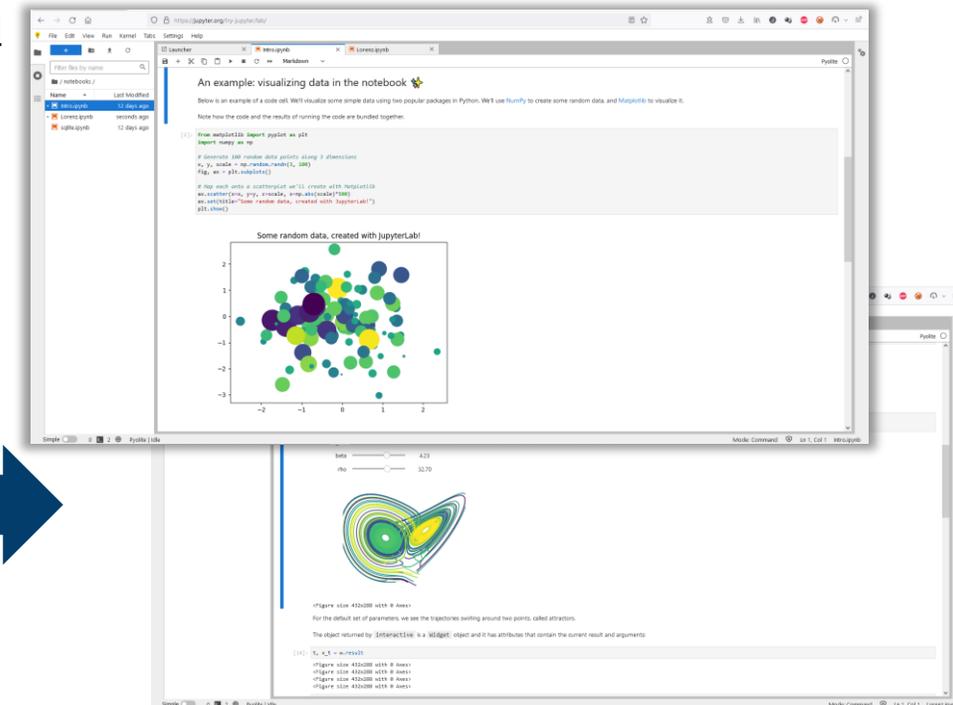
- **JupyterLab** installed using conda, mamba, pip, pipenv or docker.  
→ [https://jupyterlab.readthedocs.io/en/stable/getting\\_started/installation.html](https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html)
- **JupyterLab** installed as normal desktop application = **JupyterLab Desktop**  
→ <https://github.com/jupyterlab/jupyterlab-desktop/releases>

### Remote (cluster) installation:

- **JupyterLab** installed on a remote server and accessed through the browser
  - in \$HOME (e.g. using pip or miniconda)
  - system-wide (e.g. with Easybuild, Spark) by the admins.

### Browser-only installation (limited feature set):

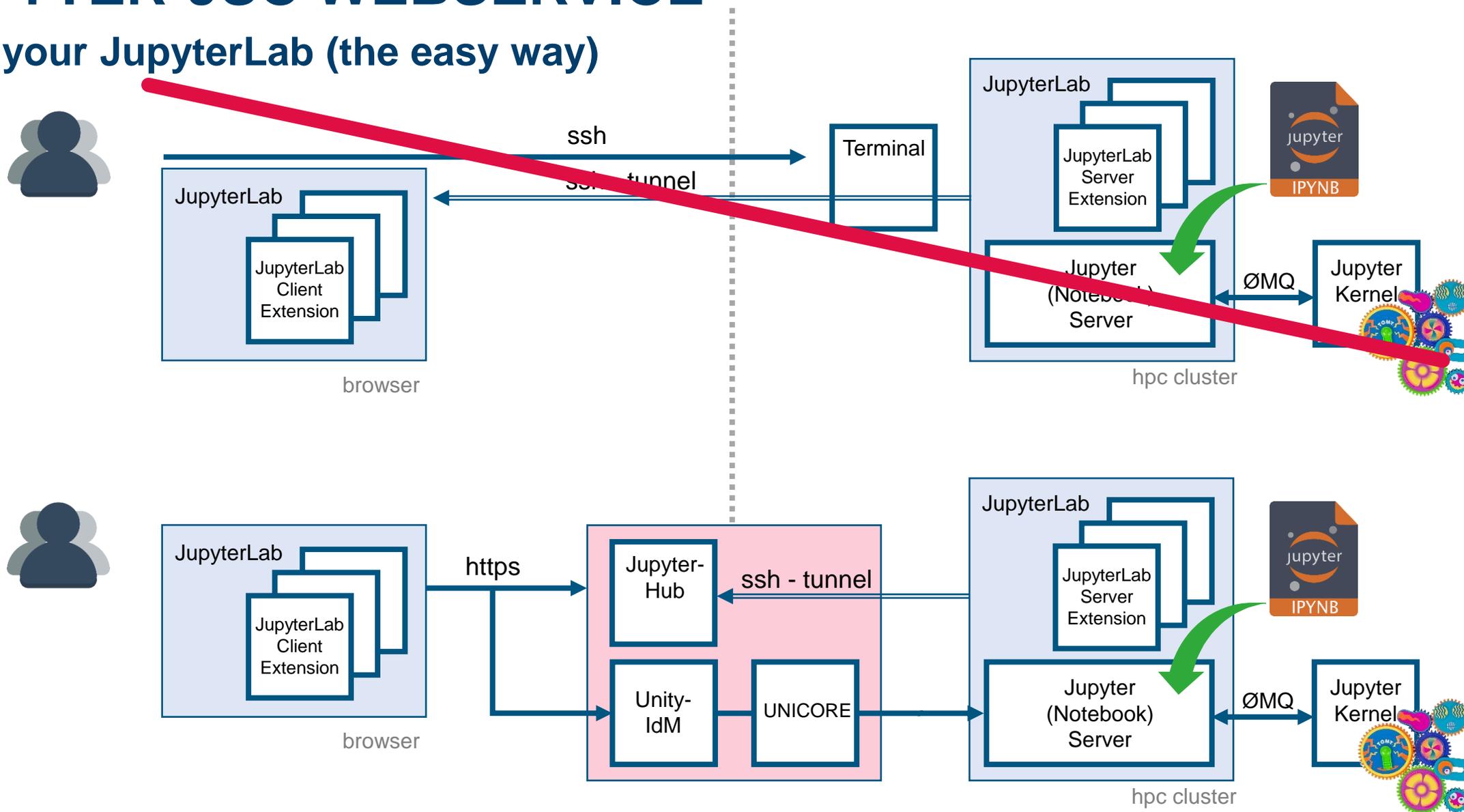
- **JupyterLab** local with server + client in your browser = **JupyterLite**  
Includes a browser-ready Python environment named Pyodide.  
→ <https://jupyter.org/try-jupyter/lab>



# START & LOGIN

# JUPYTER-JSC WEBSERVICE

## Start your JupyterLab (the easy way)



# PRE-ACCESS TODOS

## 1) Register & Login

- ✓ <https://judoor.fz-juelich.de>

## 2) Join a project

- ✓ Wait to get joined by the project PI

## 3) Sign usage agreement

- ✓ Wait for creation of HPC accounts

## 4) Check Connected Services:

- ✓ jupyter-jsc

The screenshot displays the 'Your account' page on the Jülich Supercomputing Centre portal. The page is organized into several sections:

- Account:** Fields for Salutation, E-mail address (with a checkmark), Telephone, and Address are visible.
- Mentored projects:** A button labeled 'Mentored projects' is present.
- Systems:** A table lists available systems:
  - judac:** Managed with SSH-keys, usage agreement confirmed on 18.04.2021 (marked with a green checkmark).
  - jureca:** JURECA-DC\_GPU: training 2211 (marked with a red X), with a message: 'You need to sign the usage agreement to access this system' (also marked with a red X).
- Projects:** A project titled 'Interactive High-Performance Computing with Jupyter @ JSC' is listed, associated with 'training2211' (marked with a green checkmark).
- Software:** A section for installed software.
- Connected Services:** A row of service buttons including 'trac', 'llview', 'jards', 'gitlab', and 'jupyter-jsc' (marked with a green checkmark).

# JUPYTER-JSC WEBSERVICE

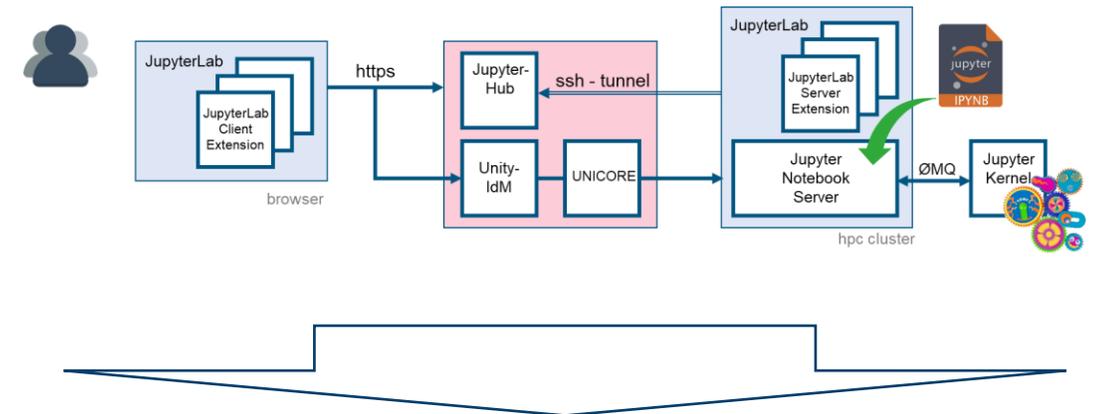
<https://jupyter-jsc.fz-juelich.de>

The image displays three overlapping screenshots of the Jupyter-JSC web interface. The top screenshot shows a 'Your server is starting up...' message with a table of server instances. The middle screenshot shows a '+ New' button and a table of existing JupyterLab instances. The bottom screenshot is a banner for 'Supercomputing in Your Browser' with a 'Login' button.

Name	System	Partition	Project	Status	Actions
juwels_cluster	JUWELS	devel	ccsvs	70%	Cancel

Name	System	Partition	Project	Status	Actions
hdfcloud_3.3	HDF-Cloud	N/A	N/A		Start
juwelsbooster_login	JUWELS	LoginNodeBooster	ccstdl		Start
juwels_cluster	JUWELS	devel	ccsvs	30% spinning	Open, Cancel



The image shows a screenshot of a Jupyter Notebook interface. The notebook contains Python code for matrix multiplication and GPU resource monitoring. The code is as follows:

```
[1]: import math
2: import numpy as np
3: from numba import cuda
4: import matplotlib.pyplot as plt
5: matplotlib.use('agg')

[2]:

[3]: len(cuda.gpus)

[4]: 1. len(cuda.gpus)
2. def mandelbrot_number(x, iterations):
3.     # matrix index
4.     i, j = cuda.grid(2)
5.     size = x.shape[0]
6.     # skip threads outside the matrix.
7.     if i >= size or j >= size:
8.         return
9.     # Run the simulation.
10.    e = 1.2 * 3.14159 / size * j
11.    z = 0
12.    for n in range(iterations):
13.        if math.isnan(z):
```

The interface also displays a 'GPU DASHBOARDS' sidebar with metrics for GPU Utilization, GPU Memory, GPU Resources, PCIe Throughput, WLink Throughput, WLink Timeout, and Machine Resources. A 'GPU Memory' graph shows usage over time, peaking at 332.49 MB.

# JUPYTER-JSC WEBSERVICE

## First time login

=> <https://jupyter-jsc.fz-juelich.de>

### Jupyter-JSC first time login

- Requirements:
  - Registered at [judoor.fz-juelich.de](https://judoor.fz-juelich.de)
    - (check "Connected Services" = jupyter-jsc)
  - Project membership + signed systems usage agreement
  - Waited ~10 minutes

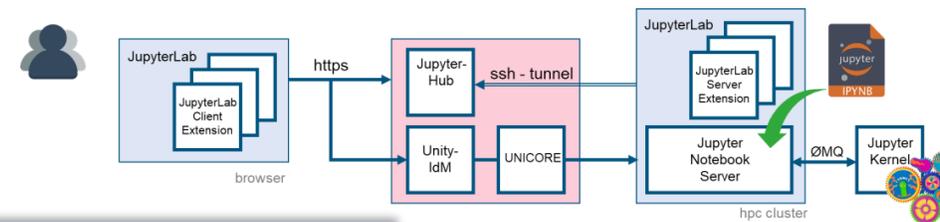
1. Login at <https://jupyter-jsc.fz-juelich.de>
2. Sign in with your JSC account
3. Register to Jupyter-JSC
4. Accept usage agreement
5. Submit the registration
6. Wait for email and confirm your email address

6. From: [unity-jsc@fz-juelich.de](mailto:unity-jsc@fz-juelich.de)  
To: [unity-jsc@fz-juelich.de](mailto:unity-jsc@fz-juelich.de)  
Subject: Jupyter-JSC Registration  
Date: Tue, 19 May 2020 11:09:53 +0200

Dear User,

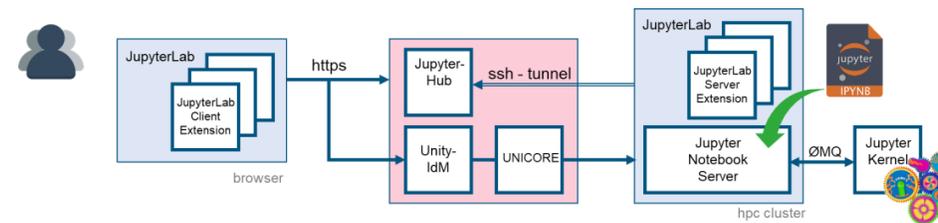
Your email address was entered into the Jupyter-JSC authentication service and must be confirmed. Afterward, you have to log in again.  
[Confirm your e-mail address.](#)

If you did not use your JuDoor account to log into <https://jupyter-jsc.fz-juelich.de>, we recommend that you change your JuDoor password.



# JUPYTER-JSC WEBSERVICE

## Control Panel



### A. New JupyterLab

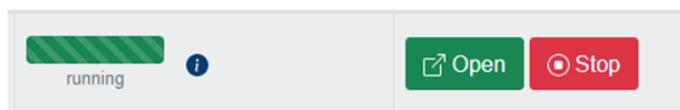


### B. Configuration Dialog

- Lab Config: set Name, Version, System, Account, Project, Partition
- Resources: if running on a compute node
- Kernels and Extensions: Optional addons

### C. Actions

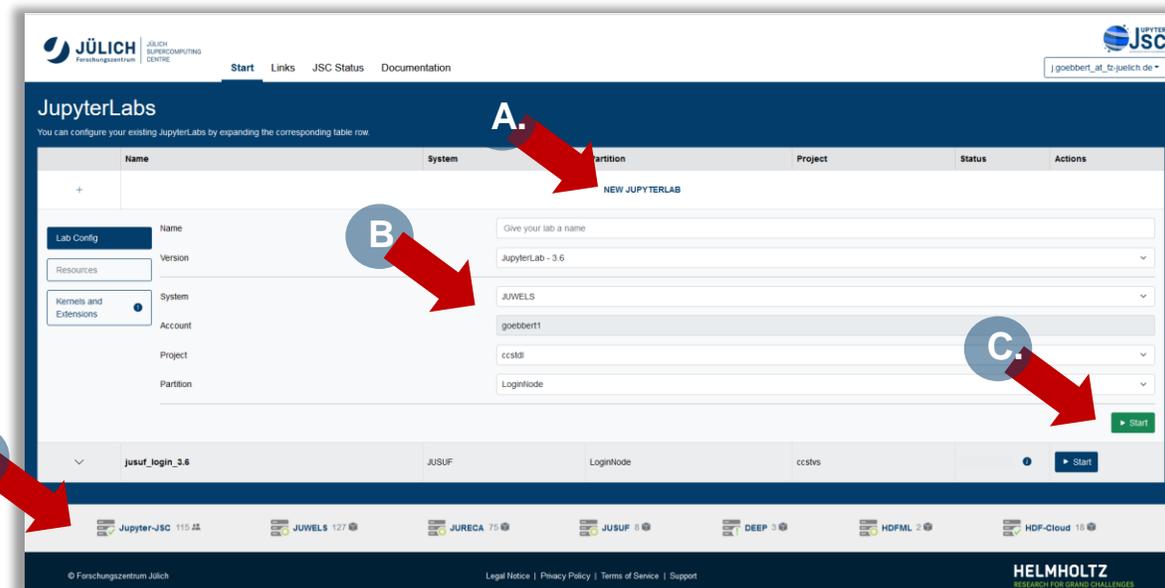
- Start/Open/Stop a JupyterLab
- Change/Delete **configuration**



### D. Statusbar

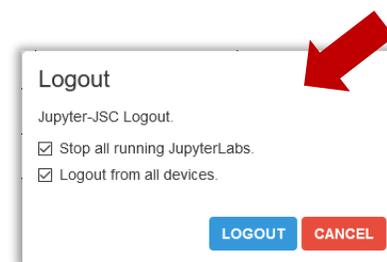


- Shows, (hover to get more details)
  - Number of active users in the last 24h
  - Number of running JupyterLabs
- Click to see system status page



### E. Logout

- Logout will ask what you want to do with the running JupyterLabs – be careful what you answer!



# JUPYTER-JSC WEBSERVICE

## JupyterLab Configuration

### Jupyter-JSC – Configuration

Available options **depend on**

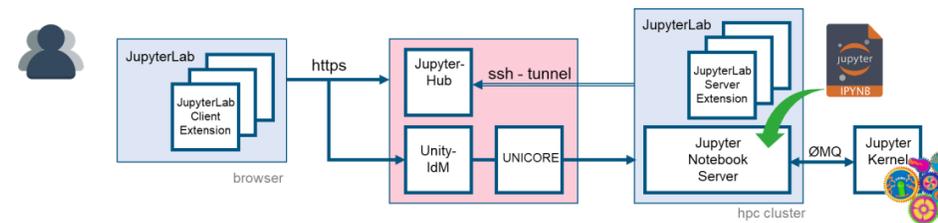
- user account settings visible in [judoor.fz-juelich.de](https://judoor.fz-juelich.de)
- system specific usage agreement on JuDoor is signed (!!!)
- currently available systems in all of your projects

### Basic options

- Version:  
multiple versions of JupyterLab are installed
- System:  
JUWELS, JURECA, JUSUF, DEEP, HDFML, HDF-Cloud
- Account:  
In general users only have a single account
- Project:  
project which have access to the selected system
- Partition:  
partition which are accessible by the project  
(this includes the decision for LoginNode and ComputeNode)

### Extra options

- Partition == compute      Resources
- Kernel and Extensions      non-default JupyterKernel, Extensions, Proxies



Name	System	Partition	Project	Status	Actions
NEW JUPYTERLAB					
Name					
Version					
System					
Account					
Project					
Partition					

**Lab Config**

**Resources**

**Kernels and Extensions**

**Login Nodes**

- LoginNode
- LoginNodeBooster
- LoginNodeVis

**Compute Nodes**

- batch
- devel
- develgpu
- gpus
- mem192

**Start**

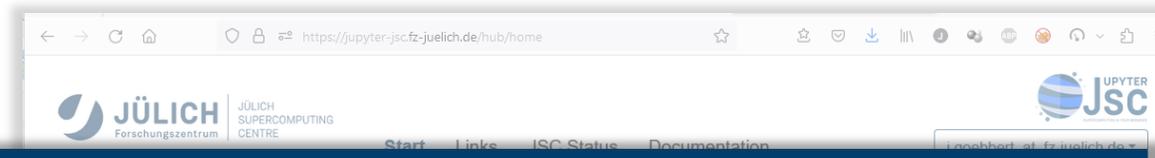
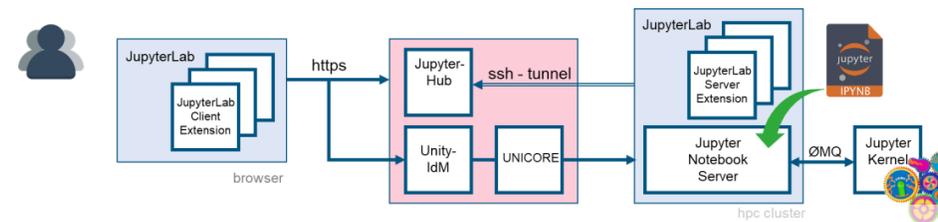
# JUPYTER-JSC WEBSERVICE

## JupyterLab Configuration

### Jupyter-JSC – Configuration

Available options **depend on**

- user account settings visible in [judoor.fz-juelich.de](https://judoor.fz-juelich.de)
- system specific usage agreement on JuDoor is signed (!!!)



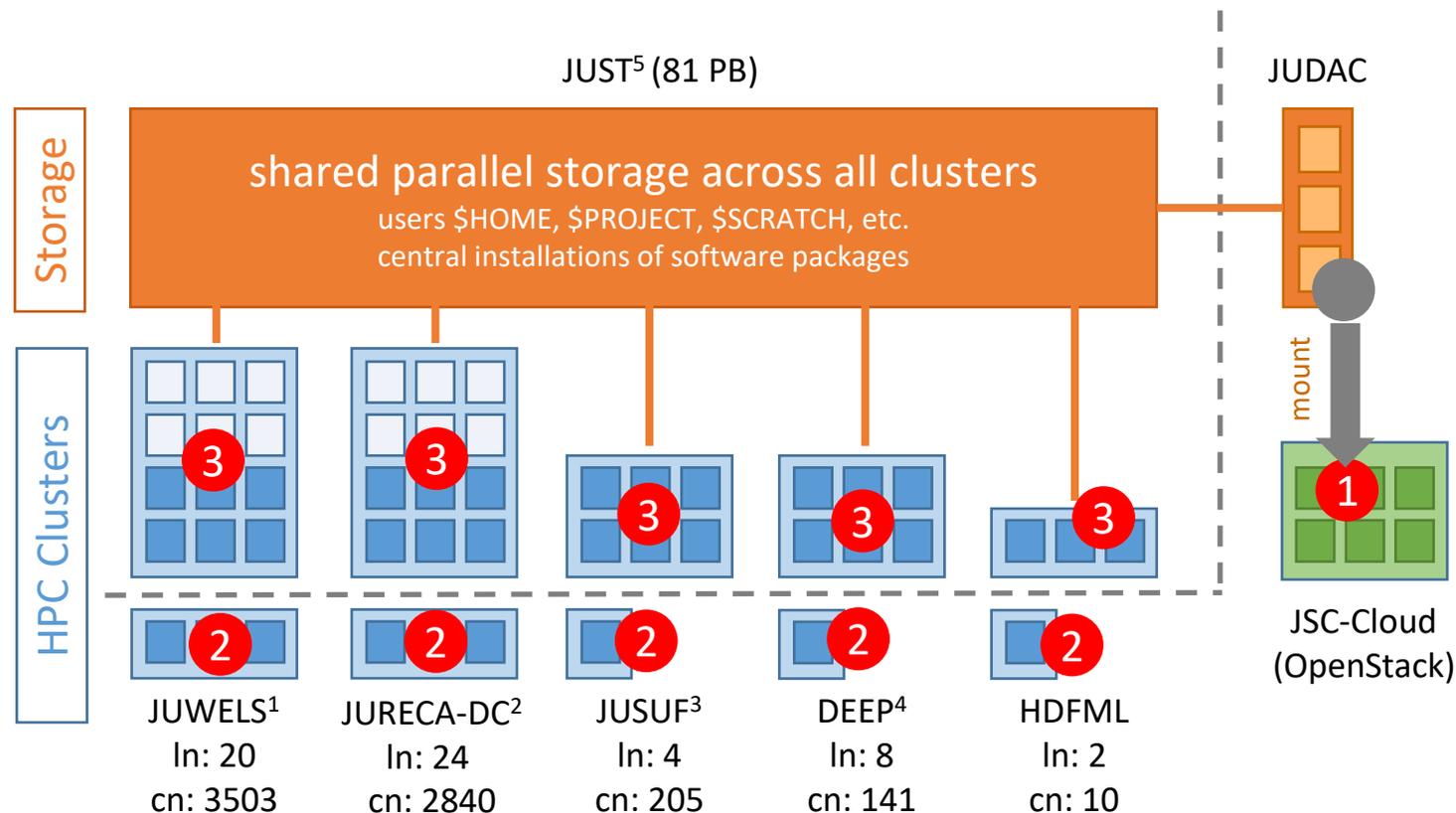
	Name	System	Partition	Project	Status	Actions
^	jusuf_login_3.4	JUSUF	LoginNode	ccstdl		<a href="#">Retry</a>
Service	<ul style="list-style-type: none"> <li>▶ 2023_03_14 14:04:23.887: Sending request to backend service to start your service on JUSUF.</li> <li>▼ 2023_03_14 14:04:33.724: Setup ssh port-forwarding. Create ssh tunnel with system user ljupyter. JupyterHub will then be able to connect to JupyterLab at jsfl05i:52243</li> <li>▼ 2023_03_14 14:04:34.044: <b>Cancel in progress</b> We're stopping your service. This may take a few seconds.</li> <li>▼ 2023_03_14 14:04:34.044: 2023_03_14 14:04:34.044: <b>Could not setup tunnel</b> Request identification: d2f8bd07a10f4534a9897f568ef3cbcb</li> </ul>					
Options						
Resources						
Reservation						
Kernels and Extensions						
Logs						

### Extra options

- Partition == compute      Resources
- Kernel and Extensions      non-default JupyterKernel, Extensions, Proxies

LoginNode
LoginNodeBooster
LoginNodeVis
<b>Compute Nodes</b>
batch
devel
develgpu
gpu
mem192

# JUPYTERLAB EVERYWHERE



no. login nodes = In  
no. compute nodes = cn

[1] <https://apps.fz-juelich.de/jsc/hps/juwels/configuration.html>

[2] <https://apps.fz-juelich.de/jsc/hps/jureca/configuration.html>

[3] <https://apps.fz-juelich.de/jsc/hps/jusuf/cluster/configuration.html>

[4] [https://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/DEEP-EST/\\_node.html](https://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/DEEP-EST/_node.html)

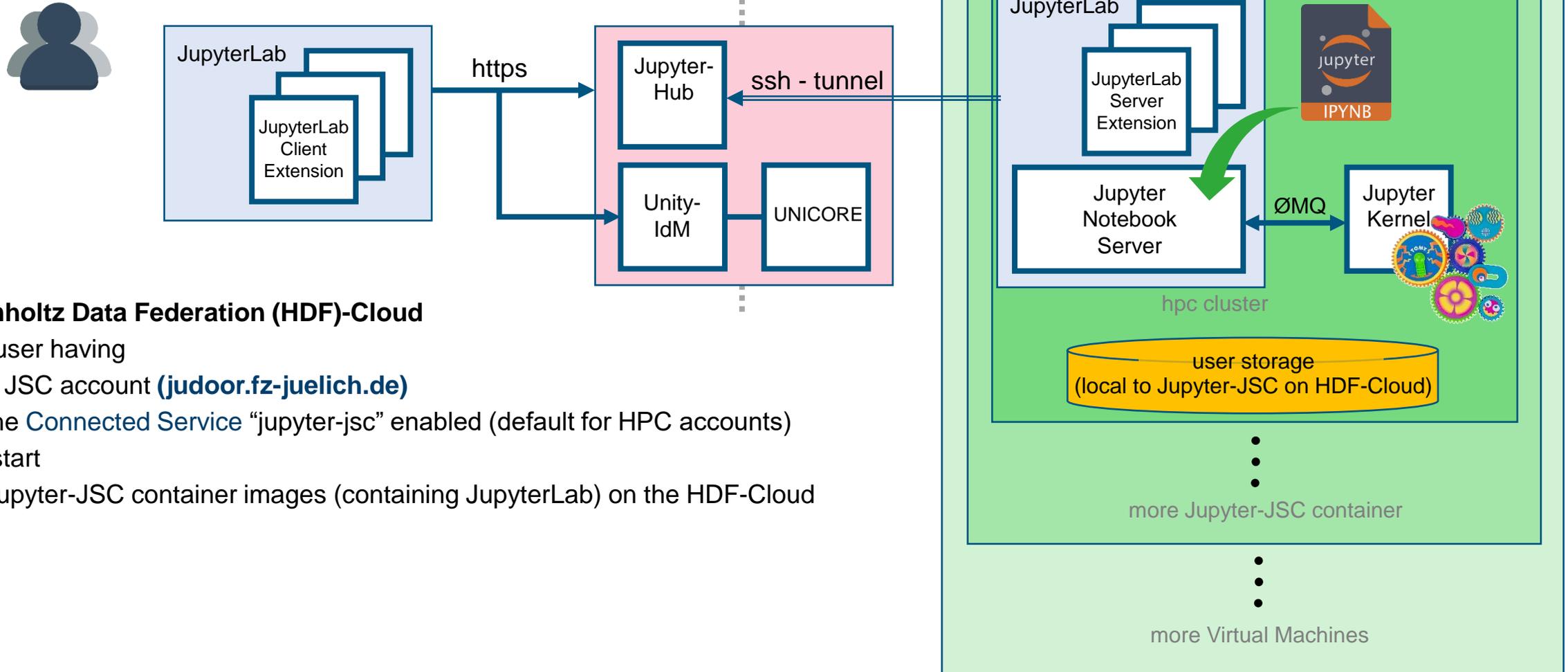
[5] [https://www.fz-juelich.de/ias/jsc/EN/Expertise/Datamanagement/OnlineStorage/JUST/Configuration/Configuration\\_node.html](https://www.fz-juelich.de/ias/jsc/EN/Expertise/Datamanagement/OnlineStorage/JUST/Configuration/Configuration_node.html)

## JupyterLab everywhere

- 1 JupyterLab on cloud
- 2 JupyterLab on login nodes
- 3 JupyterLab on compute nodes

# JUPYTER-JSC WEBSERVICE

System: HDF-Cloud



## Helmholtz Data Federation (HDF)-Cloud

Any user having

- a JSC account ([judoor.fz-juelich.de](https://www.fz-juelich.de/ias/jsc))
  - the Connected Service “jupyter-jsc” enabled (default for HPC accounts)
- can start
- Jupyter-JSC container images (containing JupyterLab) on the HDF-Cloud

# JUPYTER-JSC WEBSERVICE

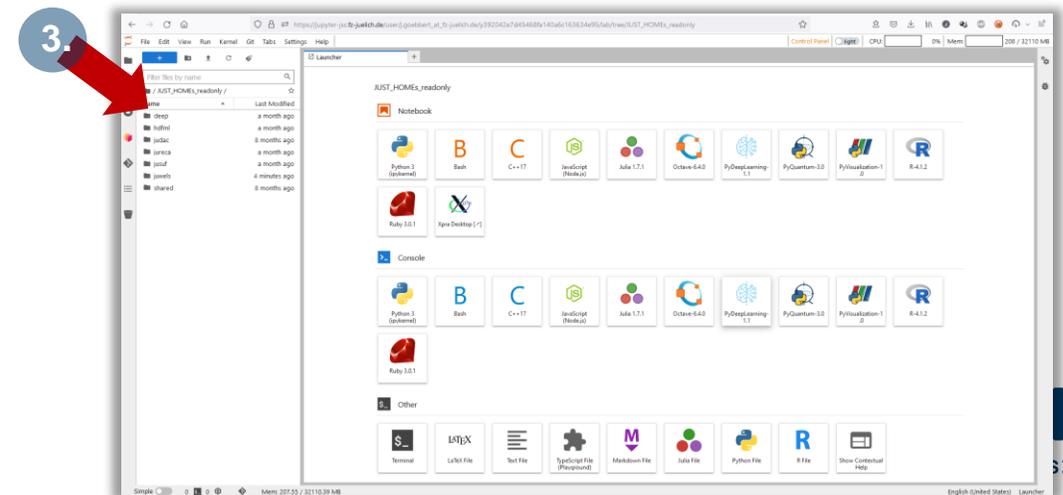
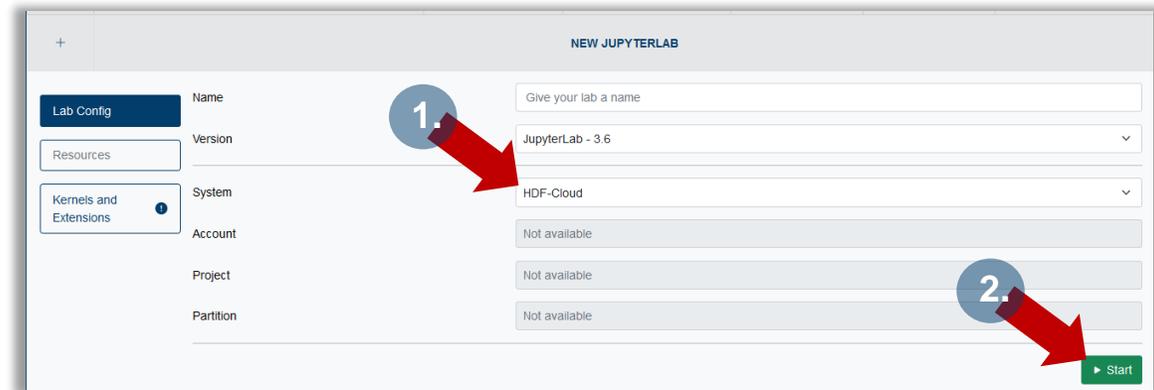
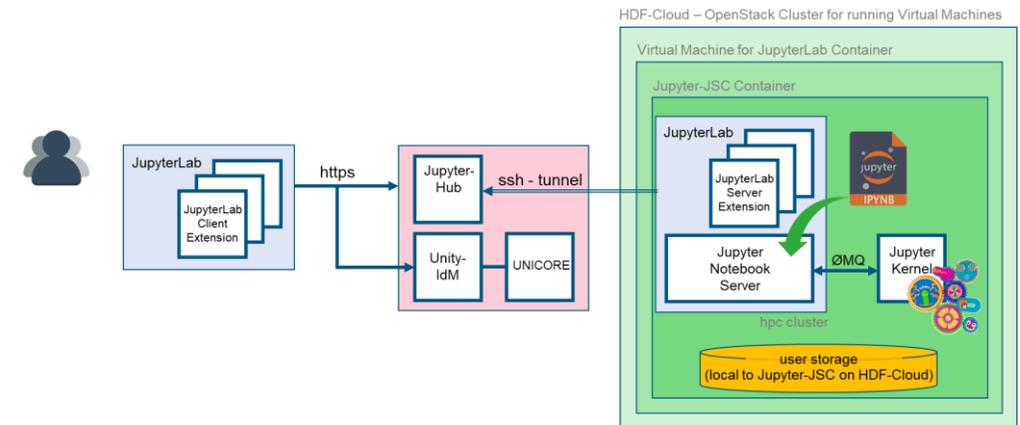
## System: HDF-Cloud

### Start JupyterLab on HDF-Cloud

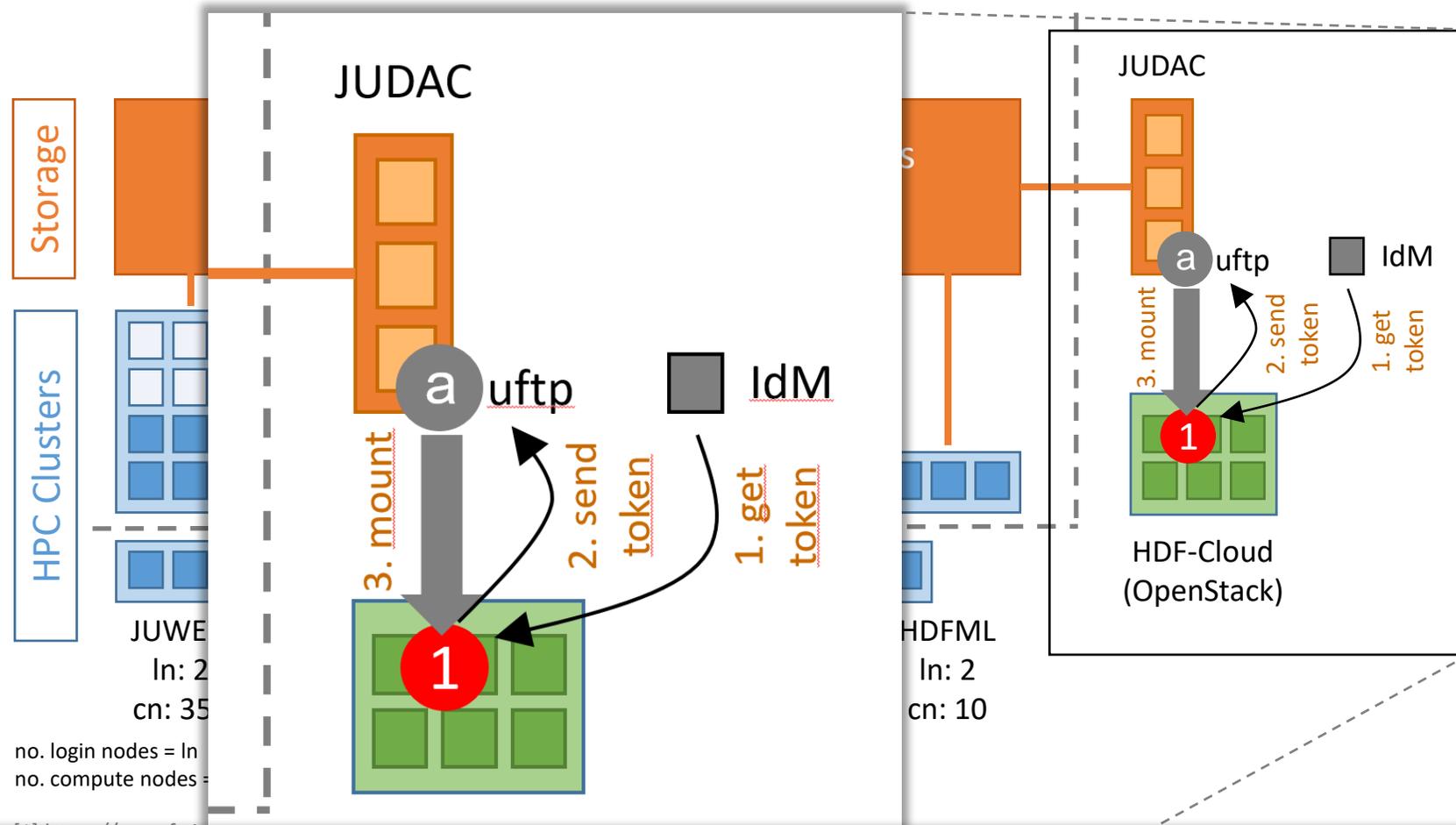
- Requirements:
  - Registered JSC account at <https://judoor.fz-juelich.de>
  - Logged in to Jupyter-JSC at <https://jupyter-jsc.fz-juelich.de>
  - Named a new JupyterLab configuration
- Start a JupyterLab:
  - Version == "JupyterLab 3.6"
  - System == "HDF-Cloud"

### Limitations on JupyterLab on HDF-Cloud

- max. **4 GB** memory
  - ATTENTION:  
the container automatically stops, when this limit is reached.
- Storage in Jupyter-JSC container
  - is **local** to the HDF-Cloud
  - HPC \$HOMEs are mounted read-only
  - only accessible from a Jupyter-JSC container
- HDF-Cloud has **no GPUs**



# HOW TO MOUNT GPFS ON HDF-CLOUD



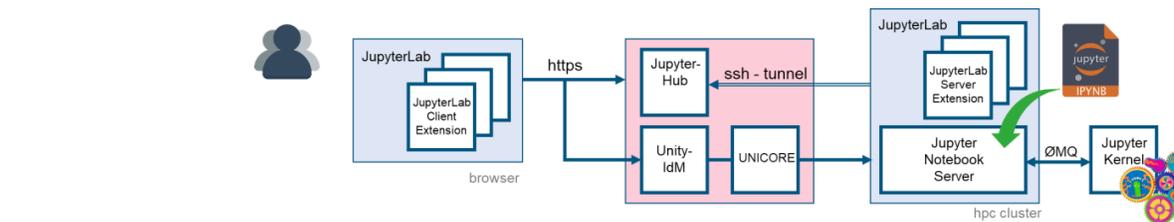
[https://gitlab.jsc.fz-juelich.de/jupyter4jsc/training-2023.04-jupyter4hpc/-/blob/main/day2\\_hpcenv/7\\_cloud-hpc\\_challenges/1-hdf-cloud\\_mount-hpc-storage.ipynb](https://gitlab.jsc.fz-juelich.de/jupyter4jsc/training-2023.04-jupyter4hpc/-/blob/main/day2_hpcenv/7_cloud-hpc_challenges/1-hdf-cloud_mount-hpc-storage.ipynb)

# JUPYTER-JSC SECRETS

## Very important to know

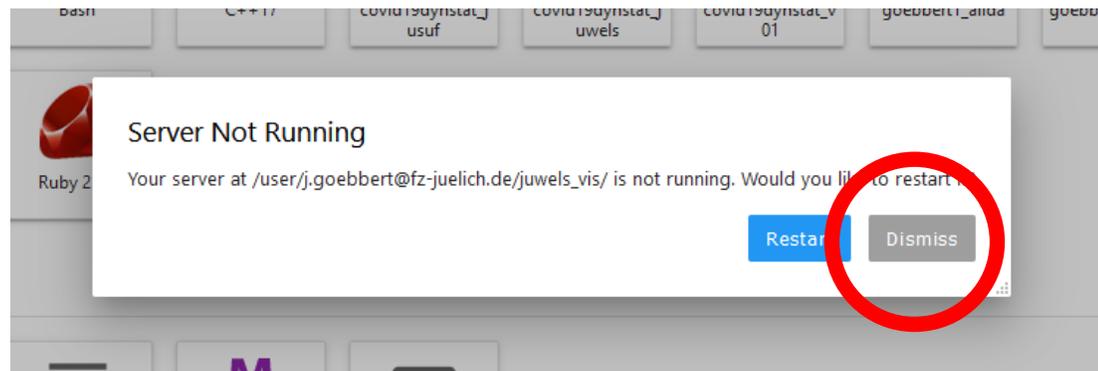
### Secret 1: Support button

- Let us know, if something does not work.  
We can only fix it, if we know it.

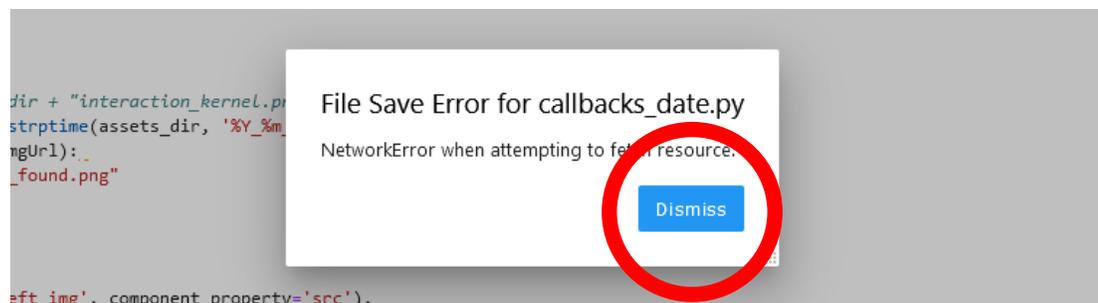


### Secret 2: Reload on connection loss

- “Server Not Running”  
means, that your browser just lost connection  
=> **Just hit “Dismiss” !!!**  
(as soon as you are online again)



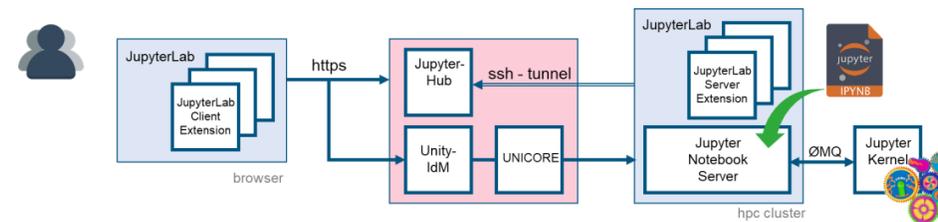
- “File Save Error for <...>”  
means, that your browser just lost connection  
=> **Just hit “Dismiss” !!!**  
(as soon as you are online again)



You can **always** safely hit the “Reload” button of your browser, if the connection to JupyterLab ever gets lost.  
(it will just restart JupyterLab on the browser-site)

# JUPYTER-JSC SECRETS

For experts only 😊

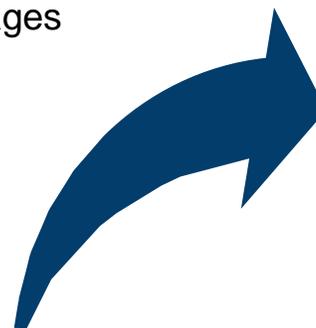


## Secret 3: Jupyter-JSC logs

- Jupyter-Lab gets started by UNICORE on our HPC systems
- On startup UNICORE created the directory `$SCRATCH_<project>/unicore-jobs/<random-hash>/`
  - In the terminal of a running JupyterLab, this directory is `$JUPYTER_LOG_DIR`
- In this directory you find
  - `stdout` -> terminal output of jupyterlab messages
  - `stderr` -> terminal output of jupyterlab error messages
  - `.start` -> details how your JupyterLab got started

## Secret 4: change to a different JupyterLab version

- In `.start` you can see, that
  - `$HOME/.jupyter/start_jupyter-jsc.sh` is used to prepare the environment for JupyterLab. This script must ensure that the command `jupyter` is available in `$PATH`.



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

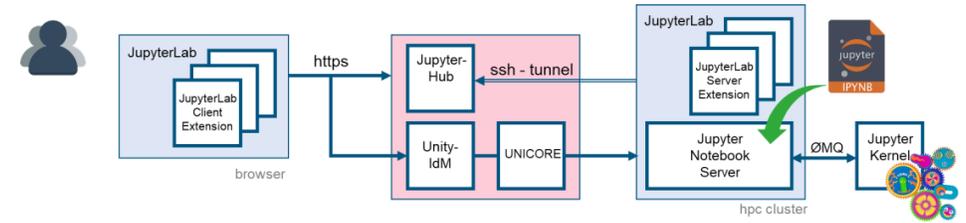
```
module purge  
module load Stages/2023  
module load GCCcore/.11.3.0  
module load JupyterCollection/2023.3.6
```

Switch to a customized JupyterLab with  
`$HOME/.jupyter/start_jupyter-jsc.sh`

It enables you to switch to an older/newer/other version of JupyterLab, if the default one gives you trouble or is missing features.

# JUPYTER-JSC WEBSERVICE

## Some comments about the UI



open filebrowser

open launcher

tutorials & examples

sidebar with core and extensions features

indicates active notebook cell

type of active notebook cell

no close, but go back to Jupyter-JSC's control panel

memory consumption (keep an eye on that!)

Type of Jupyter kernel this notebook is connected to (click to change)

notebook cell

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

Se **[\*]** indicates that cell was send to Jupyter kernel for execution

- change if you like

```
[*]: # INPUT NEEDED:
KERNEL_NAME=${USER}_kernel

export KERNEL_NAME=$(echo "${KERNEL_NAME}" | awk '{print tolower($0)}')
echo ${KERNEL_NAME} # double check
```

- List directories where JupyterLab will search for kernels

```
[ ] JUPYTER_SEARCH_PATH (for kernels-directory)
# Jupyter search paths for kernels-directories"
if [ -n "$JUPYTER_PATH" ]; then
echo "$HOME/.local/share/jupyter"
```

**[ ]** indicates that cell has never been executed by the connected Jupyter kernel

Using project kernels is need to be enable for your project first by our Jupyter-jsc admins.

# JUPYTERLAB EXTENSIONS

# JUPYTERLAB EXTENSIONS

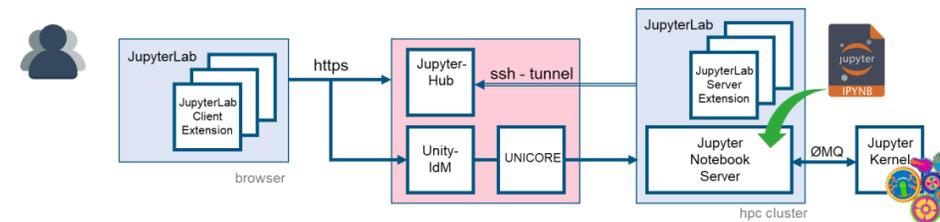
## Some general information

### List the installed JupyterLab extensions

- Open the Launcher
- Start a Terminal
- Run command `jupyter labextension list`

### Extensions are installed in JupyterLab's Application Directory, which

- stores any information that JupyterLab persists
  - including settings and built assets of extensions
- default location is `<sys-prefix>/share/jupyter/lab`
- can be relocated by setting `$JUPYTERLAB_DIR`
  - contains the JupyterLab static assets
    - (e.g. `static/index.html`)
  - **JupyterLab < 3:**  
any change requires a rebuild of the whole JupyterLab to take effect!
  - **JupyterLab >= 3:**  
introduced prebuild extensions, which are loaded at startup time



```
[goebbert1@jrllogin04 jureca]$ jupyter labextension list
JupyterLab v3.2.1
/p/software/jurecadc/stages/2020/software/Jupyter/2021.3.2-gcccoreml-10.3.0-2021.2.0-Py
jupyterlab-iframe v0.4.0 enabled OK
jupyter-leaflet v0.14.0 enabled OK
ipyvolume v0.6.0-alpha.8 enabled OK
jupyterlab-system-monitor v0.8.0 enabled OK (python, jupyterlab-system-monitor)
jupyterlab-gitlab v3.0.0 enabled OK (python, jupyterlab-gitlab)
jupyterlab-topbar-extension v0.6.1 enabled OK (python, jupyterlab-topbar)
dask-labextension v5.1.0 enabled OK (python, dask_labextension)
jupyterlab-plotly v5.3.1 enabled OK
jupyter-vue v1.6.1 enabled OK
...
Other labextensions (built into JupyterLab)
app dir: /p/software/jurecadc/stages/2020/software/Jupyter/2021.3.2-gcccoreml-10.3.0-2021.2.0-python-3.8.5/share/jupyter/lab
jupyterlab-dash v0.4.0 enabled OK
jupyterlab-theme-toggle v0.6.1 enabled OK
[goebbert1@jrllogin04 jureca]$
```

<https://jupyterlab.readthedocs.io/en/stable/user/extensions.html>

### Hint: JupyterLab Playground

A JupyterLab extension to write and load simple JupyterLab plugins inside JupyterLab.

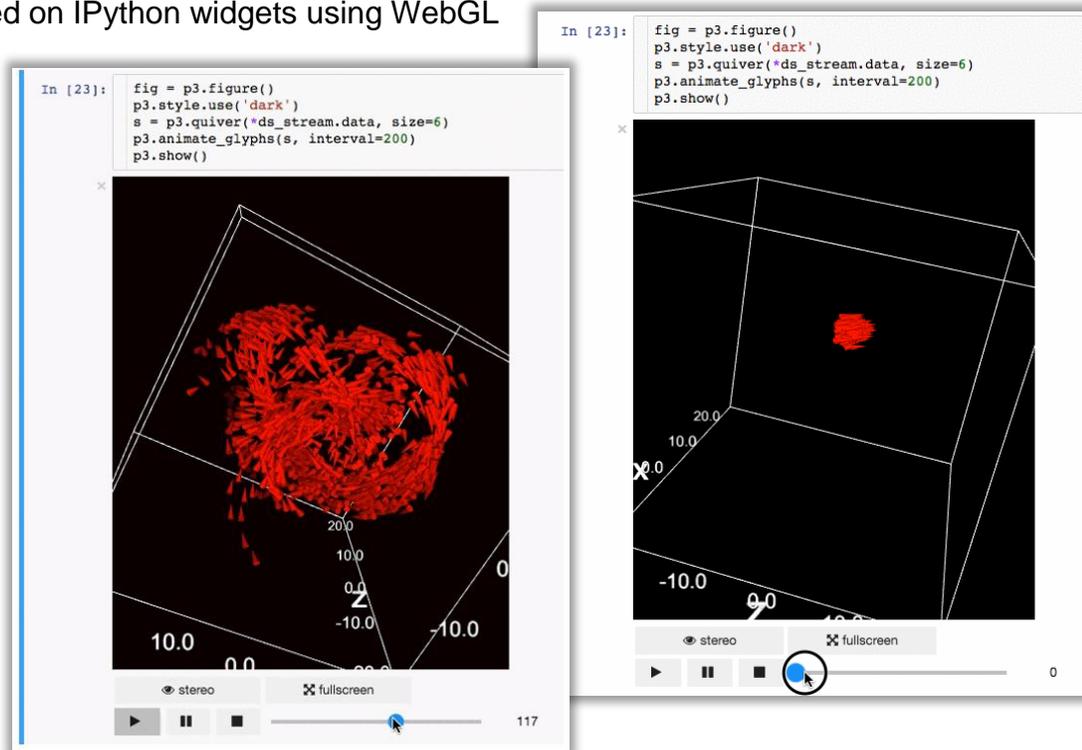
<https://github.com/jupyterlab/jupyterlab-plugin-playground>

# JUPYTERLAB EXTENSIONS

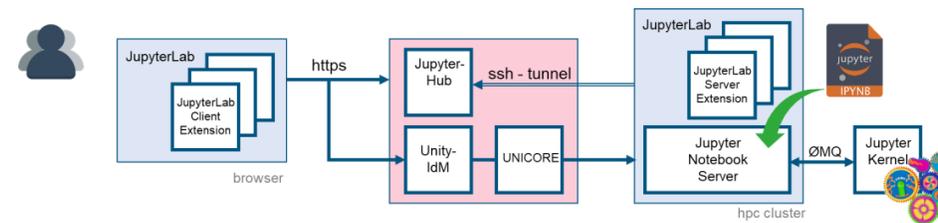
Installed by default at Jupyter-JSC

## IPyVolume

3d plotting for Python in the Jupyter notebook based on IPython widgets using WebGL

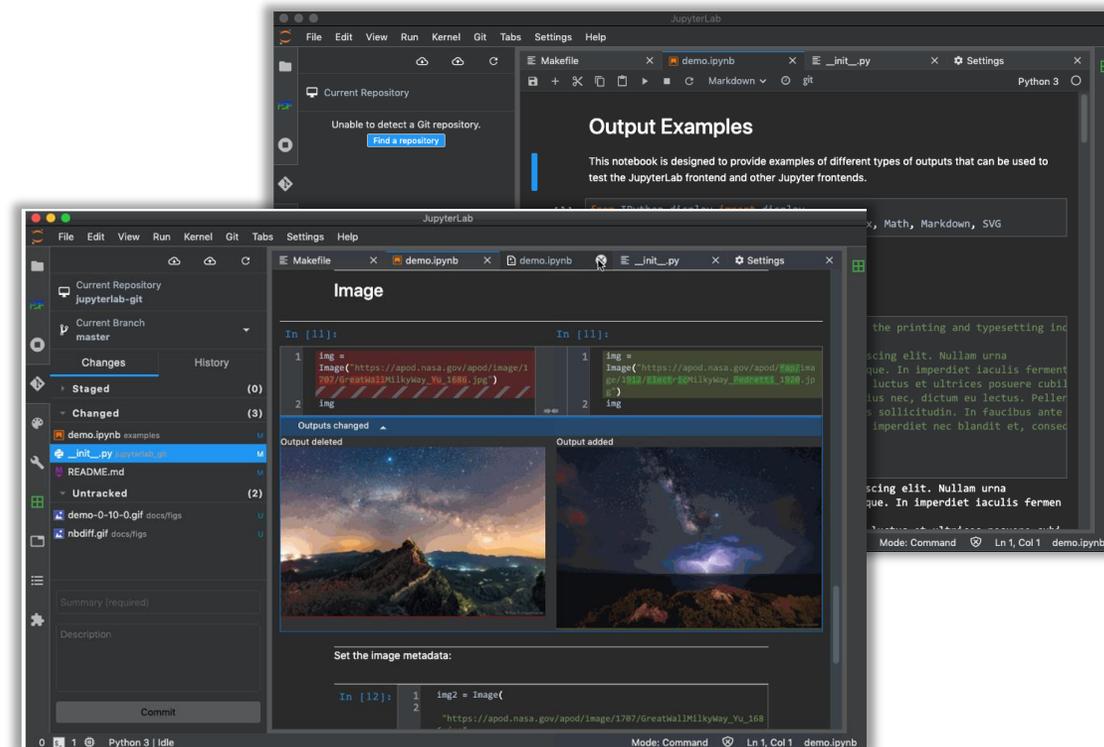


<https://github.com/maartenbreddels/ipyvolume>



## JupyterLab-Git

JupyterLab extension for version control using Git



<https://github.com/jupyterlab/jupyterlab-git>

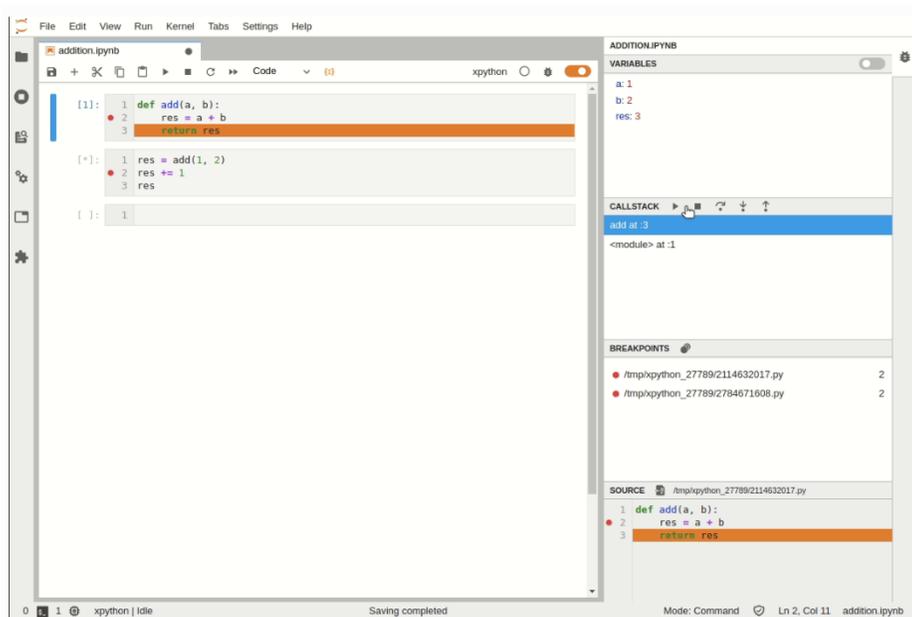
# JUPYTERLAB EXTENSIONS

## Installed by default at Jupyter-JSC

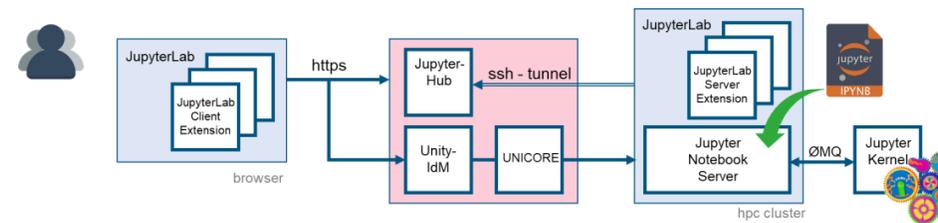
### JupyterLab - Visual Debugger

JupyterLab >= 3 ships with a Debugger front-end by default.

This means that notebooks, code consoles and files can now be debugged from JupyterLab directly! For the debugger to be enabled and visible, a kernel with support for debugging is required.

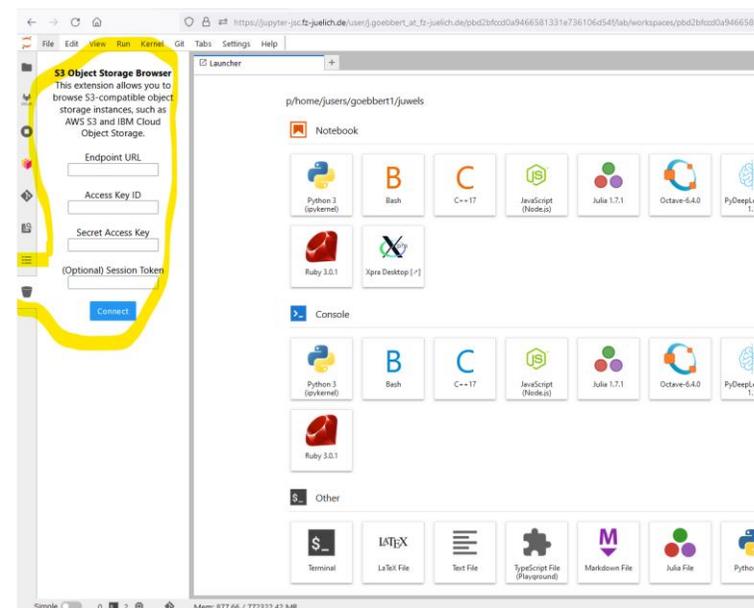


<https://jupyterlab.readthedocs.io/en/stable/user/debugger.html>



### JupyterLab-S3-browser

A JupyterLab extension for browsing S3-compatible object storage



<https://github.com/IBM/jupyterlab-s3-browser>

# JUPYTERLAB EXTENSIONS

## Installed by default at Jupyter-JSC

### PyThreeJS

A Python / ThreeJS bridge utilizing the Jupyter widget infrastructure.  
<https://threejs.org> - lightweight, 3D library with a default WebGL renderer.

```
In [9]: f = """
function f(origu,origv) {
  // scale u and v to the ranges I want: [0, 2*pi]
  var u = 2*Math.PI*origu;
  var v = 2*Math.PI*origv;

  var x = Math.sin(u);
  var y = Math.cos(v);
  var z = Math.cos(u+v);

  return new THREE.Vector3(x,y,z);
}
"""
surf_g = ParametricGeometry(func=f);

surf = Mesh(geometry=surf_g, material=LambertMaterial(color='green', side='FrontSide'))
surf2 = Mesh(geometry=surf_g, material=LambertMaterial(color='yellow', side='BackSide'))
scene = Scene(children=[surf, surf2, AmbientLight(color='#777777')])
c = PerspectiveCamera(position=[5, 5, 3], up=[0, 0, 1],
                      children=[DirectionalLight(color='white',
                                                position=[3, 5, 1],
                                                intensity=0.6)])

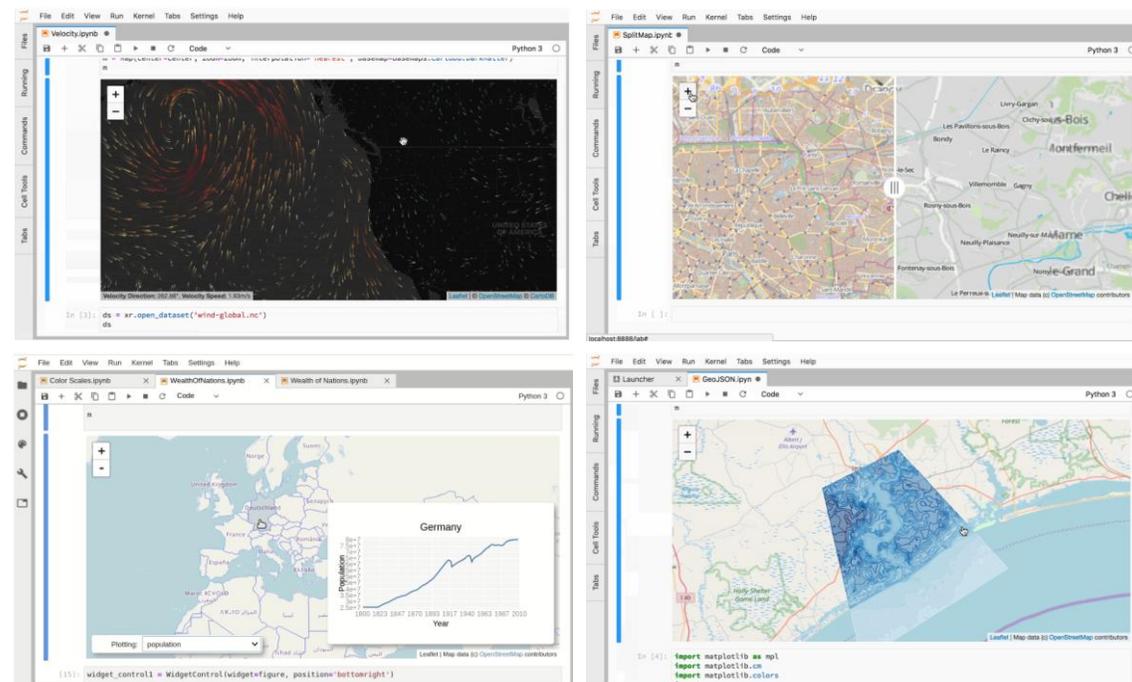
renderer = Renderer(camera=c, scene=scene, controls=[OrbitControls(controlling=c)])
display(renderer)
```



<https://github.com/jupyter-widgets/pythreejs>

### IPyLeaflet

A Jupyter / Leaflet bridge enabling interactive maps in the Jupyter notebook.



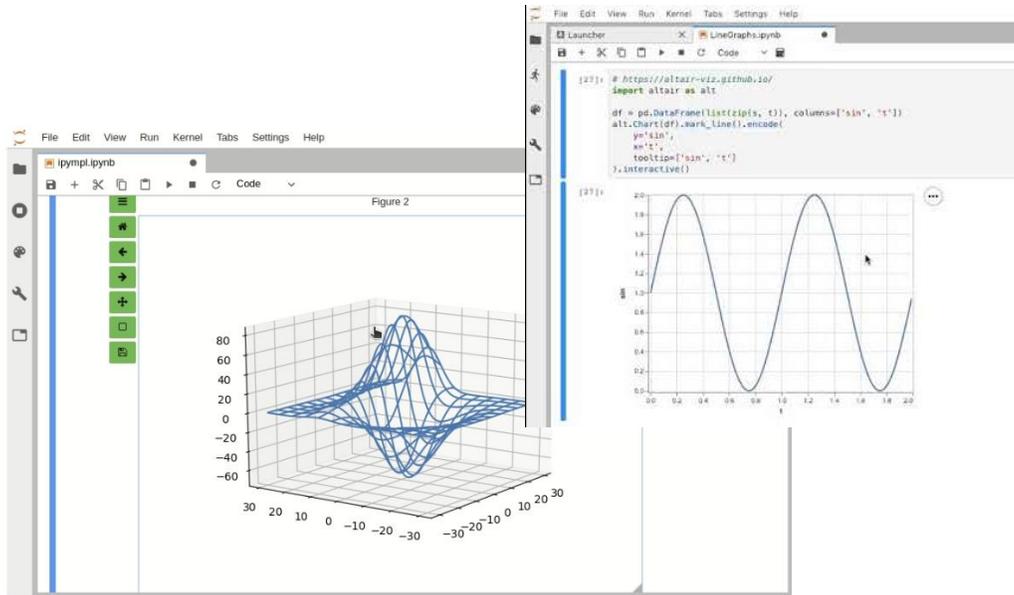
<https://github.com/jupyter-widgets/ipyleaflet>

# JUPYTERLAB EXTENSIONS

## Installed by default at Jupyter-JSC

### IPyMPL - matplotlib

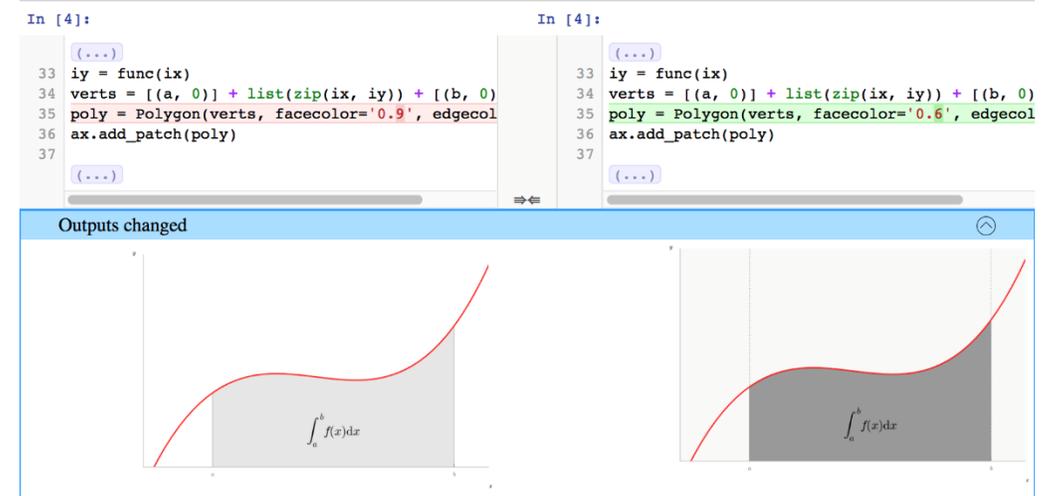
Leveraging the Jupyter interactive widgets framework, ipympl enables the interactive features of matplotlib in the Jupyter notebook and in JupyterLab.



<https://github.com/matplotlib/ipympl>

### NBDime

Tools for diffing and merging of Jupyter notebooks.



<https://github.com/jupyter/nbdime>

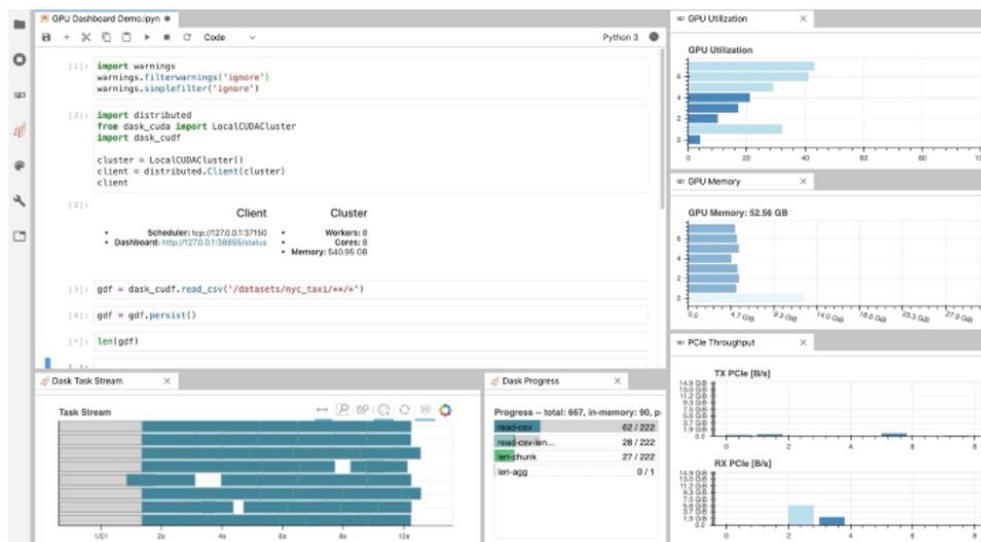


# JUPYTERLAB EXTENSIONS

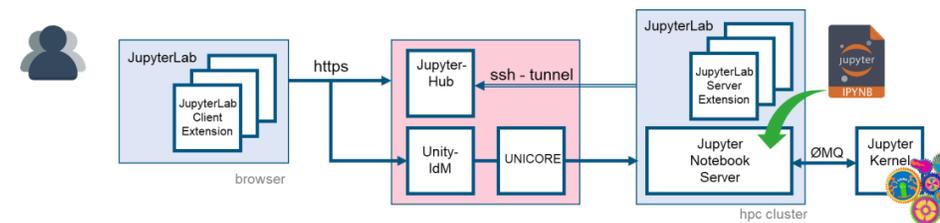
## Installed by default at Jupyter-JSC

### NVDashboard

NVDashboard is an open-source package for the real-time visualization of NVIDIA GPU metrics in interactive Jupyter Lab environments.

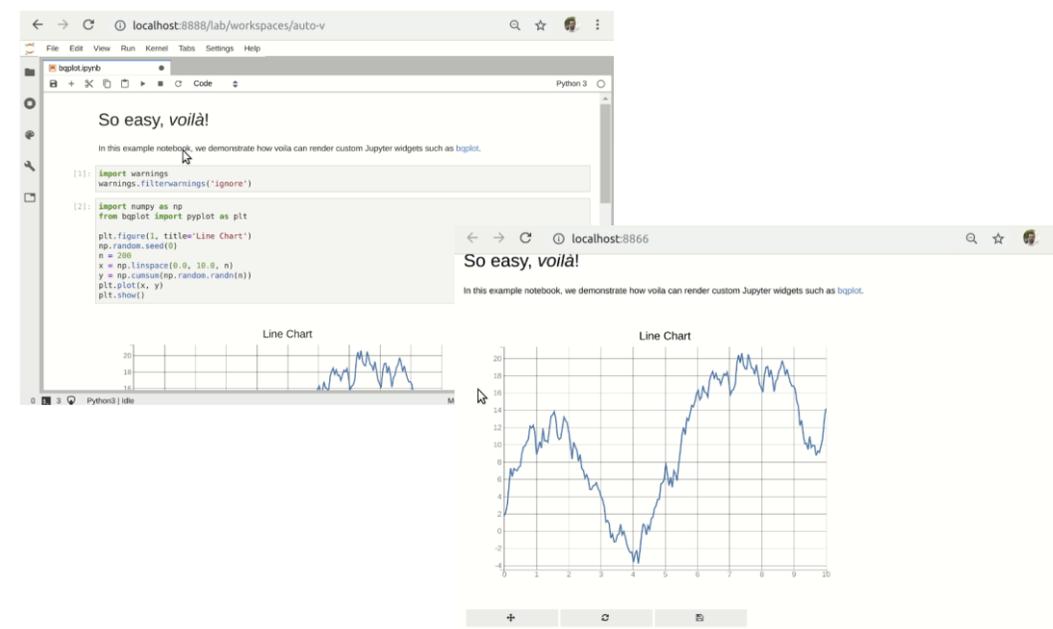


<https://github.com/rapidsai/jupyterlab-nvdashboard>  
<https://developer.nvidia.com/blog/gpu-dashboards-in-jupyter-lab/>



### Voilà

Voilà turns Jupyter notebooks into standalone web applications.



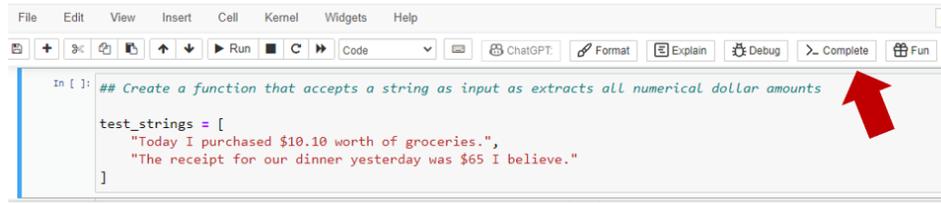
<https://github.com/voila-dashboards/voila>

# JUPYTERLAB EXTENSIONS

## ... more useful extensions

### ChatGPT for Jupyter

A browser extension to provide various helper functions in Jupyter Notebooks and Jupyter Lab, powered by ChatGPT.



```
In [ ]: ## Create a function that accepts a string as input as extracts all numerical dollar amounts

test_strings = [
    "Today I purchased $10.10 worth of groceries.",
    "The receipt for our dinner yesterday was $65 I believe."
]
```

#### ChatGPT - Complete Code

Here is the code that extracts all numerical dollar amounts from a string:

```
import re

def extract_dollar_amounts(string):
    dollar_amounts = re.findall(r'\$\d+(?!\.\d+)?', string)
    return dollar_amounts

test_strings = [
    "Today I purchased $10.10 worth of groceries.",
    "The receipt for our dinner yesterday was $65 I believe."
]

for test_string in test_strings:
    dollar_amounts = extract_dollar_amounts(test_string)
    print(f"Dollar amounts in '{test_string}': {dollar_amounts}")
```

This code will output the following:

```
Dollar amounts in 'Today I purchased $10.10 worth of groceries.': ['$10.10']
Dollar amounts in 'The receipt for our dinner yesterday was $65 I believe.': ['$65']
```

<https://github.com/TiesdeKok/chat-gpt-jupyter-extension>

# JUPYTERLAB EXTENSIONS

## Installed by default at Jupyter-JSC

Core packages	Version	Link	Description
jupyterlab	3.6.5	<a href="https://jupyterlab.readthedocs.io/en/3.6.x/">https://jupyterlab.readthedocs.io/en/3.6.x/</a>	-
nbdclassic	1.0.0	<a href="https://nbdclassic.readthedocs.io">https://nbdclassic.readthedocs.io</a>	Jupyter Notebook as a Jupyter Server extension
jupyterlab_server	2.23.0	<a href="https://jupyterlab-server.readthedocs.io">https://jupyterlab-server.readthedocs.io</a>	Server components for JupyterLab applications
jupyterhub	3.1.1	<a href="https://jupyterhub.readthedocs.io">https://jupyterhub.readthedocs.io</a>	Multi-user server for Jupyter notebooks

Optional extension	Version	Link	Description
jupyterlab-nvdashboard	0.8.0	<a href="https://github.com/rapidsai/jupyterlab-nvdashboard">https://github.com/rapidsai/jupyterlab-nvdashboard</a> , <a href="#">eb-file</a>	A JupyterLab extension for displaying dashboards of GPU usage.
jupyter-slurm-provisioner	0.6.0	<a href="https://github.com/FZJ-JSC/jupyter-slurm-provisioner">https://github.com/FZJ-JSC/jupyter-slurm-provisioner</a> , <a href="#">eb-file</a>	Allows to start Jupyter kernels as a SLURM job remote from the Jupyter server
nglview	3.0.6	<a href="http://nglviewer.org/nglview/latest/">http://nglviewer.org/nglview/latest/</a> , <a href="#">eb-file</a>	Jupyter widget to interactively view molecular structures and trajectories
jupyter-ai	0.9.0	<a href="https://jupyter-ai.readthedocs.io/">https://jupyter-ai.readthedocs.io/</a> , <a href="#">eb-file</a>	A generative AI extension for JupyterLab

Core Kernels	Version	Link	Description
Bash	0.9.0	<a href="https://github.com/takuyver/bash_kernel">https://github.com/takuyver/bash_kernel</a> , <a href="#">eb-file</a>	A bash kernel for IPython
Cling (C++)	20230205	<a href="https://github.com/root-project/cling/">https://github.com/root-project/cling/</a> , <a href="#">eb-file</a>	Jupyter kernel for the C++ programming language
Julia	1.8.5	<a href="https://github.com/JuliaPy/pyjulia">https://github.com/JuliaPy/pyjulia</a> , <a href="#">eb-file</a>	python interface to julia
LFortran	0.19.0	<a href="https://fortran.org/">https://fortran.org/</a> , <a href="#">eb-file</a>	Modern interactive LLVM-based Fortran compiler
Octave	8.2.0	<a href="https://www.octave.org/">https://www.octave.org/</a> , <a href="#">eb-file</a>	Scientific Programming Language - Powerful mathematics-oriented syntax with built-in 2D/3D plotting and visualization tools
R	4.2.1	<a href="https://irkernel.github.io">https://irkernel.github.io</a> , <a href="#">eb-file</a>	R kernel for Jupyter
Ruby	3.0.5	<a href="https://github.com/SciRuby/iruby">https://github.com/SciRuby/iruby</a> , <a href="#">eb-file</a>	Ruby kernel for Jupyter

Community Kernels	Version	Link	Description
DeepLearning	2023.5	<a href="#">eb-file</a>	Python kernel incl. a collection of extra modules/packages for Deep Learning
PyEarthSystem	2023.5	<a href="#">eb-file</a>	Python kernel incl. a collection of extra modules/packages for the Earth System community
QuantumComputing	2023.5	<a href="#">eb-file</a>	Python kernel incl. a collection of extra modules/packages for the quantum computing community
Visualization	2023.5	<a href="#">eb-file</a>	Python kernel incl. a collection of extra modules/packages for visualization

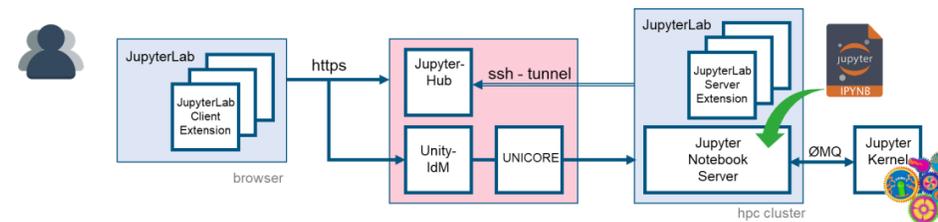
JupyterLab Applications	Version	Link	Description
Xpra	4.4.6	<a href="https://xpra.org">https://xpra.org</a> <a href="#">eb-file</a>	Remote desktop for X11 applications in the browser

Core Extensions	Version	Link	Description
jupyter-server-proxy	4.0.0	<a href="https://jupyter-server-proxy.readthedocs.io">https://jupyter-server-proxy.readthedocs.io</a>	Jupyter notebook server extension to proxy web services.
jupyterlab-lsp	4.2.0	<a href="https://github.com/jupyterlab-lsp/jupyterlab-lsp">https://github.com/jupyterlab-lsp/jupyterlab-lsp</a>	Coding assistance for JupyterLab using Language Server Protocol
ipyml	0.9.3	<a href="https://matplotlib.org/ipyml/">https://matplotlib.org/ipyml/</a>	Interactive features of matplotlib in Jupyter
ipyleaflet	0.17.3	<a href="https://ipyleaflet.readthedocs.io">https://ipyleaflet.readthedocs.io</a>	Interactive maps in the Jupyter notebook
bqplot	0.12.39	<a href="https://bqplot.github.io/bqplot/">https://bqplot.github.io/bqplot/</a>	Plotting library for IPython/Jupyter notebooks
ipyvolume	0.6.3	<a href="https://github.com/widgetti/ipyvolume">https://github.com/widgetti/ipyvolume</a>	3d plotting for Python in the Jupyter notebook based on IPython widgets using WebGL
jupyterlab_gitlab	3.0.0	<a href="https://gitlab.com/beenje/jupyterlab-gitlab">https://gitlab.com/beenje/jupyterlab-gitlab</a>	A JupyterLab extension for browsing GitLab repositories
jupyterlab_git	0.41.0	<a href="https://github.com/jupyterlab/jupyterlab-git">https://github.com/jupyterlab/jupyterlab-git</a>	A Git extension for JupyterLab
nbdime	3.2.1	<a href="https://nbdime.readthedocs.io/">https://nbdime.readthedocs.io/</a>	Tools for diffing and merging of Jupyter notebooks.
jupyterlab_latex	3.2.0	<a href="https://github.com/jupyterlab/jupyterlab-latex">https://github.com/jupyterlab/jupyterlab-latex</a>	JupyterLab extension for live editing of LaTeX documents
jupyterlab_s3_browser	0.13.0	<a href="https://github.com/IBM/jupyterlab-s3-browser">https://github.com/IBM/jupyterlab-s3-browser</a>	A JupyterLab extension for browsing S3-compatible object storage
plotly	5.15.0	<a href="https://plotly.com/python/">https://plotly.com/python/</a>	Python graphing library for interactive, publication-quality graphs.
jupyter_bokeh	3.0.4	<a href="https://github.com/bokeh/jupyter_bokeh">https://github.com/bokeh/jupyter_bokeh</a>	An extension for rendering Bokeh content in JupyterLab notebooks
panel	0.14.4	<a href="https://panel.holoviz.org/">https://panel.holoviz.org/</a>	The powerful data exploration & web app framework for Python
holoviews	1.16.0	<a href="https://holoviews.org/">https://holoviews.org/</a>	With Holoviews, your data visualizes itself.
jupyterlab_hdf	1.3.0	<a href="https://github.com/jupyterlab/jupyterlab-hdf5">https://github.com/jupyterlab/jupyterlab-hdf5</a>	Open and explore HDF5 files in JupyterLab. Can handle very large (TB) sized files, and datasets of any dimensionality
ipyparallel	8.6.1	<a href="https://ipyparallel.readthedocs.io">https://ipyparallel.readthedocs.io</a>	IPython Parallel: Interactive Parallel Computing in Python
dask_labextension	6.1.0	<a href="https://github.com/dask/dask-labextension">https://github.com/dask/dask-labextension</a>	JupyterLab extension for Dask
voila	0.5.0a4	<a href="https://voila.readthedocs.io">https://voila.readthedocs.io</a>	Voilà turns Jupyter notebooks into standalone web applications
nbdev	2.3.12	<a href="https://nbdev.fast.ai/">https://nbdev.fast.ai/</a>	Create delightful software with Jupyter Notebooks
sidecar	0.5.2	<a href="https://github.com/jupyter-widgets/jupyterlab-sidecar">https://github.com/jupyter-widgets/jupyterlab-sidecar</a>	A sidecar output widget for JupyterLab
dash	2.11.1	<a href="https://plotly.com/dash">https://plotly.com/dash</a>	Data Apps & Dashboards for Python. No JavaScript Required.
ipyvue	1.9.2	<a href="https://github.com/widgetti/ipyvue">https://github.com/widgetti/ipyvue</a>	Jupyter widgets base for Vue libraries
ipywebrtc	0.6.0	<a href="https://github.com/maartenbreddels/ipywebrtc">https://github.com/maartenbreddels/ipywebrtc</a>	WebRTC for Jupyter notebook/lab
jupyterlab-spellchecker	0.7.3	<a href="https://github.com/jupyterlab-contrib/spellchecker">https://github.com/jupyterlab-contrib/spellchecker</a>	Spellchecker for JupyterLab notebook markdown cells and file editor.
jupyterlab_code_formatter	1.6.1	<a href="https://github.com/ryantam626/jupyterlab_code_formatter">https://github.com/ryantam626/jupyterlab_code_formatter</a>	A JupyterLab plugin to facilitate invocation of code formatters.
jupyterlab_recents	3.2.0	<a href="https://github.com/NERSC/jupyterlab-recents">https://github.com/NERSC/jupyterlab-recents</a>	A JupyterLab extension that tracks recent files and directories.
jupyterlab-favorites	3.1.1	<a href="https://github.com/NERSC/jupyterlab-favorites">https://github.com/NERSC/jupyterlab-favorites</a>	Add the ability to save favorite folders to JupyterLab for quicker browsing
jupyterlab-system-monitor	0.8.0	<a href="https://github.com/tpio/jupyterlab-system-monitor">https://github.com/tpio/jupyterlab-system-monitor</a>	JupyterLab extension to display system metrics
jupyterlab_iframe	0.4.4	<a href="https://github.com/timkpaine/jupyterlab_iframe">https://github.com/timkpaine/jupyterlab_iframe</a>	View html as an embedded iframe in JupyterLab
jupyterlab-tour	3.1.4	<a href="https://github.com/jupyterlab-contrib/jupyterlab-tour">https://github.com/jupyterlab-contrib/jupyterlab-tour</a>	A JupyterLab UI tour built on jupyterlab-tutorial and react-joyride.
papermill	2.4.0	<a href="https://papermill.readthedocs.io">https://papermill.readthedocs.io</a>	Parameterize, execute, and analyze notebooks
pyunicore	0.15.0	<a href="https://github.com/HumanBrainProject/pyunicore">https://github.com/HumanBrainProject/pyunicore</a>	UNICORE REST bindings for python

# JUPYTER KERNEL

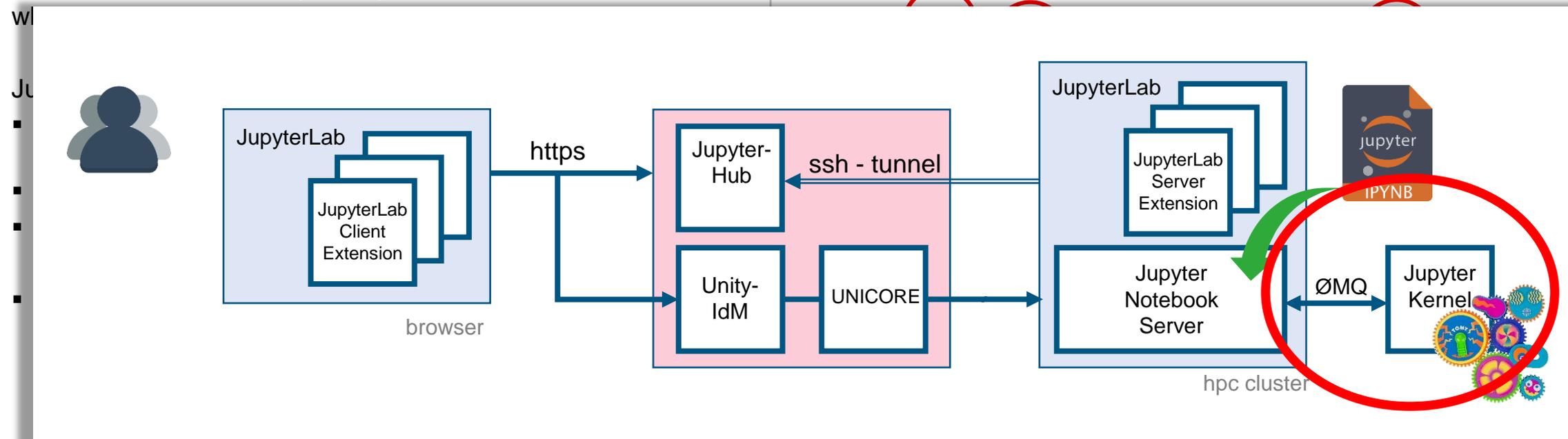
# JUPYTER KERNEL

## How to create your own Jupyter Kernel



### Jupyter Kernel

A “kernel” refers to the separate process



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

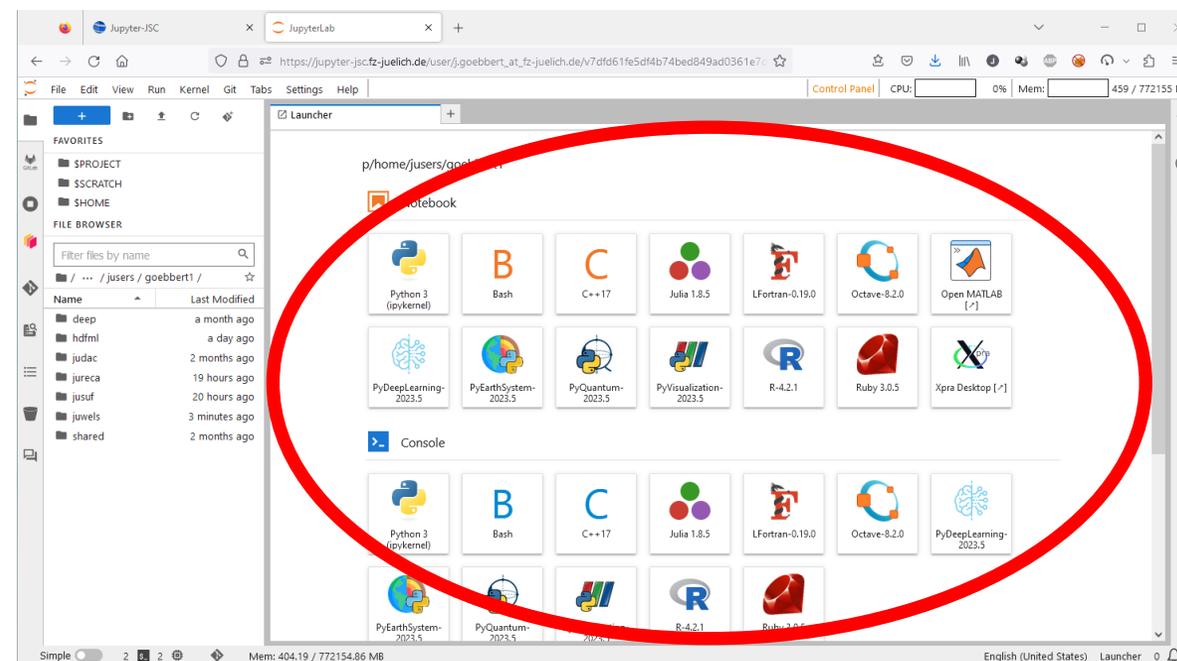
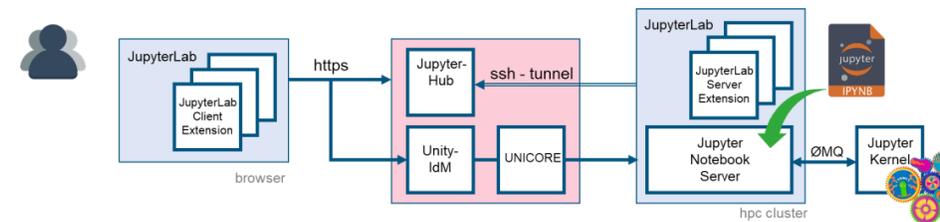
### Jupyter Kernel

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

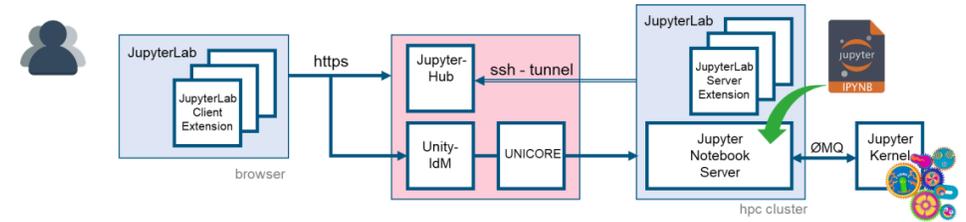
### Jupyter Kernel

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.



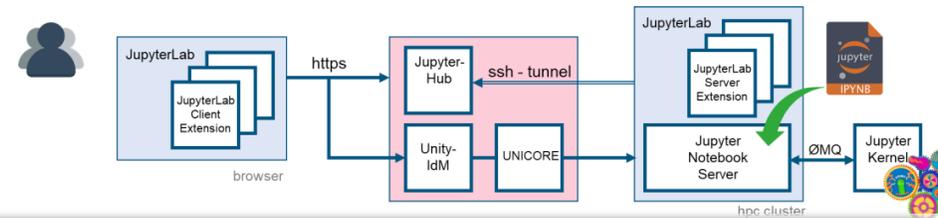
## Building your own Jupyter kernel is a three step process

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>

# 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

The screenshot shows a JupyterLab interface. On the left is a file browser for 'jupyter4jsc' with a list of notebooks. The notebook 'Create\_JupyterKernel\_general.ipynb' is selected. The main area displays the notebook content, which includes an attention box and a three-step process for building a kernel.

**Create your own Jupyter Kernel**

Often the standard kernel do not provide all features you need for your work. This might be that certain modules are not loaded or packages are not installed. With your own kernel you can overcome that problem easily and define your own environment, in which you work.

This notebook shows you how you can build your own kernel for a **python environment**.

**Attention:** This notebook is meant to run out of a JupyterLab on JSC's HPC systems.

Building your own Jupyter kernel is a three step process

- Create/Pimp new virtual Python environment
  - venv
- Create/Edit launch script for the Jupyter kernel
  - kernel.sh
- Create/Edit Jupyter kernel configuration
  - kernel.json

Settings

Set the kernel name

- must be lower case

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

# JUPYTER KERNEL

## Run your Jupyter kernel configuration

### Run your Jupyter Kernel

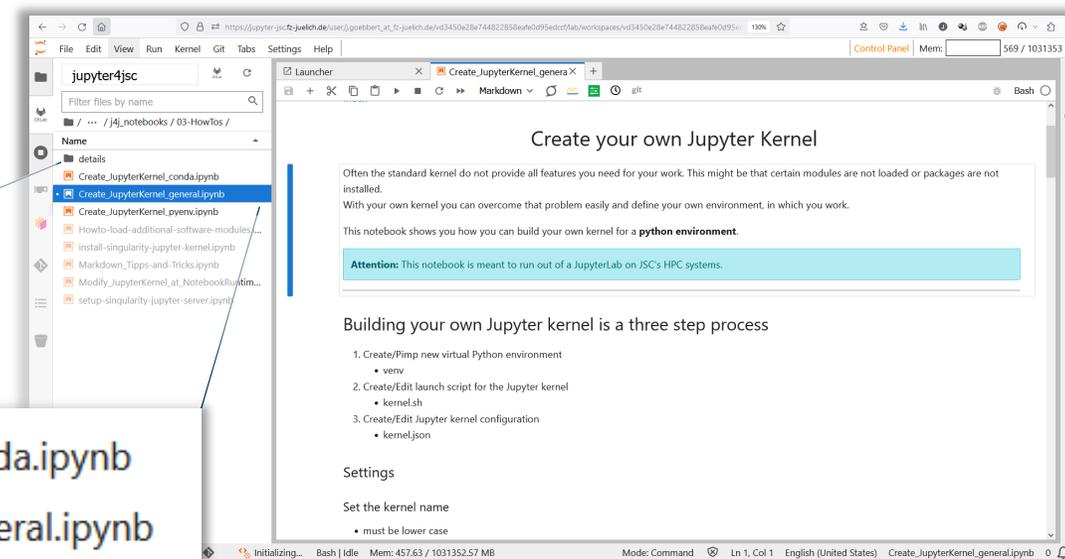
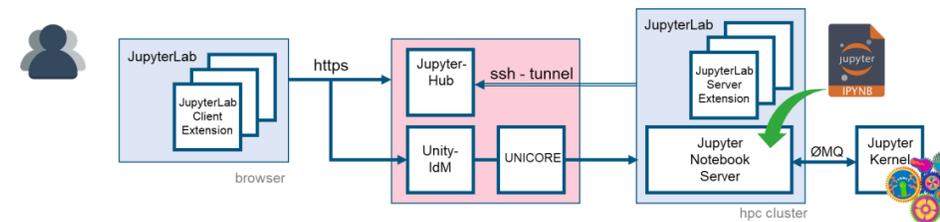
1. <https://jupyter-jsc.fz-juelich.de>
2. Choose system where your Jupyter kernel is installed in `~/ .local/share/jupyter/kernels`
3. Select your kernel in the launch pad or click the kernel name.

### 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](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).

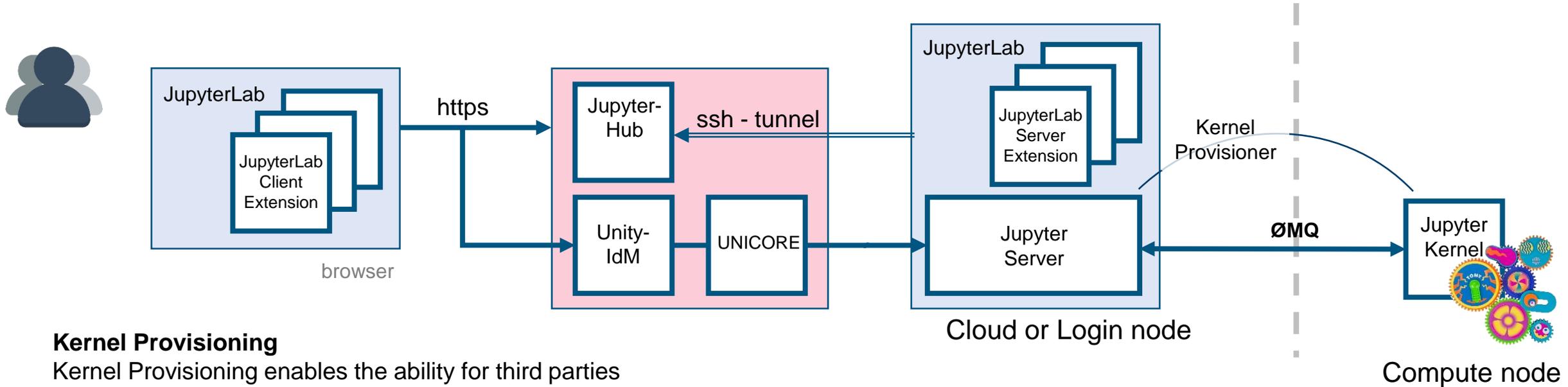
E.g.

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

# **SLURM WRAPPED KERNELS WITH SLURM-PROVISIONER**

# 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**.

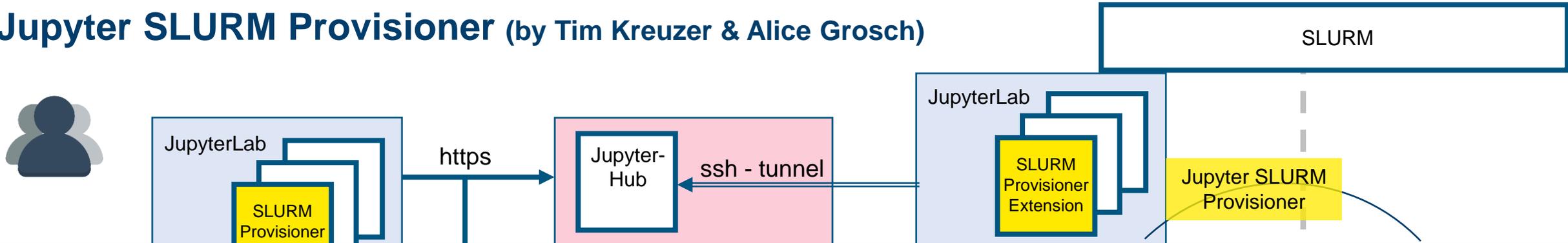
By implementing and configuring a *kernel provisioner*, third parties have the ability to **provision kernels for different environments**, typically managed by resource managers like Kubernetes, Hadoop YARN, Slurm, etc.

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 'Select allocation for slurm wrapper' dialog in JupyterLab. The 'New' button is highlighted. 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.



```
[4]: !hostname
      jsfc080

[ ]:

kreuzer1@jsf102:~/slurm-pr...
[kreuzer1@jsf102 slurm-provisioner]$ hostname
jsf102.jusuf
[kreuzer1@jsf102 slurm-provisioner]$

[goebbert1@jsf101 jusuf]$ jupyter kernelspec provisioners
Available kernel provisioners:
  local-provisioner      jupyter_client.provisioning:LocalProvisioner
  slurm-provisioner     jupyter_slurm_provisioner:SlurmProvisioner
```

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

# REMOTE HYBRID KERNELS

Jupy

The screenshot displays the JupyterLab interface. The top navigation bar includes 'File', 'Edit', 'View', 'Run', 'Kernel', 'Git', 'Tabs', 'Settings', and 'Help'. A 'Control Panel' is visible on the right, showing CPU usage at 0% and memory usage at 196 / 257468 MB. The left sidebar contains 'Current Configuration' (with a 'Configure' button), 'Kernel Allocations' (stating 'There are no allocations available.'), and a 'Configure Slurm' panel with dropdown menus for 'Select allocation for slurm' (set to 'New'), 'Select kernel for slurm' (set to 'Custom Python'), 'Select project for slurm' (set to 'ccstvs'), and 'Select partition for slurm' (set to 'develgpu'). The main area shows the 'Launcher' view for the user 'p/home/jusers/goebbert1/jusuf'. It features a grid of 24 kernel options, each with an icon and a label: Python 3 (ipykernel), Bash, C++17, covid19dynstat\_j usuf, covid19dynstat\_j uwels, esm2022\_kernel, goebbert1\_kerne l\_jsonmerge, goebbert1\_pyferr et, goebbert1\_works hop2, 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, and Xpra Desktop [↗]. A 'Try the Welcome Tour.' notification is present in the bottom right corner. The bottom status bar shows 'Mem: 195.93 / 257467.88 MB' and 'English (United States) Launcher'.

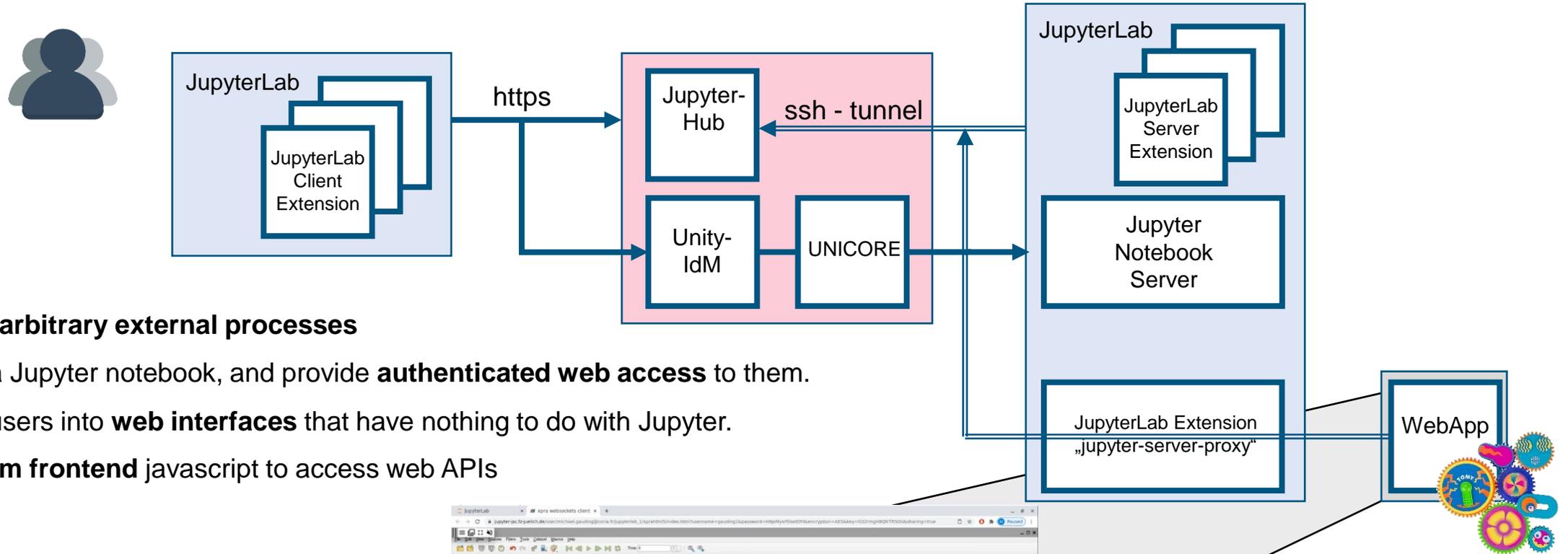
<https://github.com/FZJ-JSC/jupyter-slurm-provisioner>  
<https://github.com/FZJ-JSC/jupyter-slurm-provisioner-extension>

slurm-provisioner jupyter\_slurm\_provisioner:SlurmProvisioner  
Forschungszentrum

# JUPYTER SERVER PROXY

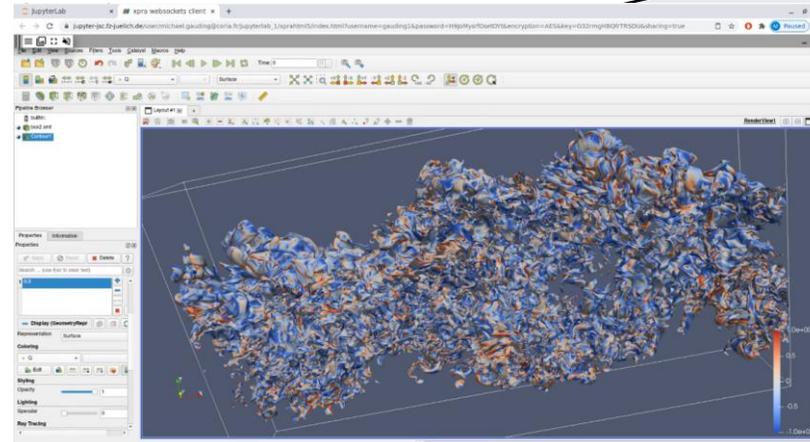
# JUPYTERLAB – WEBSERVICE PROXY

## 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

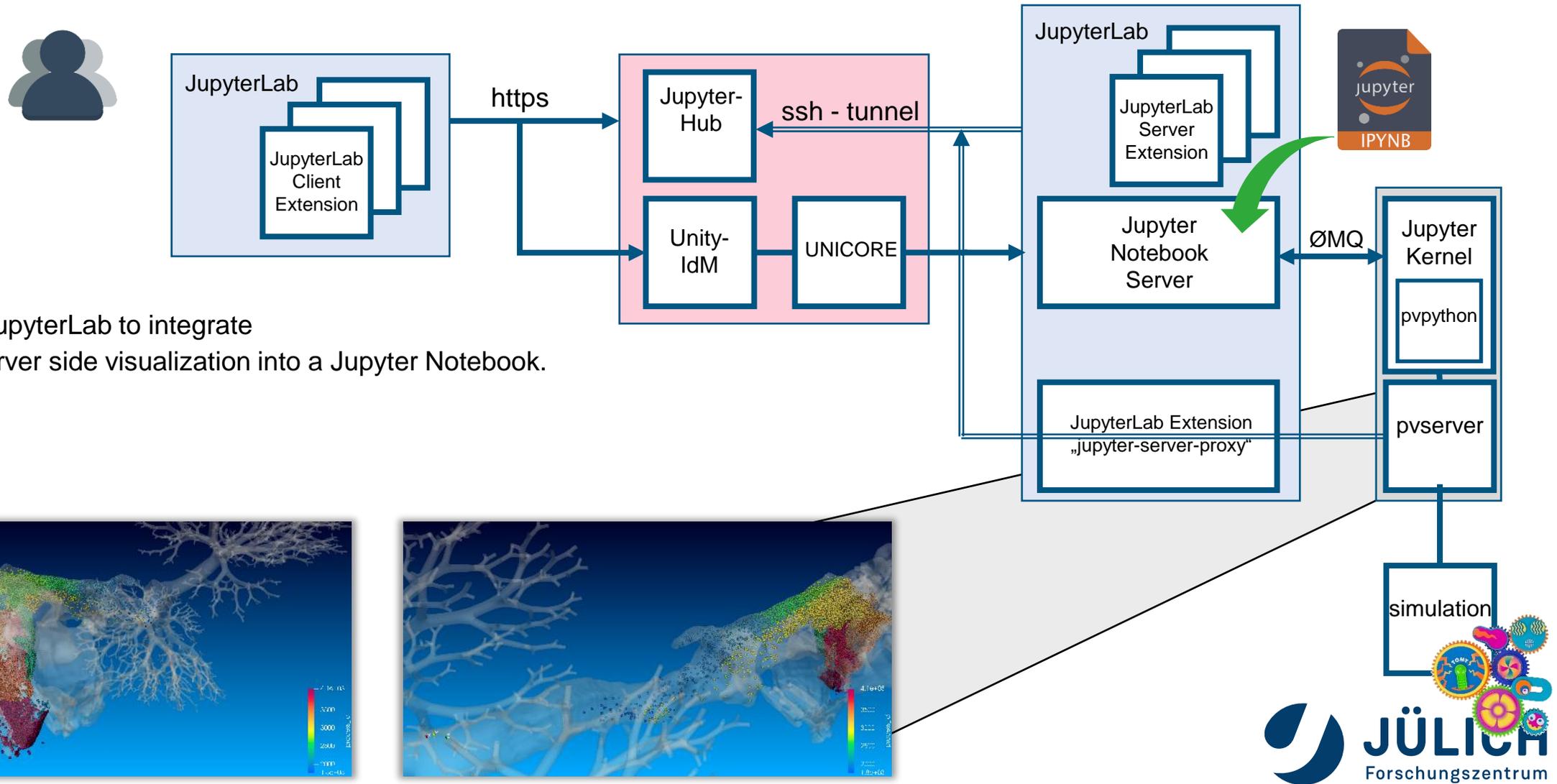


Turbulent mixing with variable density, subset of 1939x600x3584 grid points, Michael Gauding, CORIA

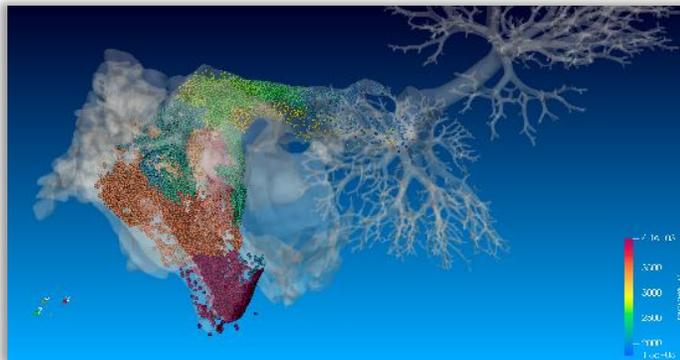
<https://github.com/jupyterhub/jupyter-server-proxy>

# JUPYTERLAB – WEBSERVICE PROXY

## Extension: jupyter-server-proxy



How to use JupyterLab to integrate interactive server side visualization into a Jupyter Notebook.



# 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>**

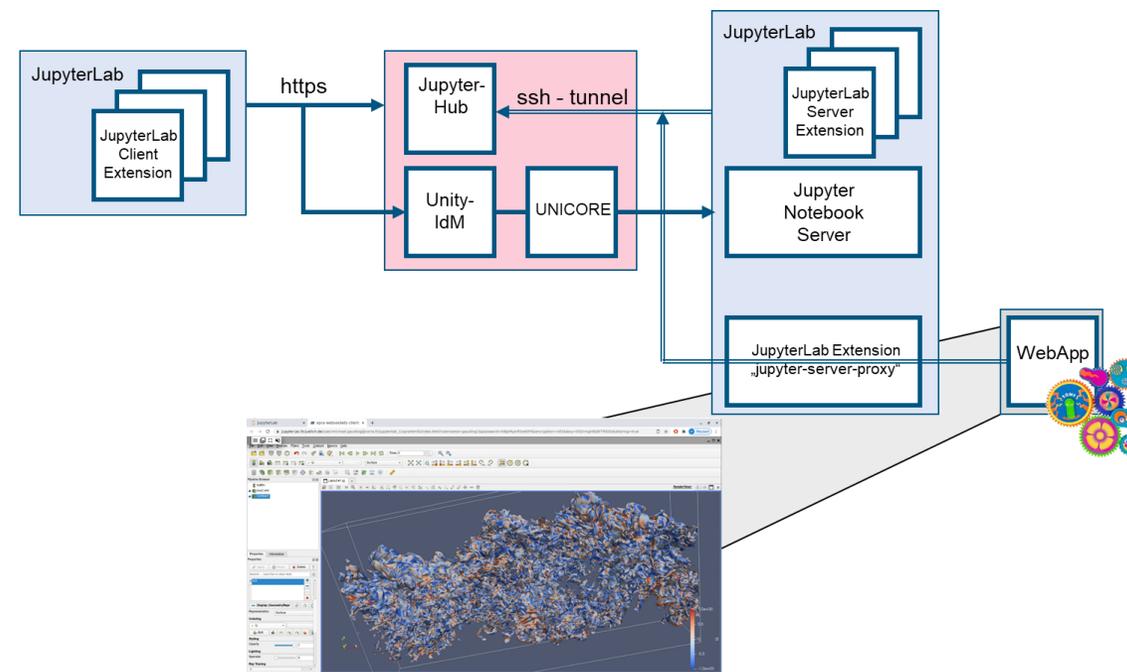
The URL will be rewritten to remove the above prefix.

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>

# JUPYTER SERVER PROXY EXAMPLES

# 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.

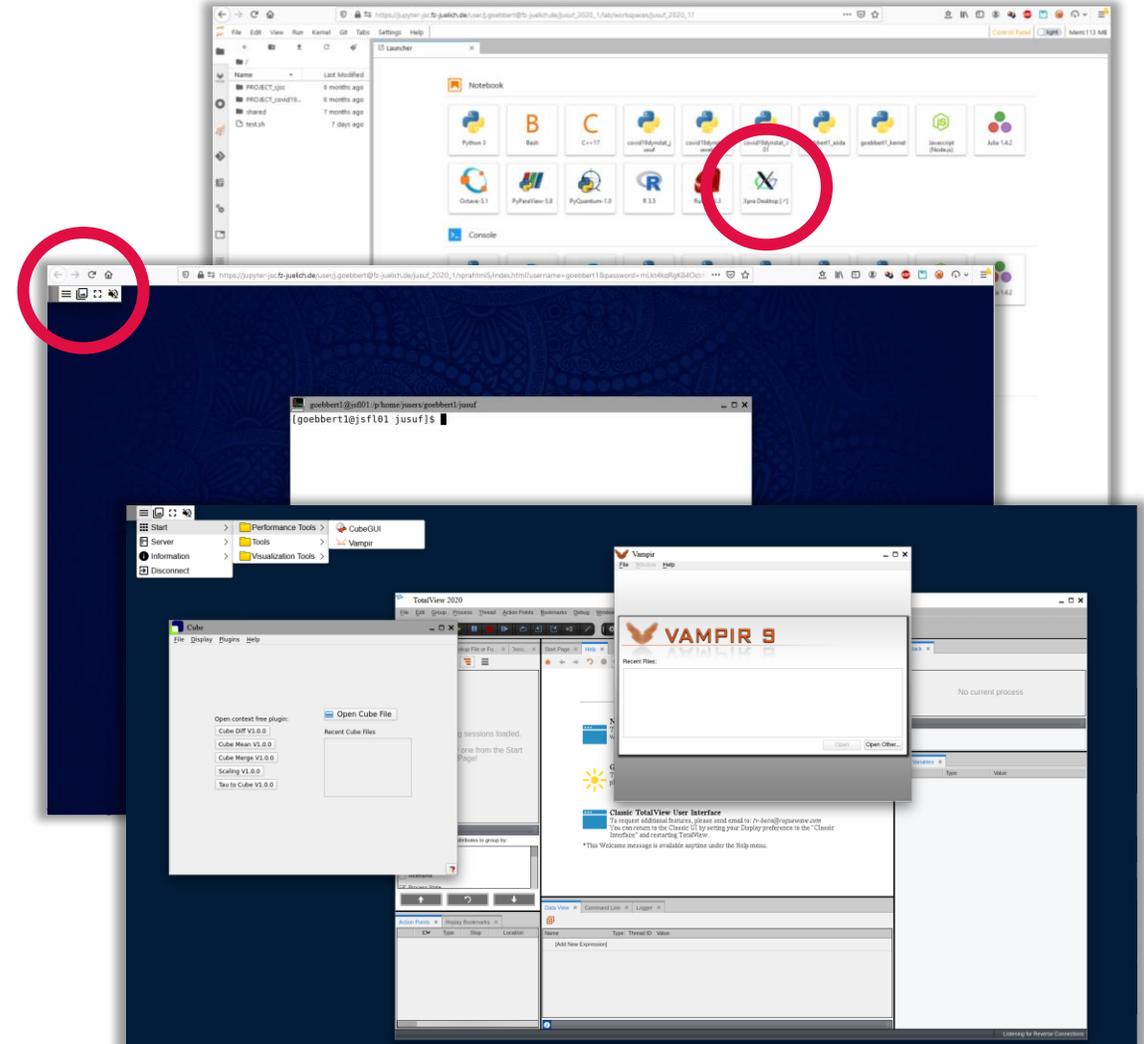
- Runs in a browser
- allows dis-/reconnection without disrupting the forwarded application
- <https://xpra.org>

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.

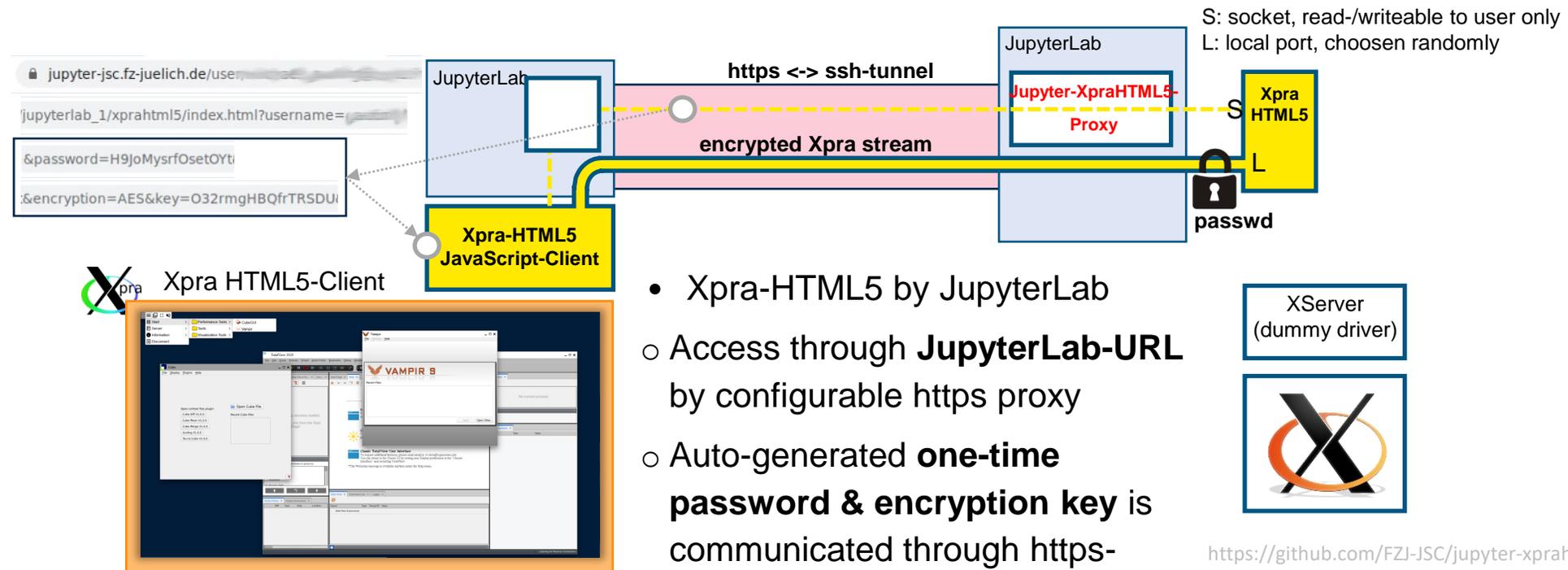
Hint:

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



# JUPYTERLAB – REMOTE DESKTOP

Run your X11-Applications in the browser



- 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>

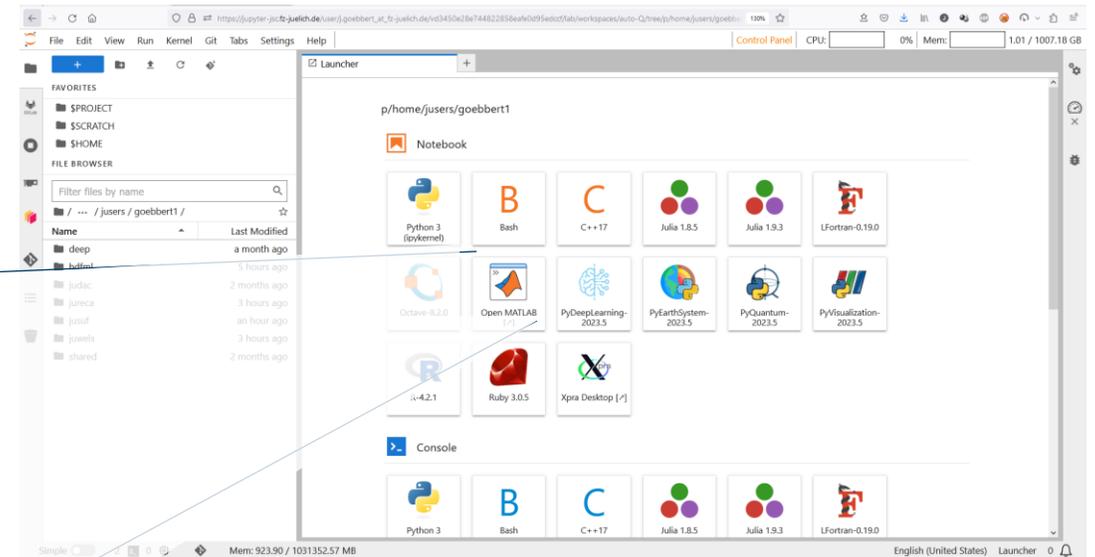
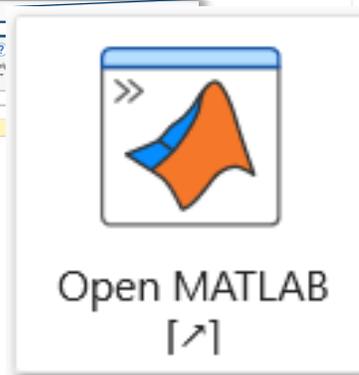
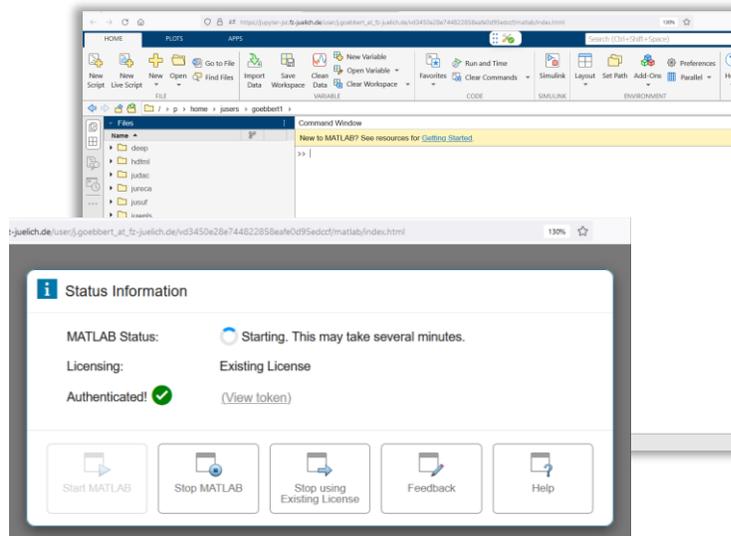
# JUPYTERLAB – MATLAB

## 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>

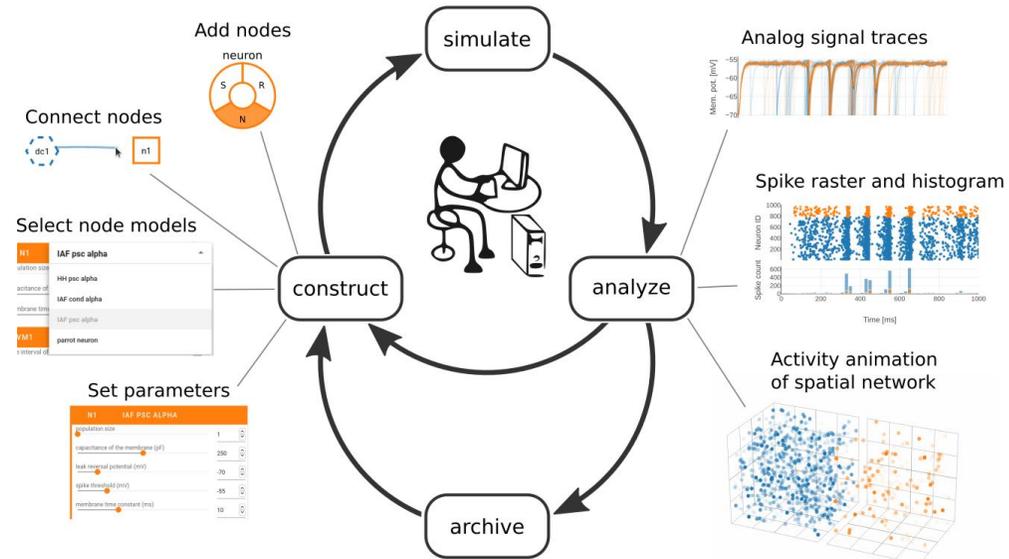
[3] <https://de.mathworks.com/products/parallel-computing.html>

# JUPYTERLAB – NEST DESKTOP

## Web-based GUI for Neuroscientists using NEST

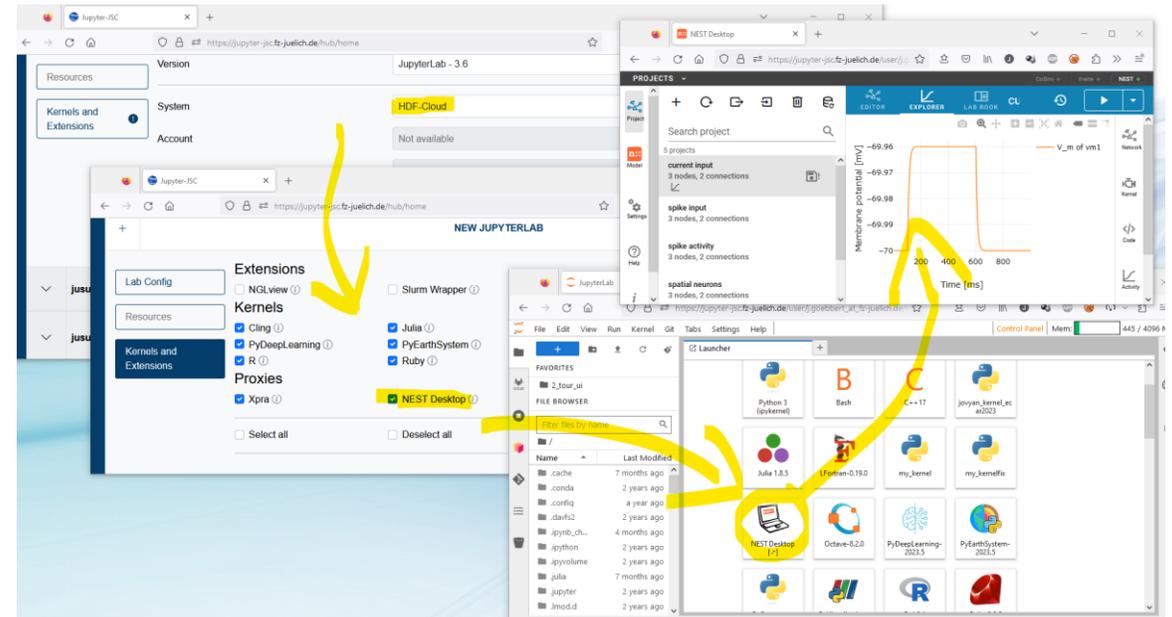
### NEST-Desktop

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)  
<https://github.com/jhgoebbert/jupyter-nestdesktop-proxy>**

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

# CONCLUSION

## Why Jupyter is so popular among Data Scientists

JupyterLab ...

- ... is a **web-based platform for interactive computing and data analysis** that is well-suited to the needs of research software engineers.
- ... provides researchers with a **comprehensive environment** for working with code, text, multimedia, and data, making it an ideal tool for a wide range of research tasks.
- ... is designed to be **flexible and customizable**, and can be modified to suit the specific needs and workflows of individual researchers.
- ... supports the creation of **reproducible research** through its support for Jupyter notebooks.
- ... supports **collaboration and sharing** of research work through its support for sharing notebooks, dashboards, and other elements of a research project.
- ... provides a wide range of **extensions and plugins** that can be used to integrate other tools and services into the environment.
- ... is an **open-source project**, which means that researchers have access to the source code and can contribute to its development.

# QUESTIONS?



More details:

<https://gitlab.jsc.fz-juelich.de/jupyter4jsc/training-2023.04-jupyter4hpc>