



# INTERACTIVE HIGH-PERFORMANCE COMPUTING WITH JUPYTERLAB

Webinar #2 – RISC2 Webinar Series

2022-09-22 | JENS H. GÖBBERT (J.GOEBBERT@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

<https://www.youtube.com/watch?v=xuNj5paMuow>

<https://jupyter.org>

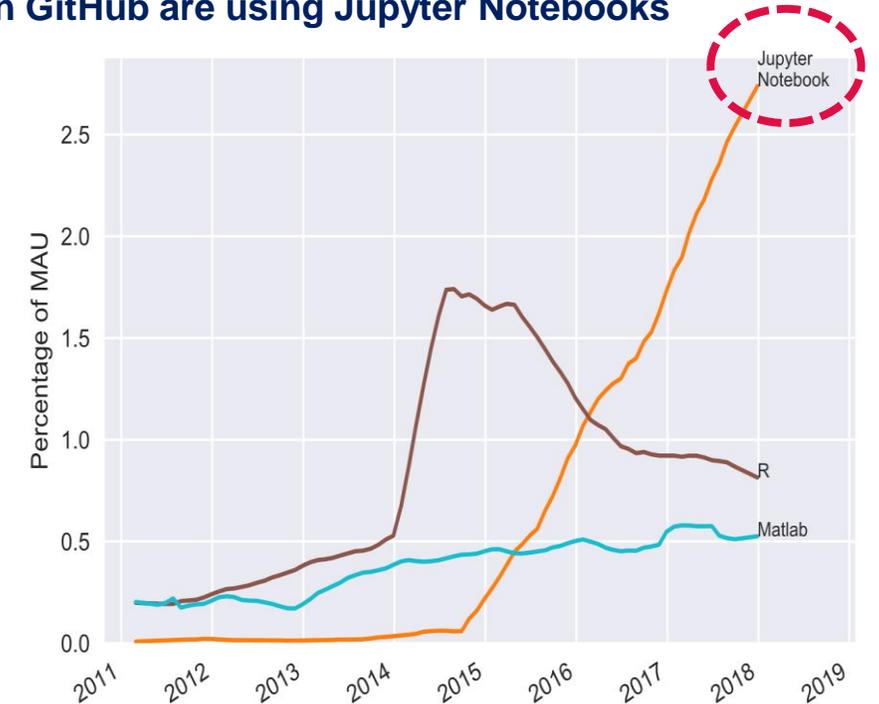
Member of the Helmholtz Association

# 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 ...

## 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>

# 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:

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

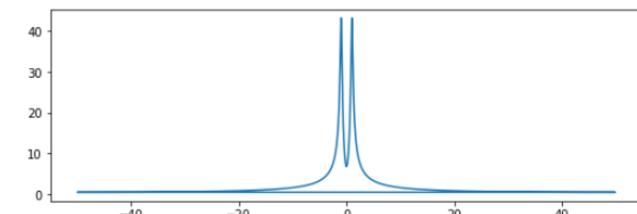
where  $x$  is the position of the oscillator,  $\omega_0$  is the frequency, and  $\zeta$  is the damping ratio. To write this second-order ODE on standard form we introduce  $p = \frac{dx}{dt}$ :

```
[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)
```

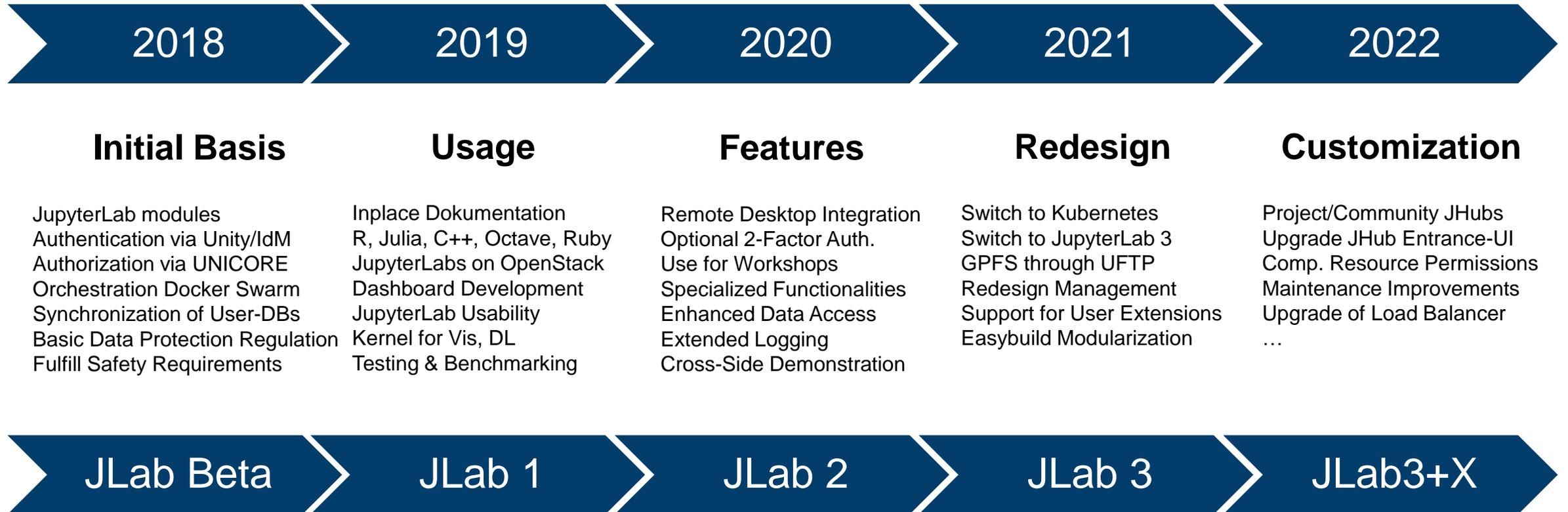
```
[44]: fig, ax = plt.subplots(figsize=(9,3))
      ax.plot(w, abs(F));
```



Output

Output

# HISTORY OF JUPYTERLAB AT JSC



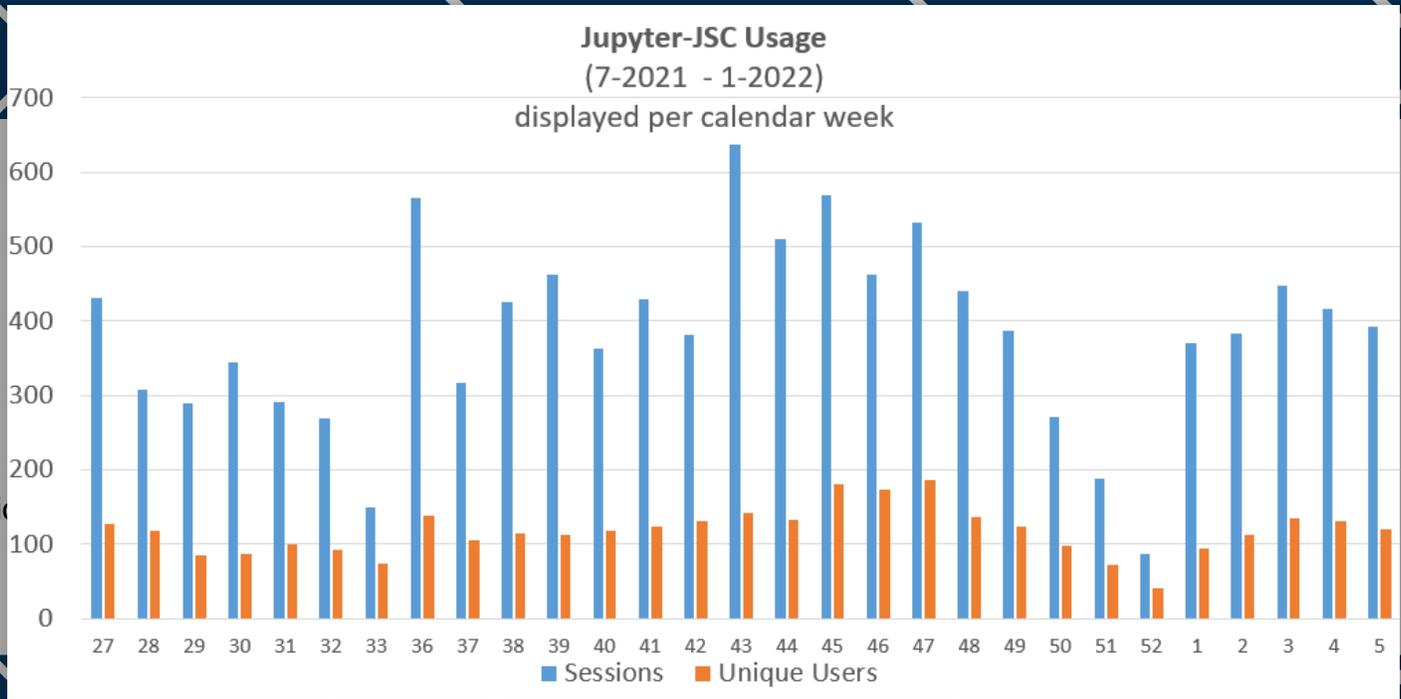
# HISTORY OF JUPYTERLAB AT JSC

2018

## Initial Basis

- JupyterLab modules
- Authentication via Unity/IdM
- Authorization via UNICORE
- Orchestration Docker Swarm
- Synchronization of User-DBs
- Basic Data Protection Regulation
- Fulfill Safety Requirements

JLab Beta



JLab 1

JLab 2

JLab 3

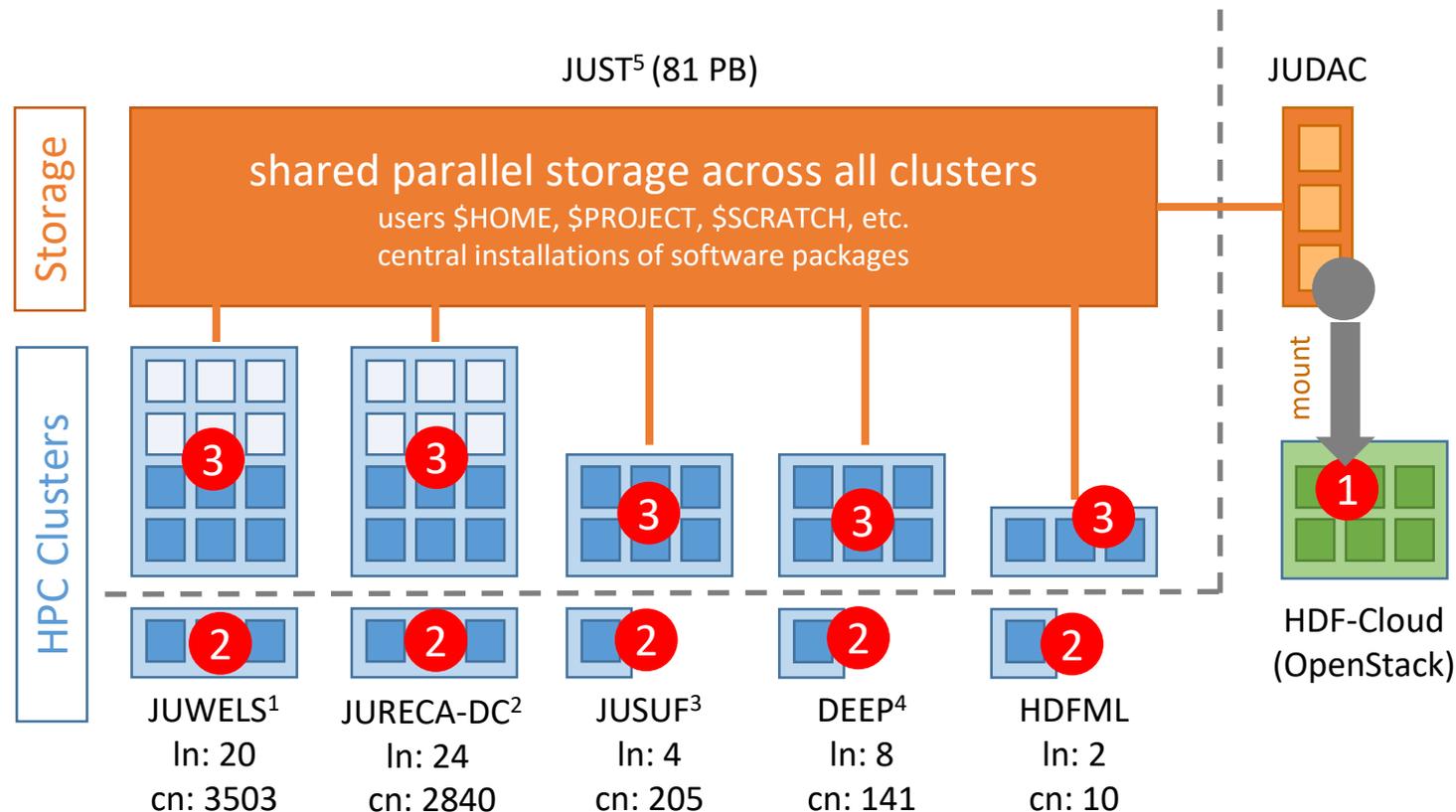
JLab3+X

2022

## Customization

- Project/Community JHubs
- Upgrade JHub Entrance-UI
- Comp. Resource Permissions
- Maintenance Improvements
- Upgrade of Load Balancer
- ...

# JUPYTERLAB EVERYWHERE



## JupyterLab everywhere

- 1 JupyterLab on HDF-Cloud
- 2 JupyterLab on login nodes
- 3 JupyterLab on compute nodes

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>

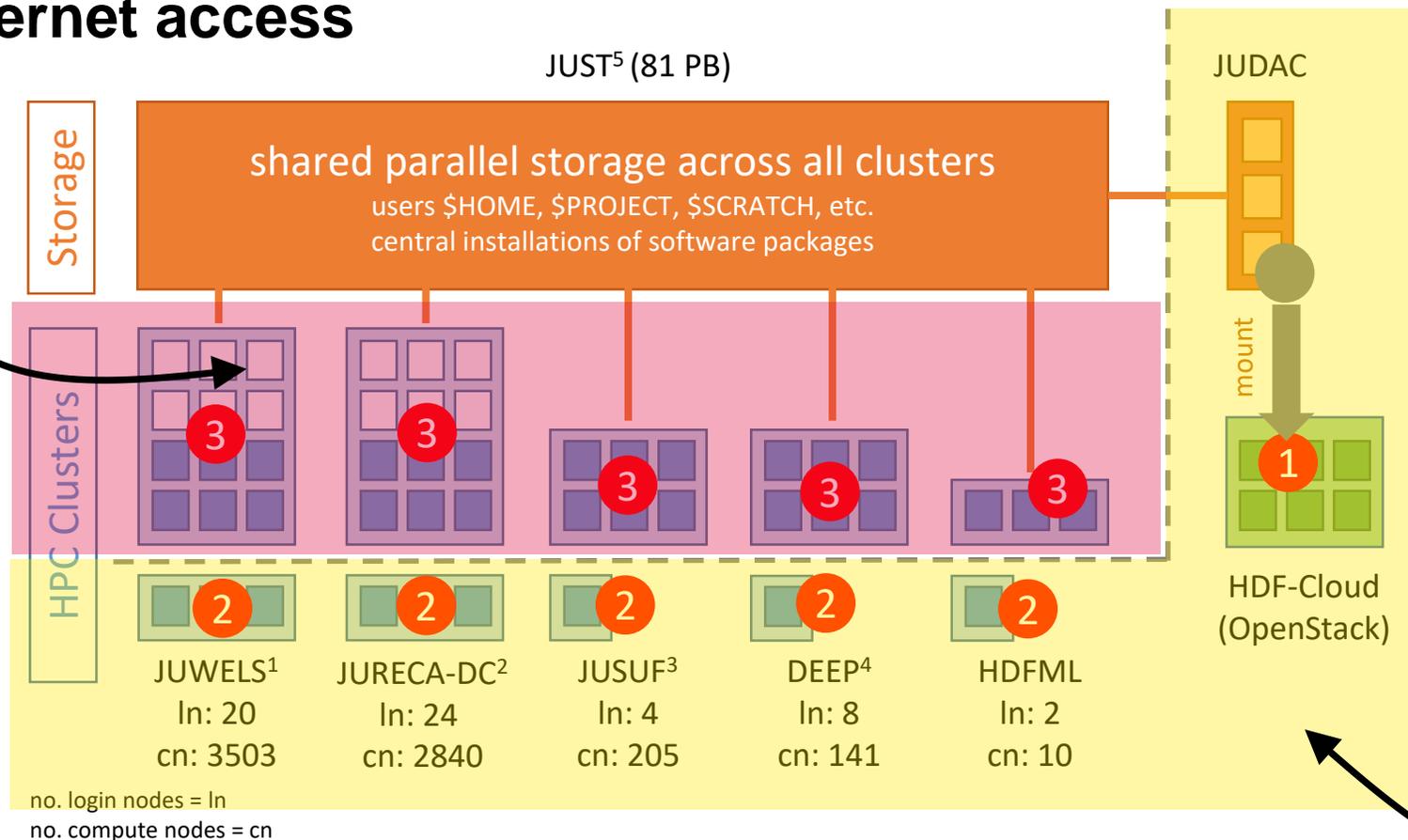
[3] <https://apps.fz-juelich.de/jsc/hps/jusuf/cluster/configuration.html>

[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

**NO internet access**



**JupyterLab everywhere**

- 1 JupyterLab on HDF-Cloud
- 2 JupyterLab on login nodes
- 3 JupyterLab on compute nodes

**internet access**

[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)

# TERMINOLOGY

## What is JupyterLab

### JupyterLab

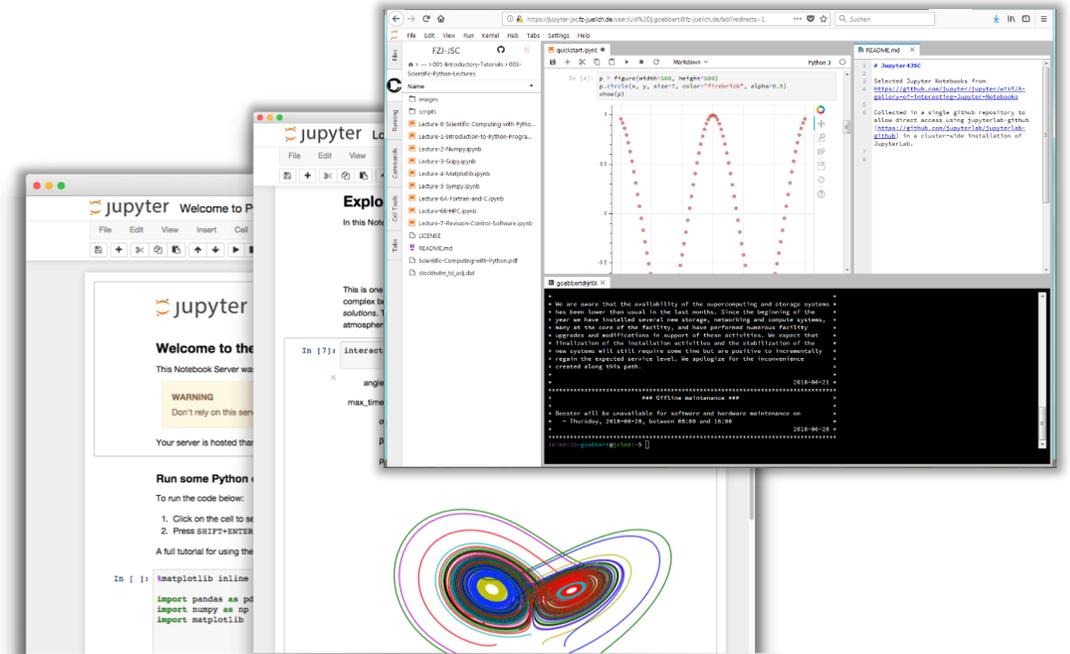
- **Interactive** working environment in the web browser
- For the creation of **reproducible** computer-aided narratives
- Very **popular** with researchers from all fields
- Jupyter = Julia + Python + R

### 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>

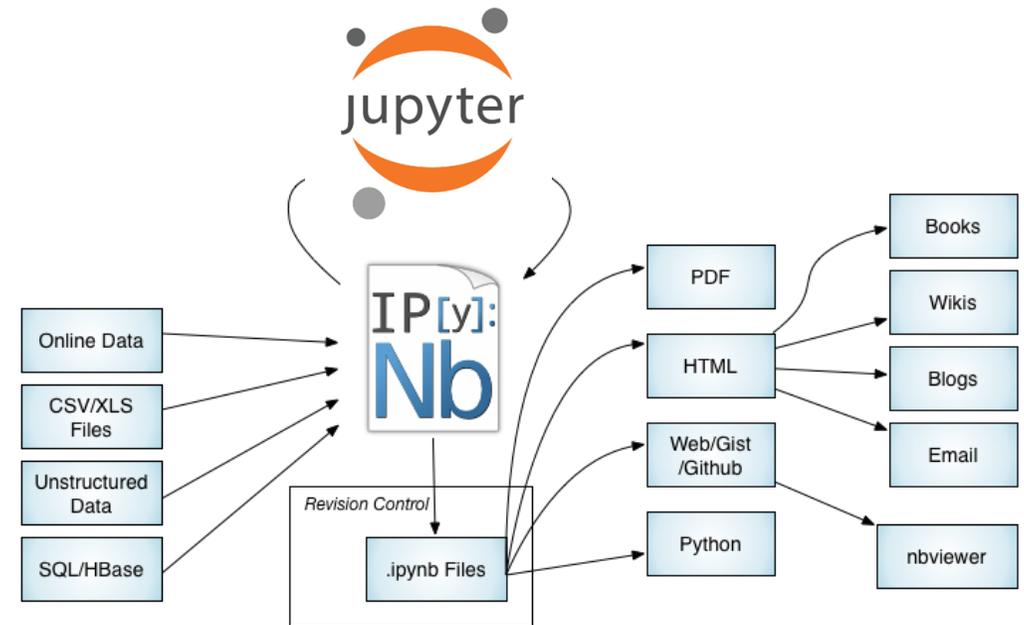
# TERMINOLOGY

## 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 deep learning, 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.

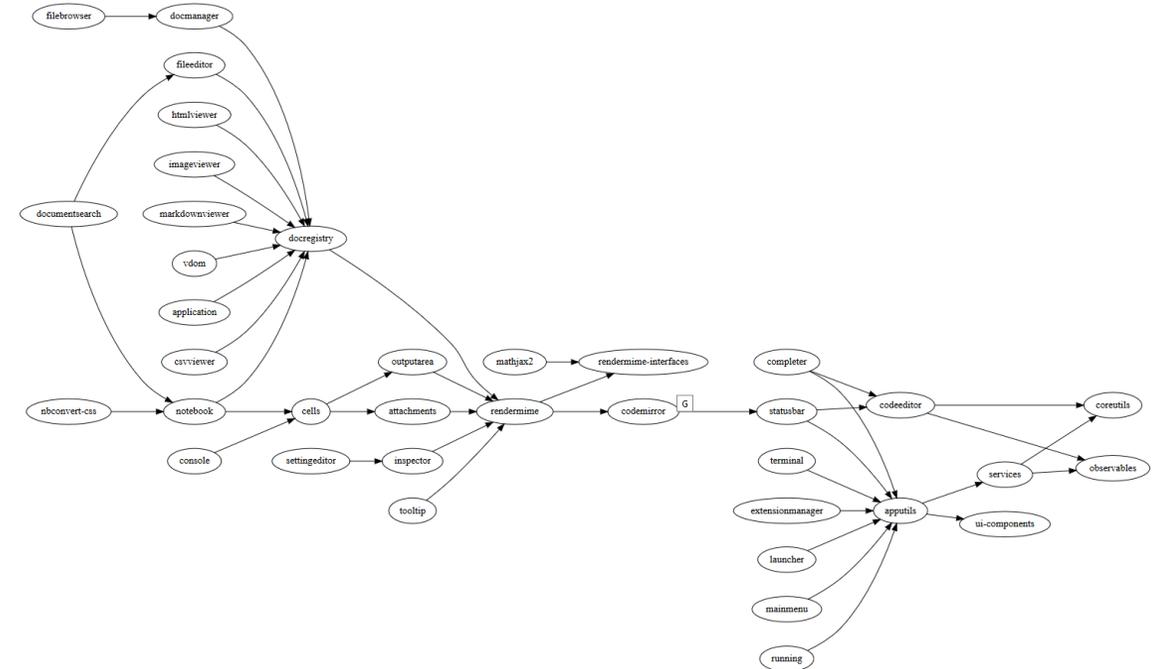
### JupyterLab Extensions

- 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.

With JupyterLab 3 prebuild extensions were introduced.

You can now (technically) extend a compiled JupyterLab 3+ with **your own extensions**.



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

# 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)

### Remote (cluster) installation:

- **JupyterLab** installed in \$HOME (e.g. using pip or miniconda)
- **JupyterLab** installed system-wide (e.g. with Easybuild, Spark)

### Browser-only installation (**alpha!**):

- **JupyterLab** (local in your browser – JupyterLite = just for testing!)  
<https://blog.jupyter.org/jupyter-everywhere-f8151c2cc6e8>  
<https://jupyter.org/try-jupyter/lab>



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

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

```
ssh -N -L <LOCAL_PORT>:<JLAB_NODE>:<JLAB_PORT> <USERID>@<LOGIN_NODE>.fz-juelich.de  
# example: ssh -N -L 8888:juwels04:8888 goebbert1@juwels01.fz-juelich.de  
  
# if you want to tunnel to juwels04 only, then you should set JLAB_NODE to "localhost"
```

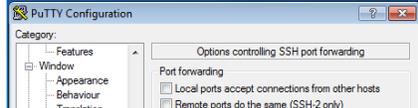
**Attention:**

- 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. juwels01.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:

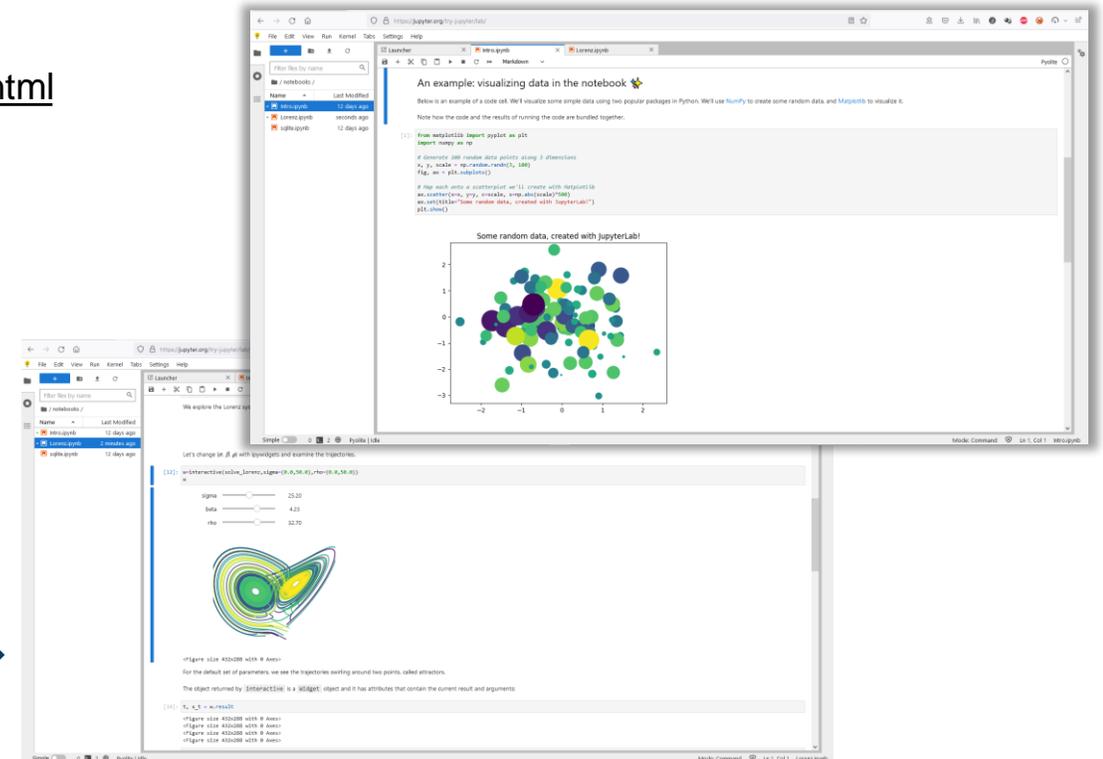
- **JupyterLab** installed using conda, mamba, pip, pipenv or docker.  
[https://jupyterlab.readthedocs.io/en/stable/getting\\_started/installation.html](https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html)

### Remote (cluster) installation:

- **JupyterLab** installed in \$HOME (e.g. using pip or miniconda)
- **JupyterLab** installed system-wide (e.g. with Easybuild, Spark)

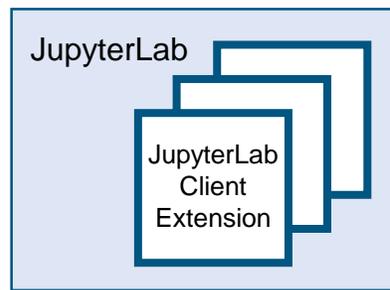
### Browser-only installation (**alpha!**):

- **JupyterLab** (local in your browser – JupyterLite = just for testing!)  
<https://blog.jupyter.org/jupyter-everywhere-f8151c2cc6e8>  
**Try it:** <https://jupyter.org/try-jupyter/lab>



# JUPYTER-JSC WEBSERVICE

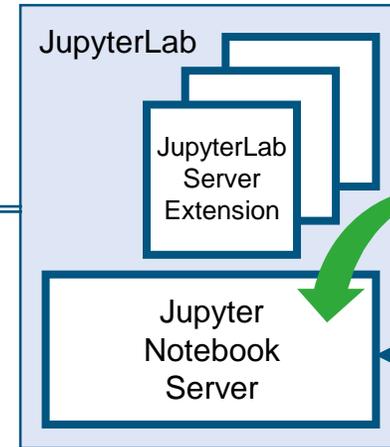
## Start your JupyterLab



browser

ssh

ssh - tunnel



hpc cluster

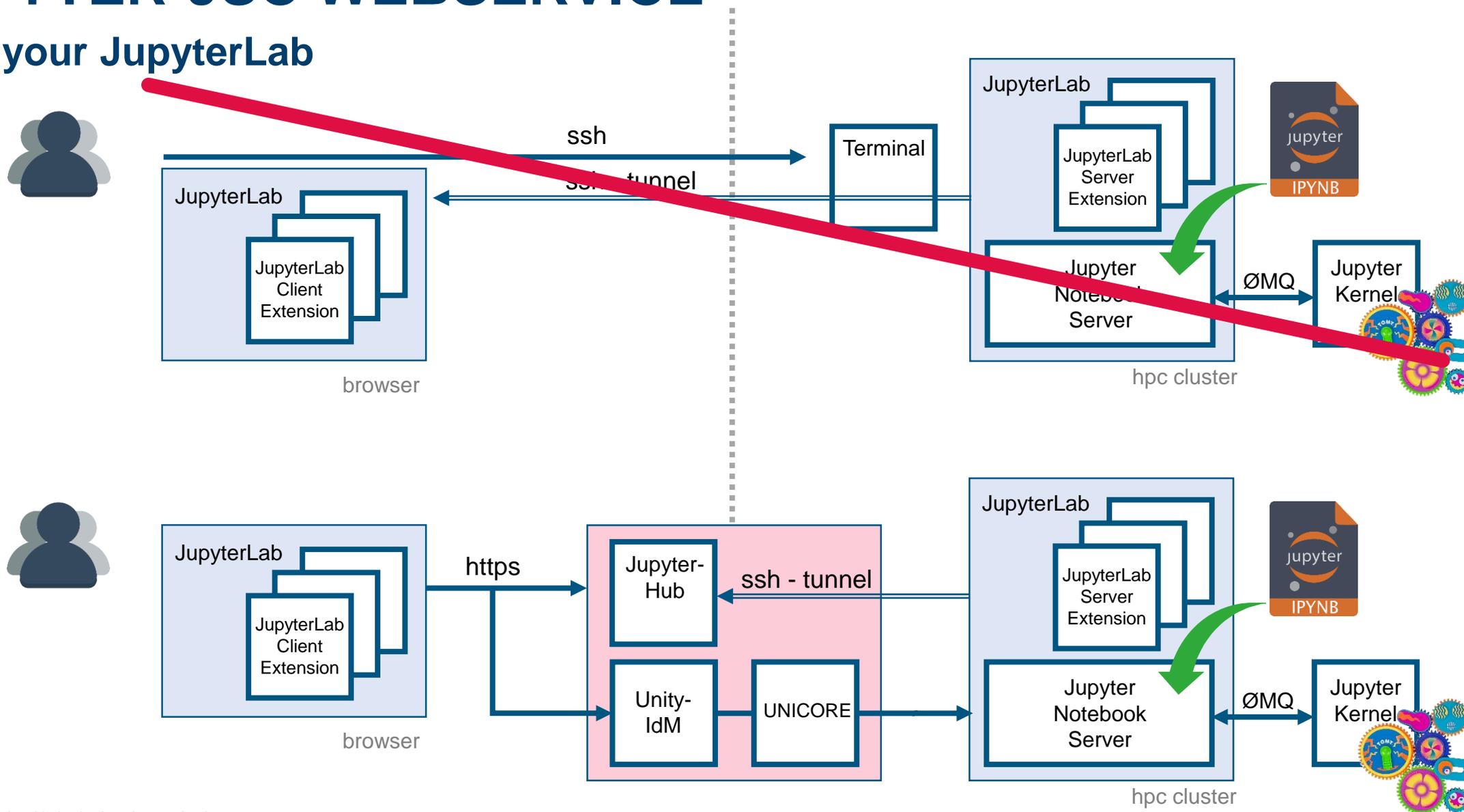


ØMQ



# JUPYTER-JSC WEBSERVICE

## Start your JupyterLab



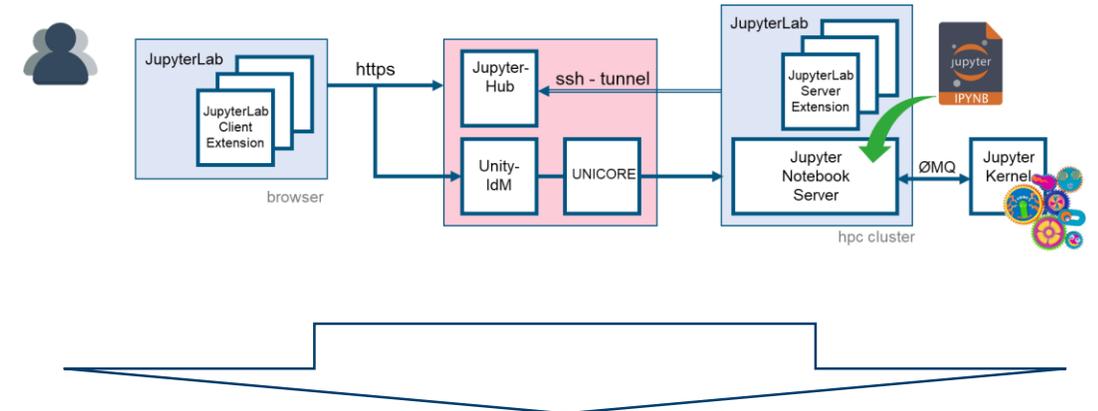
# JUPYTER-JSC WEBSERVICE

## Start your JupyterLab

The image displays three overlapping screenshots of the JupyterLab 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 shows a 'Supercomputing in Your Browser' banner 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	ccstdf		Start
juwels_cluster	JUWELS	devel	ccsvs	30% spinning	Open, Cancel



The image shows a screenshot of the JupyterLab web interface. The top part displays a 'GPU DASHBOARDS' sidebar with metrics like GPU Utilization, GPU Memory, GPU Resources, PCIe Throughput, WLink Throughput, WLink Timeout, and Machine Resources. The main area shows a code editor with Python code for matrix multiplication. The bottom part shows a 'GPU Memory' graph.

```
[1]: 1 import math
2 import numpy as np
3 from numba import cuda
4 import matplotlib.pyplot as plt
5 matplotlib.inline

[2]: 1 len(cuda.gpus)

[3]: 1 len(cuda.gpus)
[3]: b'tesla v100-500-100B'

[4]: 1 @cuda.jit
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 calculation.
10    e = 1.2 + 1j * size * j
11    z = e
12    for n in range(iterations):
13        z = z**2 + e
14        if math.isnan(z):
```

# JUPYTER-JSC WEBSERVICE

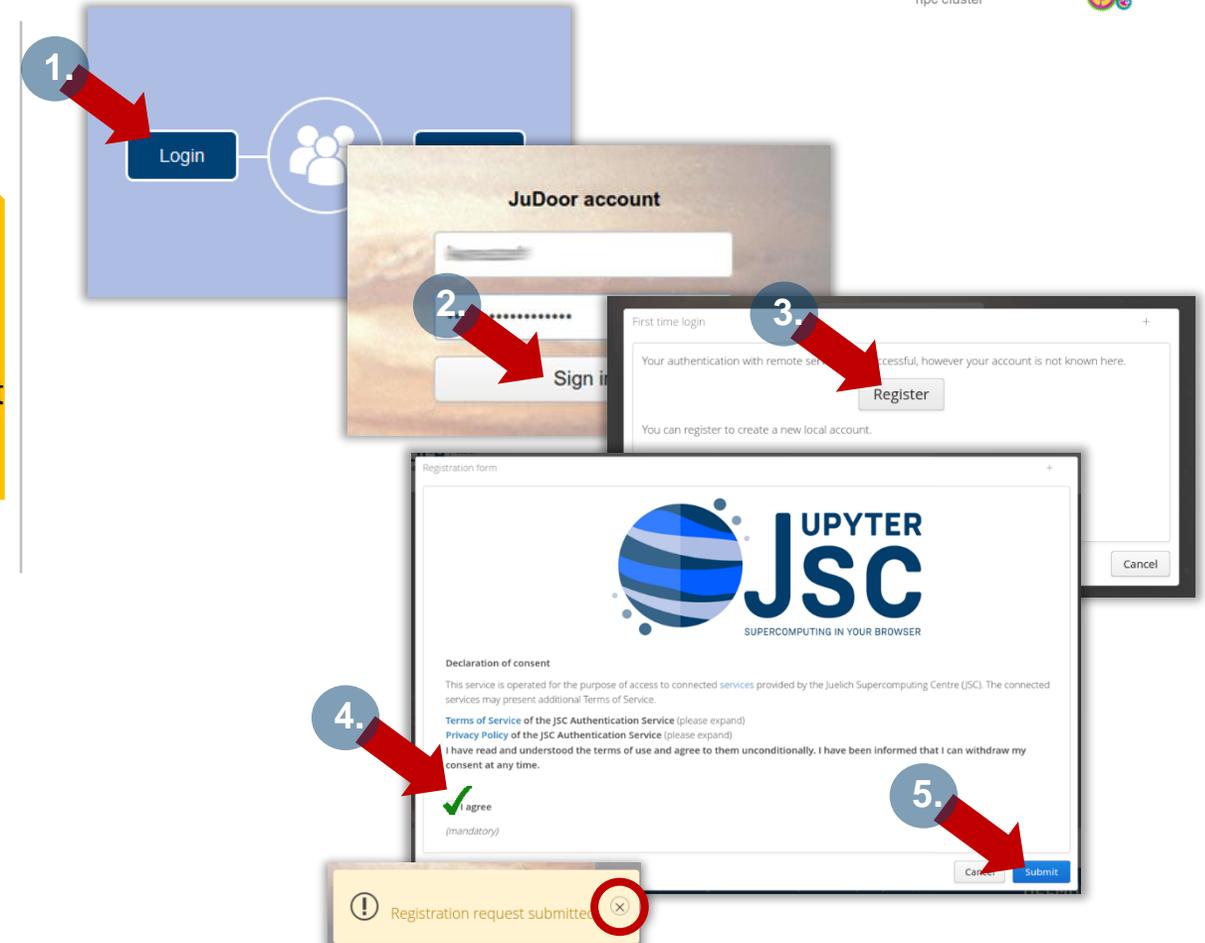
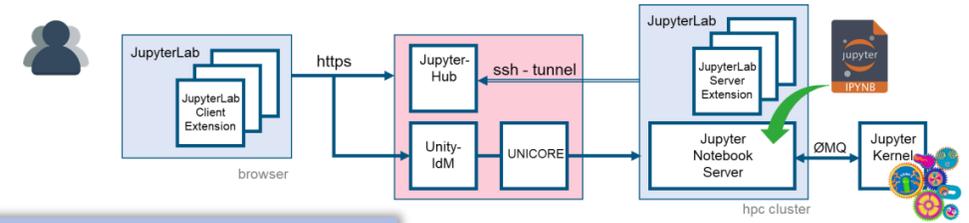
## First time login

=> <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



# JUPYTER-JSC WEBSERVICE

## Control Panel

### A. Jupyter-JSC – Add new JupyterLab

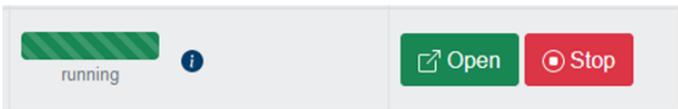


### B. Configuration Dialog

- set Name, Type, System, Account, Project, Partition

### C. Jupyter-JSC – Actions

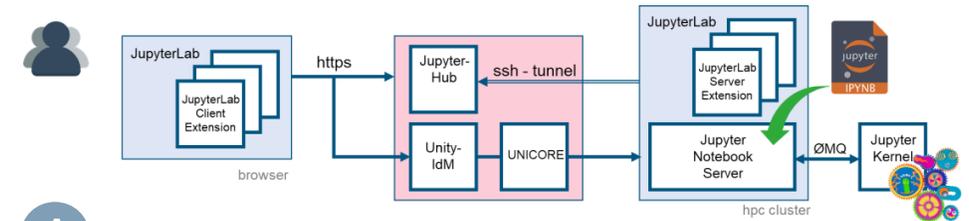
- Open/Stop a running JupyterLab
- Change/Delete **configuration**



### D. Jupyter-JSC -- Statusbar



- Upcoming maintenance (mouse hover for details)
- System offline



Name	System	Partition	Project	Status	Actions
hdfcloud_3.3	HDF-Cloud	N/A	N/A		[Start]
juwelsbooster_login	JUWELS	LoginNodeBooster	ccstid		[Start]
juwels_cluster	JUWELS	devel	ccstvs	running	[Open] [Stop]

### E. Jupyter-JSC – 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**

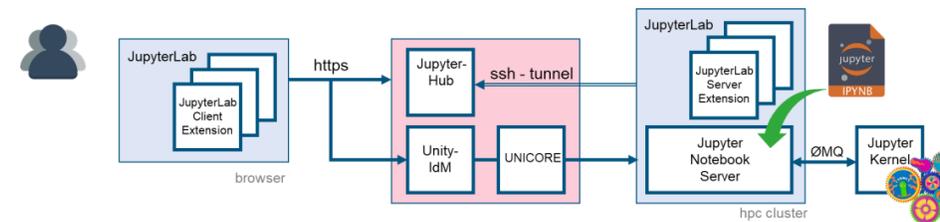
- user account settings visible in [judoor.fz-juelich.de](https://judoor.fz-juelich.de)
- currently available systems in all of your projects
  - system specific usage agreement on JuDoor is signed

### Basic options

- Type:  
JupyterLab 2 and JupyterLab 3 (default) is 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 Nodes, Runtime, GPUs, ...



Configuration

Service ⓘ Name ⓘ Please give your configuration a name.

Options ⓘ Type ⓘ JupyterLab

Resources

Reservation

Cancel Start

Configuration

Service ⓘ System ⓘ JUWELS

Options ⓘ Account ⓘ goebbert1

Resources Project ⓘ ccstdl

Reservation Partition ⓘ LoginNode

**Login Nodes**

- LoginNode
- LoginNodeBooster
- LoginNodeVis

**Compute Nodes**

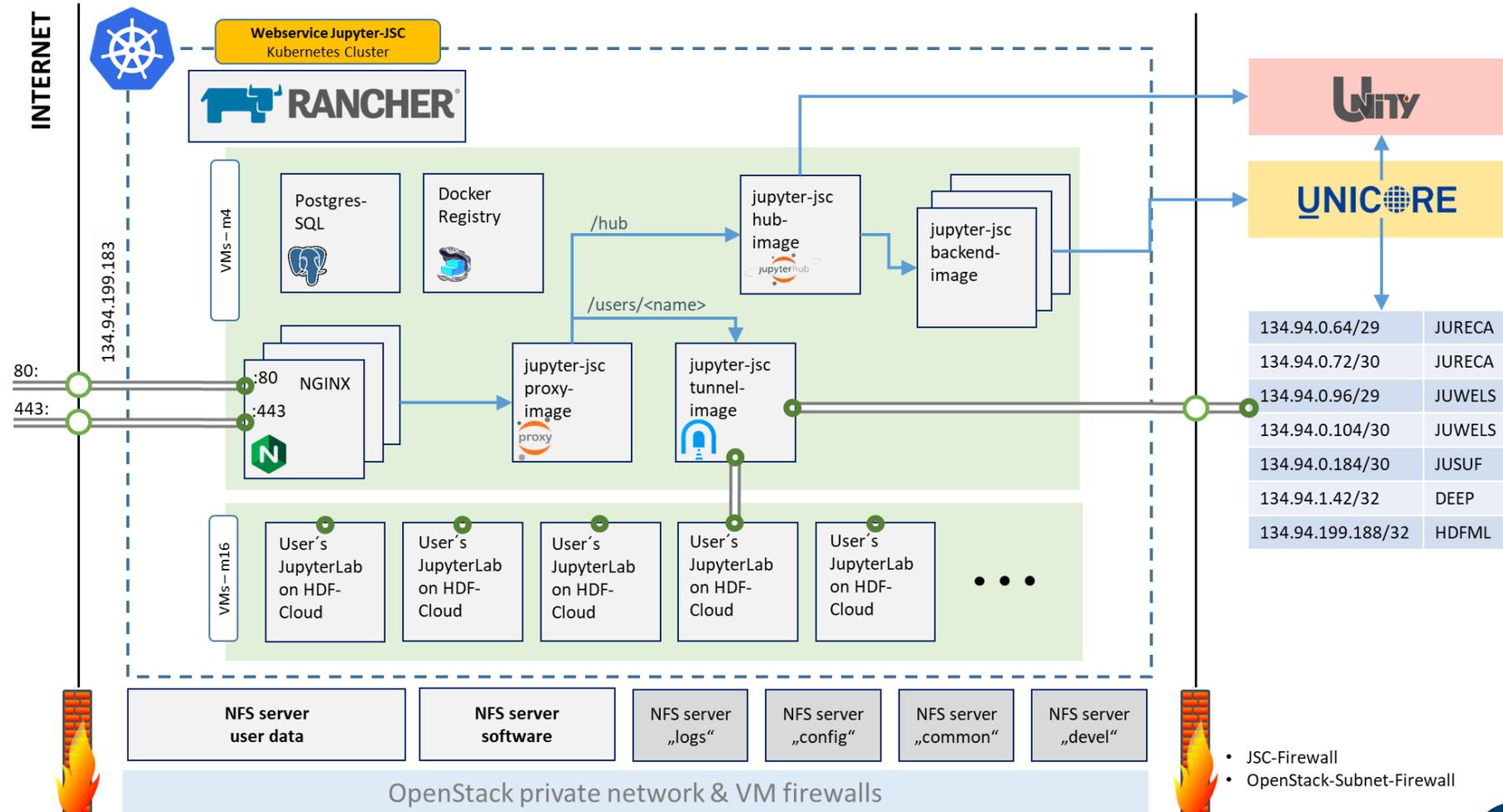
- batch
- devel
- develgpu
- gpu

Name juwels\_clus

Type JupyterLab

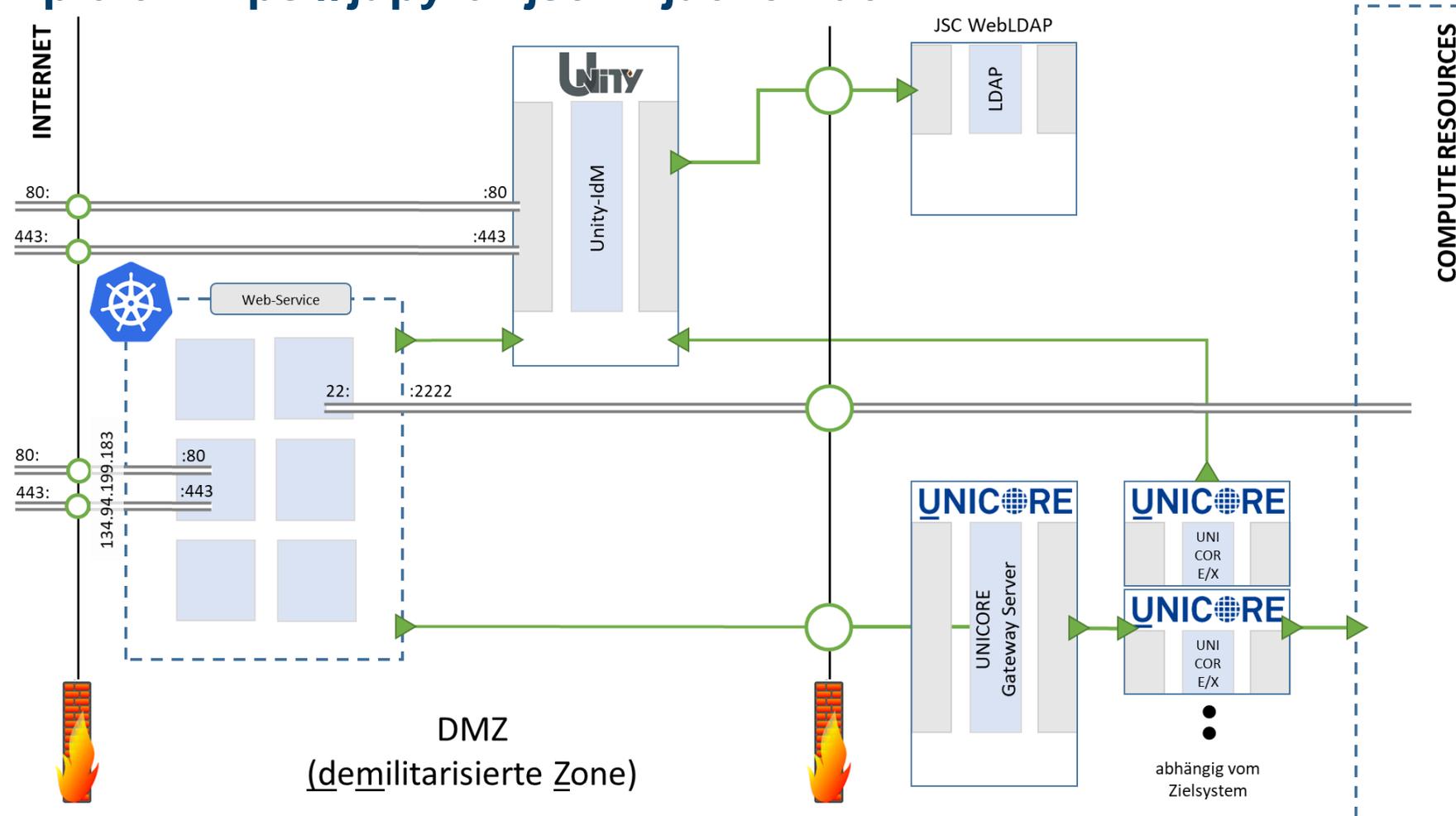
# JUPYTER HUB WEBSERVICE

On the example of <https://jupyter-jsc.fz-juelich.de>



# JUPYTER HUB WEBSERVICE

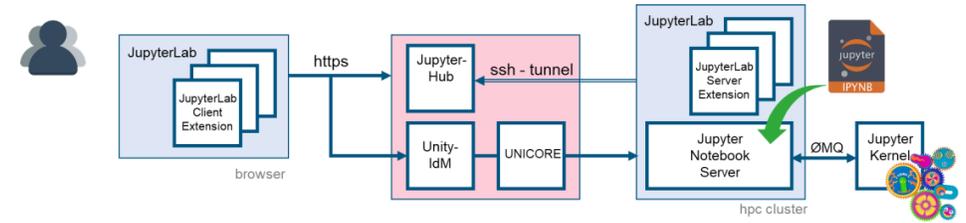
On the example of <https://jupyter-jsc.fz-juelich.de>





# JUPYTER-JSC WEBSERVICE

## Some comments about the UI



Annotations on the JupyterLab interface:

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

# JUPYTERLAB EXTENSIONS

# JUPYTER EXTENSIONS

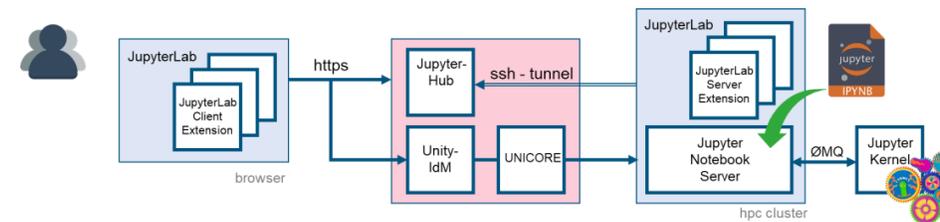
## Some general information

### 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@jrlogin04 jureca]$ jupyter labextension list
JupyterLab v3.2.1
/p/software/jurecadc/stages/2020/software/Jupyter/2021.3.2-gccoreml-10.3.0-2021.2.0-Python-3.8.5/share/jupyter/lab
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
jupyterlab-plotly v5.3.1 enabled OK
jupyterlab-git v0.12.4 enabled OK (python, jupyterlab-git)
@krasnowski/jupyterlab-lsp v3.9.0 enabled OK (python, jupyterlab-lsp)
@jupyter-server/resource-usage v0.6.0 enabled OK (python, jupyter-resource-usage)
@jupyter-widgets/jupyterlab-manager v3.0.1 enabled OK (python, jupyterlab_widgets)
@jupyter-widgets/jupyterlab-sidecar v0.6.1 enabled OK (python, sidecar)
@ryantam26/jupyterlab-code-formatter v1.4.10 enabled OK (python, jupyterlab-code-formatter)
@pyviz/jupyterlab-pyviz v2.1.0 enabled OK (python, pyviz_comms)
Bokesh/jupyter-bokesh v3.0.4 enabled OK (python, jupyter-bokesh)
@jlab-enhanced/favorites v3.0.0 enabled OK (python, jupyterlab-favorites)
@jlab-enhanced/recent v3.0.1 enabled OK (python, jupyterlab-recent)
@voila-dashboards/jupyterlab-previewer v2.1.0-alpha.2 enabled OK (python, voila)
@jimbarr/jupyterlab-spellchecker v0.7.2 enabled OK (python, jupyterlab-spellchecker)

Other labextensions (built into JupyterLab)
app dir: /p/software/jurecadc/stages/2020/software/Jupyter/2021.3.2-gccoreml-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@jrlogin04 jureca]$
```

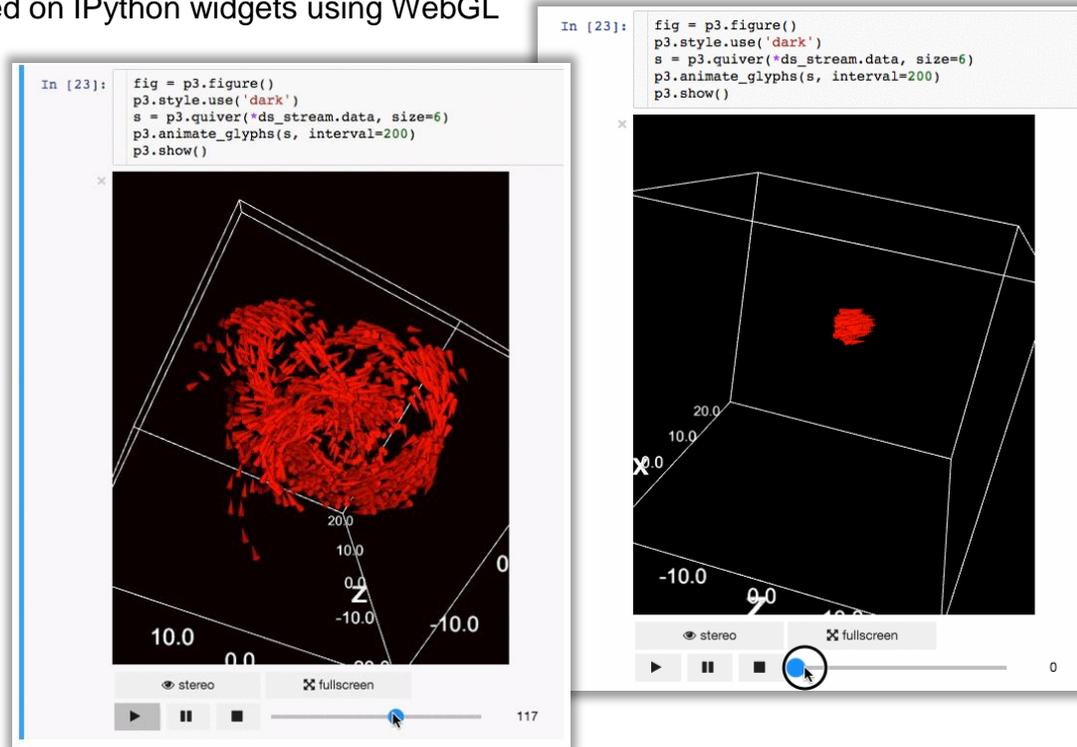
<https://jupyterlab.readthedocs.io/en/stable/user/extensions.html>

# JUPYTER-JSC EXTENSIONS

## Installed by default

### IPyVolume

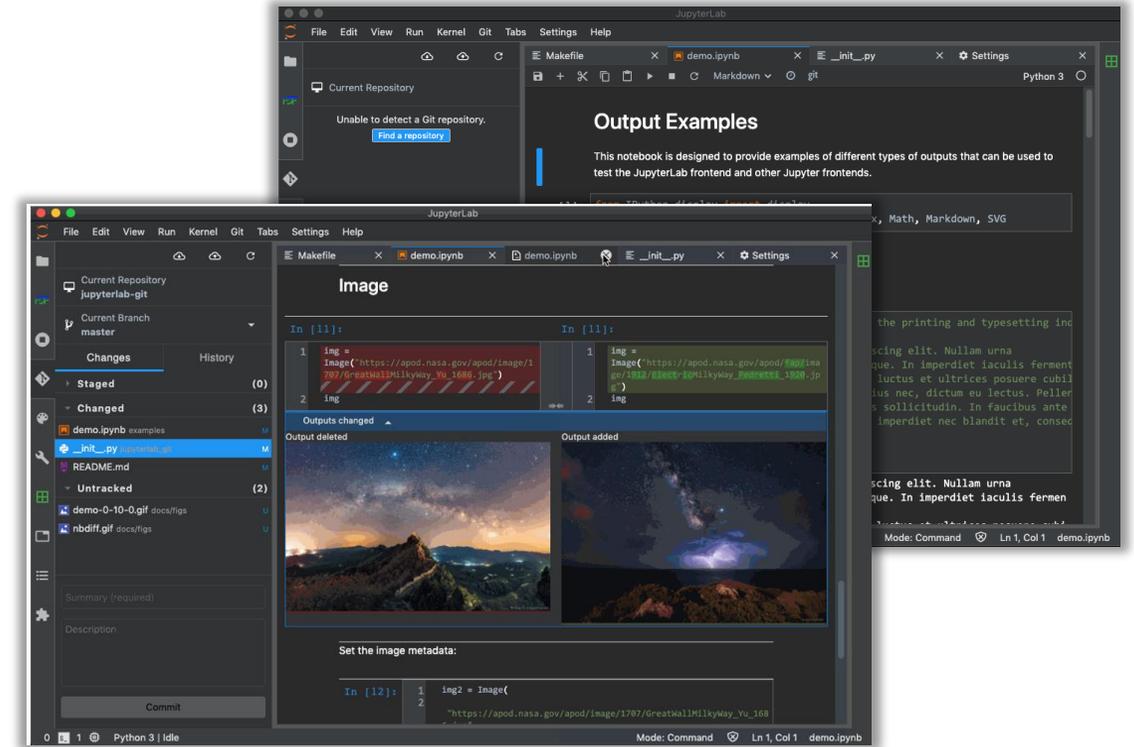
3d plotting for Python in the Jupyter notebook based on IPython widgets using WebGL



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

### JupyterLab-Git

JupyterLab extension for version control using Git



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

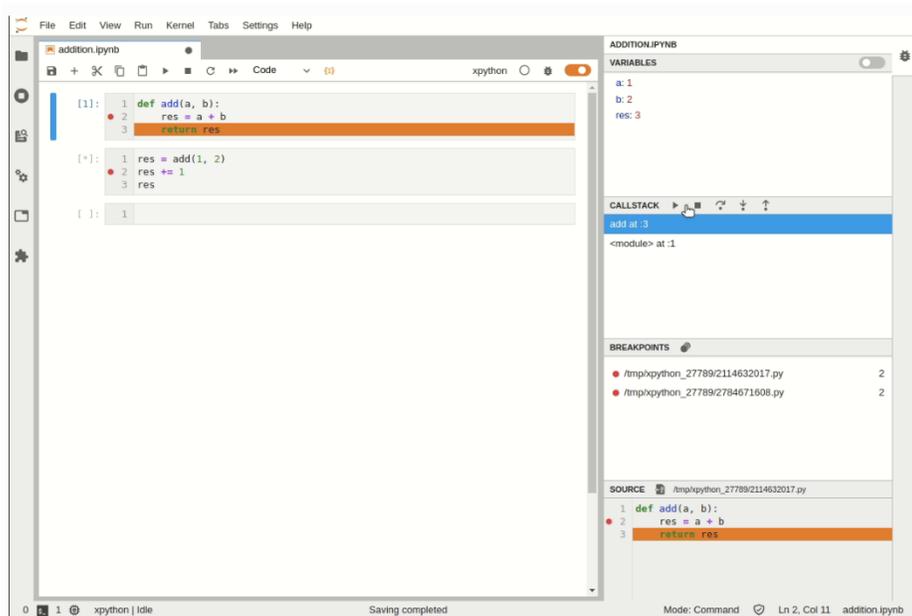
# JUPYTER-JSC EXTENSIONS

## Installed by default

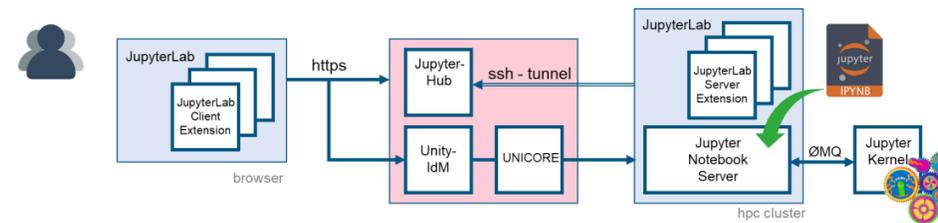
### JupyterLab - Visual Debugger

JupyterLab 3.0 now 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.



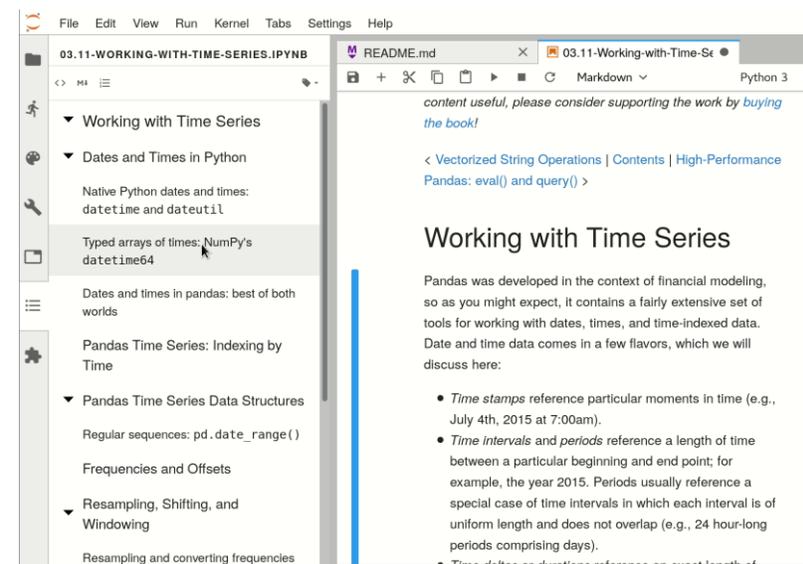
<https://jupyterlab.readthedocs.io/en/stable/user/debugger.html>



### JupyterLab-toc

A Table of Contents extension for JupyterLab.

This auto-generates a table of contents in the left area when you have a notebook or markdown document open. The entries are clickable, and scroll the document to the heading in question.



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

# JUPYTER-JSC EXTENSIONS

## Installed by default

### 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)
```

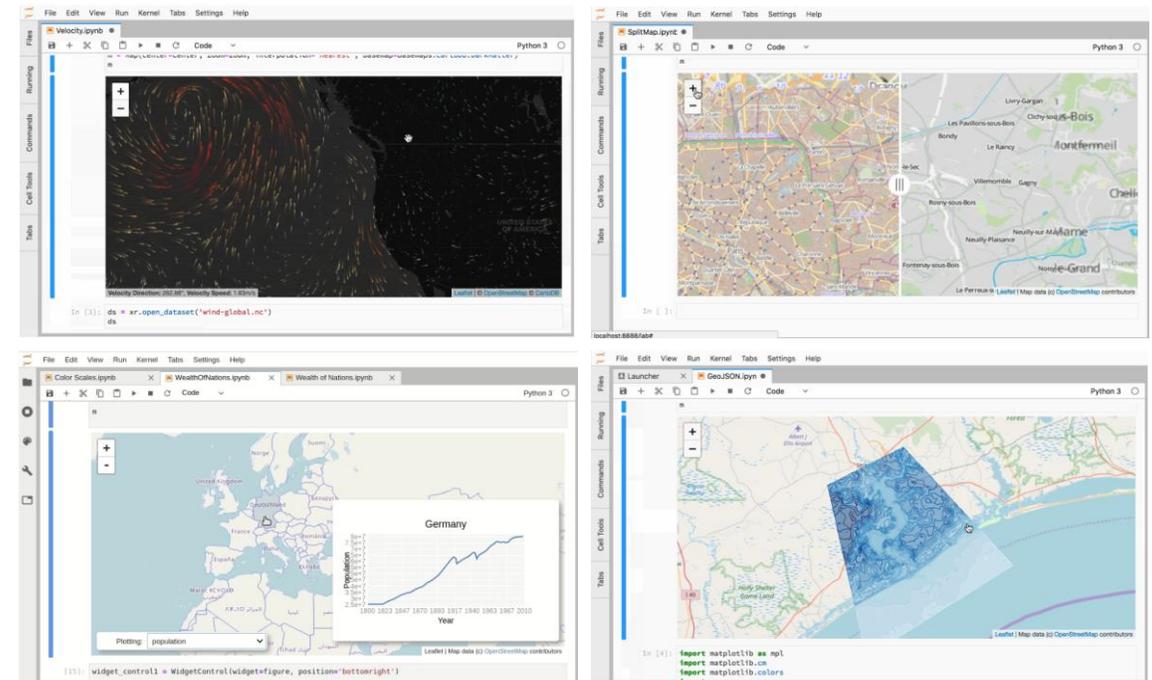


<https://github.com/jupyter-widgets/pythreejs>

Member of the Helmholtz Association

### IPyLeaflet

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



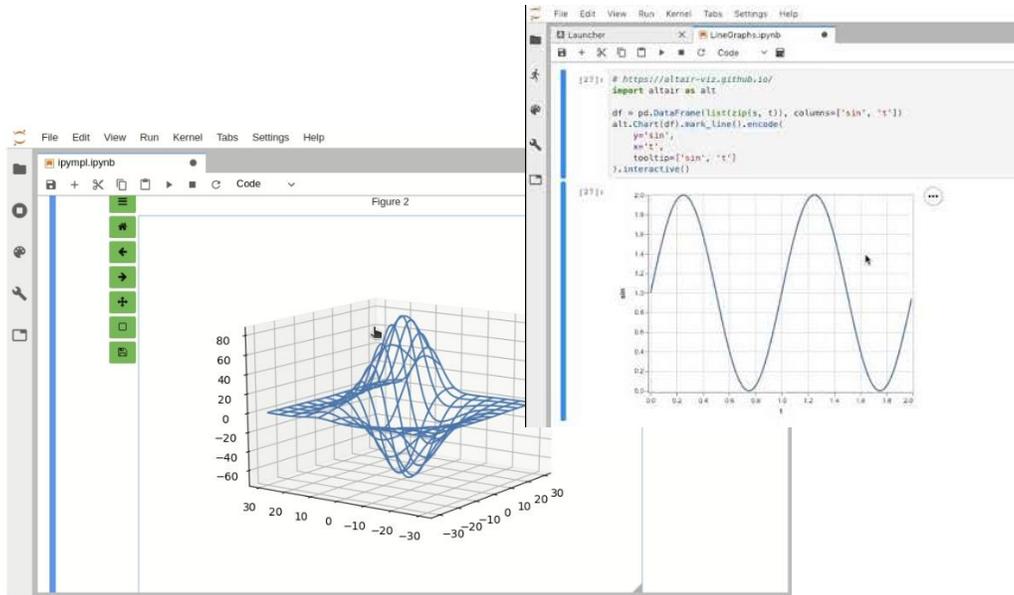
<https://github.com/jupyter-widgets/ipyleaflet>

# JUPYTER-JSC EXTENSIONS

## Installed by default

### IPyMPL - matplotlib

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

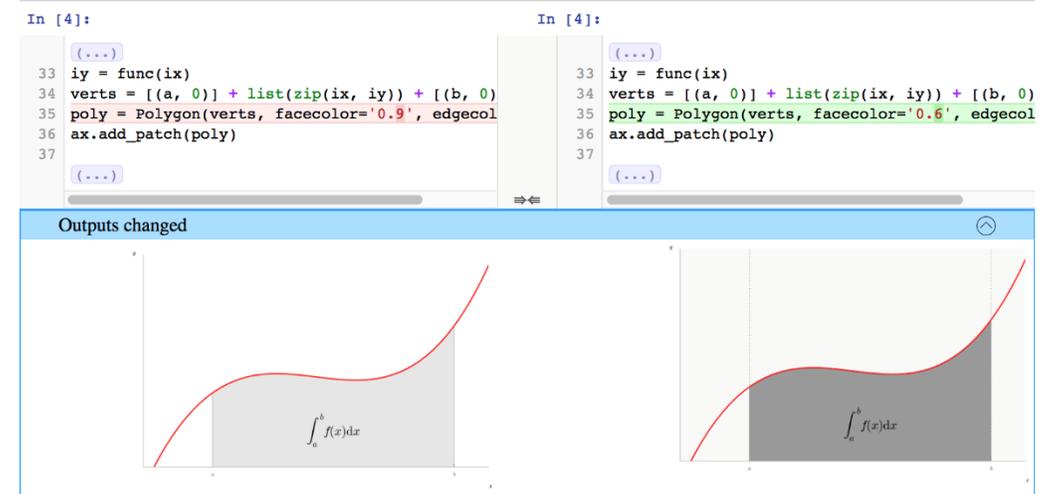


<https://github.com/matplotlib/ipympl>

Member of the Helmholtz Association

### NBDime

Tools for diffing and merging of Jupyter notebooks.



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

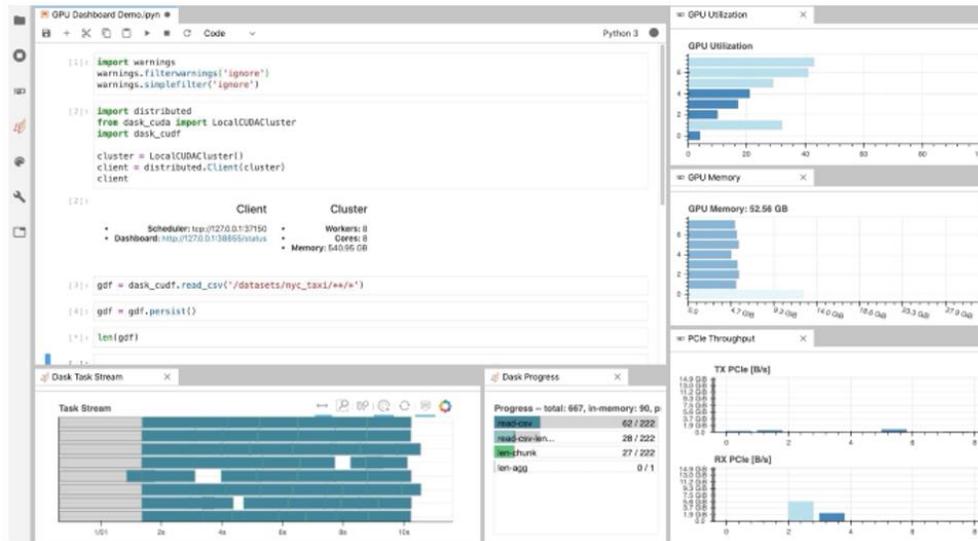


# JUPYTER-JSC EXTENSIONS

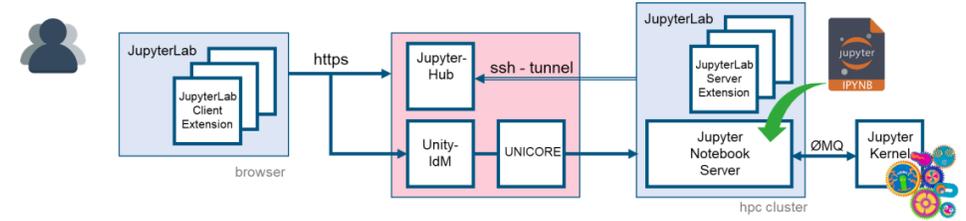
## Installed by default

### NVDashboard

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

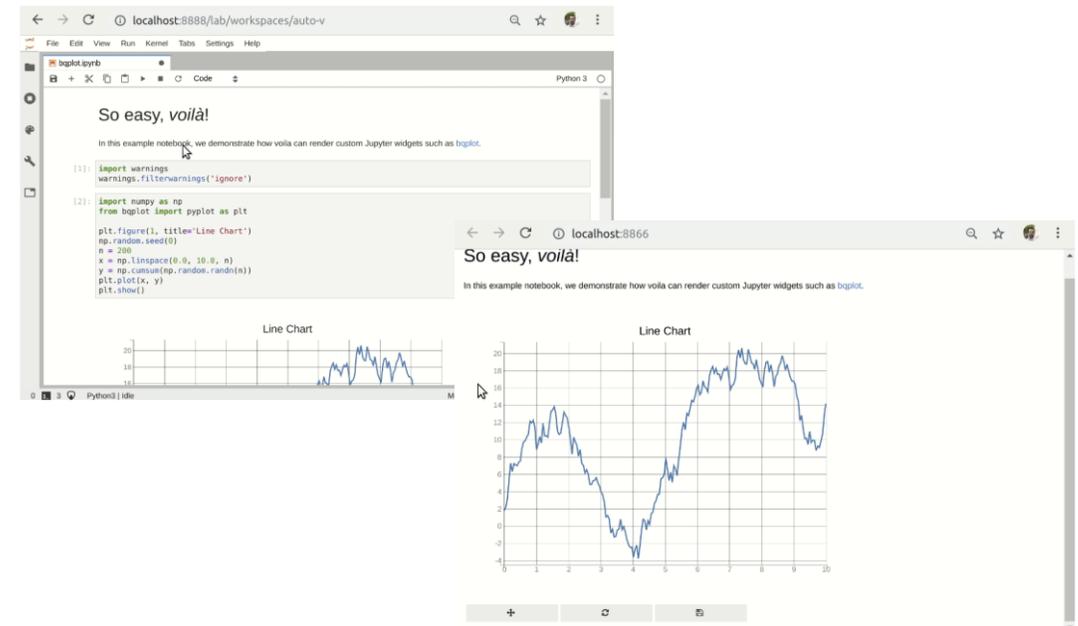


<https://github.com/rapidsai/jupyterlab-nvdashboard>  
<https://developer.nvidia.com/blog/gpu-dashboards-in-jupyter-lab/>



### Voilà

Voilà turns Jupyter notebooks into standalone web applications.

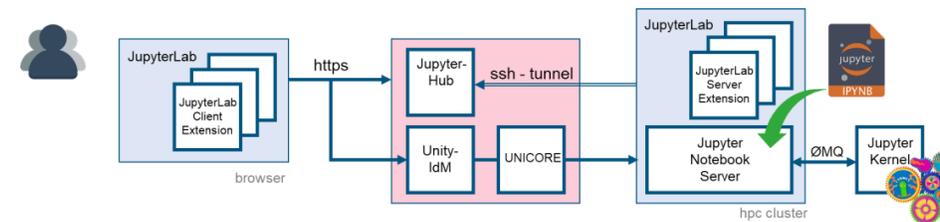


<https://github.com/voila-dashboards/voila>

# JUPYTER-JSC EXTENSIONS

## Installed by default

Extensions	old version	new version	type
<b>Core</b>			
<a href="#">@jupyterlab/server-proxy</a>	v2.1.0	v3.1.0	prebuild
<a href="#">@jupyter-widgets/jupyterlab-manager</a>	v2.0.0	v3.0.1	prebuild
<a href="#">jupyterlab-datawidgets</a>	v6.3.0	v7.0.0	source
<b>UI Enhancements</b>			
<a href="#">@jlab-enhanced/recents</a>		v3.0.1	prebuild
<a href="#">@jlab-enhanced/favorites</a>	v2.0.0	v3.0.0	prebuild
<a href="#">jupyterlab-topbar-extension</a>	v0.5.0	v0.6.1	
<a href="#">jupyterlab-system-monitor</a>	v0.6.0	v0.8.0	prebuild
<a href="#">@jupyter-server/resource-usage</a>		v0.6.0	n/a
<a href="#">jupyterlab-theme-toggle</a>	v0.5.0	v0.6.1	source
<a href="#">jupyterlab-controlbtn</a>	<a href="#">jupyterlab-control</a>	v0.5.0	n/a
<a href="#">@jupyterlab/toc</a>	v4.0.0	integrated into JupyterLab 3	
<b>Developer Tools</b>			
<a href="#">@jupyterlab/git</a>	v0.23.3	v0.32.4	prebuild
<a href="#">jupyterlab-gitlab</a>	v2.0.0	v3.0.0	prebuild
<a href="#">@krassowski/jupyterlab-lsp</a>	v2.1.3	v3.9.0	prebuild
<a href="#">nbdime-jupyterlab</a>	v2.1.0	v3.1.0	prebuild
<a href="#">@ryantam626/jupyterlab_code_formatter</a>	v1.3.8	v1.4.10	prebuild
<a href="#">@jjmbarr/jupyterlab_spellchecker</a>	v0.2.0	v0.7.2	prebuild
<a href="#">jupyterlab-nvdashboard</a>		v0.6.0	prebuild



### Data Visualization

<a href="#">jupyter-matplotlib</a>	v0.7.4	v0.9.0	prebuild
<a href="#">@bokeh/jupyter_bokeh</a>	v2.0.4	v3.0.4	prebuild
<a href="#">jupyterlab-plotly</a>	v4.14.3	v5.3.1	
<a href="#">bqplot</a>	v0.5.22	v0.5.32	prebuild
<a href="#">@pyviz/jupyterlab_pyviz</a>	v1.0.4	v2.1.0	prebuild
<a href="#">jupyter-leaflet</a>	v0.13.3	v0.14.0	prebuild
<a href="#">ipyvolume</a>	v0.6.0-alpha.5	v0.6.0-alpha.8	prebuild
<a href="#">jupyter-threejs</a>	v2.2.0	v2.3.0	prebuild
<a href="#">@jupyter-widgets/jupyterlab-sidecar</a>	v0.5.0	v0.6.1	prebuild

### Framework Integrations

<a href="#">dask-labextension</a>	v3.0.0	v5.1.0	prebuild
<a href="#">@jupyterlab/latex</a>	v2.0.1	v3.1.0	prebuild
<a href="#">jupyter-webrtc</a>	v0.5.0	v0.6.0	prebuild

### Dashboard Development

<a href="#">jupyter-vue</a>	v1.5.0	v1.6.1	
<a href="#">jupyter-vuetify</a>	v1.6.1	v1.8.1	
<a href="#">@voila-dashboards/jupyterlab-preview</a>	v1.1.0	v2.1.0-alpha.2	prebuild
<a href="#">jupyterlab-dash</a>	v0.4.0	v0.4.0	prebuild

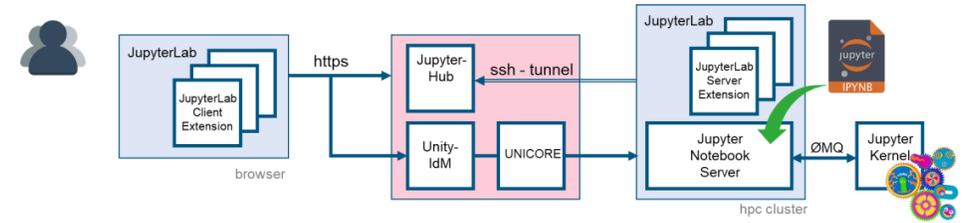
### Welcome

<a href="#">jupyterlab_iframe</a>	v0.3.0	v0.4.0	source
<a href="#">jupyterlab-tour</a>		v3.1.3	prebuild

# JUPYTER KERNEL

# JUPYTER KERNEL

## How to create your own Jupyter Kernel



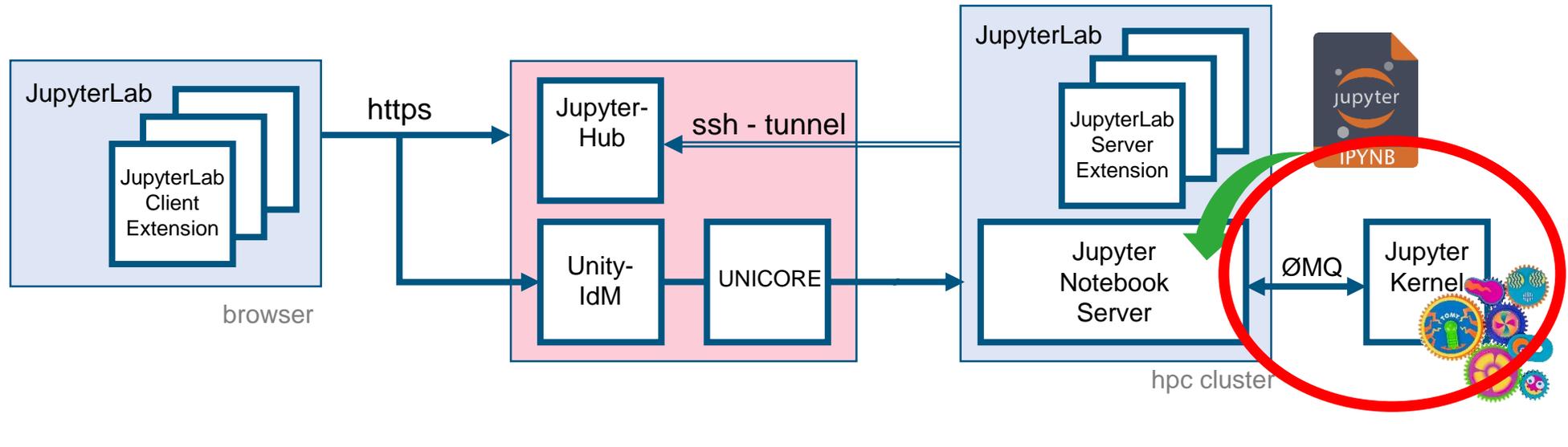
### Jupyter Kernel

A “kernel” refers to the separate process

w/

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

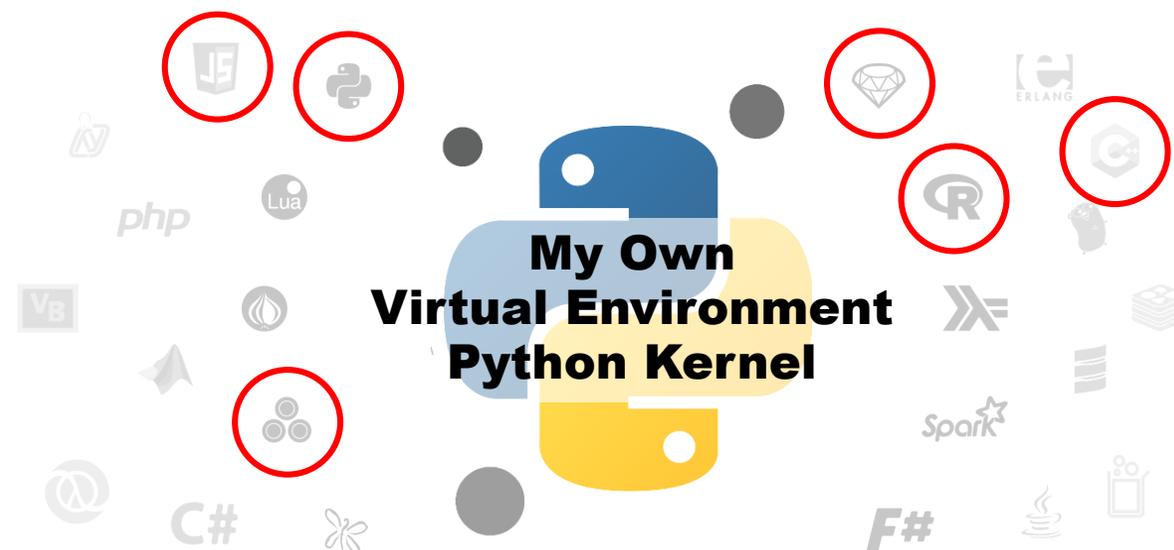
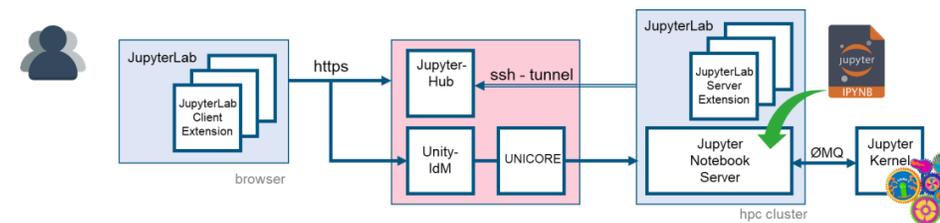
### Jupyter Kernel

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, quantumcomputing

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

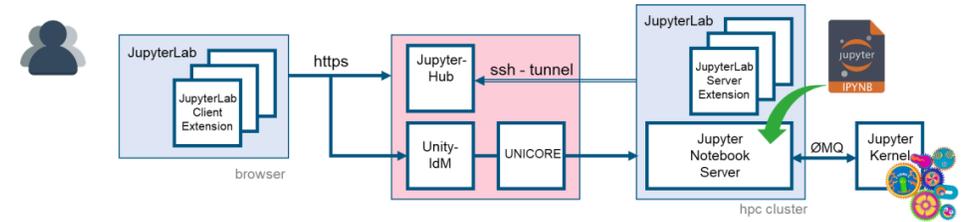
### Jupyter Kernel

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, quantumcomputing

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



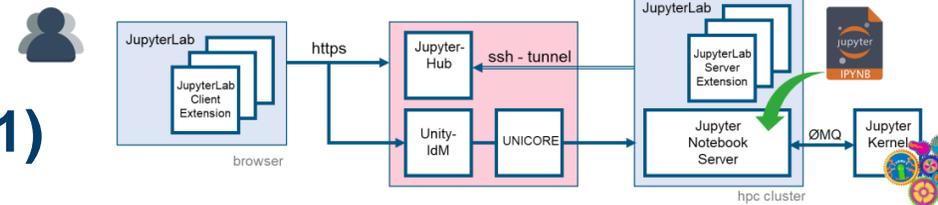
### Building your own Jupyter kernel is a three step process

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

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



1. Login to JupyterLab and open terminal

2. Load required modules

```
Inode:> module purge
Inode:> module use $OTHERSTAGES
Inode:> module load Stages/2020
Inode:> module load GCCcore/.10.3.0
Inode:> module load Python/3.8.5
```

3. Load extra modules you need for your kernel

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

1. Create a virtual environment named <venv\_name> at a path of your choice:

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

2. Activate your environment

```
Inode:> 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

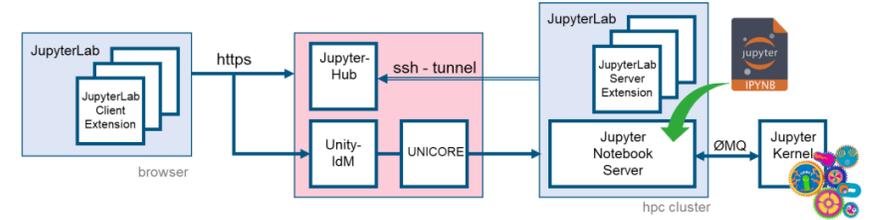
```
(<venv_name>) Lnode:> export PYTHONPATH=\n${VIRTUAL_ENV}/lib/python3.8/site-packages:${PYTHONPATH}
```

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:

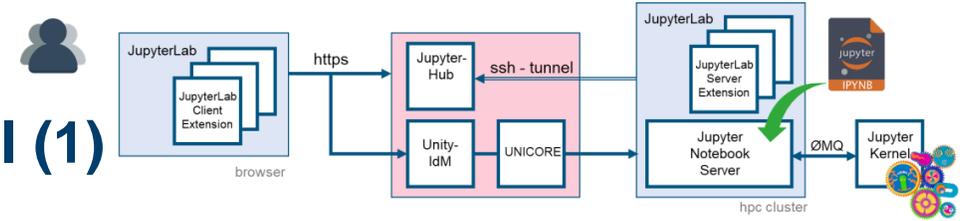
```
(<venv_name>) Lnode:> touch ${VIRTUAL_ENV}/kernel.sh
```

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
```

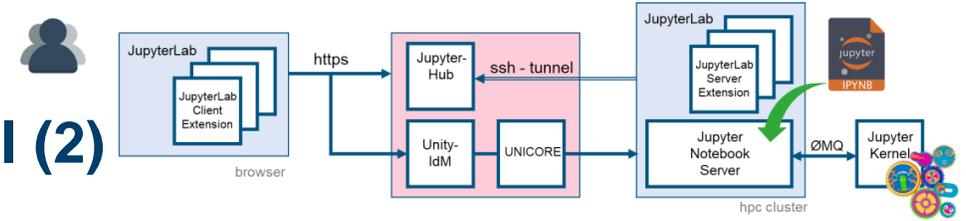


### Building your own Jupyter kernel is a three step process

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 $OTHERSTAGES
```

```
module load Stages/2020
```

```
module load GCCcore/.10.3.0
```

```
module load Python/3.8.5
```

```
# 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=${VIRTUAL_ENV}/lib/python3.8/site-packages:${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

[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)

Member of the Helmholtz Association

# JUPYTER KERNEL

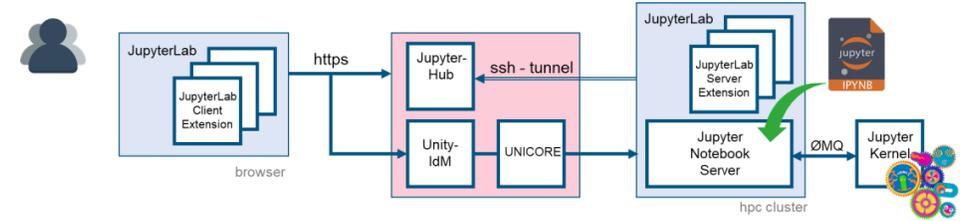
## 3. Create/Edit Jupyter kernel configuration (1)

### 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

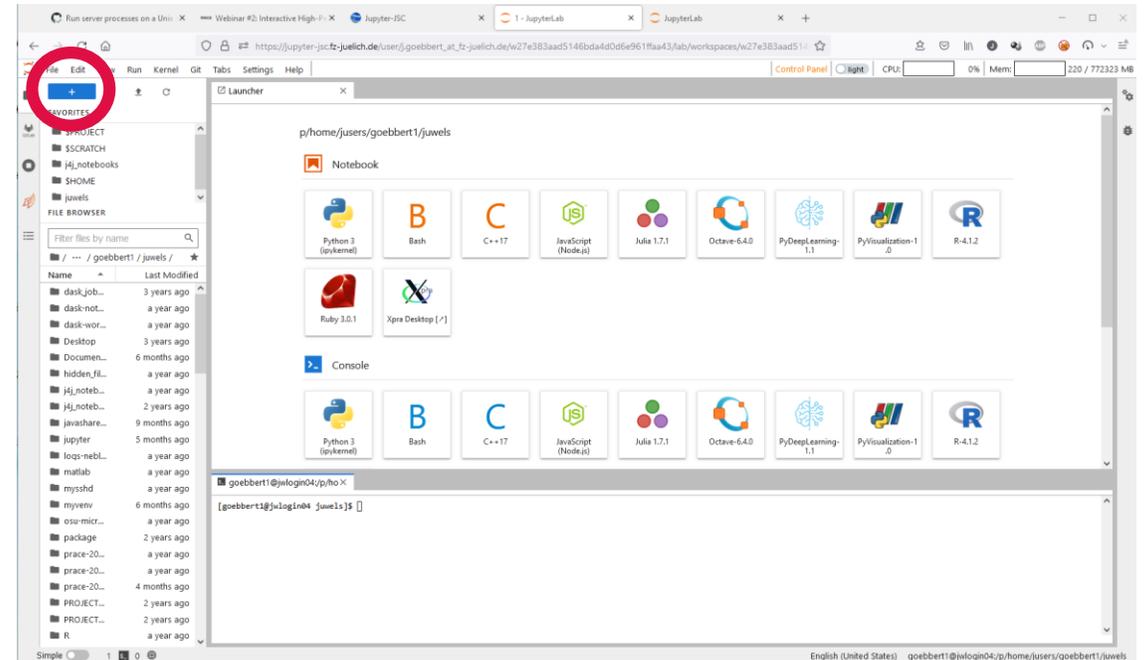
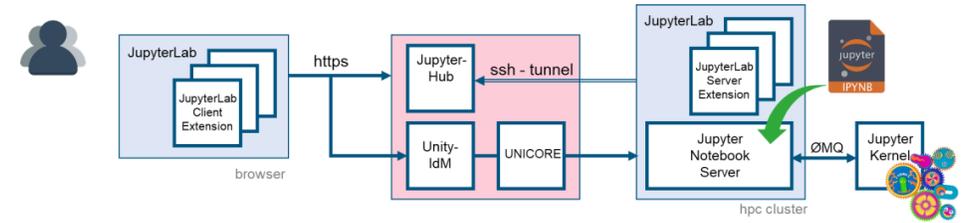
### Run your Jupyter Kernel

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.

### Conda

How to 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)

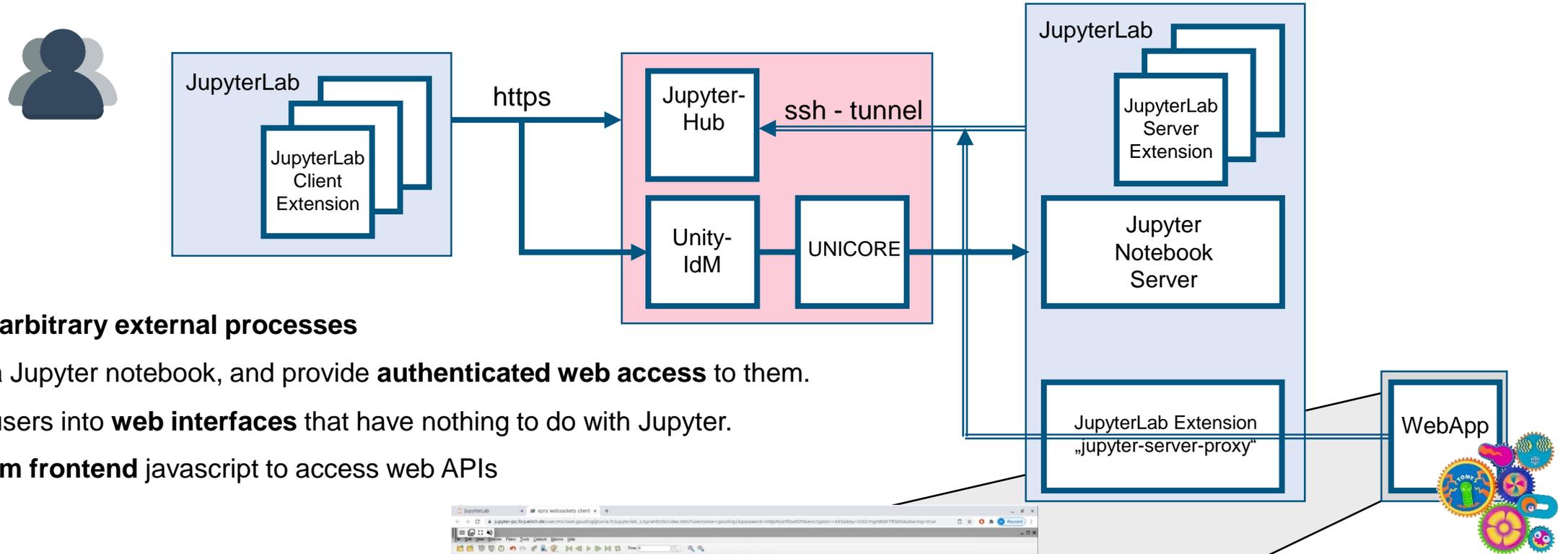


[https://gitlab.version.fz-juelich.de/jupyter4jsc/j4j\\_notebooks/-/blob/master/001-Jupyter/Create\\_JupyterKernel\\_general.ipynb](https://gitlab.version.fz-juelich.de/jupyter4jsc/j4j_notebooks/-/blob/master/001-Jupyter/Create_JupyterKernel_general.ipynb)

# JUPYTER CAN DO MORE

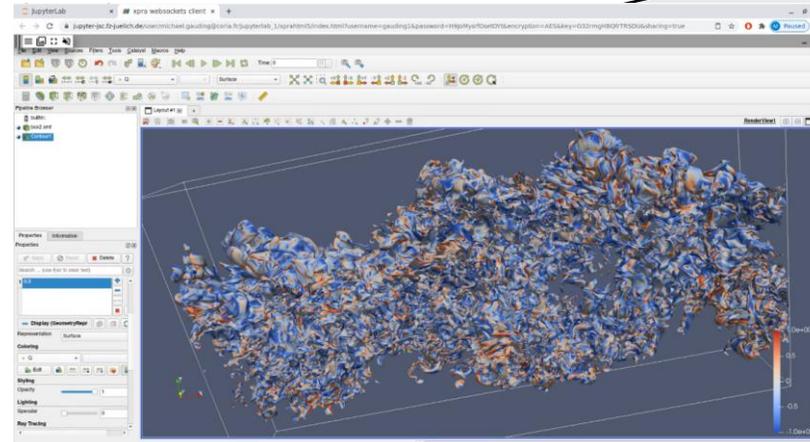
# JUPYTERLAB – WEBSERVICE PROXY

## 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



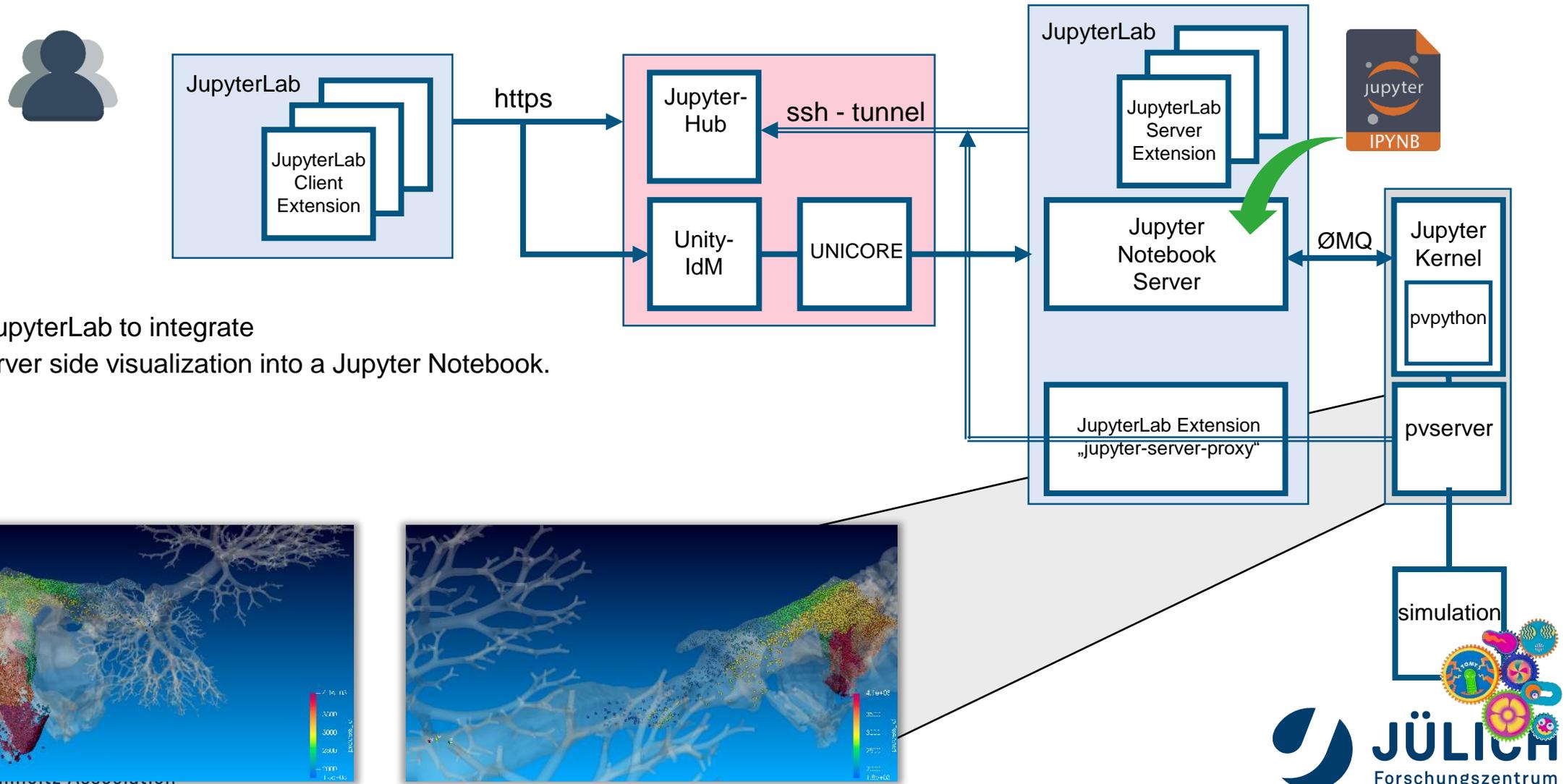
Turbulent mixing with variable density, subset of 1939x600x3584 grid points, Michael Gauding, CORIA

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

Member of the Helmholtz Association

# JUPYTERLAB – WEBSERVICE PROXY

## Extension: jupyter-server-proxy



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

# JUPYTERLAB – WEBSERVICE PROXY

## Extension: jupyter-server-proxy

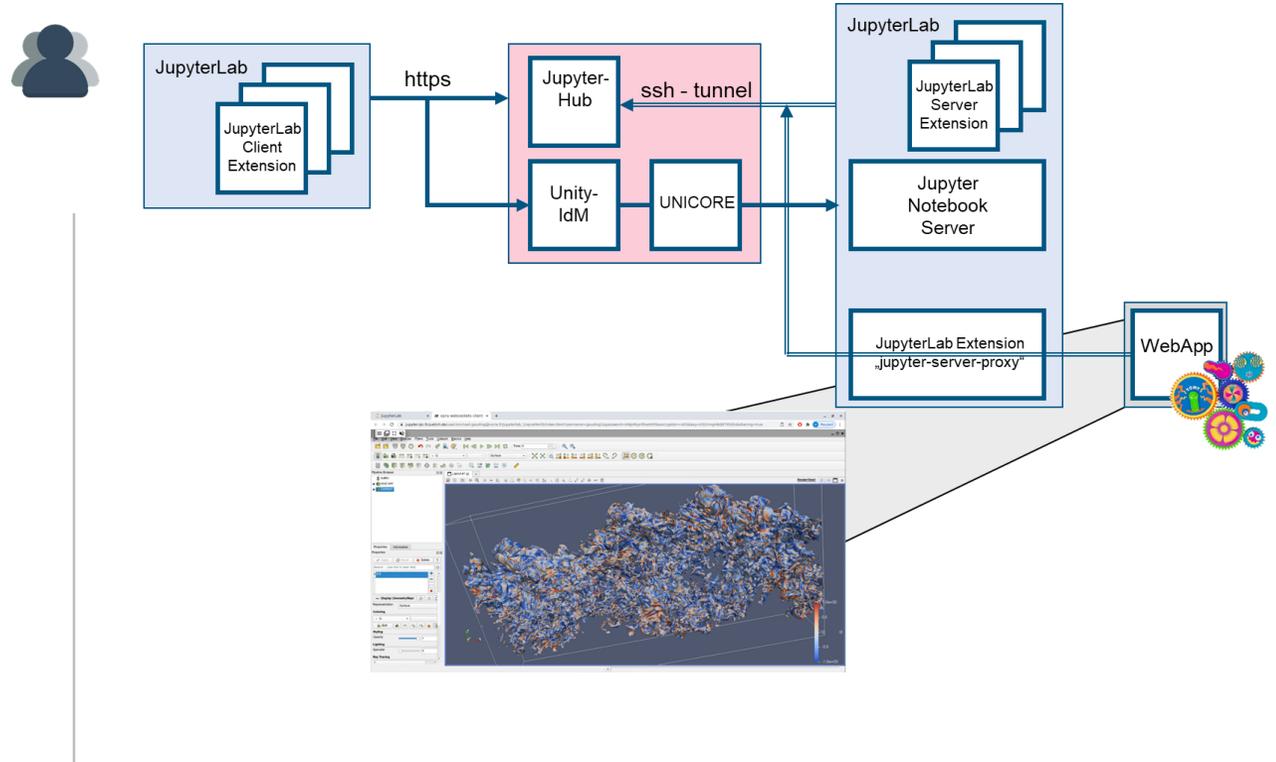
### Accessing Arbitrary Ports or Hosts

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/12345](https://jupyter-jsc.fz-juelich.de/user/j.goebbert@fz-juelich.de/juwels_login/proxy/12345)

# 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.

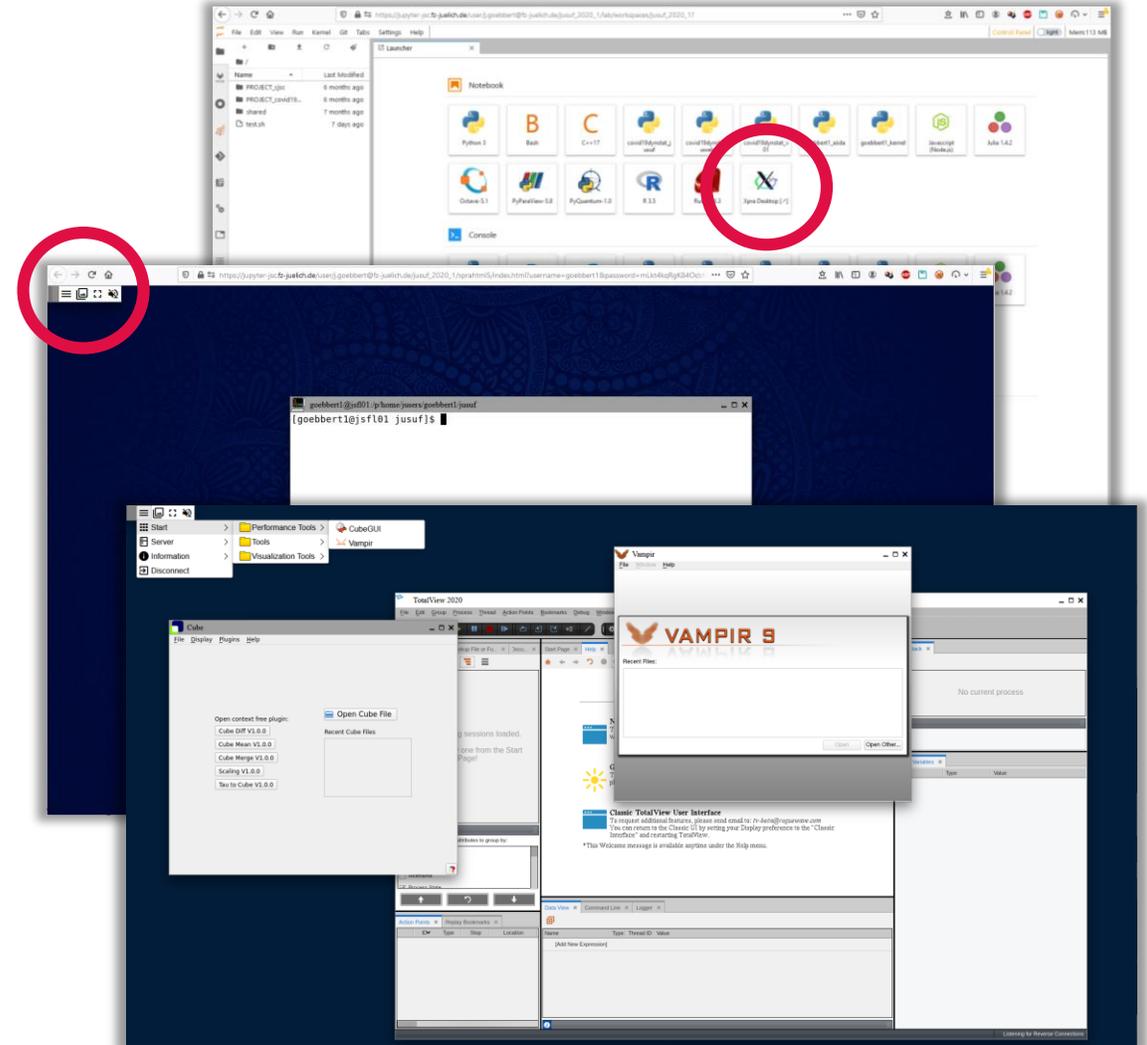
- Runs in a browser
- allows dis-/reconnection without disrupting the forwarded application
- <https://xpra.org>

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.

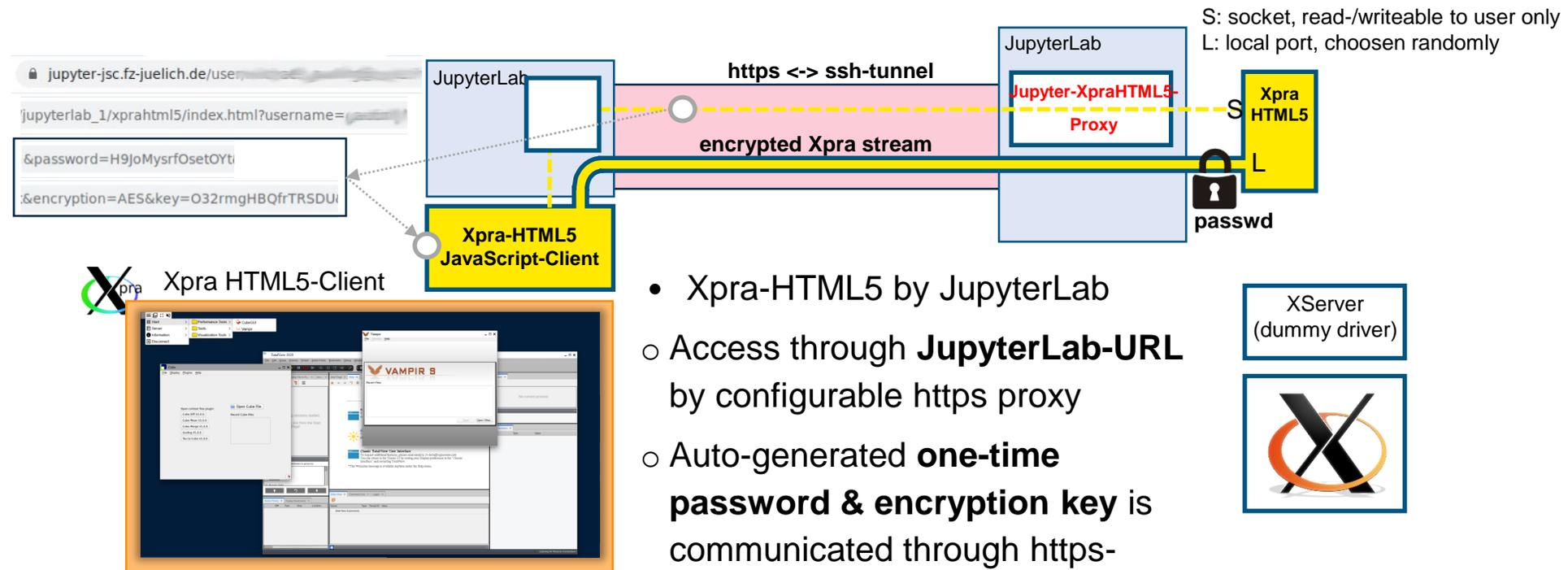
Hint:

- 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>

Member of the Helmholtz Association

# BENEFITS

## Why Jupyter is so popular among Data Scientists

Some of the reasons ...

- Jupyter allows to **view the results of the code in-line** without the dependency of other parts of the code.
- Jupyter mixes easy for users who extend their code **line-by-line with feedback** attached all along the way
- Jupyter Notebooks support visualization and include rendering data in **live-graphics and charts**.
- Jupyter is maintaining the **state of execution of each cell** automatically.
- Supports IPyWidget packages, which provide **standard user interface** for exploring code and data interactively.
- Platform and language **independent** because of its representation in JSON format.

# TUTORIALS

## Get started with Jupyter – published 2018 but still a great starting point

Possible start to enter the world of interactive computing with IPython in Jupyter:

- Leverage the Jupyter Notebook for interactive data science and visualization
- High-performance computing and visualization for data analysis and scientific modeling
- A comprehensive coverage of scientific computing through many hands-on, example-driven recipes with detailed, step-by-step explanations



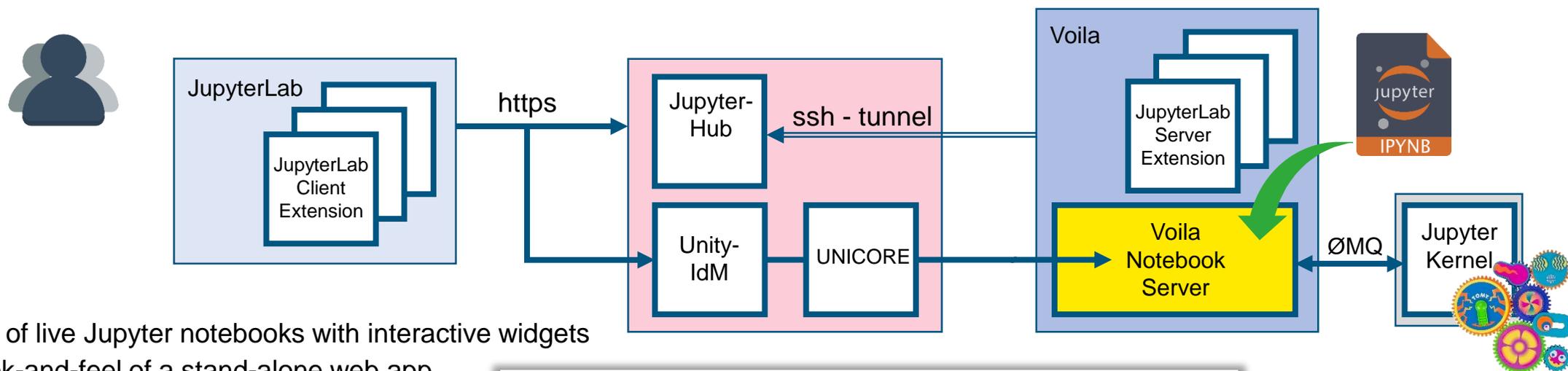
<https://ipython-books.github.io>  
<https://github.com/ipython-books/cookbook-2nd>

# QUESTIONS?

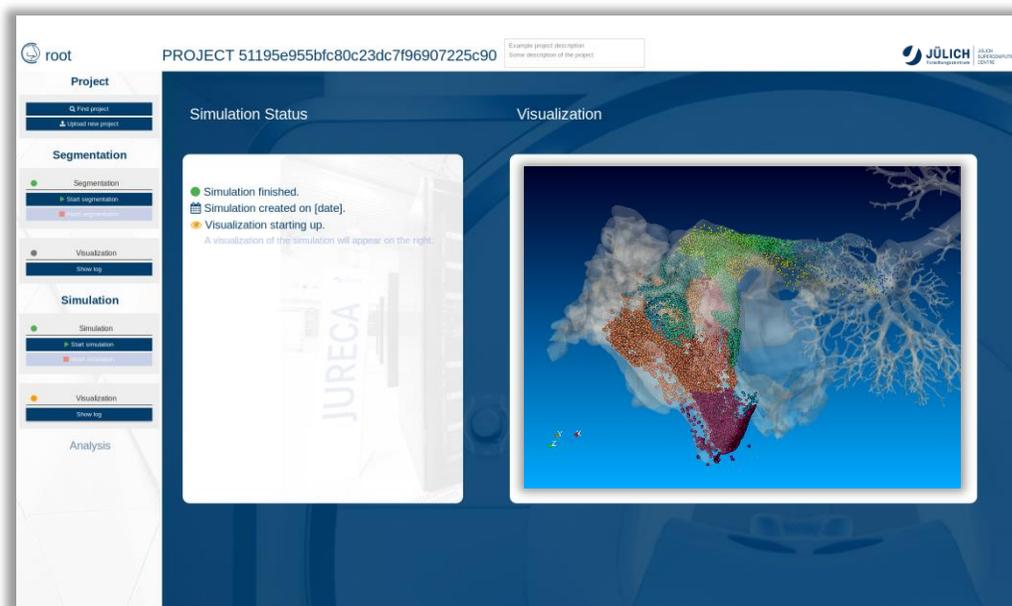


# DASHBOARDS WITH JUPYTER/VOILA

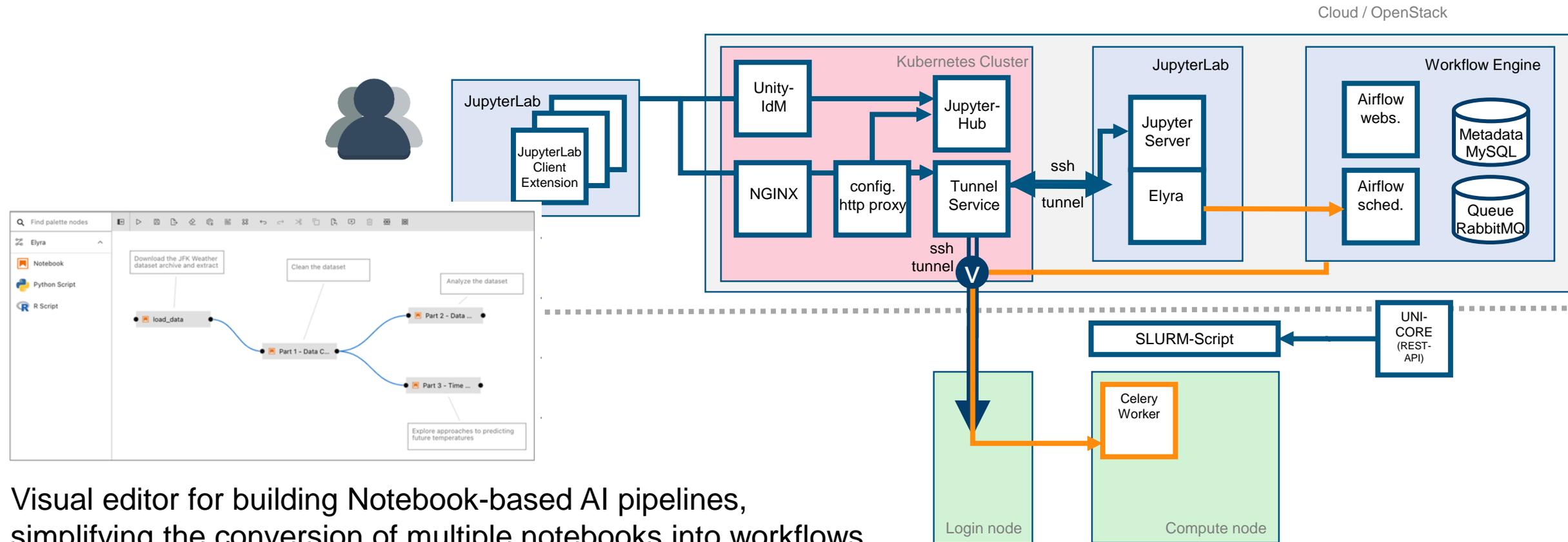
Voilà turns Jupyter notebooks into standalone web applications



- **Rendering** of live Jupyter notebooks with interactive widgets with the look-and-feel of a stand-alone web app.
- Voilà disallows execute requests from the front-end, **preventing** execution of arbitrary code.
- **Enables** HPC users to develop easily web applications from their Jupyter notebooks.



# ELYRA – WORKFLOW UI FOR JUPYTERLAB



Visual editor for building Notebook-based AI pipelines, simplifying the conversion of multiple notebooks into workflows. With custom components new functionality can be added [3]

[1] <https://elyra.readthedocs.io>

[2] <https://developer.ibm.com/blogs/open-source-elyra-ai-toolkit-simplifies-data-model-development>

[3] <https://medium.com/ibm-data-ai/elyra-3-3-pipelines-custom-components-and-catalogs-74cf198fdf48>

[4] <https://github.com/IBM/claimed>