



JUPYTERLAB - SUPERCOMPUTING IN YOUR BROWSER

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

2020-11-25 | JENS. H. GÖBBERT (J.GOEBBERT@FZ-JUELICH.DE)

TIM KREUZER (T.KREUZER@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://jupyter.org>

Member of the Helmholtz Association

MOTIVATION

Rise of Jupyter's popularity

If popularity can be counted by

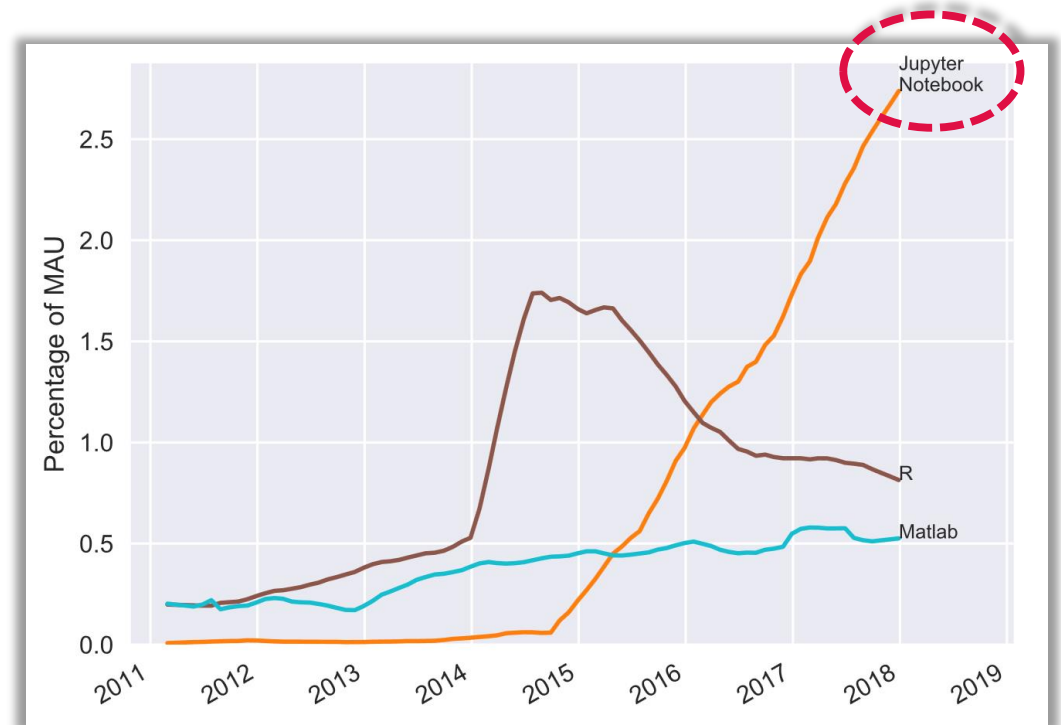
- Monthly aggregated number of user interactions with GitHub repos (= Monthly Active Users (MAU))

and

- Each repository is assigned to a single language (by looking at which language has the most bytes in the repo)

Jupyter Notebooks have seen significant and steady growth over the last years (still rising).

- Of course the popularity of Python in general is pushing this trend.



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

TERMINOLOGY

What is JupyterLab

JupyterLab

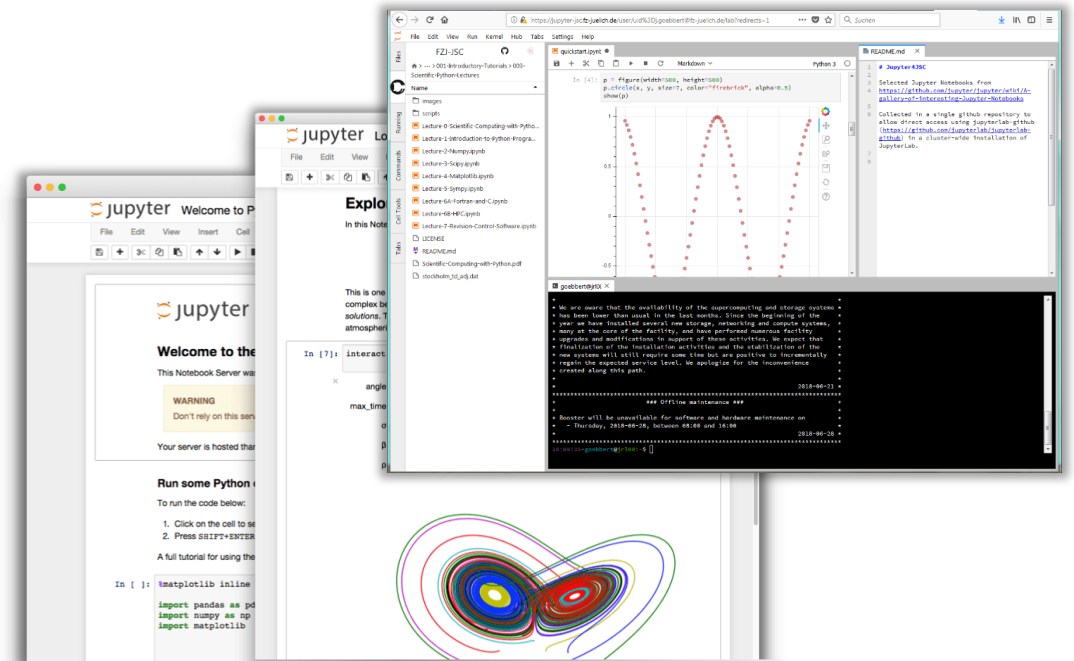
- **Interactive** working environment in the web browser
- For the creation of **reproducible** computer-aided narratives
- Very **popular** with researchers from all fields
- Jupyter = Julia + Python + R

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>

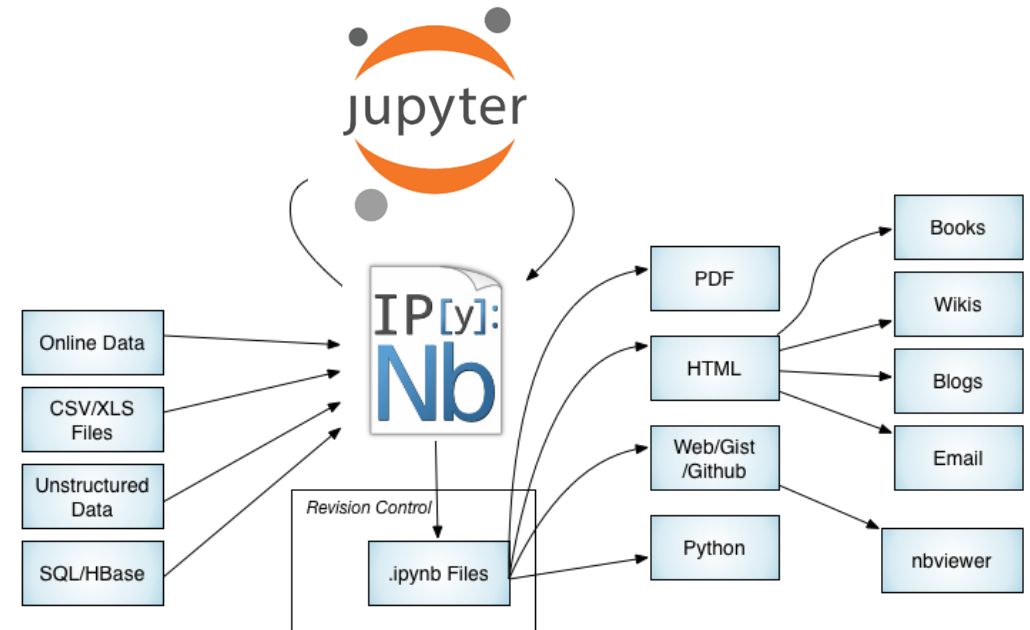
TERMINOLOGY

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, quantumcomputing
- 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?

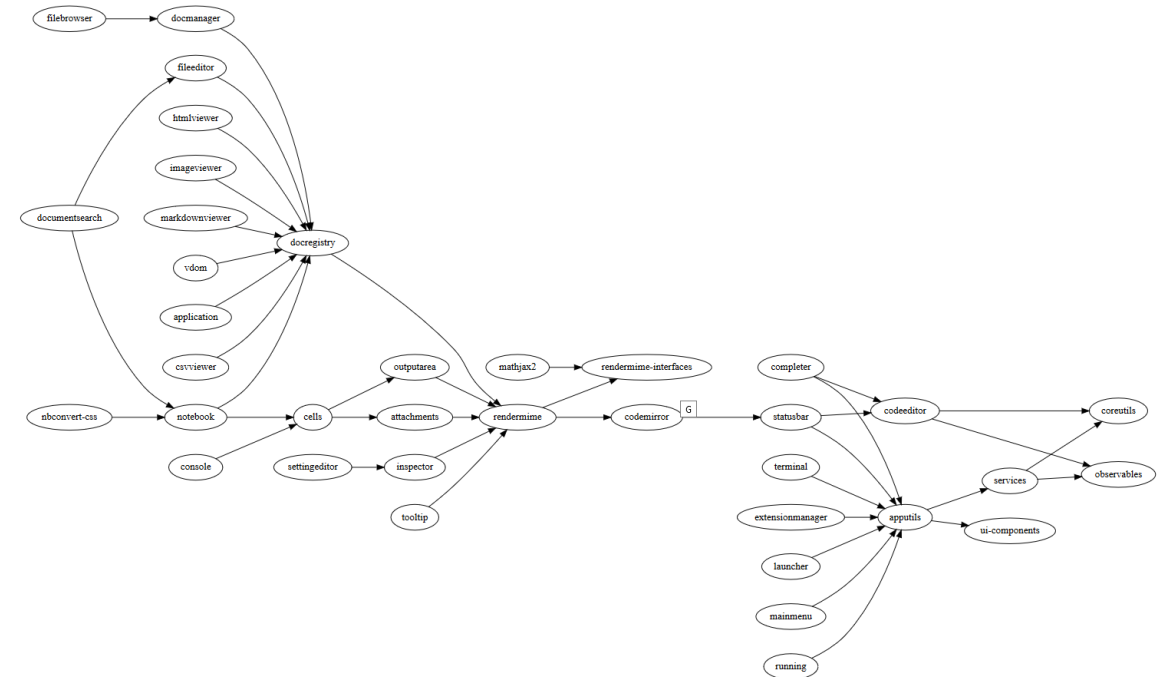
JupyterLab Extension

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.

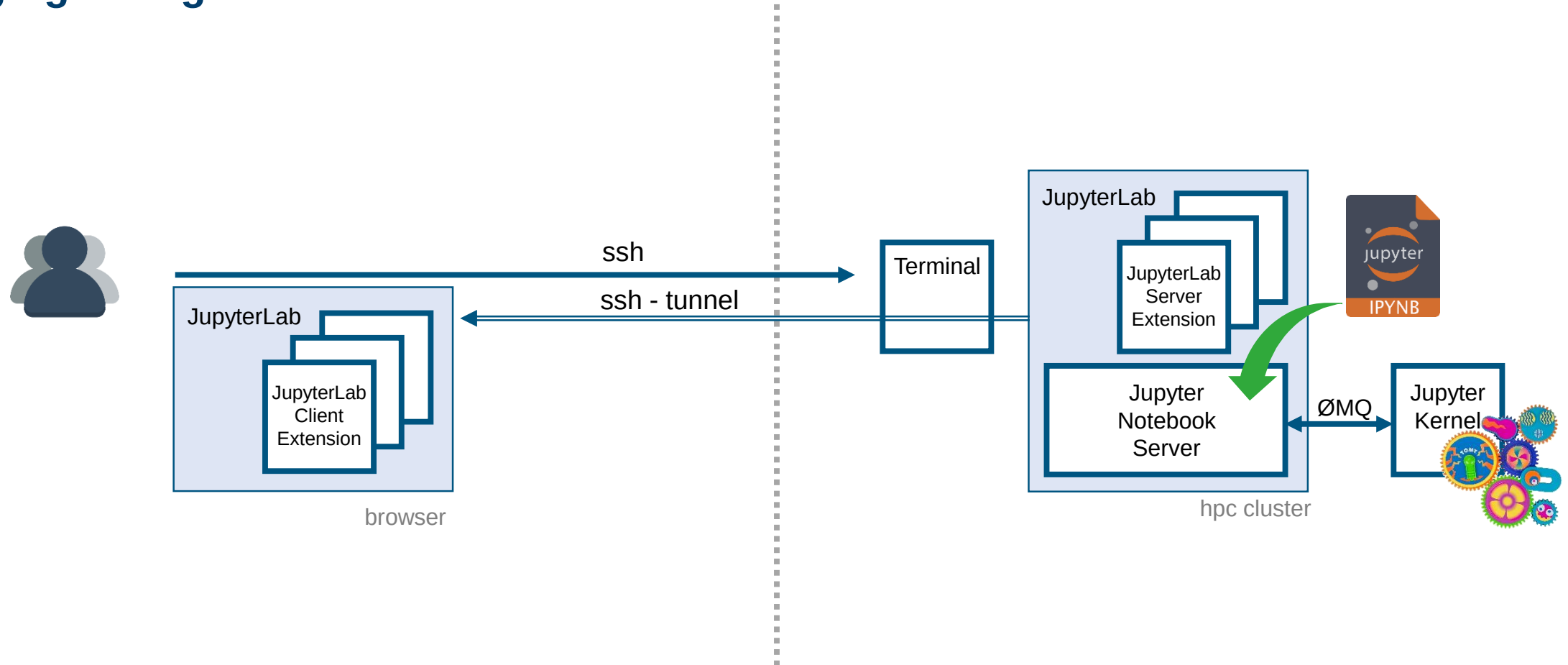
The whole JupyterLab itself is simply a **collection of extensions** that are no more powerful or privileged than any custom extension.



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

TERMINOLOGY

Bringing all together



JUPYTER – START & TUNNEL

Start your JupyterLab (the hard way)

Start Jupyter on the login node

```
Lnode:> module purge
Lnode:> module use $OTHERSTAGES
Lnode:> module load Stages/Devel-2019a
Lnode:> module load GCC/8.3.0
Lnode:> module load JupyterCollection/2019a.2.4

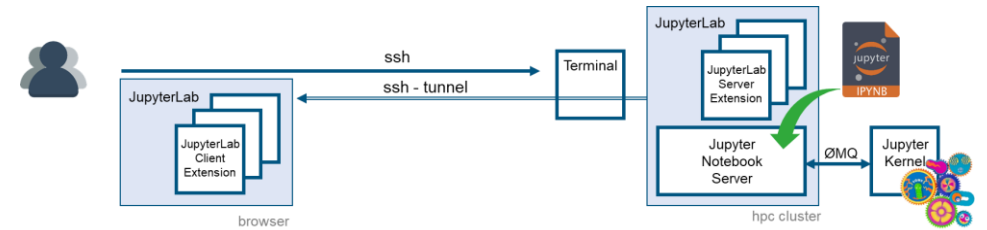
Lnode:> cd $PROJECT_<my_project>
Lnode:> jupyter lab
```

```
[I 20:44:05.916 NotebookApp] Writing notebook server cookie secret to
/run/user/12885/jupyter/notebook_cookie_secret
```

[...]

Copy/paste this URL into your browser when you connect for the first time, to login with a token:

<http://localhost:8888/?token=7f1f8d7d9414a8b72j2e2cc2c2866c29fb557677e9a08042>

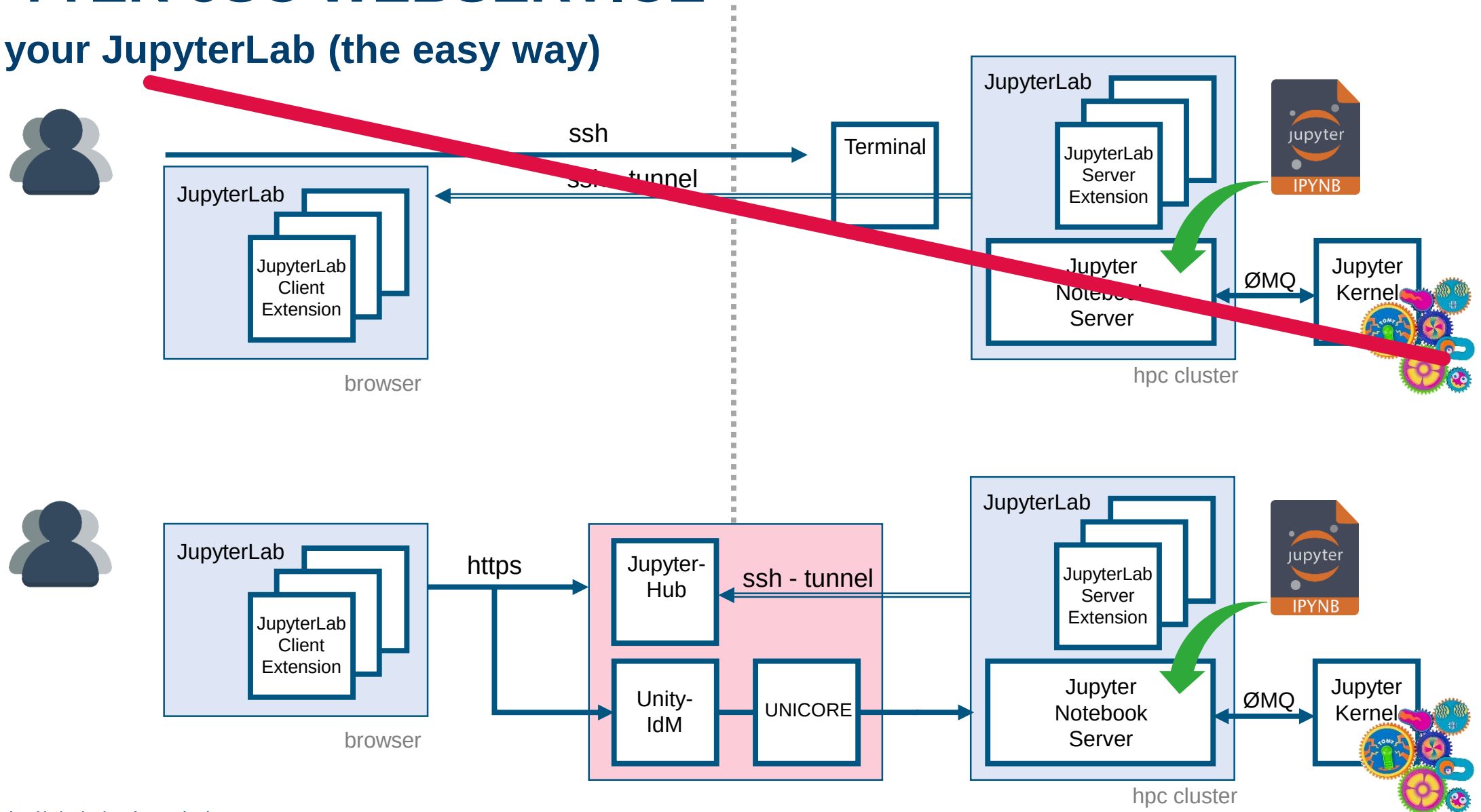


JupyterCollection is a meta-module, which loads the modules:

- JupyterKernel-Bash/0.7.1-2019a.2.4
- JupyterKernel-Cling/0.6-2019a.2.4
- JupyterKernel-JavaScript/5.2.0-2019a.2.4
- JupyterKernel-Julia/1.3.1-2019a.2.4
- JupyterKernel-Octave/5.1.0-2019a.2.4
- JupyterKernel-PyParaView/5.8.0-2019a.2.4
- JupyterKernel-PyQuantum/1.0-2019a.2.4
- JupyterKernel-R/3.5.3-2019a.2.4
- JupyterKernel-Ruby/2.6.3-2019a.2.4
- Jupyter/2019a.2.4-Python-3.6.8

JUPYTER-JSC WEBSERVICE

Start your JupyterLab (the easy way)



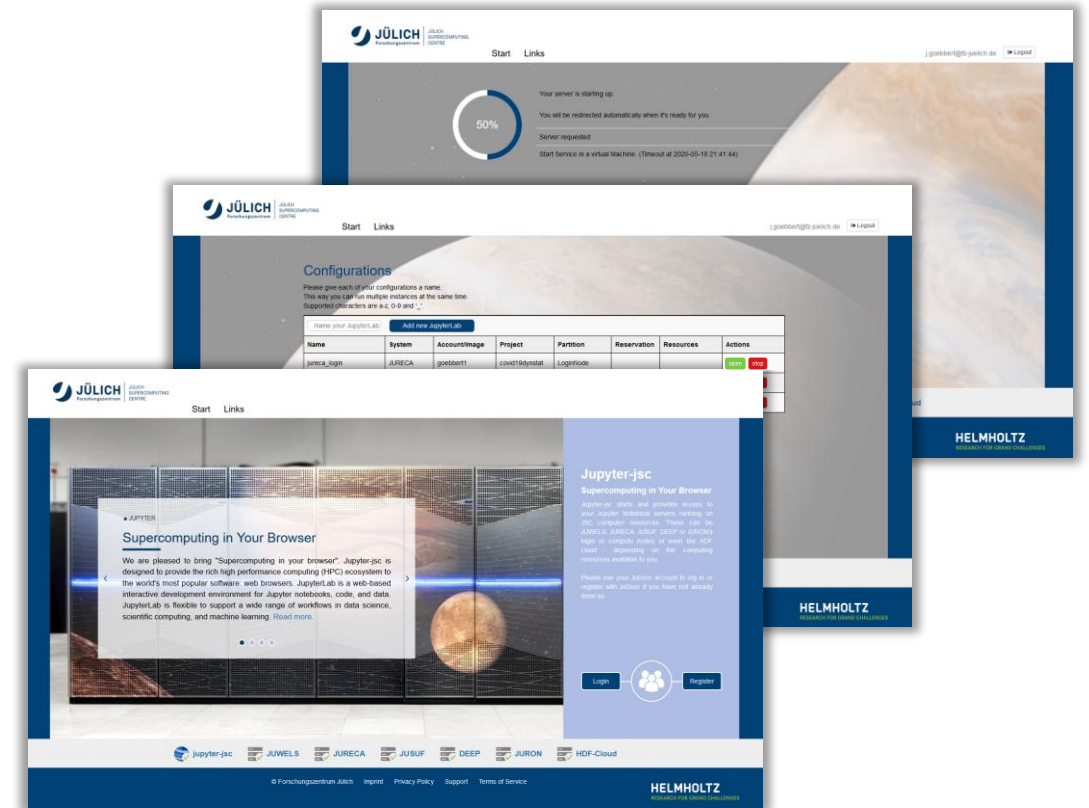
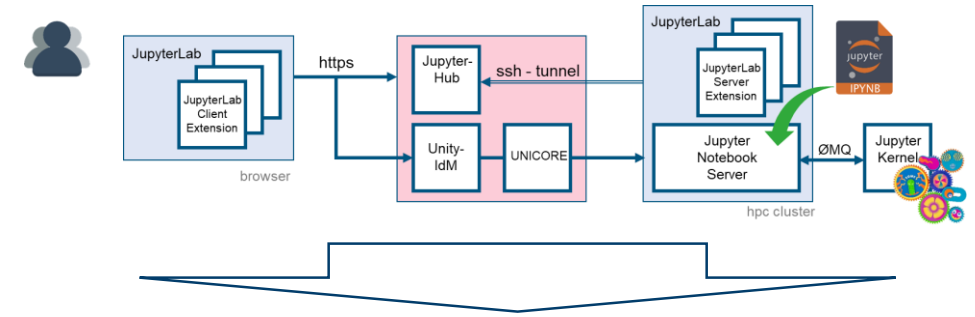
JUPYTER-JSC WEBSERVICE

Start your JupyterLab (the easy way)

JupyterHub

is used to make Jupyter available to a group of HPC users.

- Creates/manages JupyterLabs for single users.
- Connects JupyterLabs to users via a configurable HTTP proxy.
- Supports custom spawners
 - UNICORE at JSC
- Supports custom authenticators
 - Unity-IdM at JSC



JUPYTER-JSC WEBSERVICE

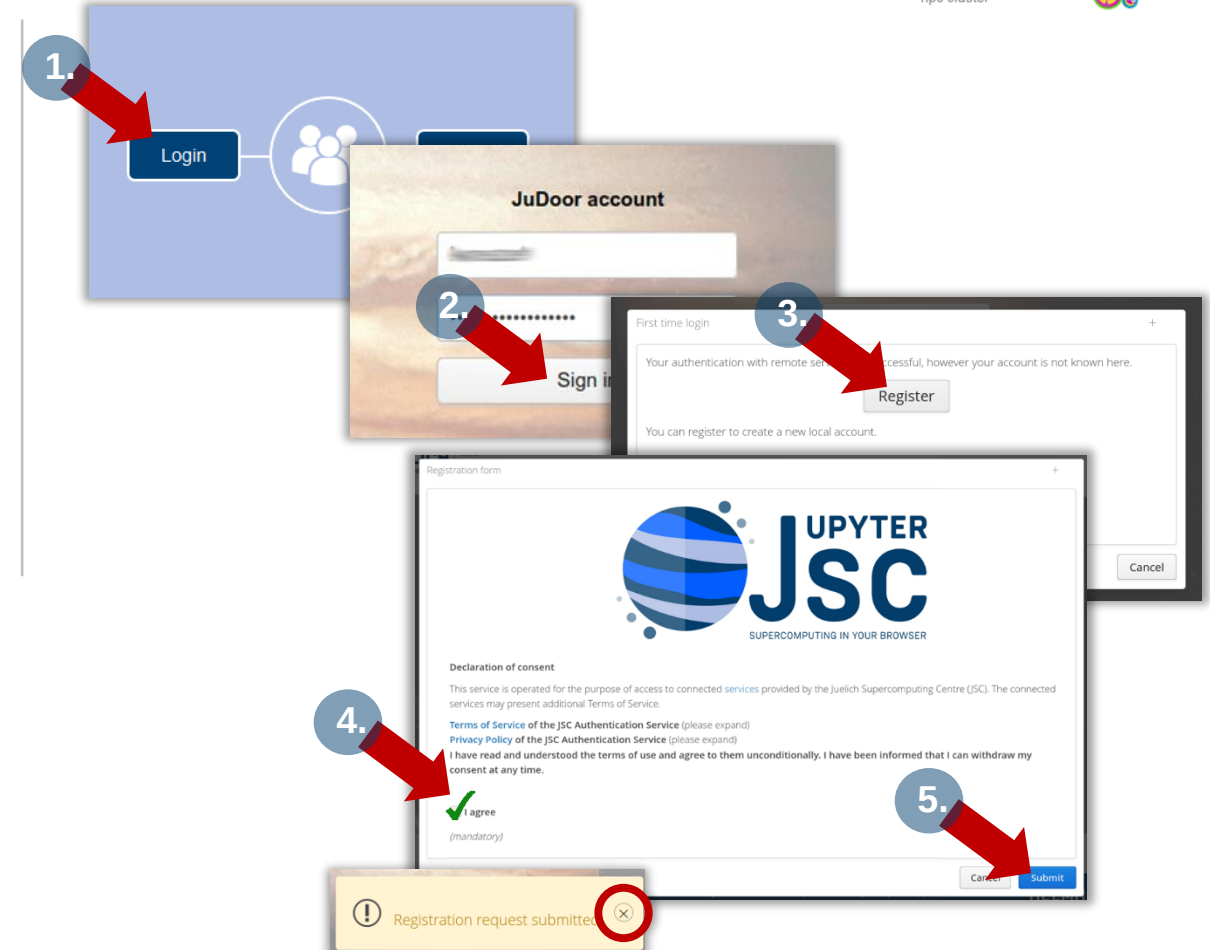
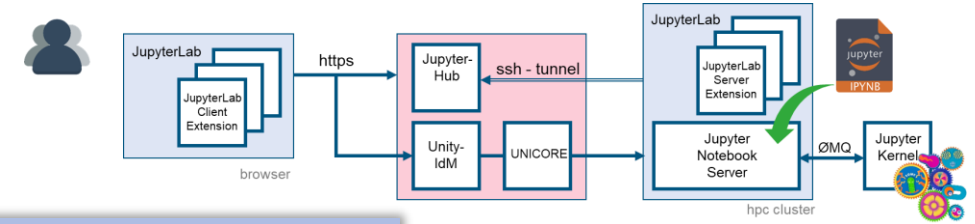
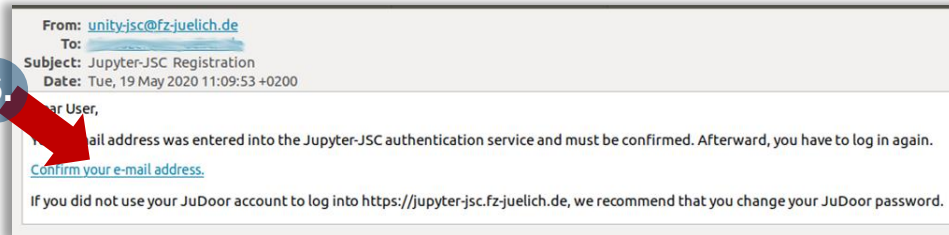
First time login

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

Jupyter-JSC first time login

- Requirements:
 - Registered at judoor.fz-juelich.de
 - (with “Connected Services” = jupyter-jsc)

1. Login at 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

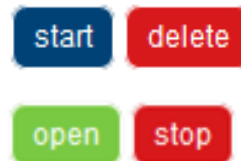
A. Jupyter-JSC – Add new JupyterLab

- Name your new JupyterLab configuration
 - Unique Jupyter workspace in `~/jupyter`
- => the **JupyterLab Options** page will open

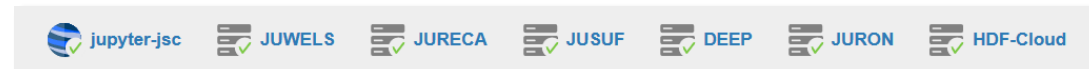
B. Jupyter-JSC – Actions

If a configuration has been added

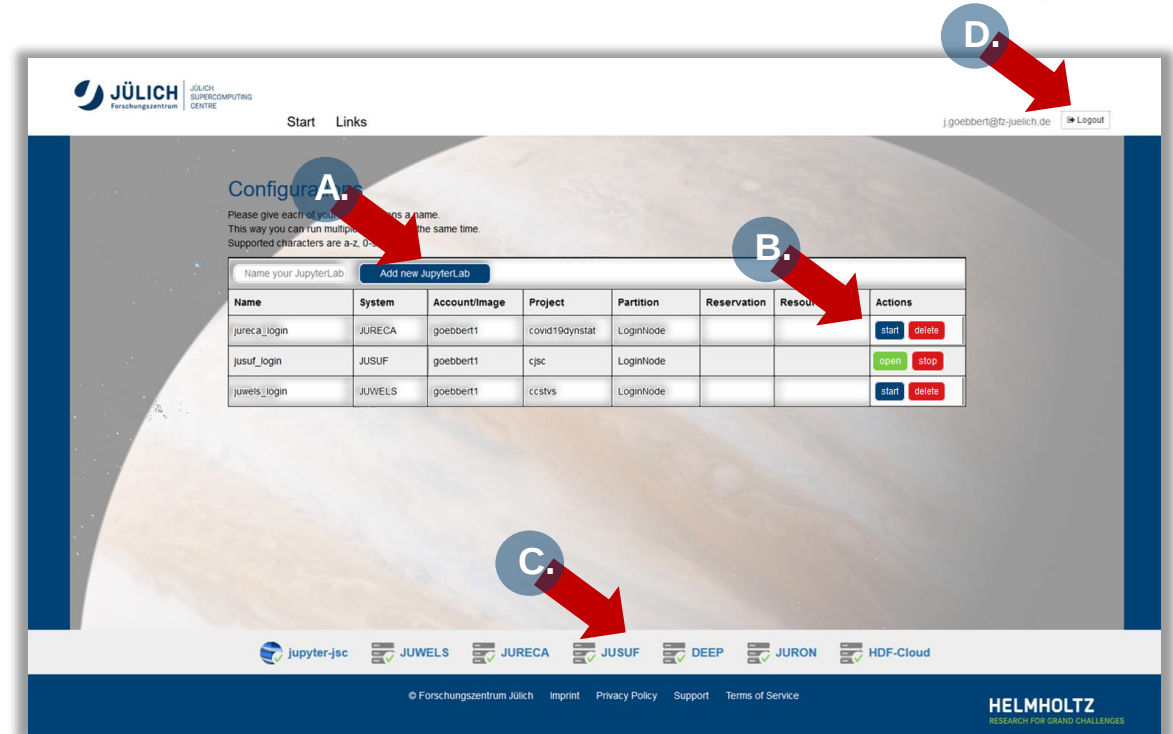
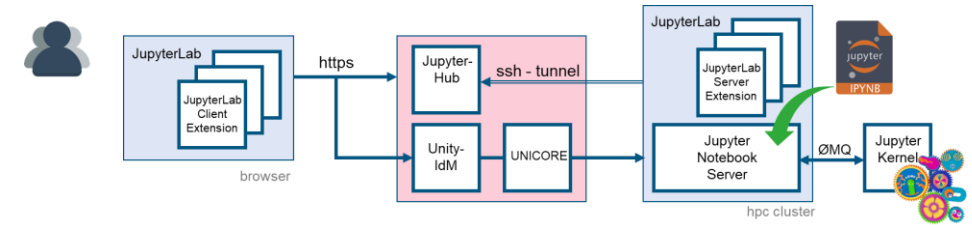
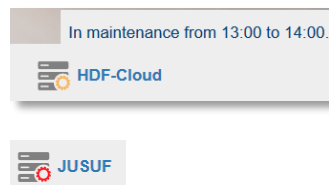
- Start/delete the named configuration (workspace will not be deleted)
- Open/stop a **running** JupyterLab



C. Jupyter-JSC -- Statusbar

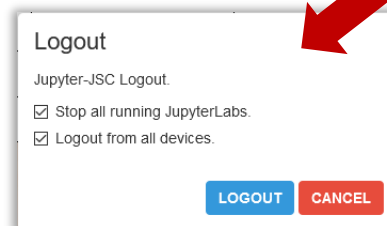


- Upcoming maintenance (mouse hover for details)
- System offline



B. Jupyter-JSC – Logout

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



JUPYTER-JSC WEBSERVICE

JupyterLab Options

Jupyter-JSC – Options

Available options **depend on**

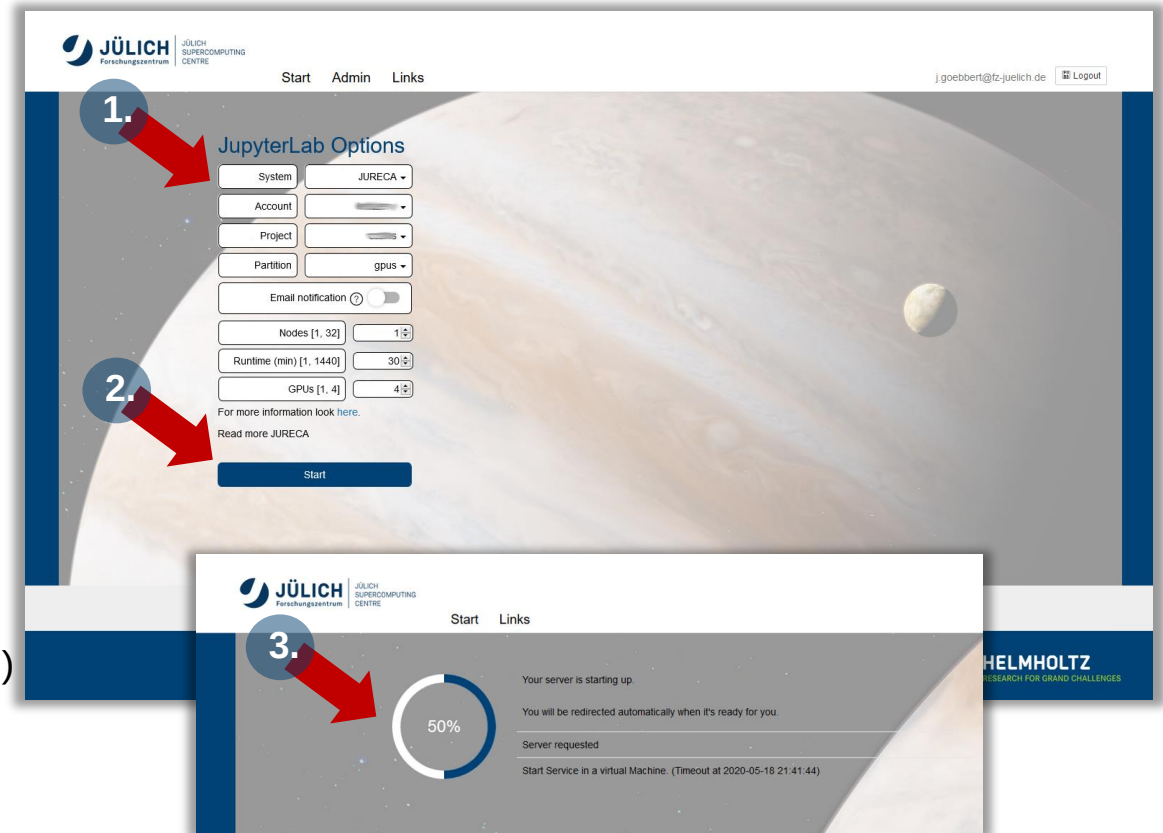
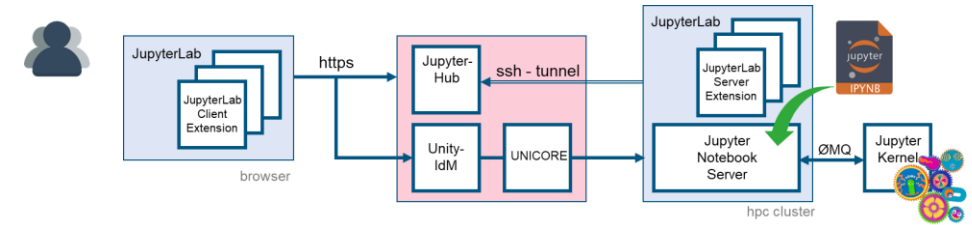
- user account settings visible in judoor.fz-juelich.de
- currently available systems in all of your projects
 - system specific usage agreement on JuDoor is signed

Basic options

- 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)
- Email notification:
Send an email when the JupyterLab has started
(useful if the JupyterLab starts on a compute node)

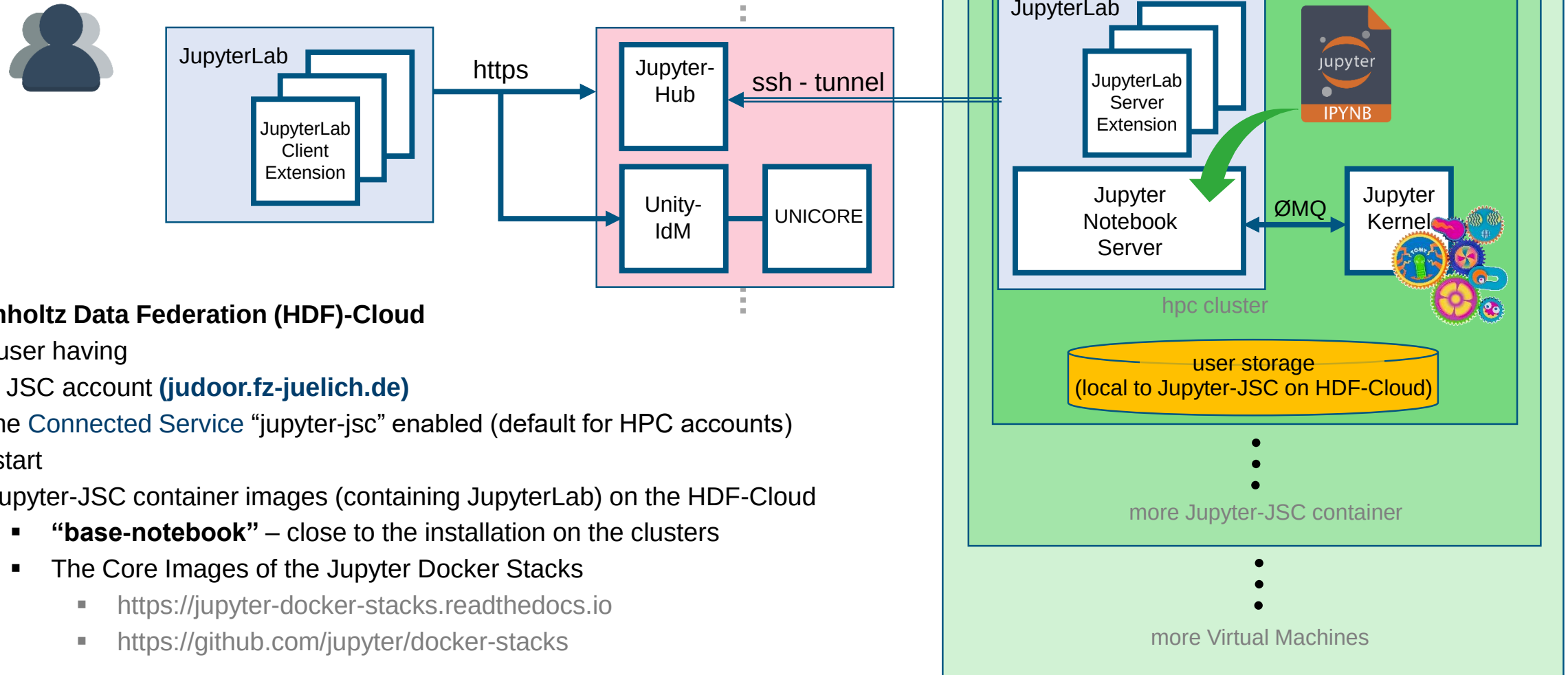
Extra options

- Partition == compute Nodes, Runtime, GPUs, ...
- System == HDF-Cloud Image



JUPYTER-JSC WEBSERVICE

System: HDF-Cloud



Helmholtz Data Federation (HDF)-Cloud

Any user having

- a JSC account (judoor.fz-juelich.de)
 - the [Connected Service](#) "jupyter-jsc" enabled (default for HPC accounts)
- can start
- Jupyter-JSC container images (containing JupyterLab) on the HDF-Cloud
 - **"base-notebook"** – close to the installation on the clusters
 - The Core Images of the Jupyter Docker Stacks
 - <https://jupyter-docker-stacks.readthedocs.io>
 - <https://github.com/jupyter/docker-stacks>

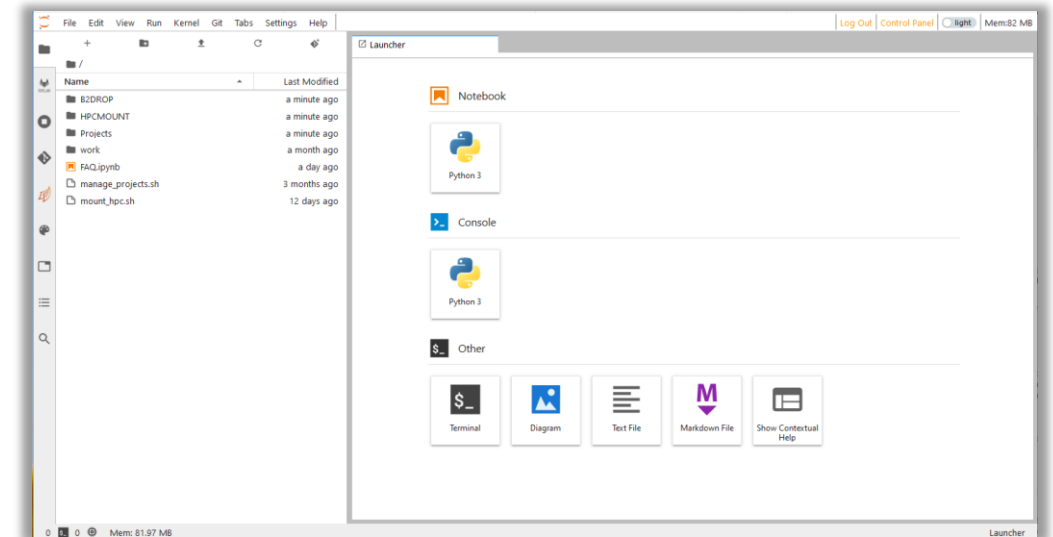
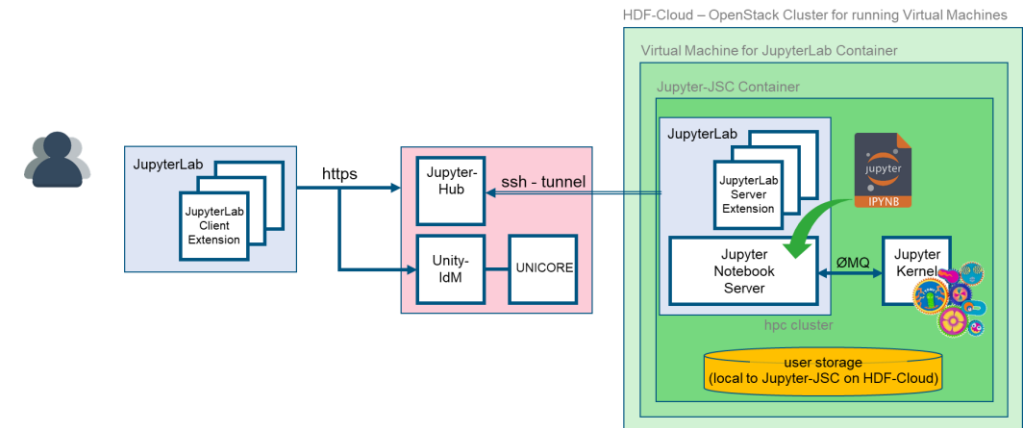


JUPYTER-JSC WEBSERVICE

System: HDF-Cloud

Limitations on JupyterLab on HDF-Cloud

- max. **2 GB** memory
- Installed Jupyter Kernel limited
- Storage in Jupyter-JSC container
 - **is local** to the HDF-Cloud
 - only accessible from a Jupyter-JSC container
 - stored persistently in a personal data container if in
 - `~/work` (max. 10 GB)
 - `~/Projects` (max. 10 GB)
 - backup of `~/work` and `~/Projects` every day to tape
- Depending on the load of the OpenStack you might be limited in the **number of** simultaneous running JupyterLab containers
- HDF-Cloud has at the moment **no GPUs**

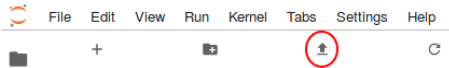


JUPYTER-JSC WEBSERVICE

System: HDF-Cloud

How can I share/backup my work from JupyterLab?

1. Download the file



2. `~/Projects`
for sharing data between JupyterLab-users on HDF-Cloud

3. Mount your HPC cluster directory with sshfs

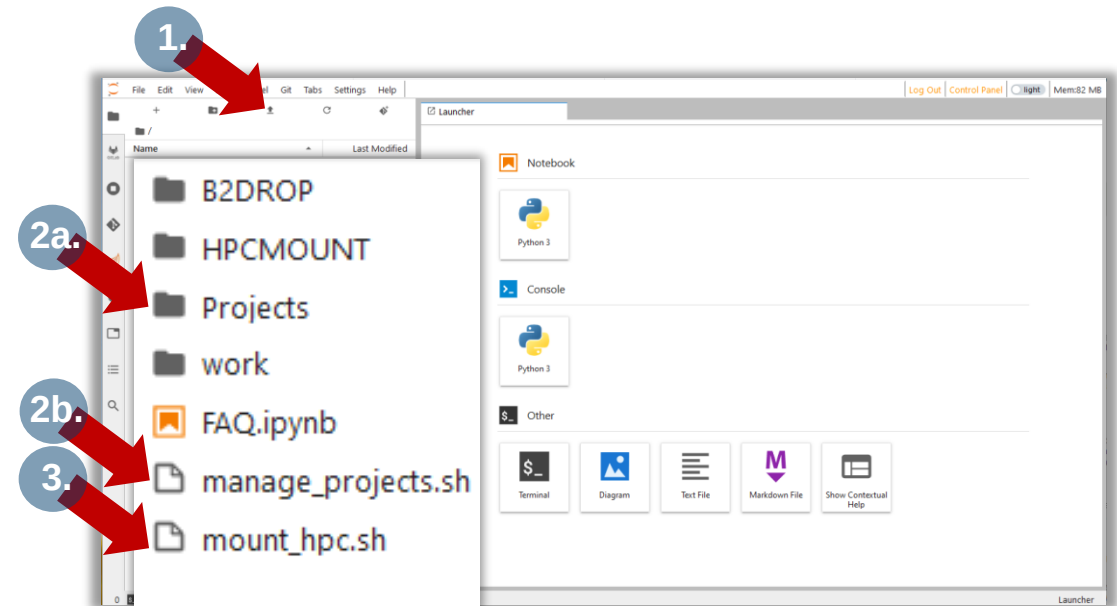
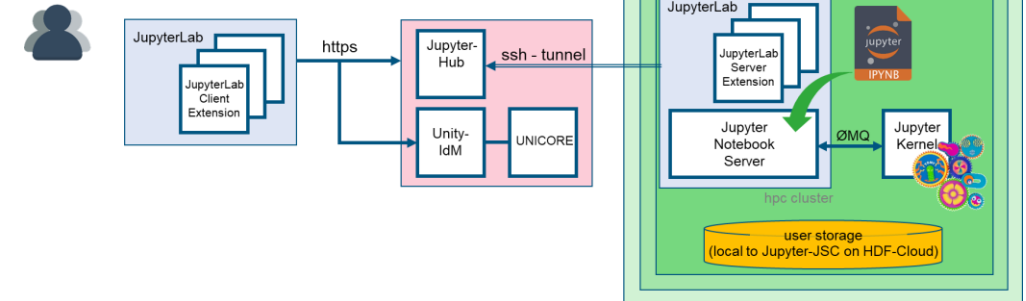
- B2Drop – <https://b2drop.eudat.eu>
- Git / GitHub / GitLab

NEVER forget:

- Data is **ONLY** persistent in `~/Projects` and `~/work`

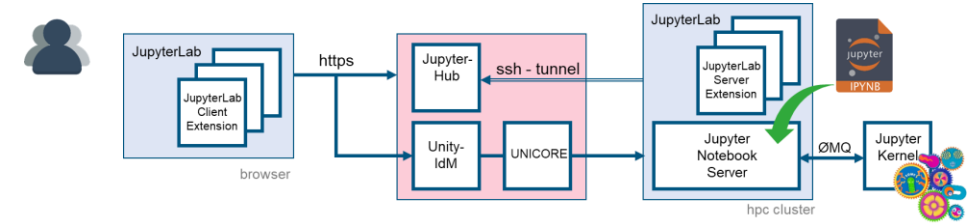
For more details please visit:

https://gitlab.version.fz-juelich.de/jupyter4jsc/j4j_notebooks/-/blob/master/FAQ_HDFCloud.ipynb



JUPYTER-JSC WEBSERVICE

Some comments about the UI



Annotations on the JupyterLab interface:

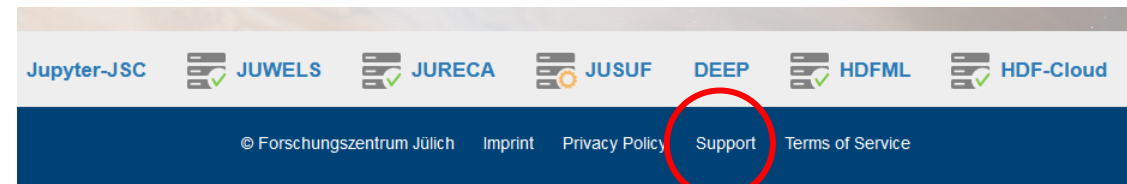
- open filebrowser**: Points to the file browser icon in the sidebar.
- tutorials & examples**: Points to the 'tutorials' and 'examples' links in the sidebar.
- sidebar with core and extensions features**: Points to the sidebar area.
- open launcher**: Points to the 'launcher' icon in the sidebar.
- indicates active notebook cell**: Points to the 'Untitled.ipynb' file in the sidebar.
- type of active notebook cell**: Points to the 'Code' button in the top toolbar.
- [*] indicates that cell was send to Jupyter kernel for execution**: Points to the asterisk icon in the top toolbar.
- [] indicates that cell has never been executed by the connected Jupyter kernel**: Points to the square icon in the top toolbar.
- no close, but go back to Jupyter-JSC's control panel**: Points to the 'Control Panel' button in the top right.
- memory consumption (keep an eye on that!)**: Points to the 'Mem:211 MB' indicator in the top right.
- Type of Jupyter kernel this notebook is connected to (click to change)**: Points to the 'Python 3' dropdown menu in the top right.
- notebook cell**: Points to a code cell in the main editor.

JUPYTER-JSC SECRETS

Very important to know

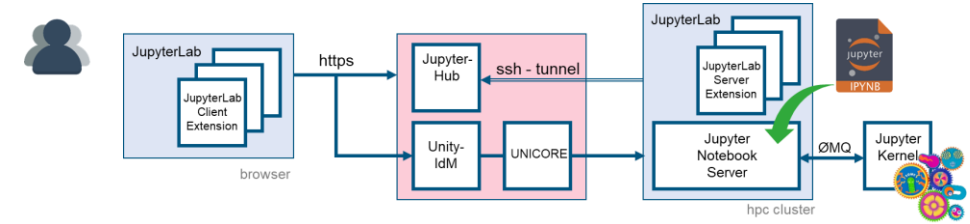
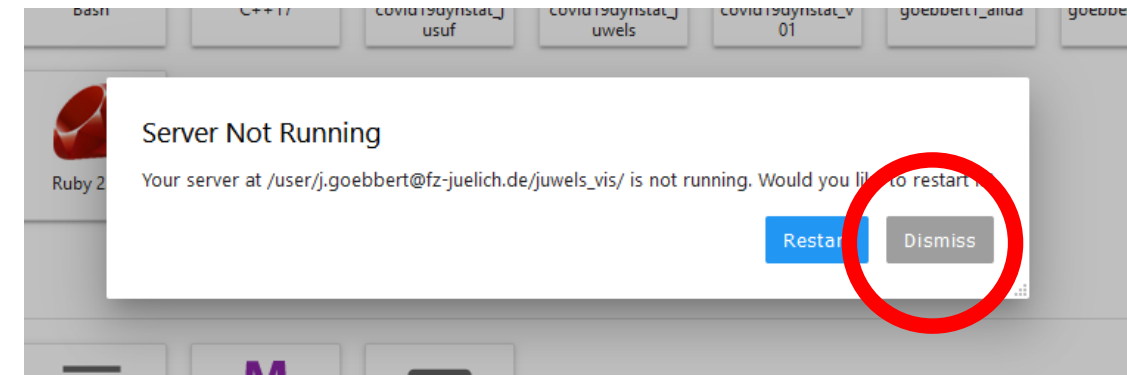
Secret 1: Support button

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



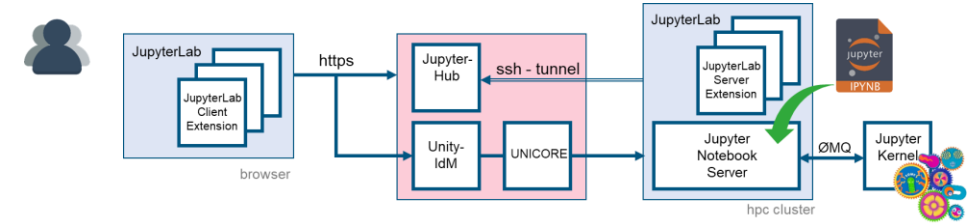
Secret 2: Reload on connection loss

- “Server Not Running” means, that your browser just lost connection => **Just hit “Dismiss” !!!**
- You can safely hit the “Reload” button of your browser, if the connection to JupyterLab ever gets lost.
(but that takes more time)



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 use $OTHERSTAGES  
module load Stages/Devel-2019a  
module load GCC/8.3.0  
module load JupyterCollection/2019a.2.3
```

Switch to a different 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.

JUPYTERLAB EXTENSIONS

JUPYTER EXTENSIONS

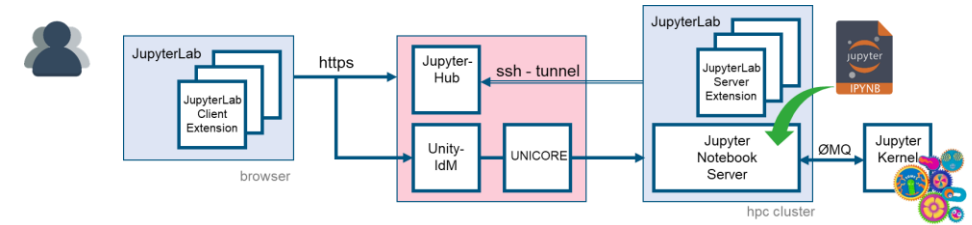
Some general information

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`
- is immutable
 - **any change requires a rebuild** of the whole JupyterLab to take effect!
 - contains the JupyterLab static assets
 - (e.g. `static/index.html`)



```
jovyan@dad3db89c836:~$ jupyter labextension list
JupyterLab v1.2.1
Known labextensions:
  app dir: /opt/conda/share/jupyter/lab
    @bokeh/jupyter_bokeh v1.1.1 enabled OK
    @jupyter-voila/jupyterlab-preview v0.1.3 enabled OK
    @jupyter-widgets/jupyterlab-manager v1.0.3 enabled OK
    @jupyter-widgets/jupyterlab-sidecar v0.4.0 enabled OK
    @jupyterlab/celltags v0.2.0 enabled OK
    @jupyterlab/git v0.8.2 enabled OK
    itkwidgets v0.22.0 enabled OK
    jefileupload v0.1.0 enabled OK
    jupyter-leaflet v0.11.4 enabled OK
    jupyter-matplotlib v0.4.2 enabled OK
    jupyter-threejs v2.1.1 enabled OK
    jupyter-vue v1.0.0 enabled OK
    jupyter-verify v1.1.1 enabled OK
    jupyter-webrtc v0.5.0 enabled OK
    jupyterlab-control v1.0.1 enabled OK
    jupyterlab-datawidgets v6.2.0 enabled OK
    jupyterlab-drawio v0.6.0 enabled OK
    jupyterlab-gitlab v0.3.0 enabled OK
    jupyterlab-logout v0.4.0 enabled OK
    jupyterlab-plotly v1.2.0 enabled OK
    jupyterlab-system-monitor v0.4.1 enabled OK
    jupyterlab-theme-toggle v0.4.2 enabled OK
    jupyterlab-topbar-extension v0.4.0 enabled OK
    jupyterlab-iframe v0.2.1 enabled OK
    nbdtm-jupyterlab v1.0.0 enabled OK
    plotlywidget v1.2.0 enabled OK
    prlink v0.1.2 enabled OK
jovyan@dad3db89c836:~$
```

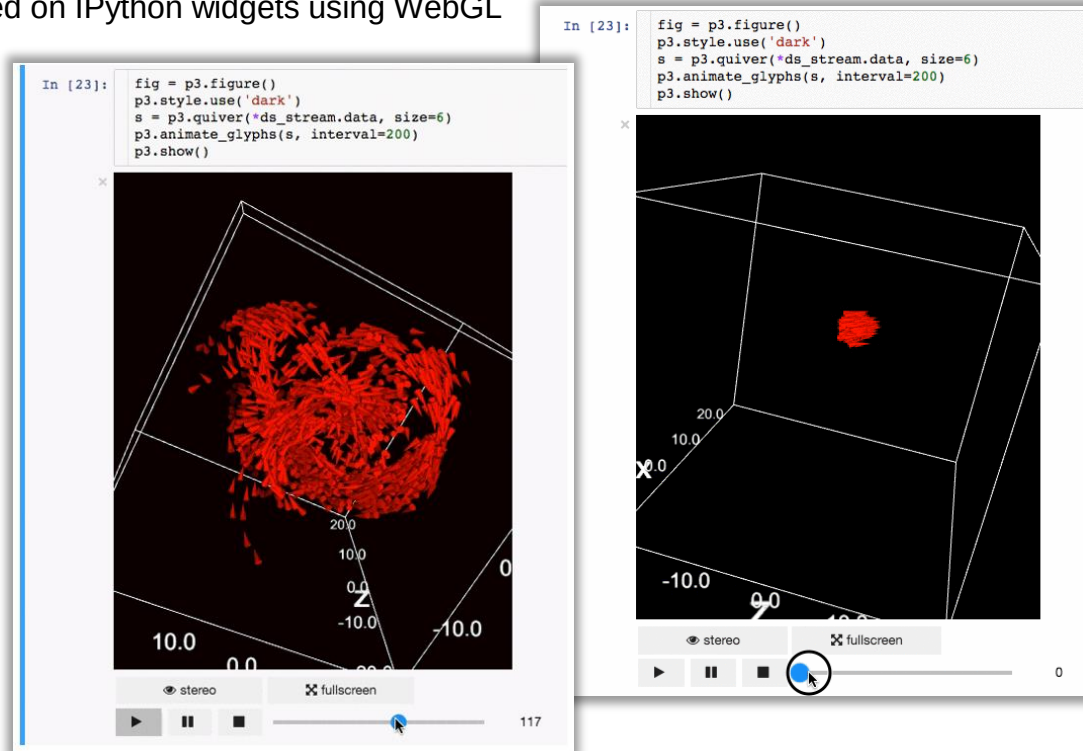
<https://jupyterlab.readthedocs.io/en/stable/user/extensions.html>

JUPYTER-JSC EXTENSIONS

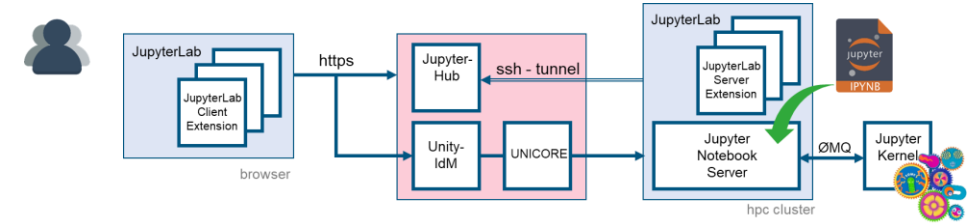
Installed by default

IPyVolume

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

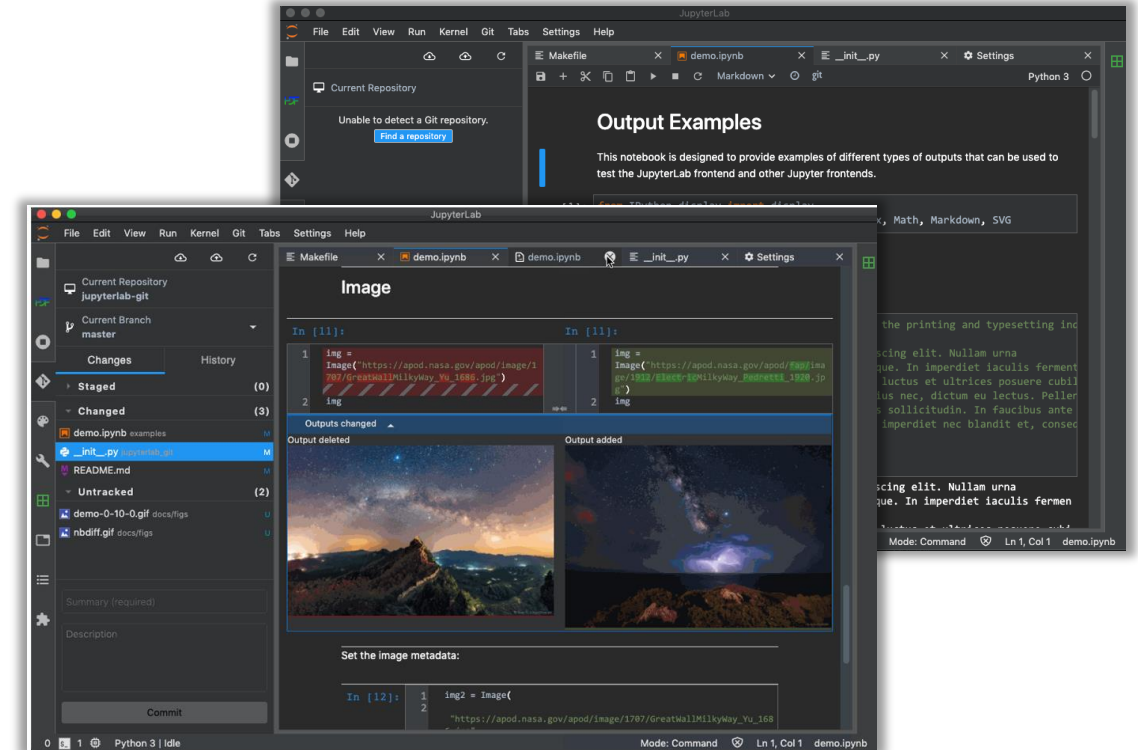


<https://github.com/maartenbreddels/ipyvolume>



JupyterLab-Git

JupyterLab extension for version control using Git



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

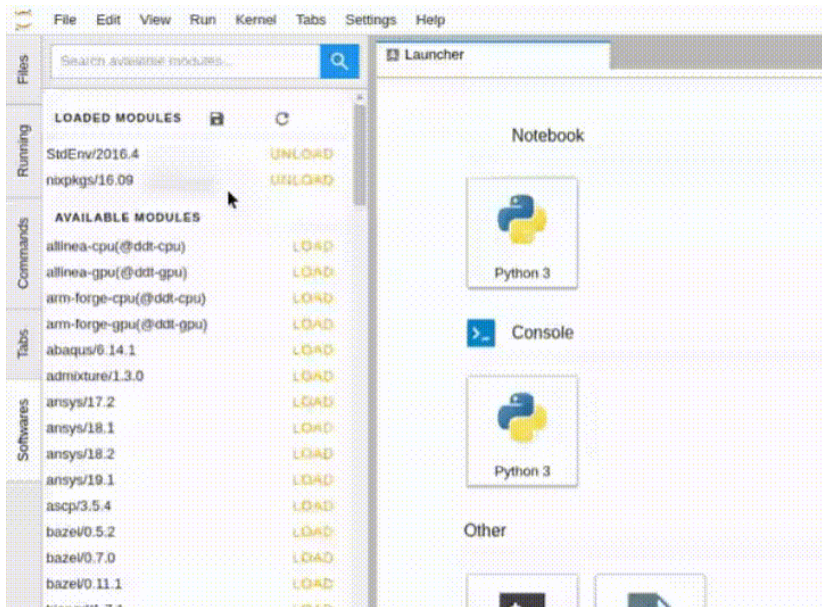
JUPYTER-JSC EXTENSIONS

Installed by default

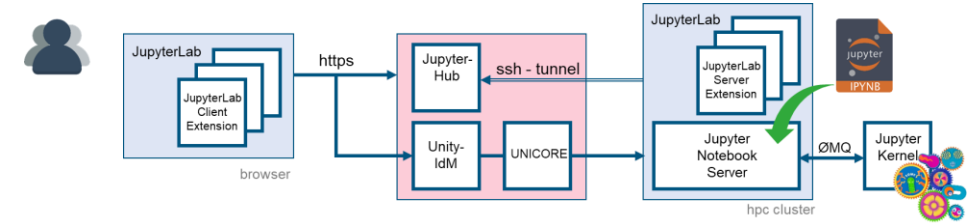
JupyterLab-Lmod

JupyterLab extension that allows user to interact with environment modules before launching kernels.

- **Remember** to restart the kernel after loading other modules.



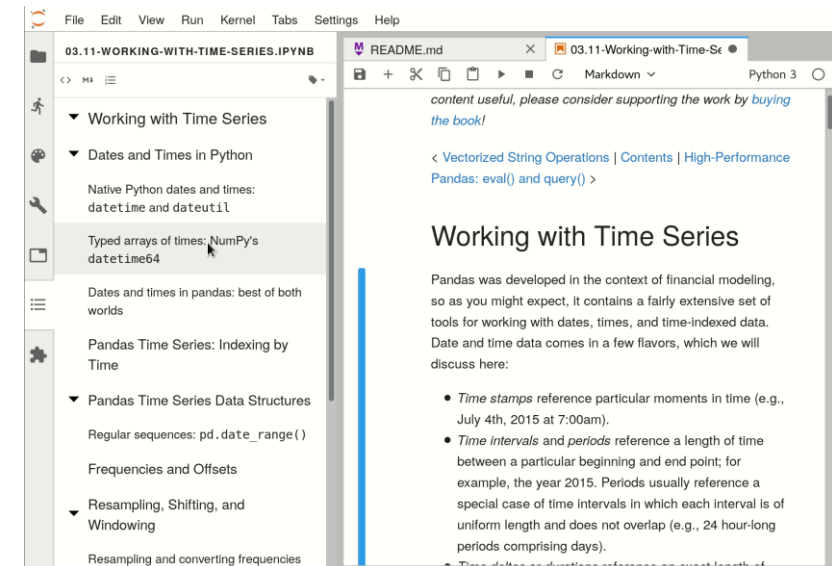
<https://github.com/cmd-ntrf/jupyter-lmod>



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;

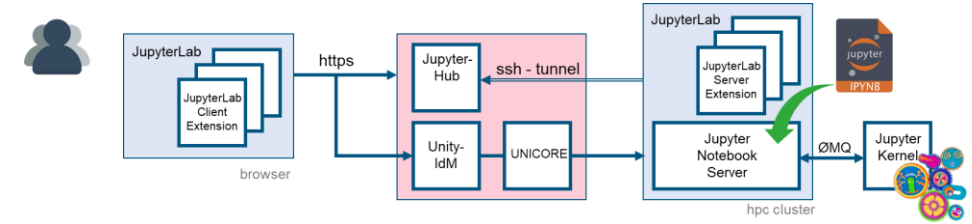
  var x = Math.sin(u);
  var y = Math.cos(v);
  var z = Math.cos(u+v);

  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)
```



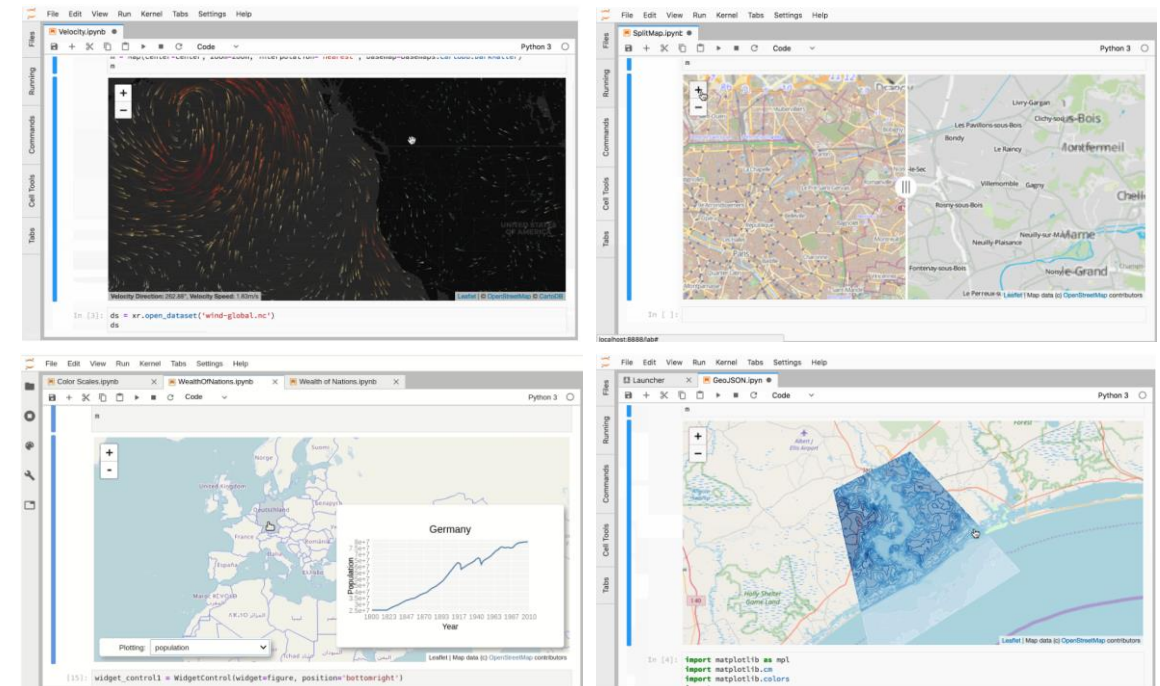
<https://github.com/jupyter-widgets/pythreejs>

Member of the Helmholtz Association



IPyleaflet

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



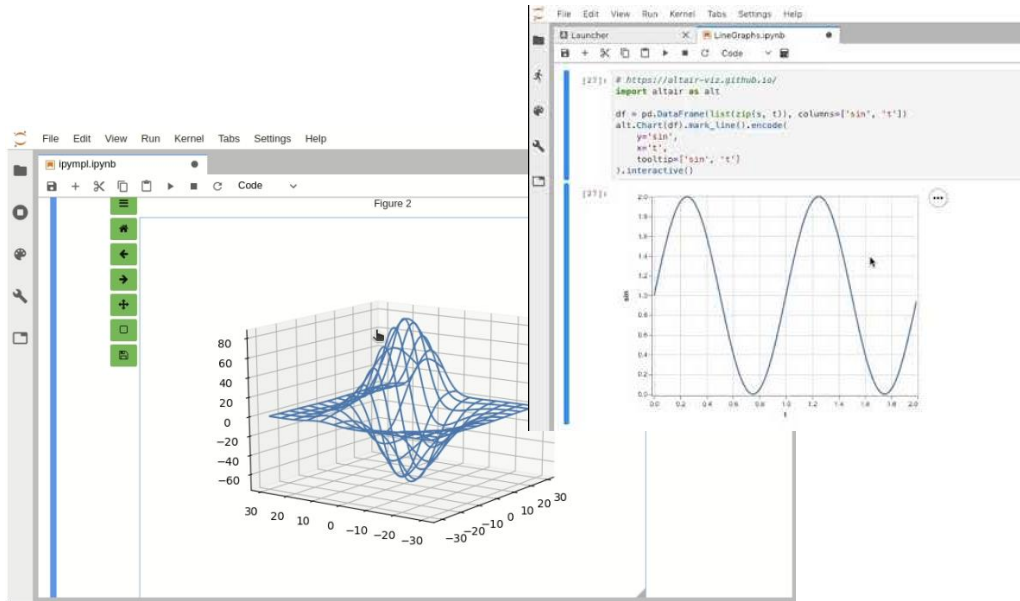
<https://github.com/jupyter-widgets/ipyleaflet>

JUPYTER-JSC EXTENSIONS

Installed by default

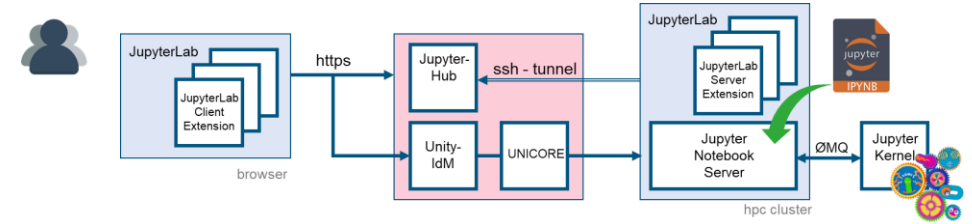
IPyMPL - matplotlib

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



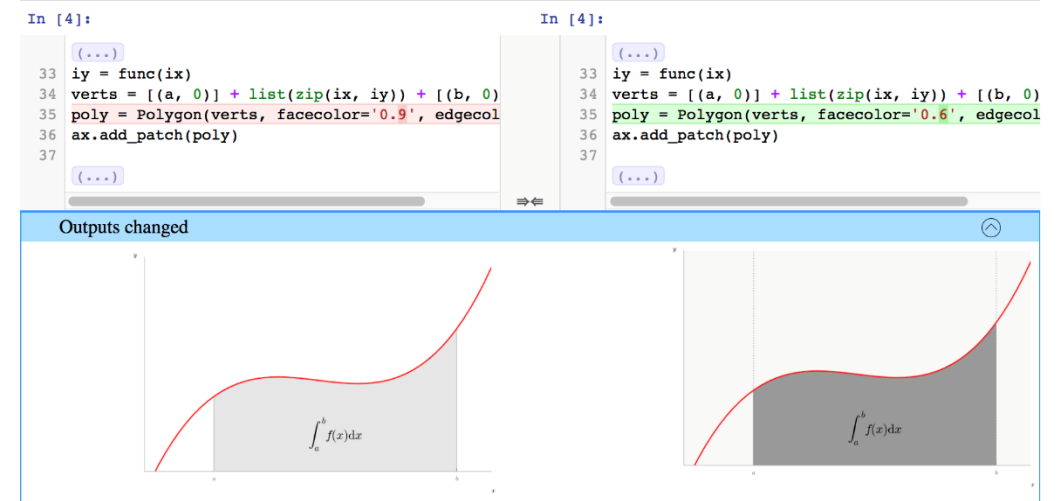
<https://github.com/matplotlib/ipympl>

Member of the Helmholtz Association



NBDime

Tools for diffing and merging of Jupyter notebooks.



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

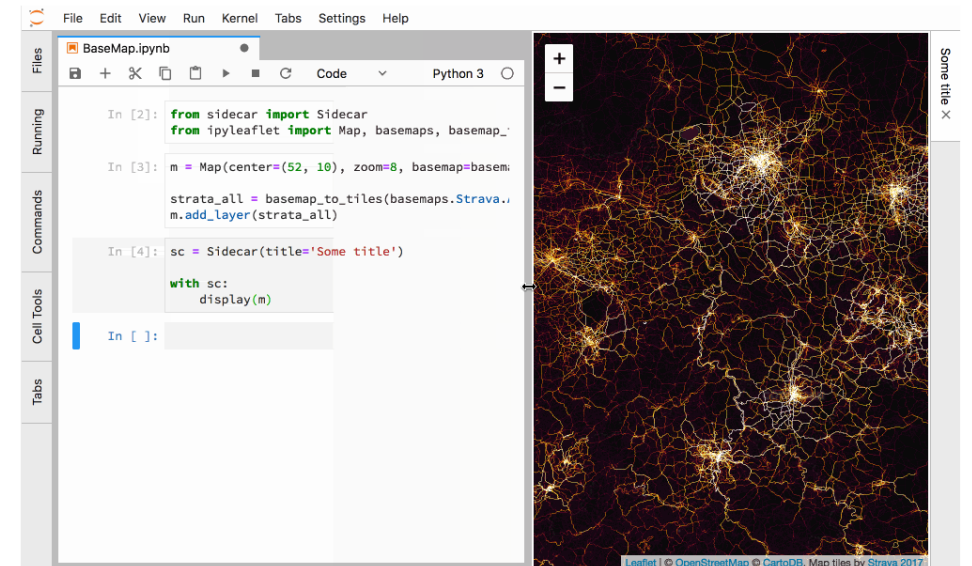
Installed by default

JupyterLab extension for the interactive and browser-based graphing library Plotly.
<https://plotly.com/python/>



Member of the Helmholtz Association

A sidecar output widget for JupyterLab.

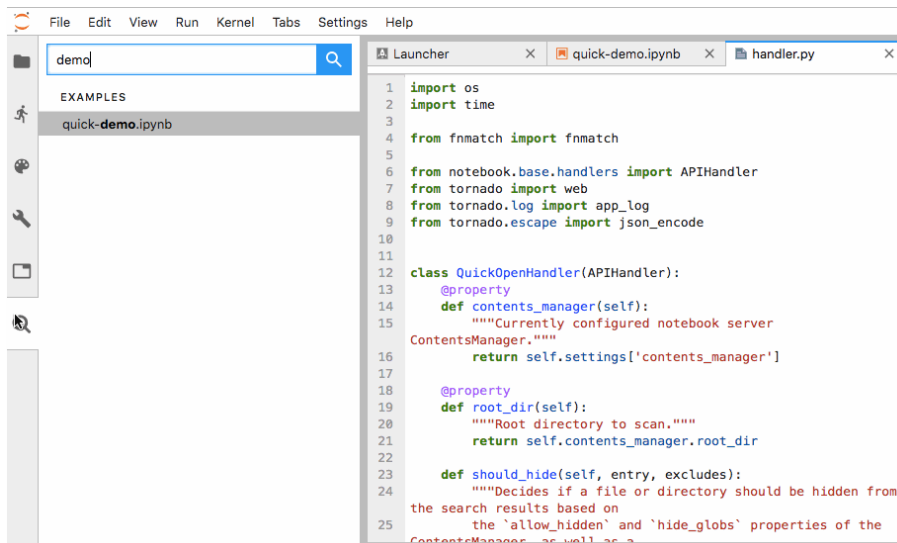


JUPYTER-JSC EXTENSIONS

Installed by default

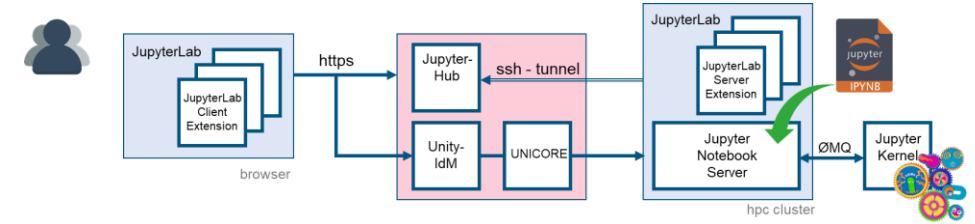
JupyterLab-Quickopen

Quickly open a file in JupyterLab by typing part of its name



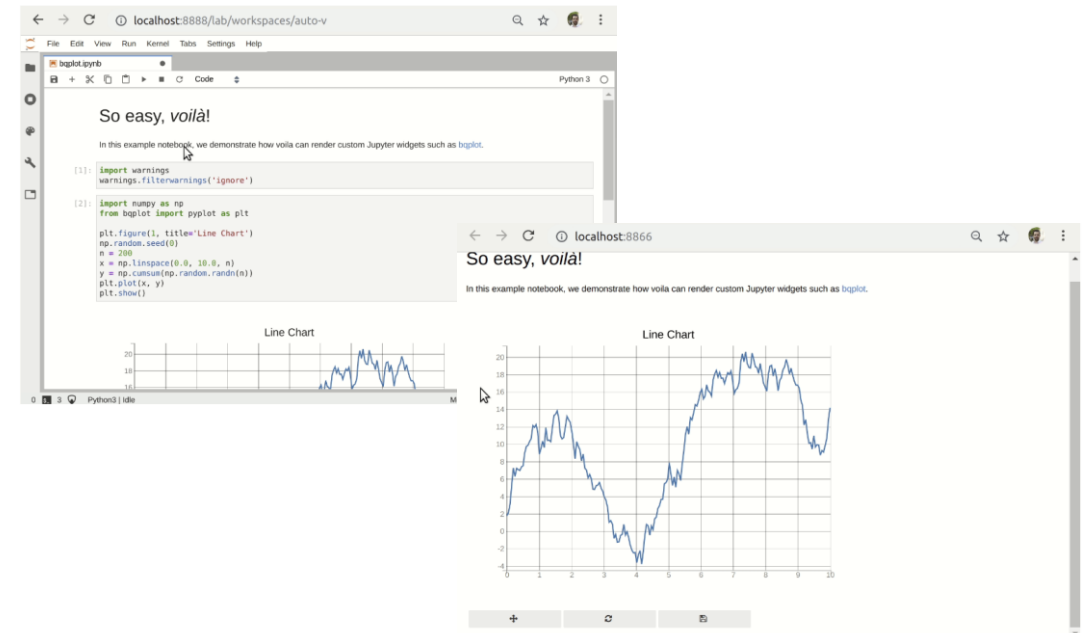
<https://github.com/parente/jupyterlab-quickopen>

Member of the Helmholtz Association



Voilà

Voilà turns Jupyter notebooks into standalone web applications.



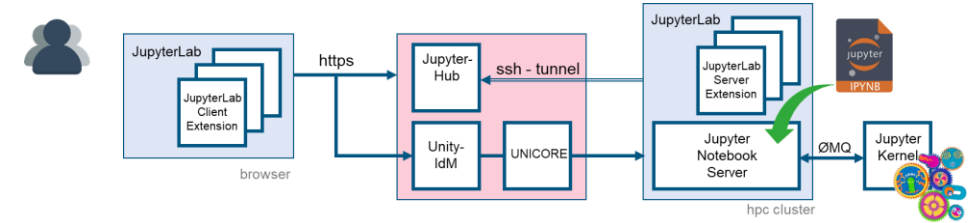
<https://github.com/voila-dashboards/voila>

JUPYTER-JSC EXTENSIONS

Installed by default

Presented JupyterLab extensions

- ipyvolume
- @jupyterlab/git
- **jupyterlab-lmod**
- @jupyterlab/toc
- jupyter-threejs
- jupyter-leaflet
- jupyter-matplotlib
- jupyterlab-plotly
- @jupyter-widgets/jupyterlab-sidecar
- @parente/jupyterlab-quickopen
- @jupyter-voila/jupyterlab-preview



More installed JupyterLab extensions

- @bokeh/jupyter_bokeh
- **dask-labextension**
- jupyterlab-gitlab
- bqplot
- @jupyterlab/latex
- @krassowski/jupyterlab_go_to_definition
- @pyviz/jupyterlab_pyviz
- @ryantam626/jupyterlab_code_formatter
- **@jupyterlab/server-proxy**
- itkwidgets
- jupyter-vue
- @jupyterlab/celltags
- jupyterlab-drawio

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

<https://npmjs.com>

JUPYTER KERNEL

JUPYTER KERNEL

How to create your own Jupyter Kernel

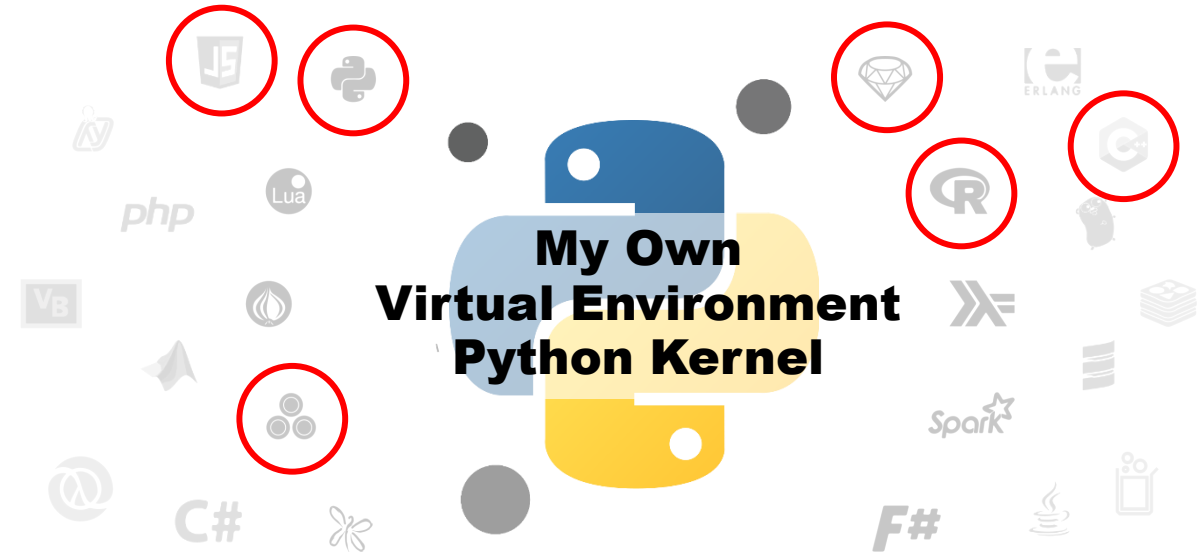
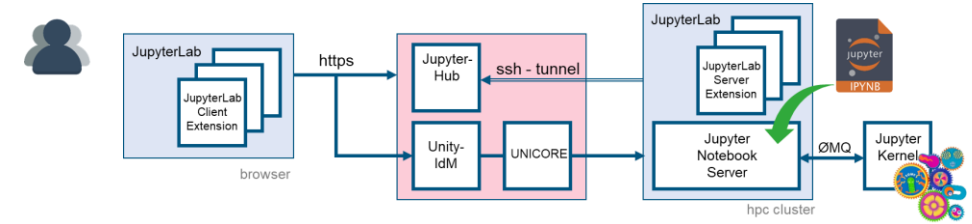
Jupyter Kernel

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

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

1. Login to JupyterLab and open terminal

2. Load required modules

```
lnode:> module purge
lnode:> module use $OTHERSTAGES
lnode:> module load Stages/Devel-2019a
lnode:> module load GCC/8.3.0
lnode:> module load Jupyter
```

3. Load extra modules you need for your kernel

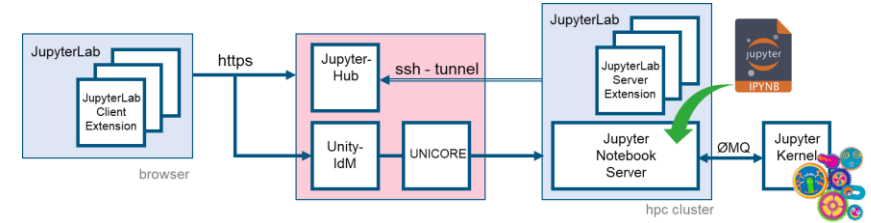
```
lnode:> module load <module you need>
```

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

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

2. Activate your environment

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



Building your own Jupyter kernel is a three step process

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

JUPYTER KERNEL

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

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

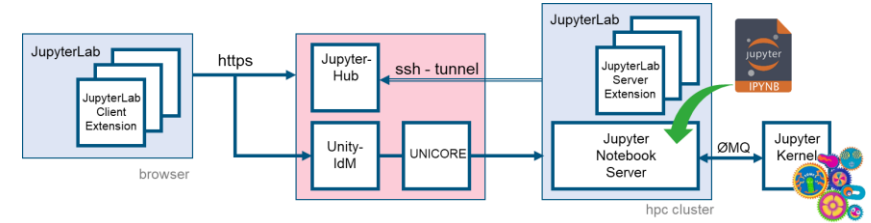
```
(<venv_name>) Lnode:> export PYTHONPATH=\n${VIRTUAL_ENV}/lib/python3.6/site-packages:${PYTHONPATH}
```

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:

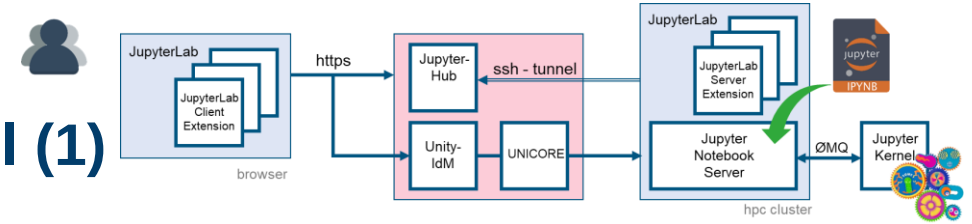
```
(<venv_name>) Lnode:> touch ${VIRTUAL_ENV}/kernel.sh
```

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
```

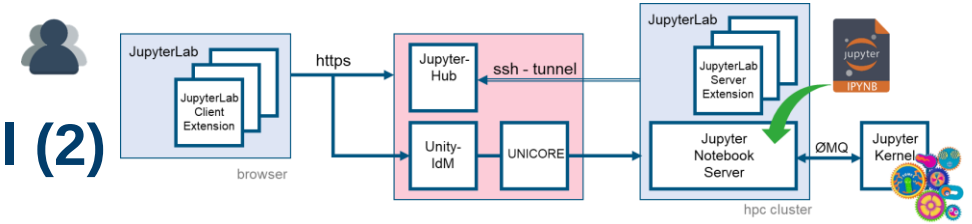


Building your own Jupyter kernel is a three step process

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/Devel-2019a
```

```
module load GCC/8.3.0
```

```
module load Jupyter
```

```
# 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=${VIRTUAL_ENV}/lib/python3.6/site-packages:${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

Member of the Helmholtz Association

JUPYTER KERNEL

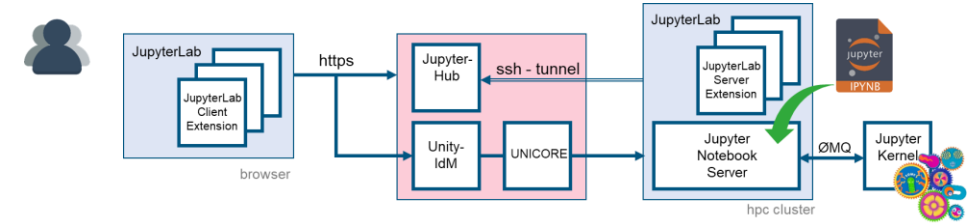
3. Create/Edit Jupyter kernel configuration (1)

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

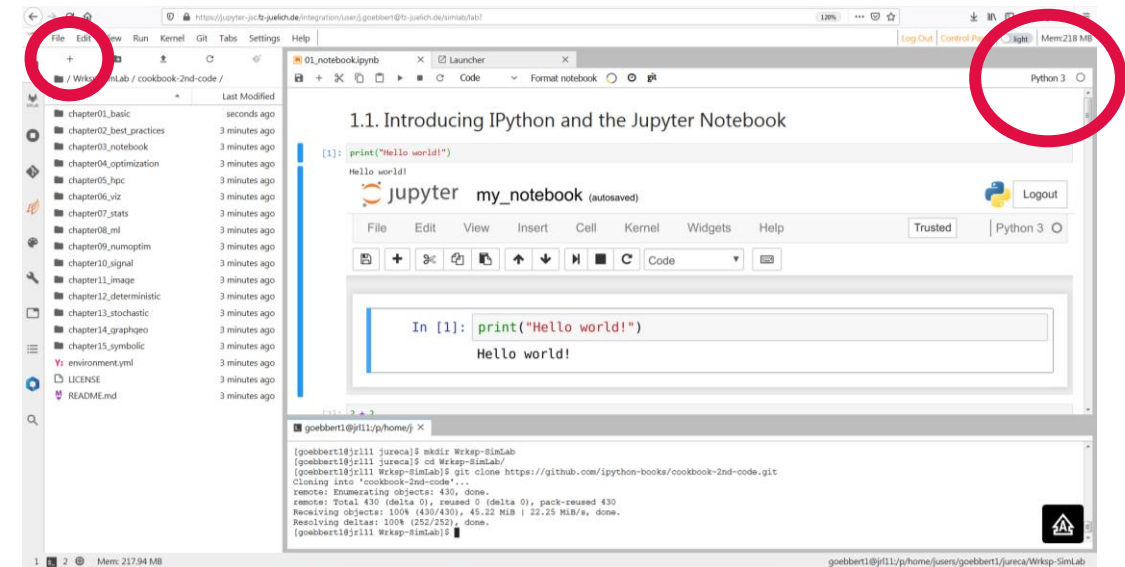
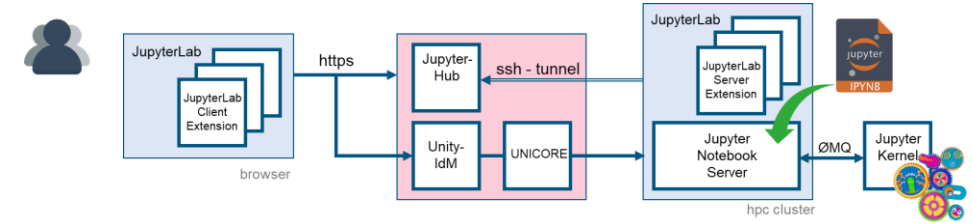
How to base your Jupyter Kernel on a Conda environment:

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

Project kernel

On request Jupyter kernel can be made available to a whole project. They are installed then to

`$PROJECT/.local/share/jupyter/kernels`



JUPYTER KERNEL

Shortcut!

You do NOT want to build your own kernel, every time you QUICKLY need a package or module.

You are lucky – we can show you a workaround / hack(!):

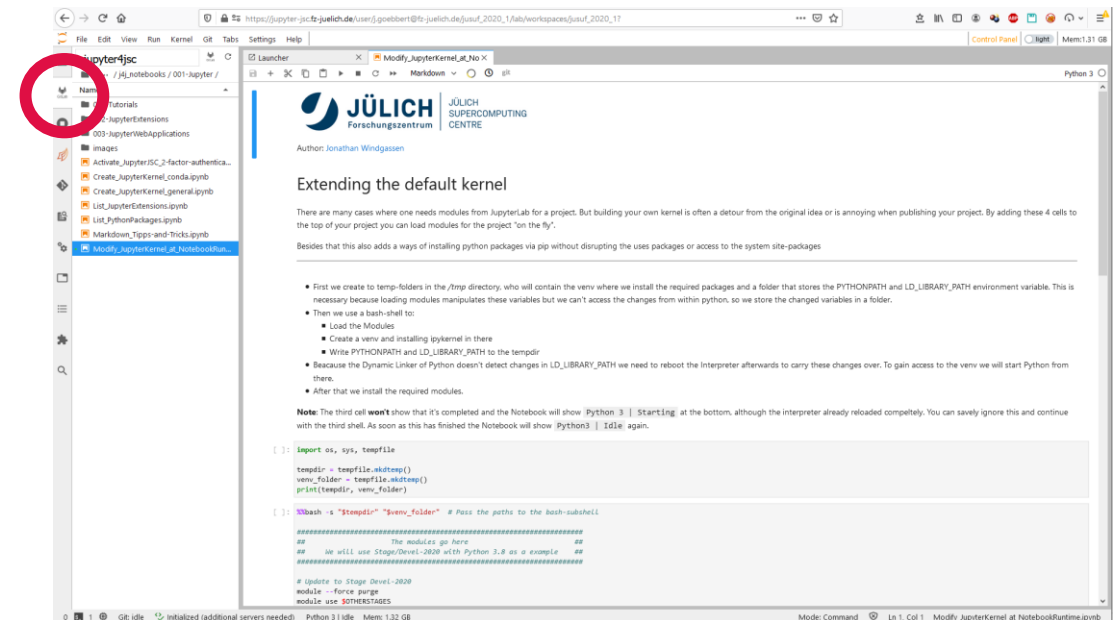
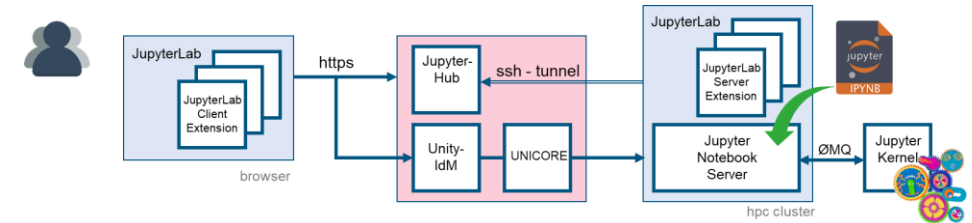
1. <https://jupyter-jsc.fz-juelich.de>
2. Click on the Github-Icon in the sidebar
3. Go to “001-Jupyter”
4. Open `Modify_JupyterKernel_at_NotebookRuntime.ipynb`

What's the trick

```
os.execve(f"{venv_folder}/bin/python", args, env)
```

Workflow

1. Create a Python virtual environment at any location.
2. **WITHIN** the notebook
 - restart the kernel's python interpreter
 - of that Python virtual environment
 - with the correct environment variables set.



https://github.com/FZJ-JSC/jupyter-jsc-notebooks/blob/master/001-Jupyter/Modify_JupyterKernel_at_NotebookRuntime.ipynb

Member of the Helmholtz Association

JUPYTER CAN DO MORE

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” (not available on JURECA, yet)

Xpra - X Persistent Remote Applications

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

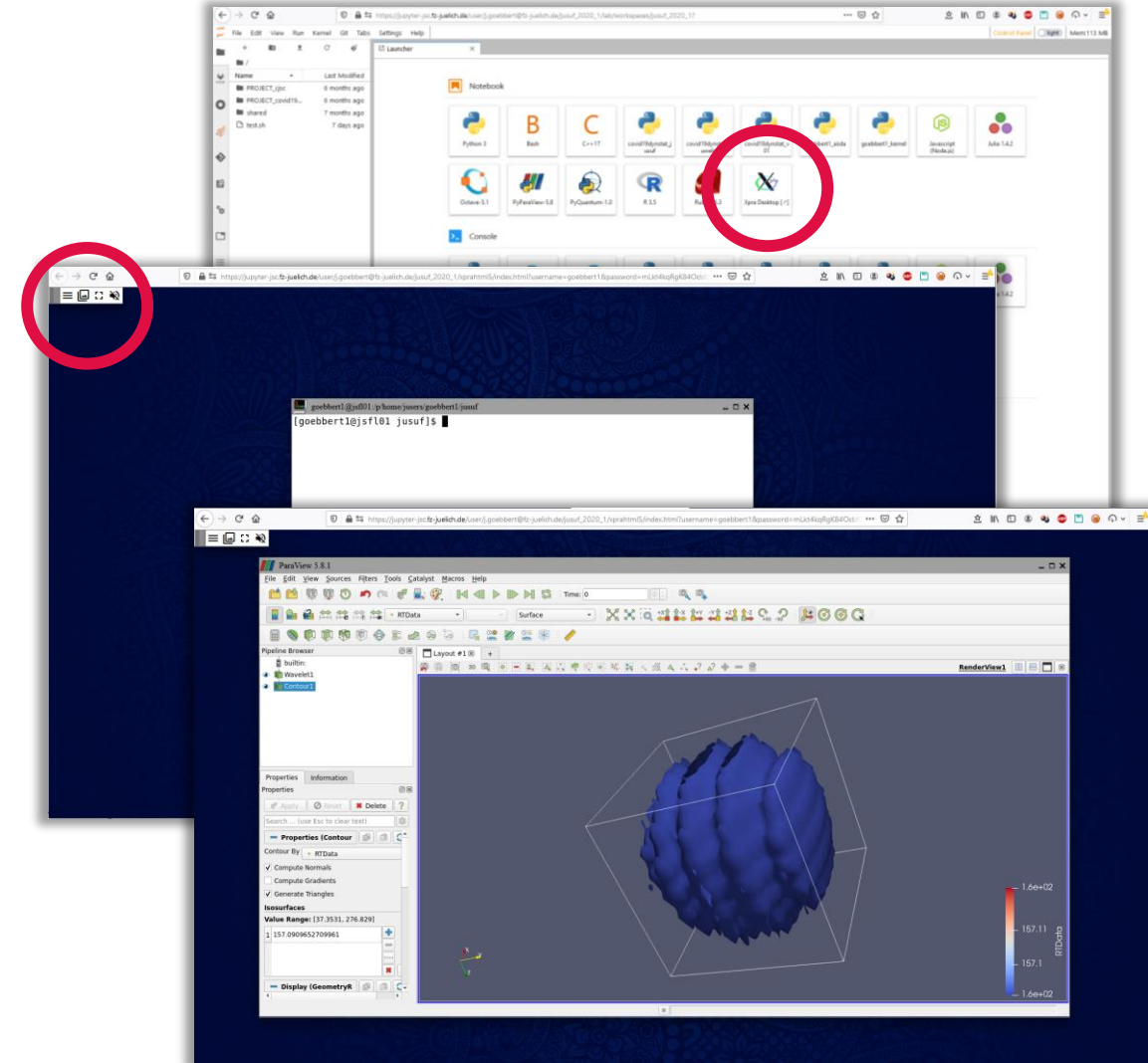
- Runs in a browser
- allows dis-/reconnection without disrupting the forwarded application
- <https://xpra.org>

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

It gets killed, when you stop your JupyterLab session.

Hint:

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



JUPYTERLAB – REMOTE DESKTOP

Run your X11-Applications in the browser

Jupyter-JSC gives you easy access to a remote desktop

1. <https://jupyter-jsc.fz-juelich.de>
2. Click on “Xpra” (not available on JURECA, yet)

Xpra - X Persistent Remote Applications

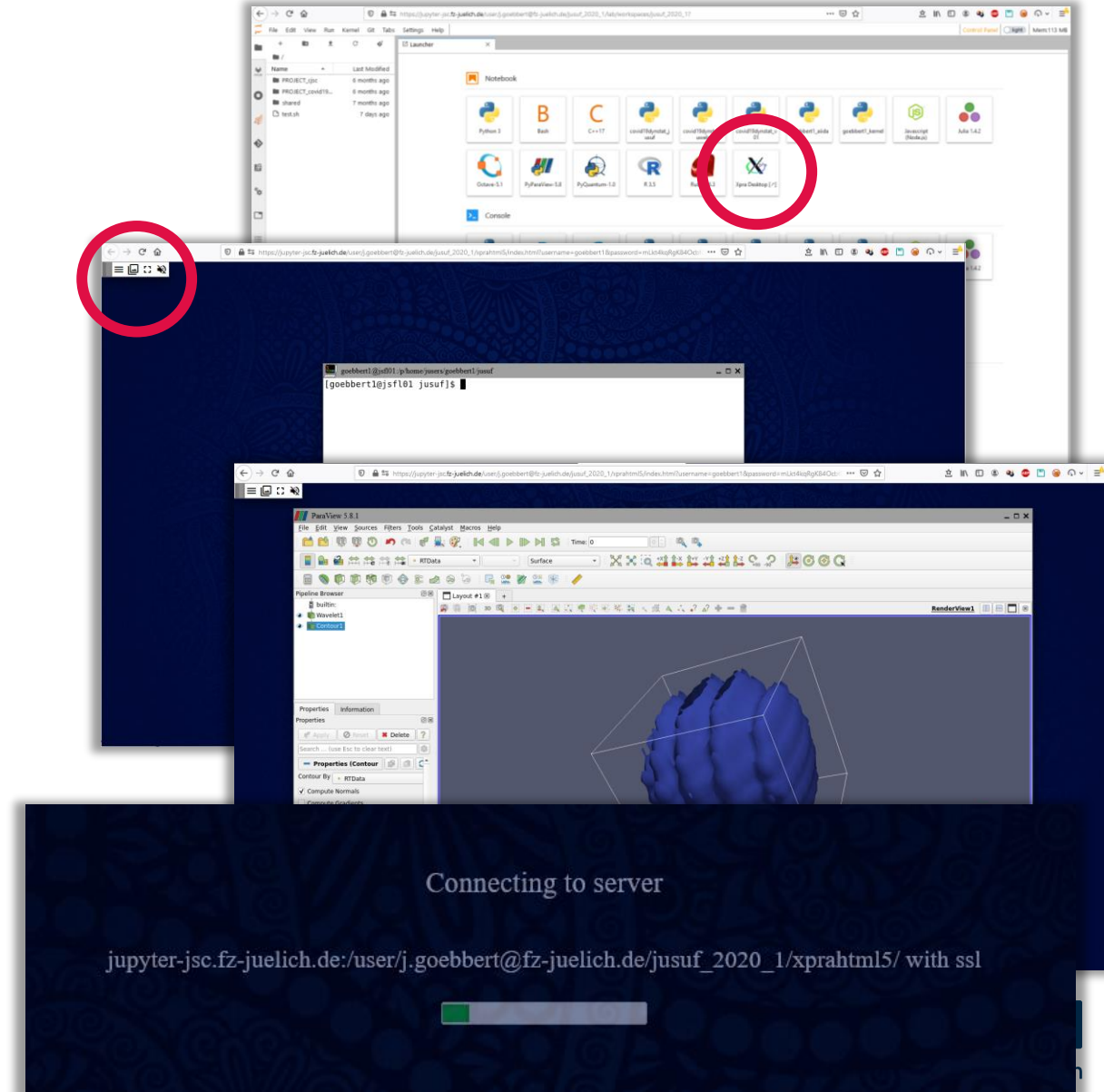
is a tool which runs X clients on a remote host and directs their display to the local machine.

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

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

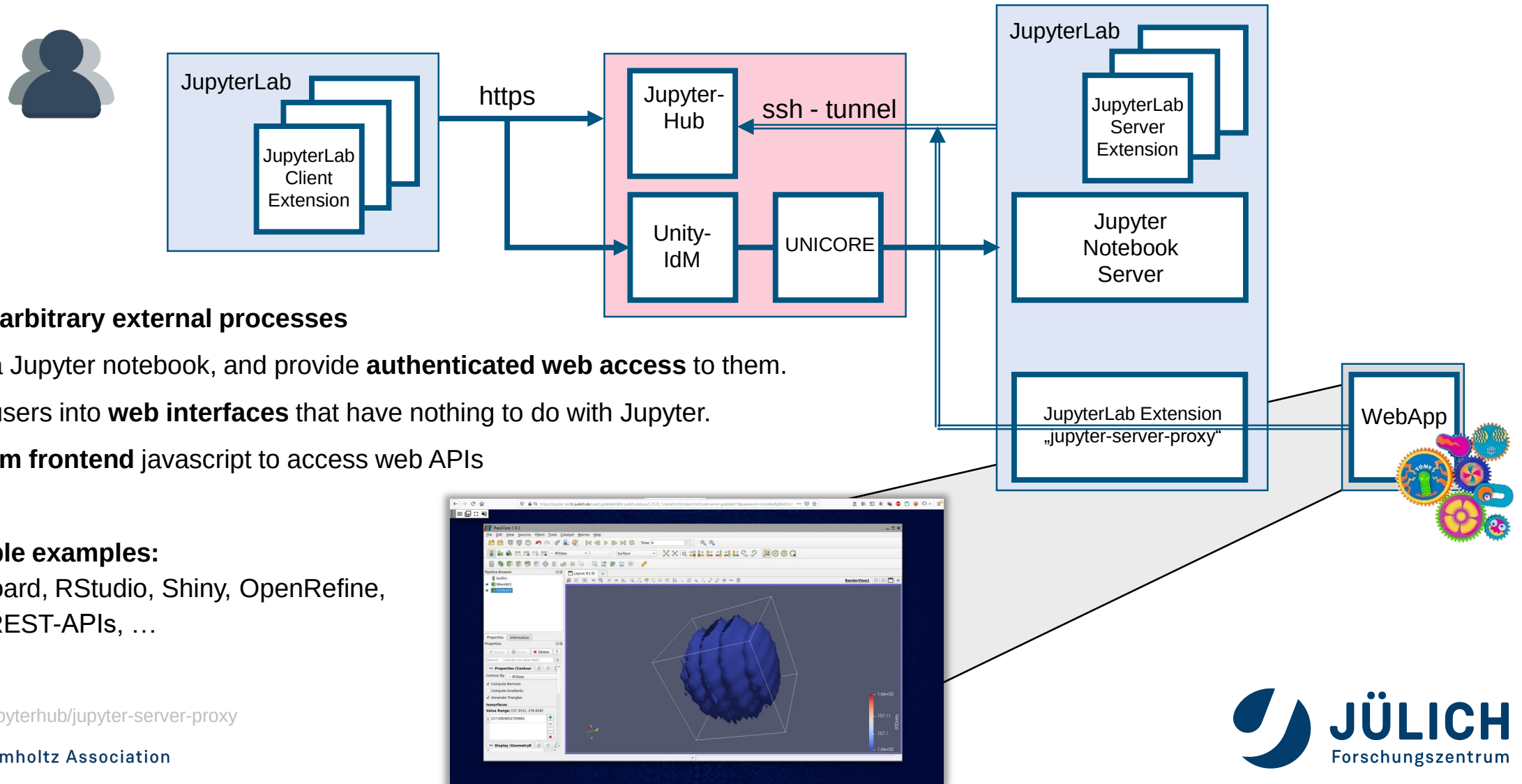
Hint:

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



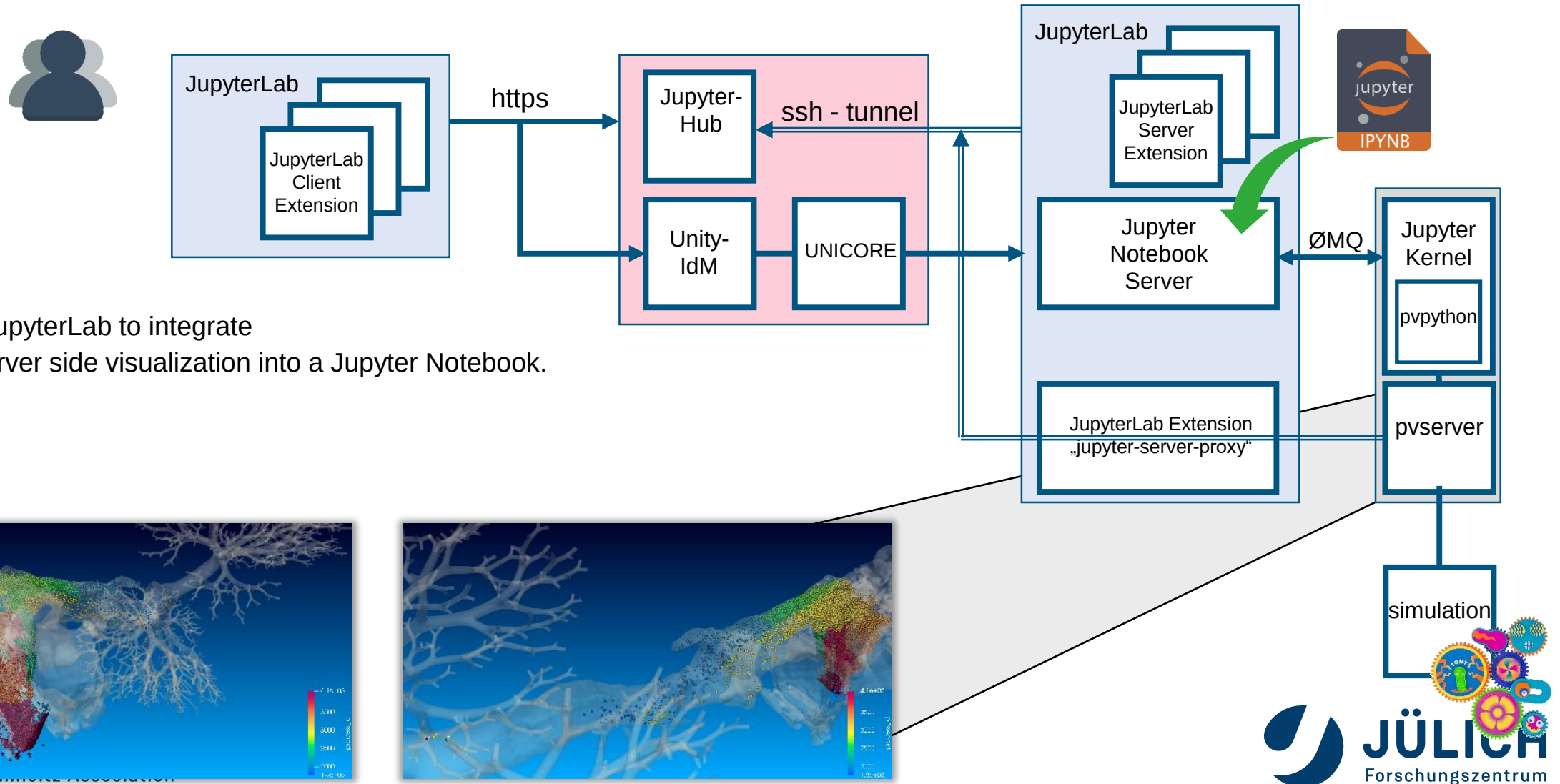
JUPYTERLAB – WEBSERVICE PROXY

Extension: jupyter-server-proxy



JUPYTERLAB – WEBSERVICE PROXY

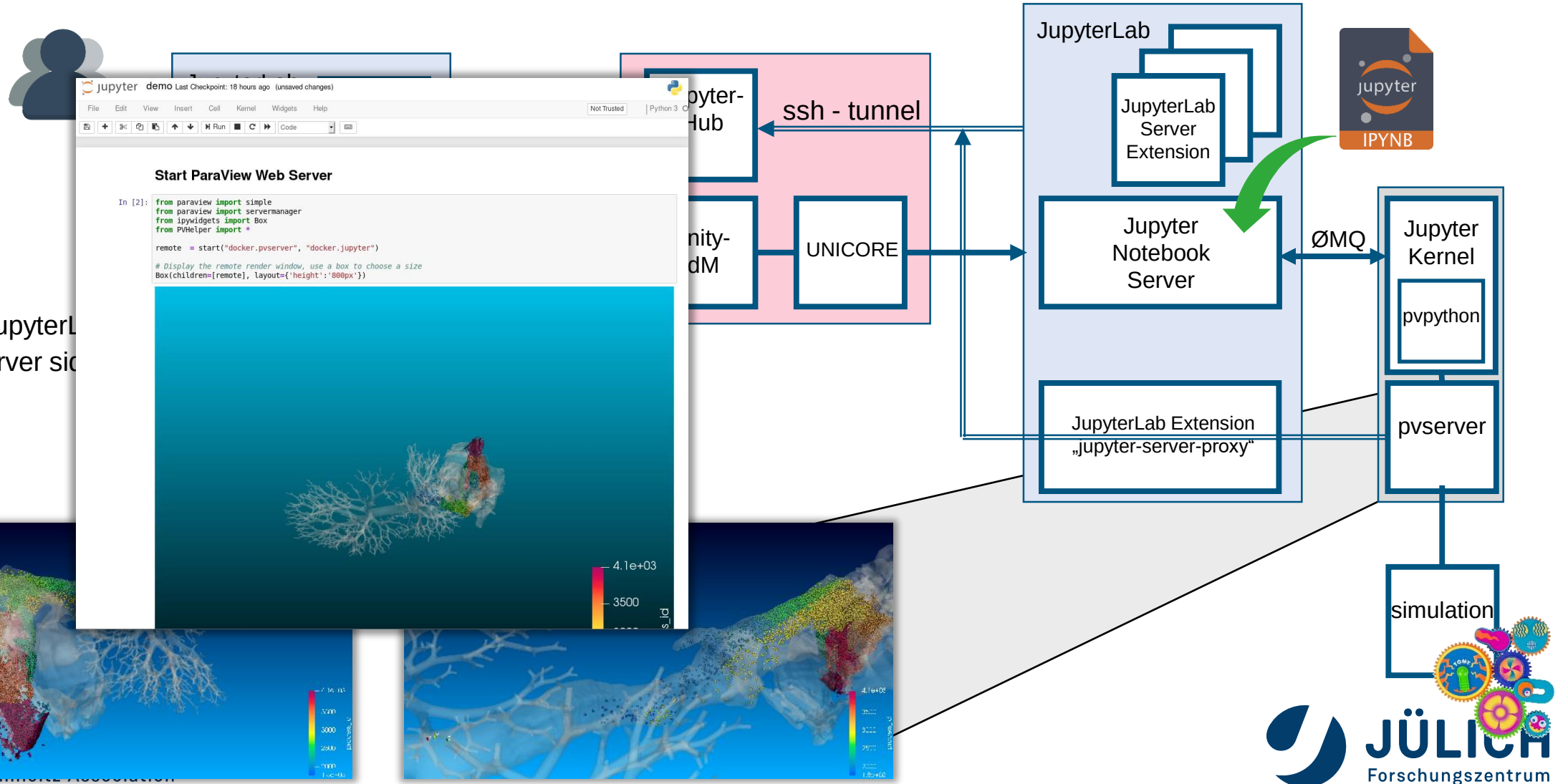
Extension: jupyter-server-proxy



JUPYTERLAB – WEBSERVICE PROXY

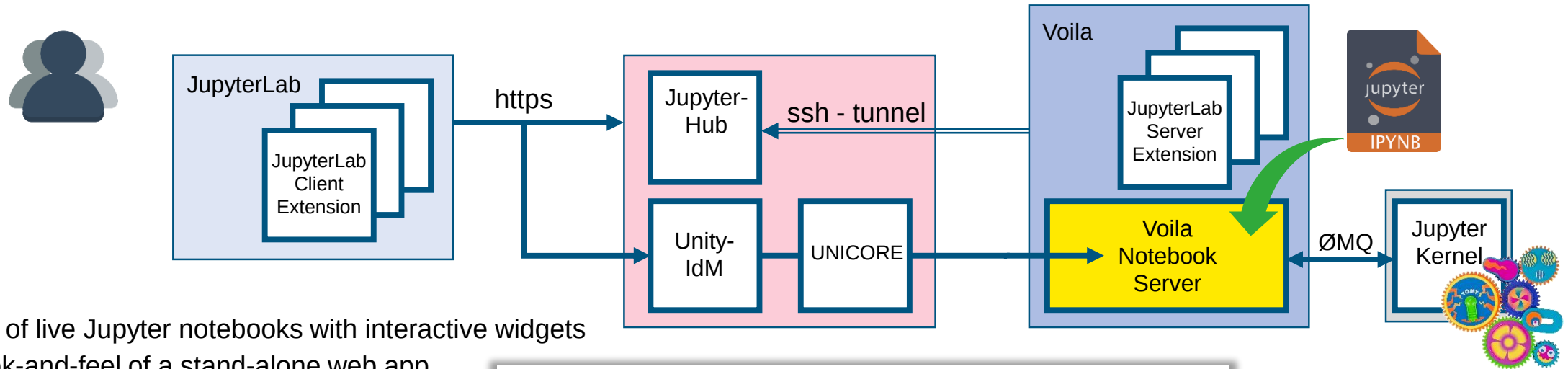
Extension: jupyter-server-proxy

How to use JupyterLab
interactive server side

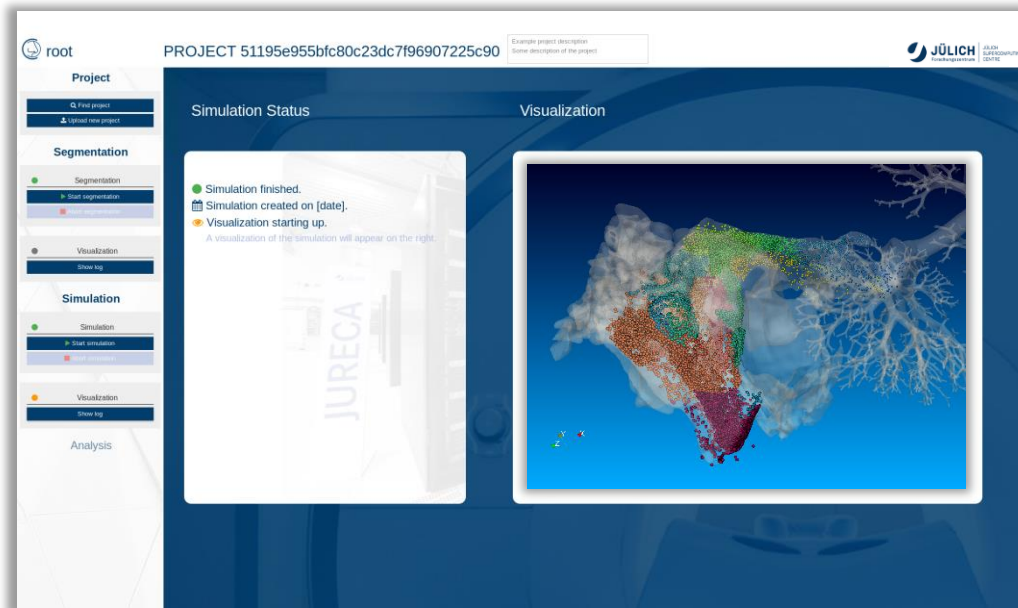


DASHBOARDS WITH JUPYTER/VOILA

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.



TUTORIALS

Get started with Jupyter

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>

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.

QUESTIONS?

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

