



# INTERACTIVE HCP WITH JUPYTER

PRACE Training Course

2021-04-20..22 | JENS. H. GÖBBERT  
ALICE GROSCH

(J.GOEBBERT@FZ-JUELICH.DE)  
(A.GROSCH@FZ-JUELICH.DE)

# SURVEY

Please take some time and fill out the survey.

The screenshot shows the PRACE Events Portal interface. At the top, the PRACE logo (a circle of stars) and the text 'PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE' are visible. To the right, it says 'EVENTS PORTAL'. Below this, a blue banner contains the course title '[ONLINE] Interactive High-Performance Computing with Jupyter'. Underneath the banner, the dates '20-22 April 2021' and 'CET timezone' are listed. A navigation menu on the left includes 'Overview', 'Registration', and 'Surveys' (which is highlighted). The main content area describes the course as an interactive exploration of large data from scientific simulations, mentioning Jupyter and JupyterLab. It states that the course provides a detailed introduction to interactive high-performance computing. A list of topics covered is provided, including Introduction to JupyterLab, Customizing JupyterLab, JupyterLab on HPC resources, Using JupyterLab as a proxy, Remote visualization within JupyterLab, Jupyter Interactive Widget Ecosystem, Utilizing supercomputers with JupyterLab, Extending JupyterLab, and Jupyter-JSC under the hood. Prerequisites listed are 'Experience in Python'.

<https://jupyter.org>

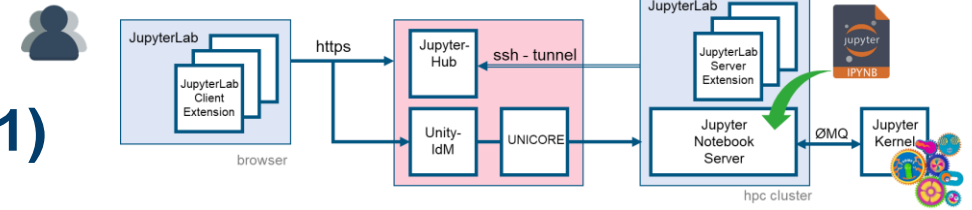
<https://events.prace-ri.eu/event/1162/>

# JUPYTER KERNEL



# 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/2020
lnode:> module load GCCcore/.9.3.0
lnode:> module load Python/3.8.5
```

3. Load extra modules you need for your kernel

```
lnode:> module load <module 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

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
```

[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

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

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

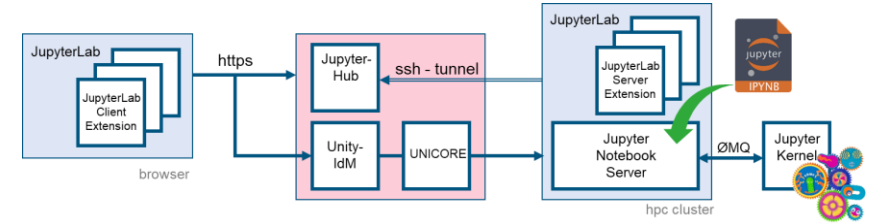
```
(<venv_name>) Lnode:> export PYTHONPATH=\n${VIRTUAL_ENV}/lib/python3.8/site-packages:${PYTHONPATH}
```

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:

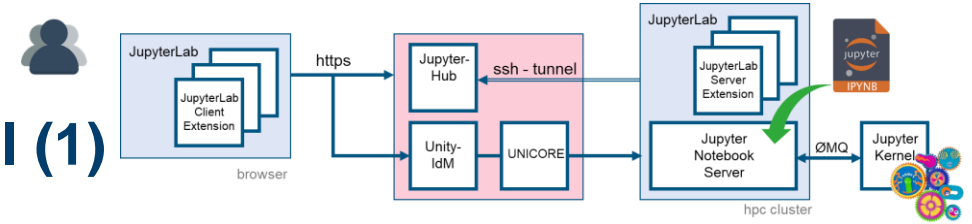
```
(<venv_name>) Lnode:> touch ${VIRTUAL_ENV}/kernel.sh
```

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
```

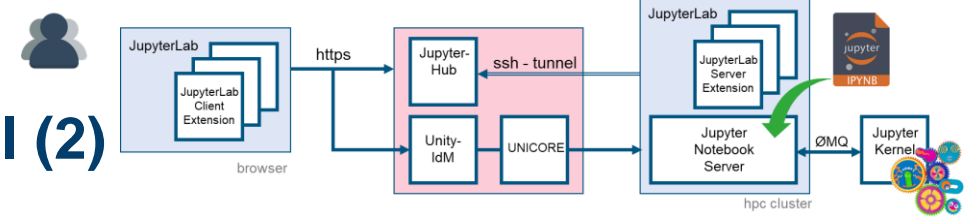


### Building your own Jupyter kernel is a three step process

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/.9.3.0
```

```
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



# JUPYTER KERNEL

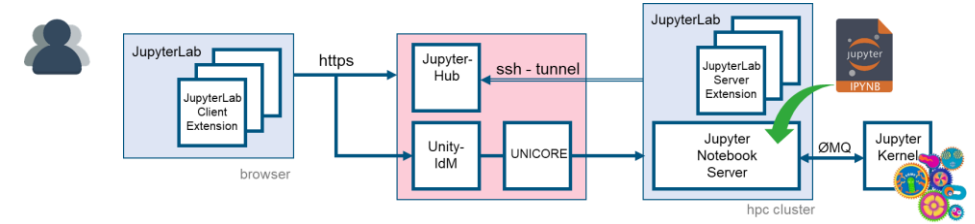
## 3. Create/Edit Jupyter kernel configuration (1)

### 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

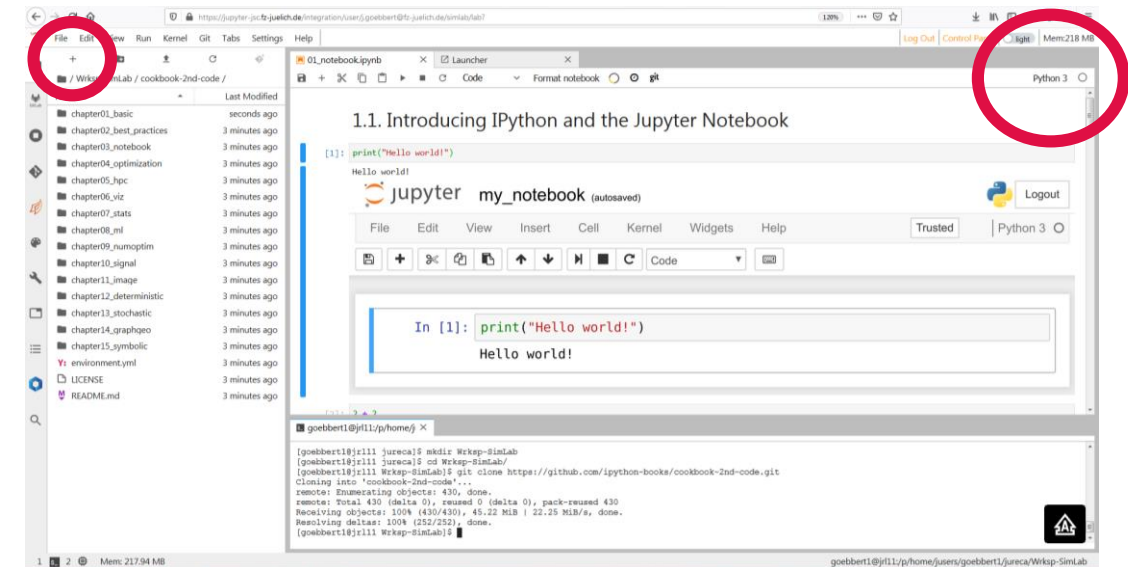
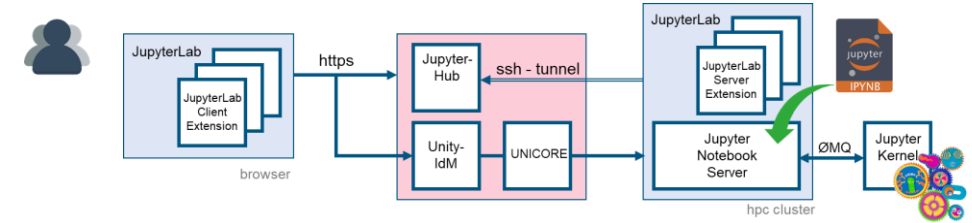
How to base your Jupyter Kernel on a Conda environment:

[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)

### 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.

