



## Interactive HPC with Jupyter (4)

training course, 26+27.05.2021

Jens Henrik Göbbert, [j.goebbert@fz-juelich.de](mailto:j.goebbert@fz-juelich.de)

Christian Witzler, [c.witzler@fz-juelich.de](mailto:c.witzler@fz-juelich.de)

Jülich Supercomputing Centre (JSC)  
Forschungszentrum Jülich (FZJ)



European  
Commission

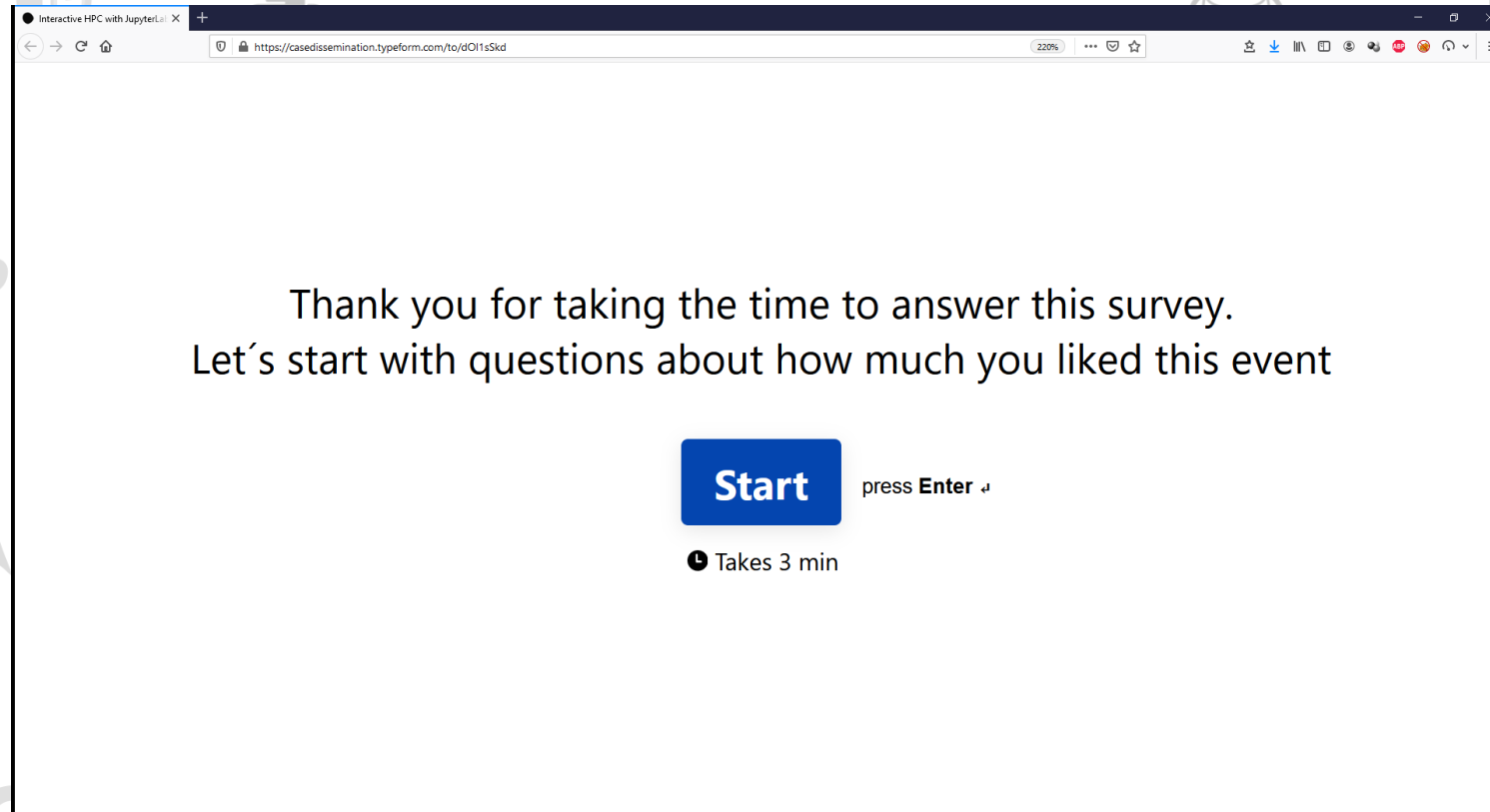
Horizon 2020  
European Union funding  
for Research & Innovation

*The CoEC project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 952181.*

*The CoE RAISE project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 951733.*

# SURVEY

Please take some time and fill out the survey.



The screenshot shows a web browser window with the address bar displaying <https://casedissemination.typeform.com/to/dOI1sSkd>. The main content area of the browser displays a message: "Thank you for taking the time to answer this survey. Let's start with questions about how much you liked this event". Below this message is a large blue button labeled "Start". To the right of the button, it says "press Enter ↵". Below the button, there is a clock icon and the text "Takes 3 min". The browser window has a dark theme and standard navigation buttons.

<https://casedissemination.typeform.com/to/dOI1sSkd>

# BUILD YOU OWN JUPYTER KERNEL

# JUPYTER KERNEL

## How to create your own Jupyter Kernel

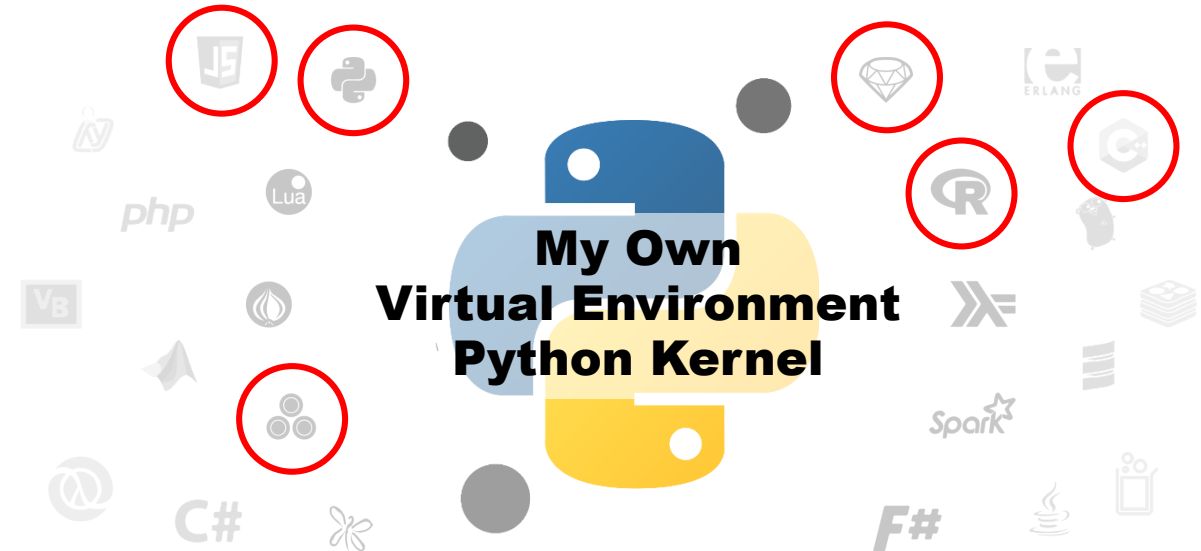
### Jupyter Kernel

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/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>
```

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

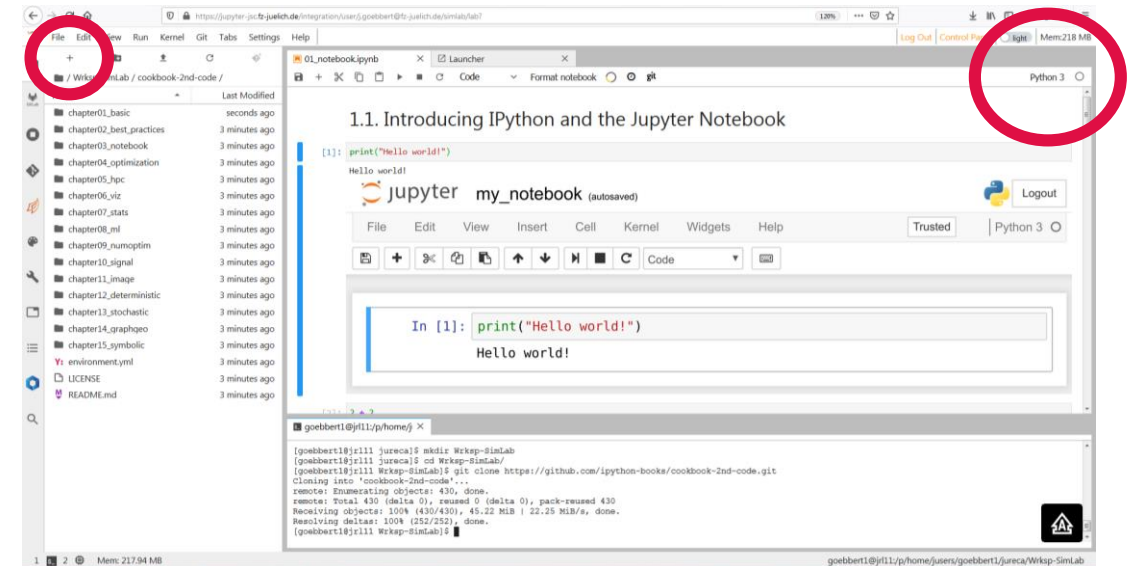
How to base your Jupyter Kernel on a Conda environment:

[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.

[https://github.com/FZJ-JSC/jupyter-jsc-notebooks/blob/master/001-Jupyter/Modify\\_JupyterKernel\\_at\\_NotebookRuntime.ipynb](https://github.com/FZJ-JSC/jupyter-jsc-notebooks/blob/master/001-Jupyter/Modify_JupyterKernel_at_NotebookRuntime.ipynb)

