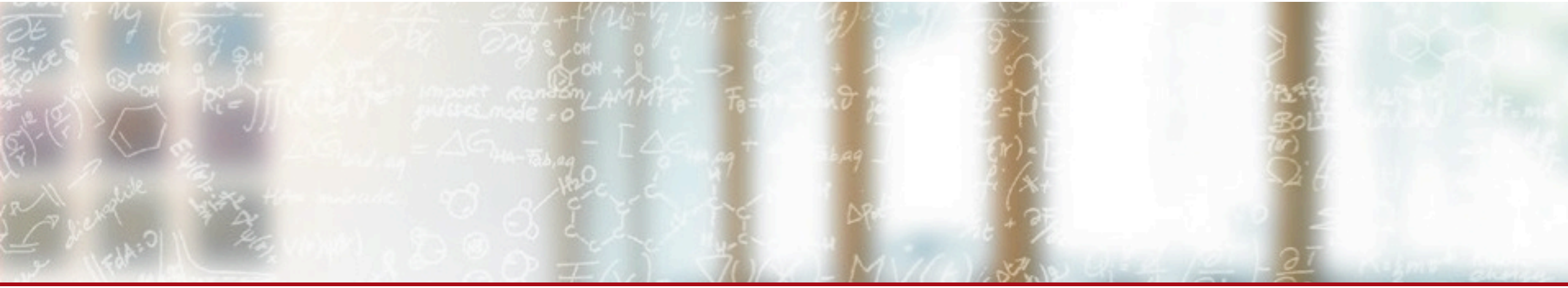
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



# Interactive Supercomputing with Jupyter Notebooks
# Tutorial on Python for HPC

CSCS-USI Summer School 2019

Tim Robinson and Rafael Sarmiento (ETH Zurich / CSCS)

July 22, 2019

# This morning's agenda

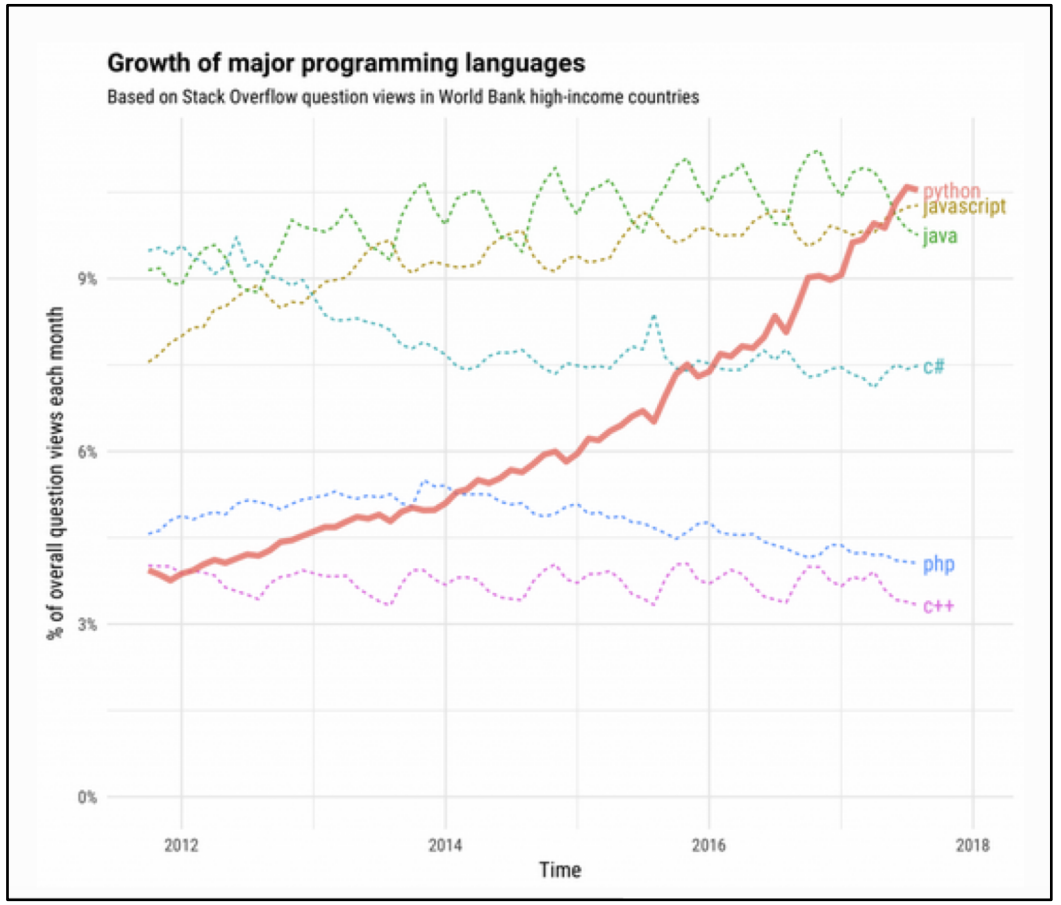
1. Interactive Supercomputing with Jupyter Notebooks

   *Coffee*

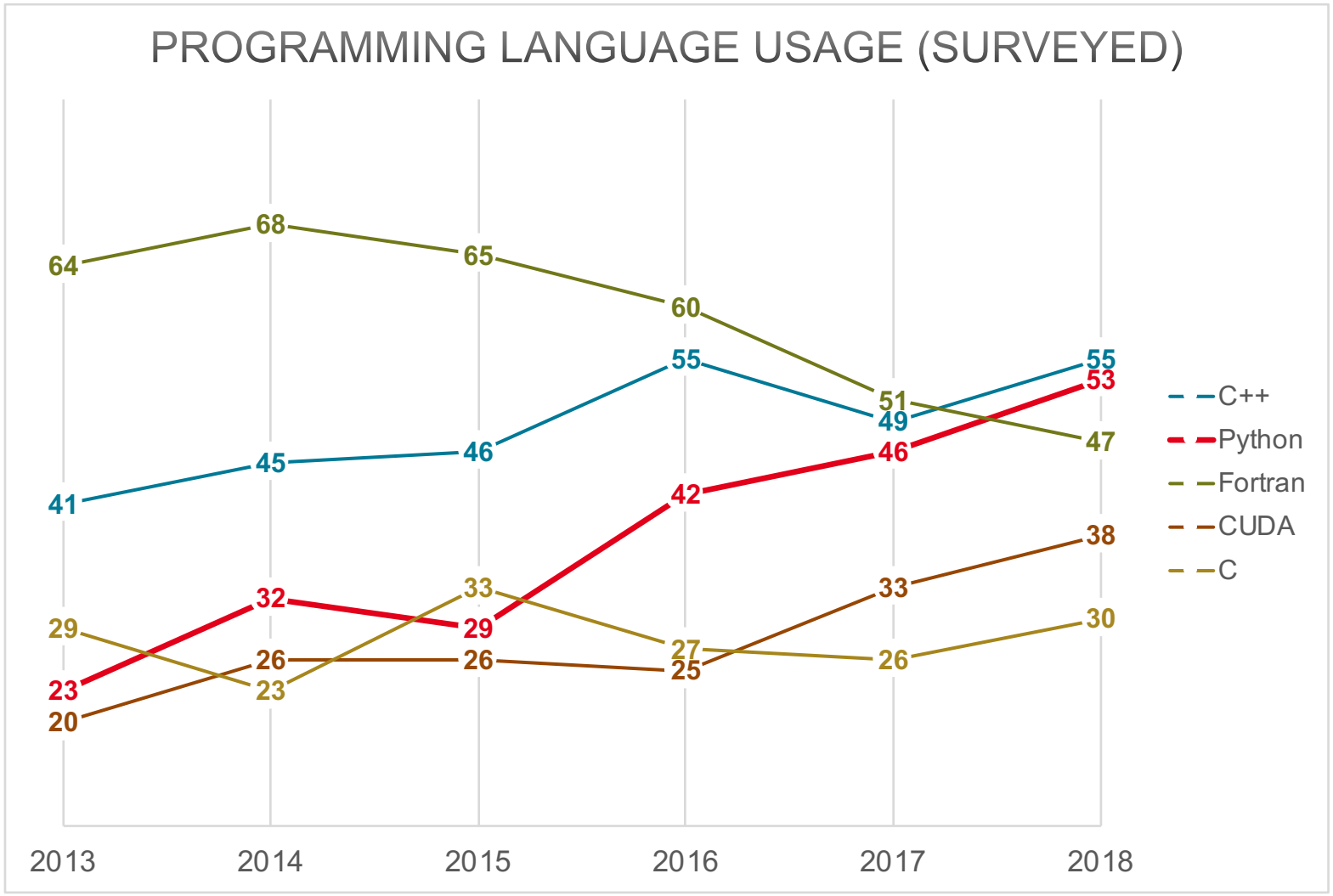
2. Tutorial on Python for HPC
3. [demo of distributed dask]

# The rise of Python

- Python has grown to become the dominant language both in data analytics and general programming
- Rise fueled by computational libraries like Numpy, Pandas, and Scikit-Learn and libraries for visualization, interactive notebooks, collaboration, etc.
- Python long used as glue, for pre- and/or post-processing.. but increasingly used for simulation as well

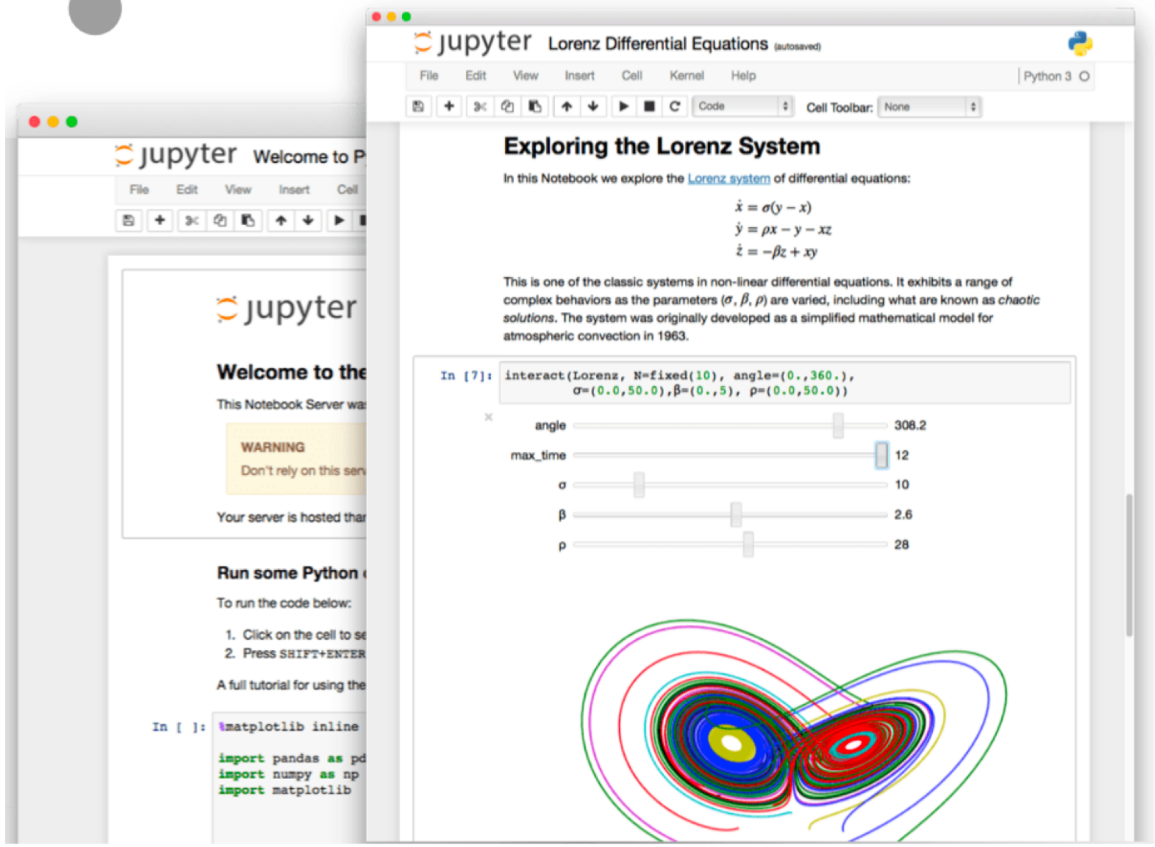


# Python usage at CSCS



# What is Jupyter?

- Project Jupyter – creating **reproducible computational narratives**
- Mainly known for the Jupyter Notebook
- Web server and web app to **create documents that contain narrative text, equations, code, results, visualizations, and rich media**
- Attached to a “kernel”, which does the computation

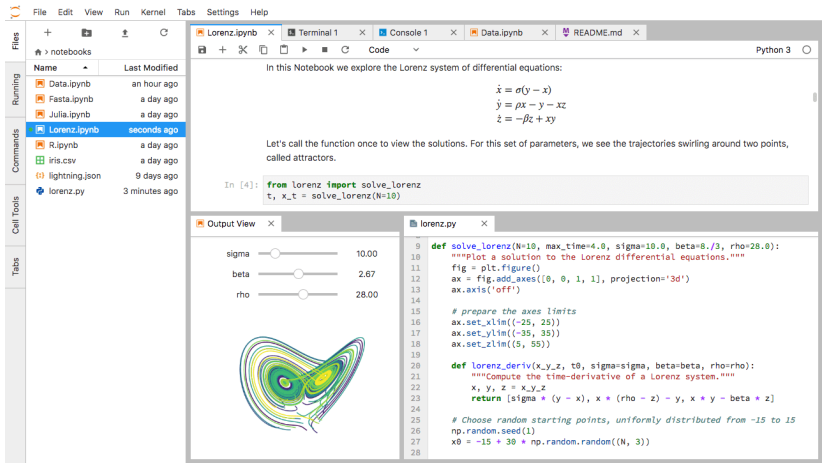


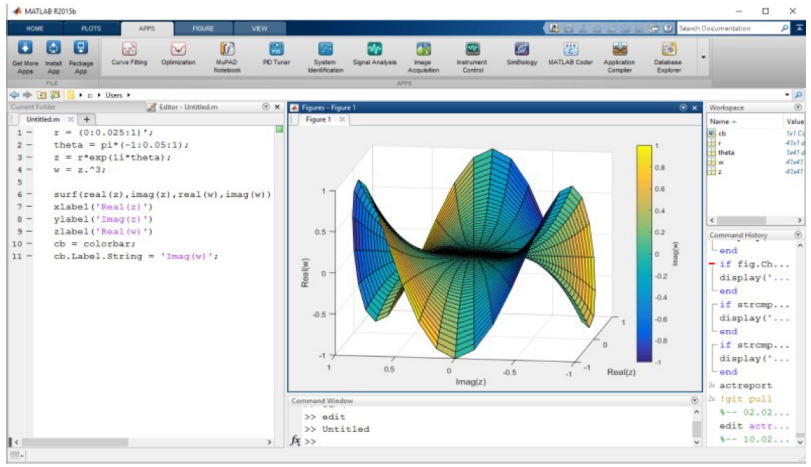
# Jupyter Notebook

- The all-in-one document is also called a Jupyter Notebook" (.ipynb, JSON format)
  - Easily shared with others
  - Can convert to PDF, HTML, LaTeX
- The working environment includes
  - In-browser terminal
  - File browsing
  - Support for many languages: Python, R, Julia, C++, …
  - Extensible design
  - Many server/client plugins
- Default implementation (IPython)


# JupyterLab

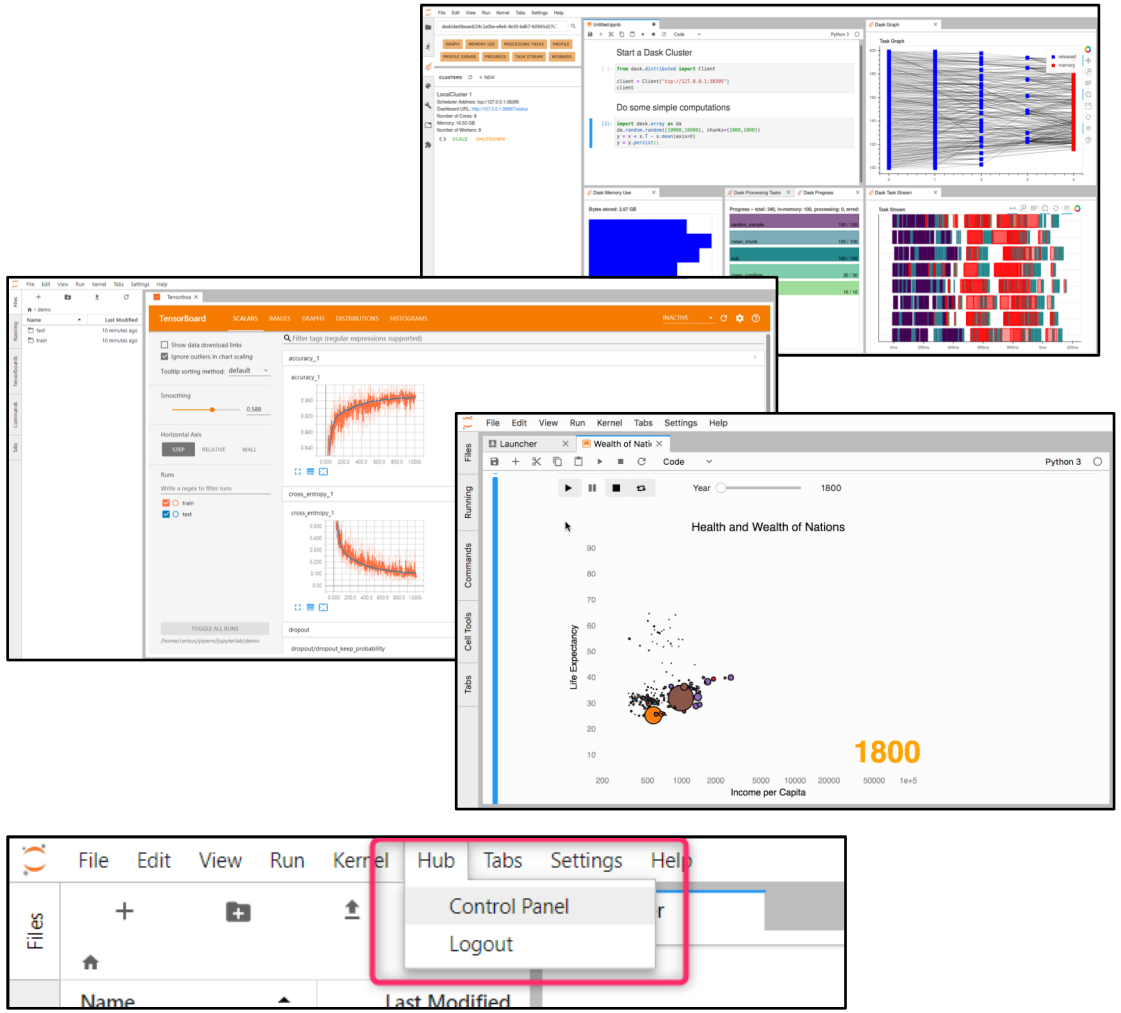
- Next-generation web-based user interface for Jupyter
- Provides higher degree of interaction between notebooks, documents, text editors and other activities (arrange with tabs/splitters)
- Advanced interactive development environment
- Served from same server and uses same notebook document format



# JupyterLab Extensions

- JuptyerLab is designed as a customizable, extensible environment
- Extensions can provide new themes, file viewers and editors, and renderers for rich output
- 100 GitHub repos tagged “jupyterlab-extensions”
- Examples include:
  - dask-labextension
  - jupyterlab-tensorboard
  - bqplot
  - jupyterlab-hub



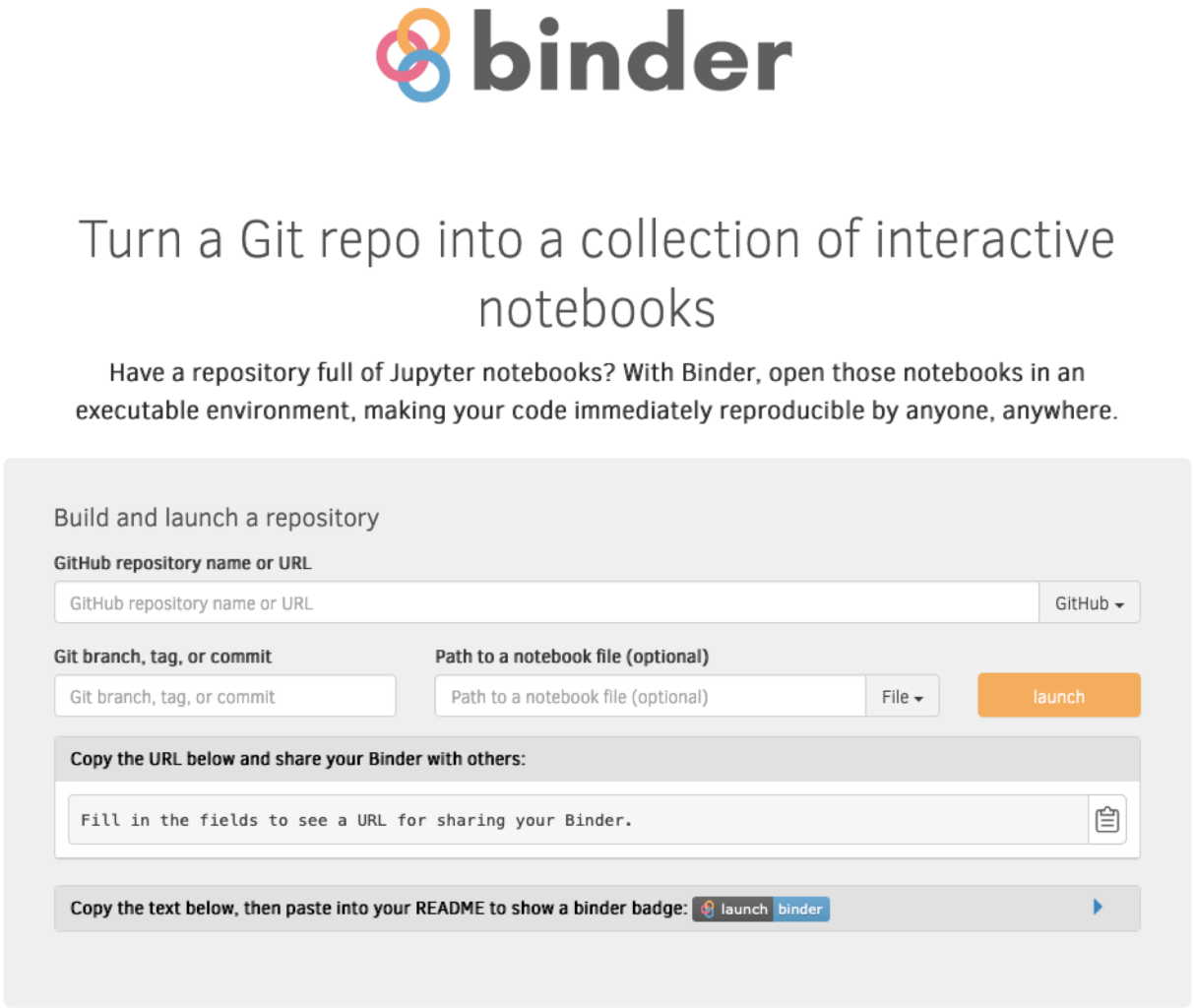
# Sharing notebooks

# Sharing static notebooks

https://nbviewer.jupyter.org/

# Sharing live notebooks

https://mybinder.org/



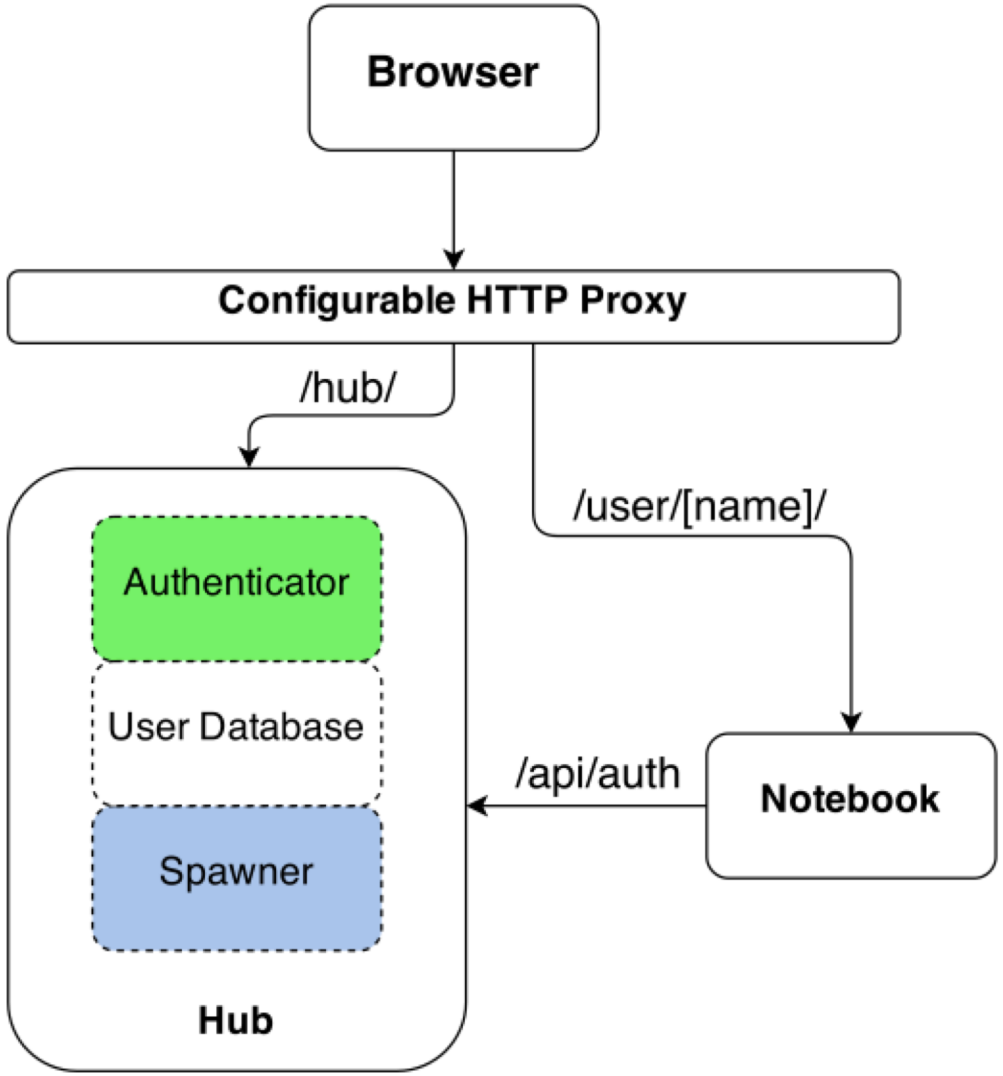
# JupyterHub

# JupyterHub

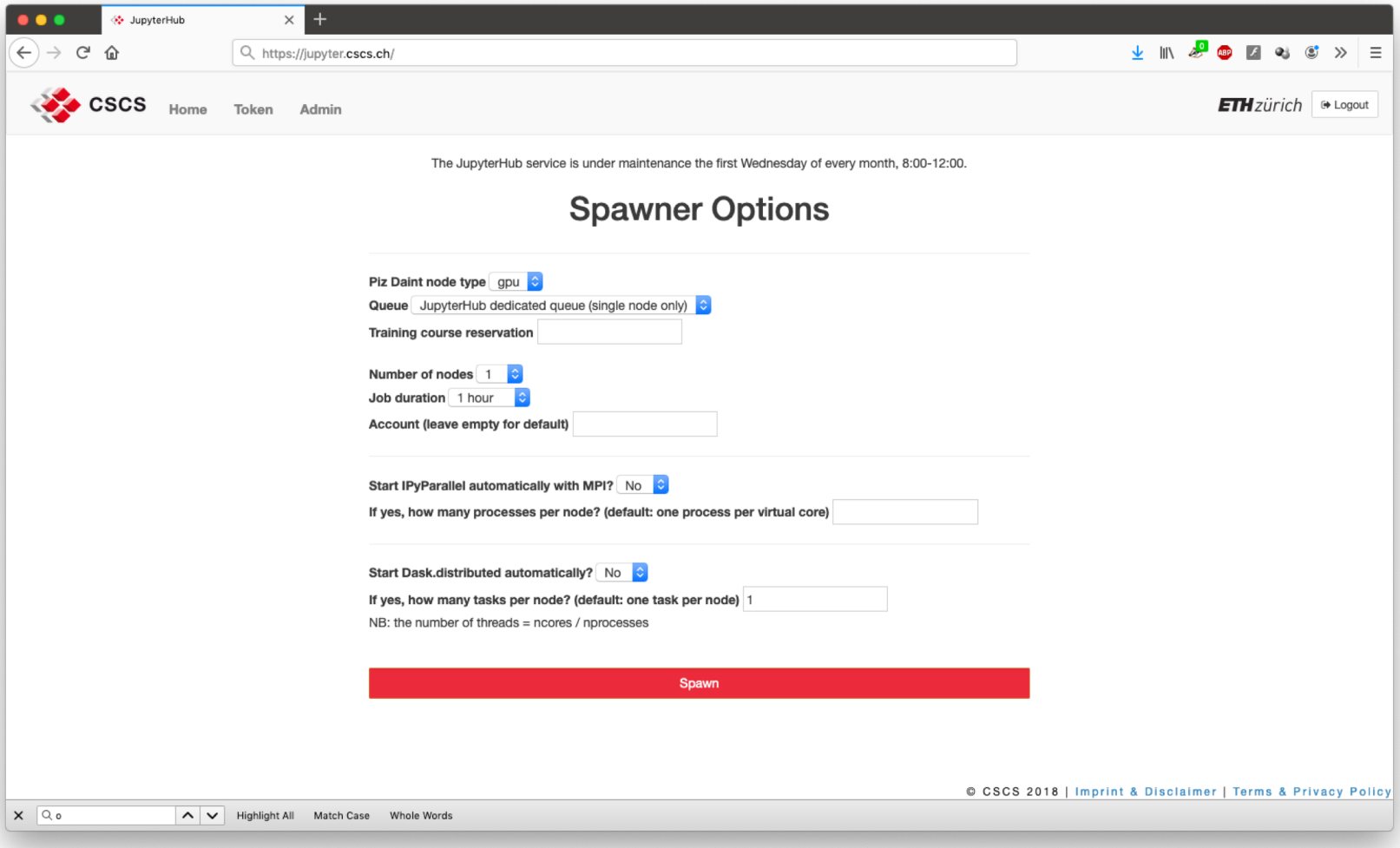
- JupyterHub is a way to give Jupyter notebook servers to a group of users – e.g. users of a supercomputing site, a research group, a classroom of summer school students, …

# JupyterHub

- Multi-user server for Jupyter Notebooks (designed for classrooms, research labs, Universities...)
- Spawns, manages and proxies multiple instances of the single-user Jupyter Notebook server
- Three main subsystems
  - **Multi-user Hub**
  - Configurable **http proxy** (node-http-proxy)
  - Multiple **single-user Jupyter notebook servers** (Python/IPython/tornado)
- The key pluggable components are the authenticator and spawner



# JupyterHub at CSCS



# Demo of JupyterLab interface and exercises

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

# Custom kernels in Jupyter

# Installing custom kernels for JupyterLab

- Jupyter kernels live in `${HOME}/.local/share/jupyter/kernels`
- You can create environments using virtualenv or conda and install them as kernels
- E.g. create a virtual environment “twr” and activate:

```
> module load daint-gpu jupyterlab
> python3 -m venv --system-site-packages twr
> source ~/twr/bin/activate
(twr)> pip install <required_modules> --user
```

- Now create the kernel inside the environment

```
(twr)> export PYTHONPATH=~/twr/lib/python3.6/site-packages:${PYTHONPATH}
(twr)> pip install --ignore-installed ipykernel --user
(twr)> python3 -m ipykernel install --user --name twr --display-name TWR
Installed kernelspec twr in /users/robinson/.local/share/jupyter/kernels/twr
```

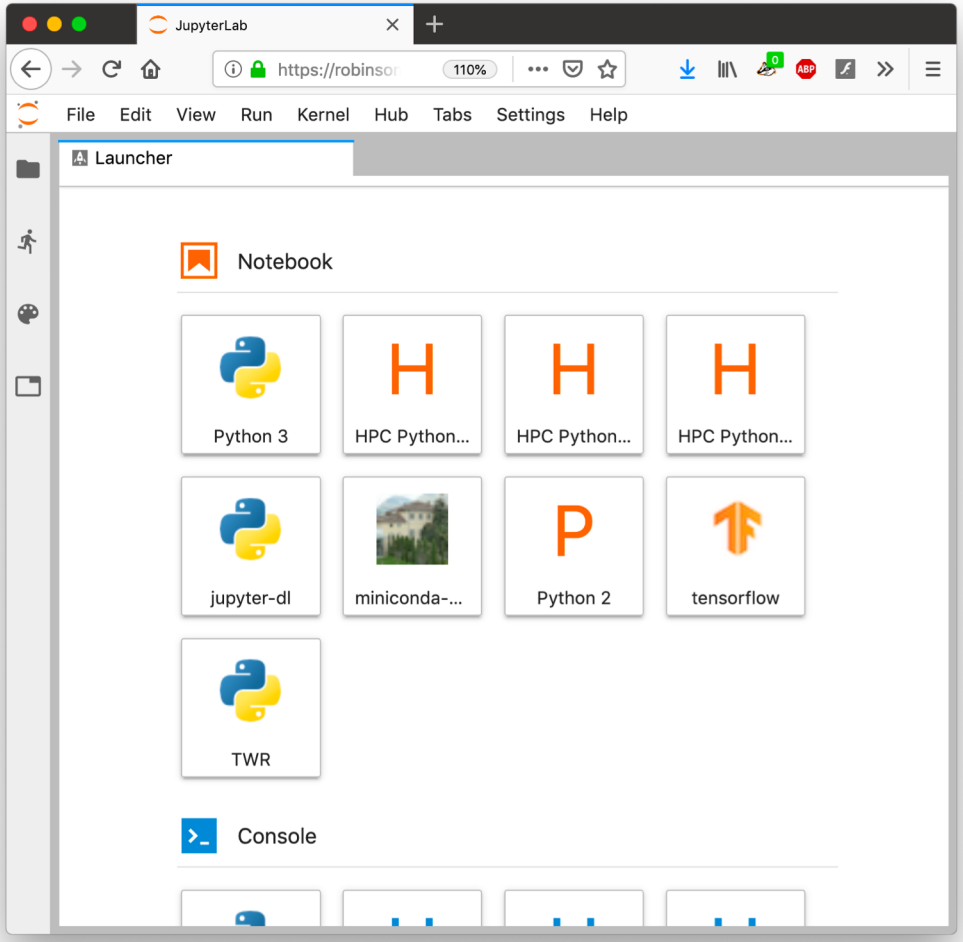
- This has created and installed a “kernel spec” file (kernel.json) at the path described in the output

# Installing custom kernels for JupyterLab (2)

- The kernel spec file looks like:

```
{
 “argv”: [
 “/users/robinson/twr/bin/python3”,
 “-m”,
 “ipykernel_launcher”,
 “-f”,
 “connection_file”
 ],
 “display_name”: “TWR”,
 “language”: “python”
}
```

- The kernel will be available in the JupyterLab launcher after a browser refresh



# Installing custom kernels for JupyterLab (3)

- If you need access to executables from a custom `PATH` or libraries in `LD_LIBRARY_PATH` you can add these to the kernel spec in an `env` dictionary:

```
{
 “argv”: [
 “/users/robinson/twr/bin/python3”,
 “-m”,
 “ipykernel_launcher”,
 “-f”,
 “connection_file”
 ],
 “display_name”: “TWR”,
 “language”: “python”,
 "env": {
     "PATH":
         "/users/robinson/bin:/usr/local/bin:/usr/bin:/bin",
     "LD_LIBRARY_PATH":
          "/project/<id>/myproject/lib:/users/robinson/lib”
 }
}
```

# Further customizing your kernel with a helper shell script

- A better method is to use a kernel helper shell script, and call it from kernel spec:

```
{
 “argv”: [
 “/users/robinson/.local/share/jupyter/kernels/twr/kernel-helper.sh”,
 “-f”,
 “connection_file”
 ],
 “display_name”: “TWR”,
 “language”: “python”
}
```

- You can put anything you need to configure your environment in the helper script. It must end with the ipykernel_launcher command, and the script must be executable:

```
#!/bin/bash
module load TensorFlow/1.12.0-CrayGNU-18.08-cuda-9.1-python3
module load Horovod/0.16.0-CrayGNU-18.08-tf-1.12.0
source ~/twr/bin/activate
export PYTHONPATH=~/twr/lib/python3.6/site-packages:${PYTHONPATH}
python -m ipykernel_launcher $@
```

# Exercise: Adding a custom kernel

- We have prepared a conda environment that (amongst other things) includes RAPIDS, which are software libraries from NVIDIA for data science and analytics pipelines on GPUs
- You will need to use this kernel for the next set of exercises
- The kernel is called “miniconda-sumsch”
- Make the kernel available in your own JupyterLab
- Instructions can be found in the notebook “ss2019_install_kernel.ipynb"

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

# Tutorial on Python for HPC

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

# Distributed Dask

# Dask Distributed

- Dask enables parallelization on larger-than-memory data using blocked algorithms and task scheduling: scale python without rewriting code
  - Originally designed for numpy and pandas – now more generic (e.g. dask delayed)
  - Distributed extends dask to clusters
- A dask distributed network consists of
  - scheduler node
  - one or more worker nodes
- Launch `dask-scheduler` on first compute node of the allocation. Then launch `dask-worker` on all nodes, providing address (IP, port) of `dask-scheduler`

# Dask Distributed with Enron Email Corpus


- Enron was an American energy, commodities and services company in Houston Texas
- The Enron Corpus is a dataset of 600K emails related to the investigation of the collapse of the company
- One of the few publicly available mass collections of real email
- Commonly-used dataset for text-based data analytics
- Dask Bag is able to store and process collections of Pythonic objects that are unable to fit into memory: useful for processing logs and collections of JSON documents, for example