

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

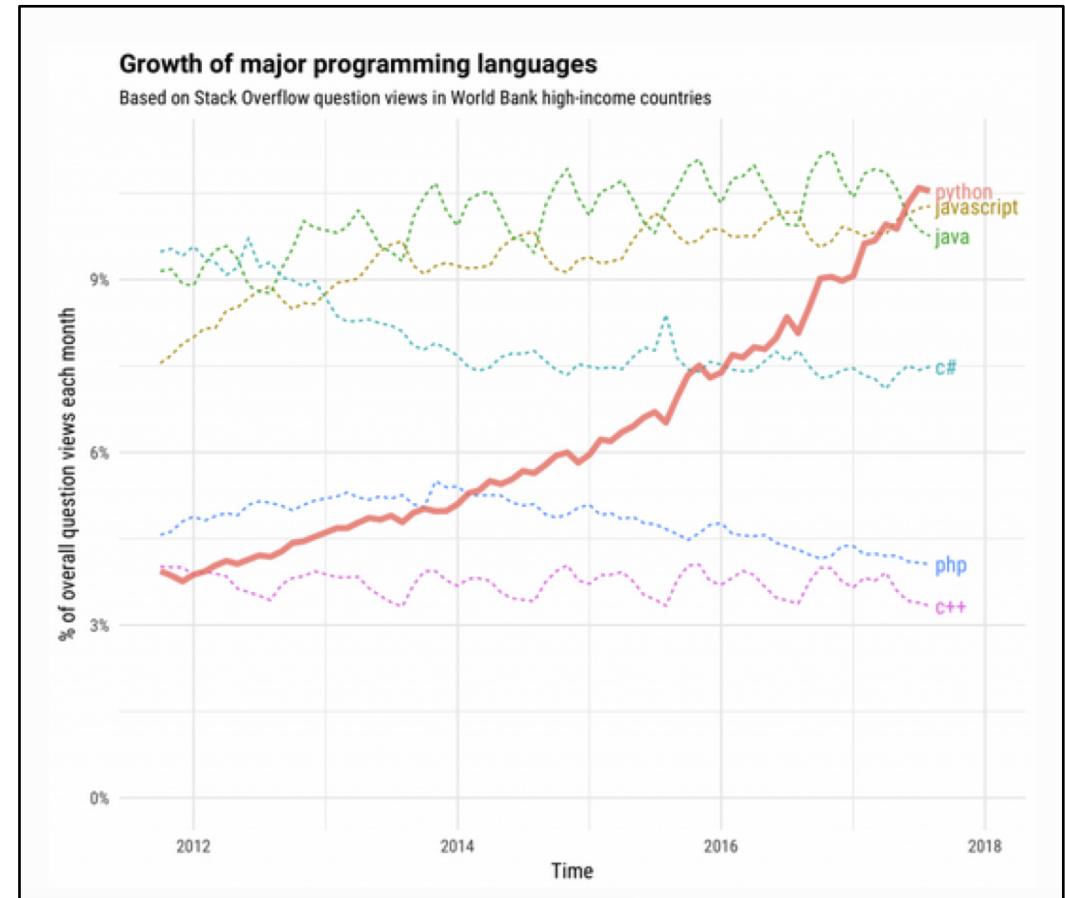
Coffee

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

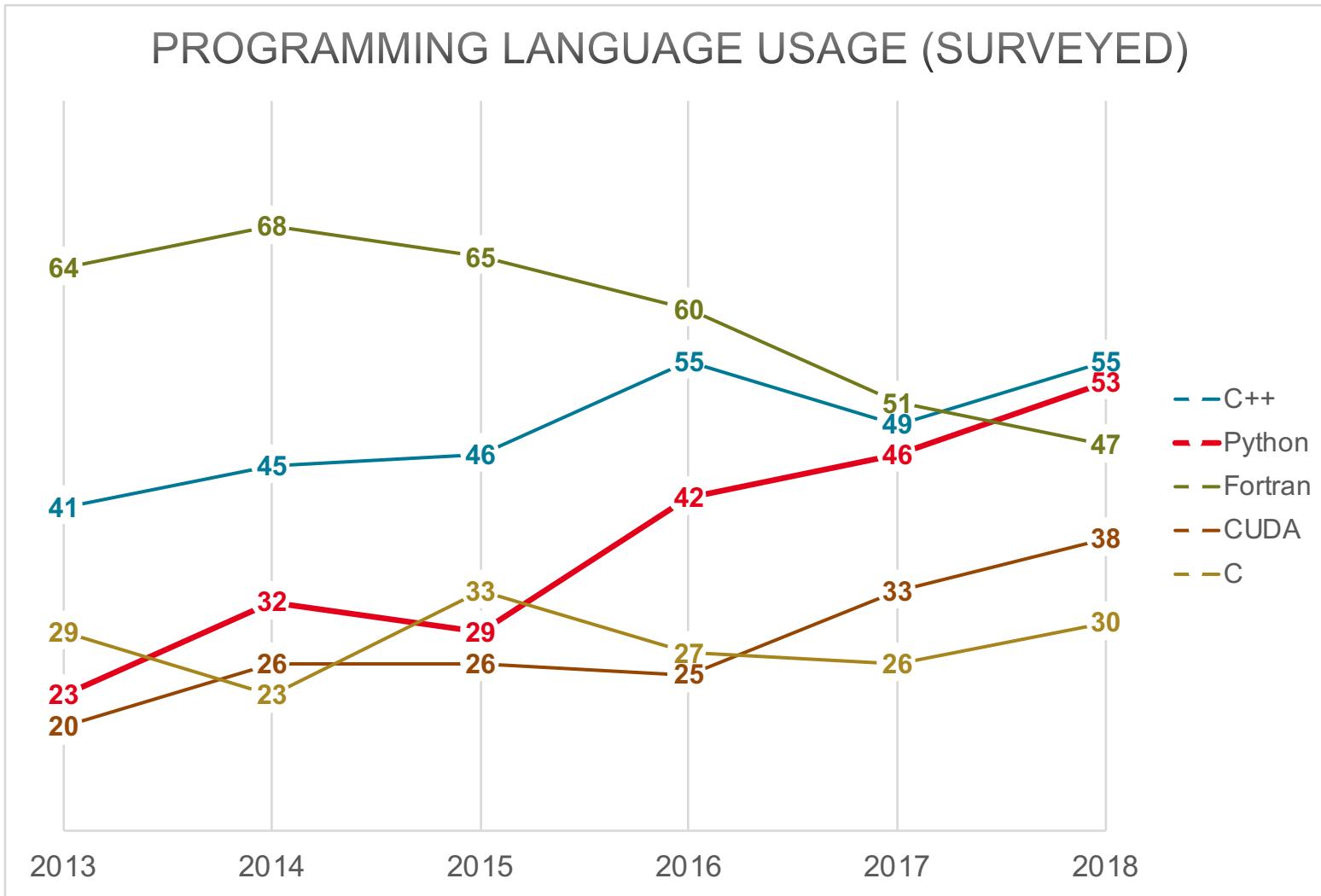
The rise of Python



- 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

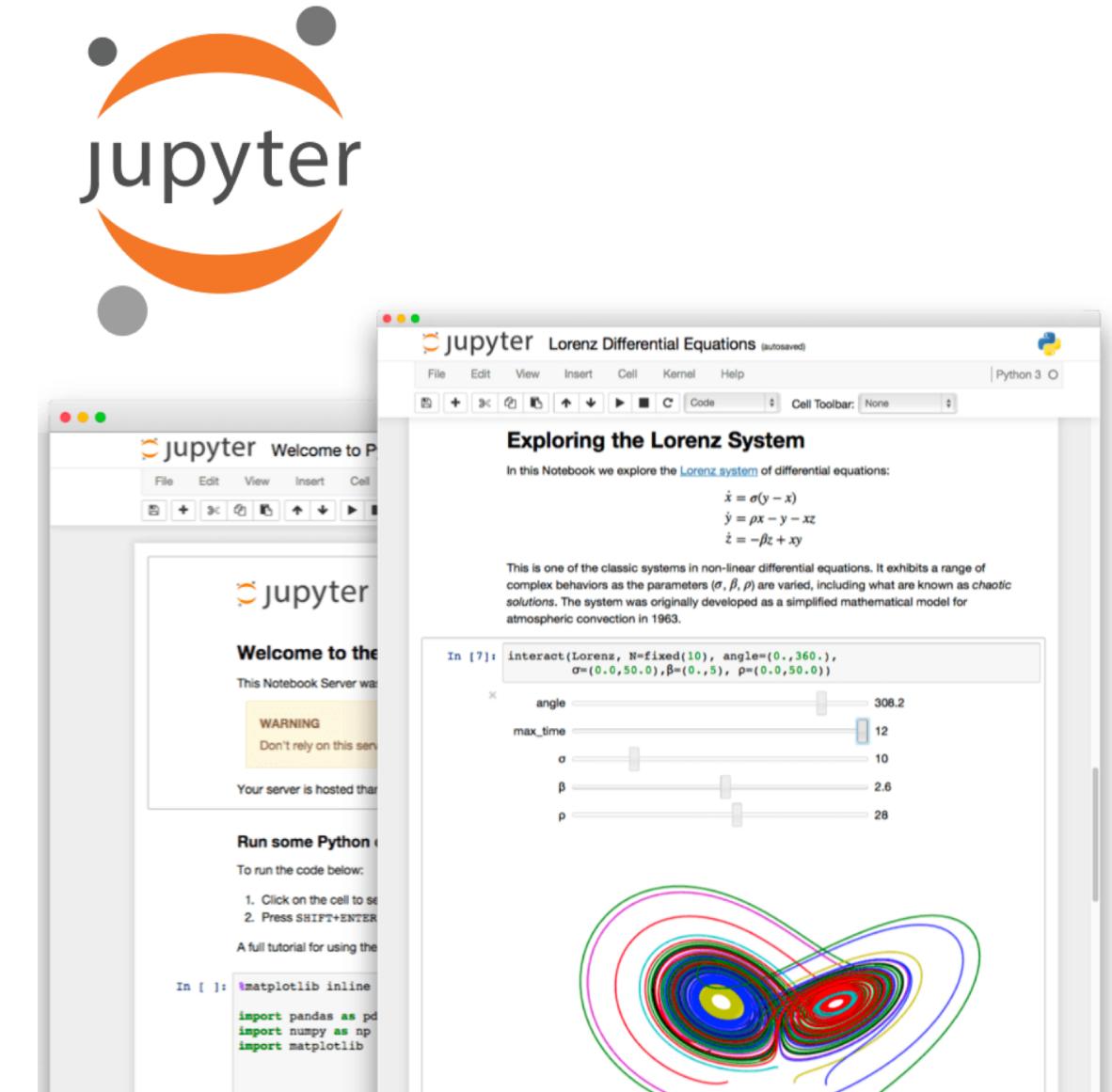


Python usage at CSCS



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)

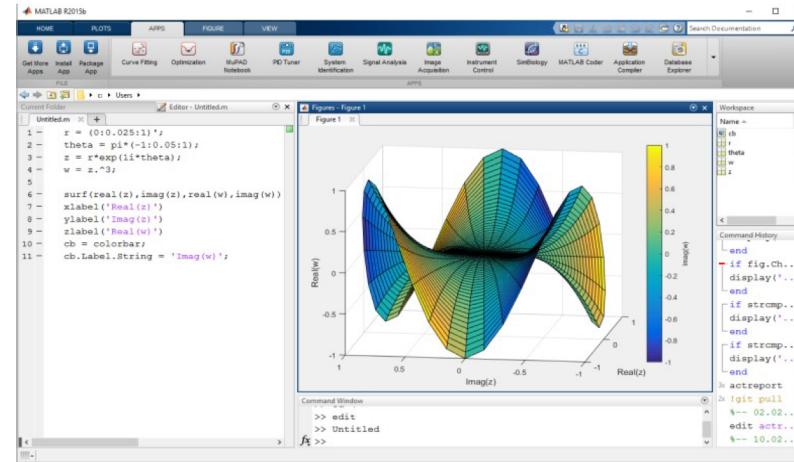
IP[y]:
IPython



JupyterLab

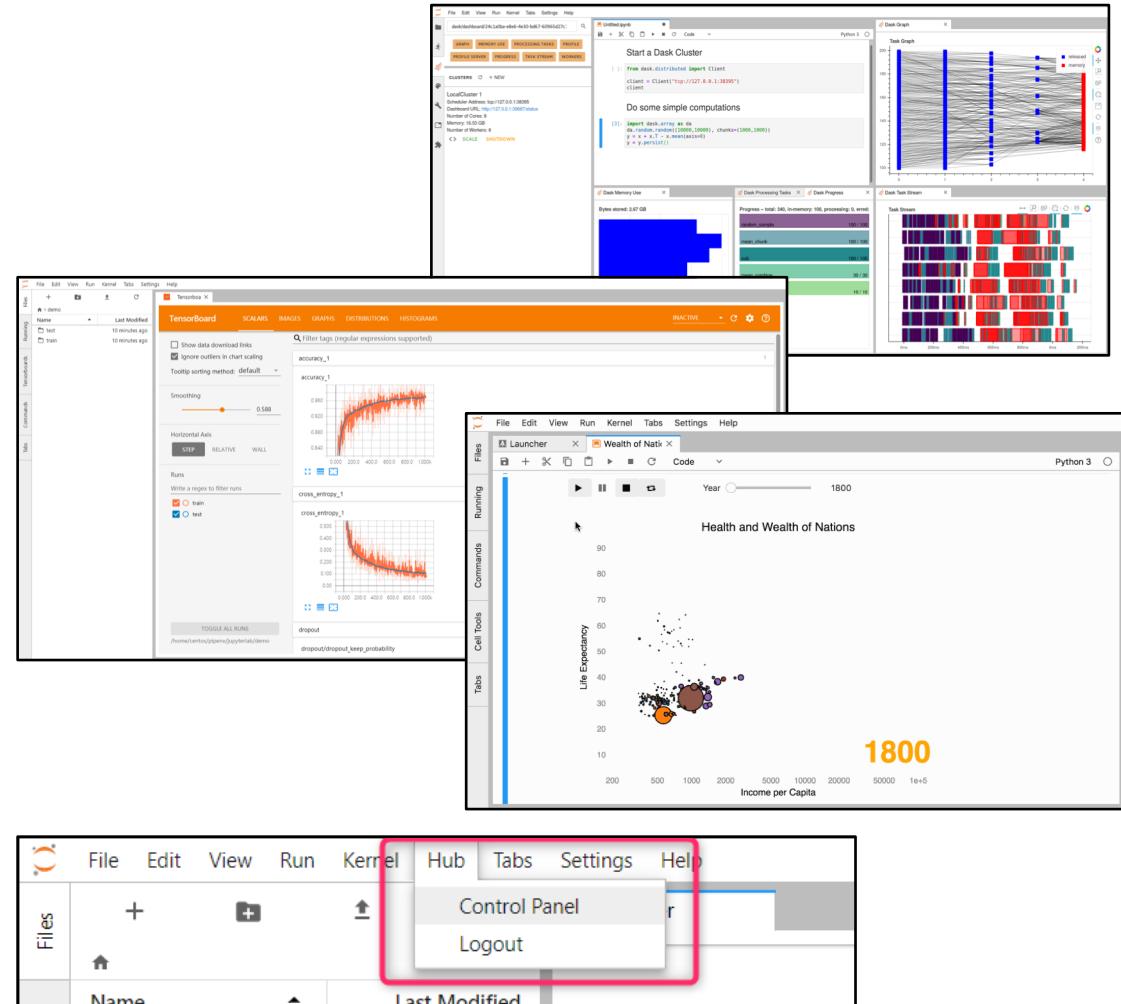
- Next-generation web-based user interface for Jupyter
- Provides higher degree of interaction between notebooks, documents, text editors and other activities (arrange with tabs/splitters)
- Advanced interactive development environment
- Served from same server and uses same notebook document format

The screenshot shows the JupyterLab interface. On the left, there's a sidebar with 'File', 'Edit', 'View', 'Run', 'Kernel', 'Tabs', 'Settings', and 'Help'. Below it are sections for 'Notebooks' (with files like Lorenz.ipynb, Data.ipynb, Fasta.ipynb, Julia.ipynb, R.ipynb, iris.csv, lightning.json, and lorenz.py), 'Running' (seconds.egg), 'Commands', 'Cell Tools', and 'Tabs'. The main area has tabs for 'Lorenz.ipynb', 'Terminal 1', 'Console 1', 'Data.ipynb', and 'README.md'. A code cell in 'Lorenz.ipynb' contains the Lorenz equations and imports. Another cell shows a plot of the Lorenz attractor with sliders for parameters sigma, beta, and rho. The 'Output View' shows the resulting 3D plot.



JupyterLab Extensions

- JupyterLab 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





CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETHzürich

Sharing notebooks

Sharing static notebooks

<https://nbviewer.jupyter.org/>

nbviewer

A simple way to share Jupyter Notebooks

Enter the location of a Jupyter Notebook to have it rendered here:

URL | GitHub username | GitHub username/repo | Gist ID

Go!

Programming Languages

IPython



IRuby



IJulia

An IJulia Preview

This notebook is a preview demo of IJulia: a [Julia language](#) backend combined with the [IPython](#) interactive environment. This combination allows you to interact with the Julia language using IPython's powerful [graphical notebook](#), which combines code, formatted text, math, and multimedia in a single document.



• Note: this is a preview, because it relies on pre-release bleeding-edge versions of Julia, IPython, and several Julia packages, as explained on the [Julia GitHub page](#), and functionality is evolving rapidly. We hope to have a more polished release soon.

Basic Julia interaction

Sharing live notebooks

<https://mybinder.org/>



Turn a Git repo into a collection of interactive notebooks

Have a repository full of Jupyter notebooks? With Binder, open those notebooks in an executable environment, making your code immediately reproducible by anyone, anywhere.

Build and launch a repository

GitHub repository name or URL

GitHub repository name or URL

GitHub ▾

Git branch, tag, or commit

Git branch, tag, or commit

Path to a notebook file (optional)

Path to a notebook file (optional)

File ▾

launch

Copy the URL below and share your Binder with others:

Fill in the fields to see a URL for sharing your Binder.

📋

Copy the text below, then paste into your README to show a binder badge:

launch binder

▶



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETHzürich

JupyterHub

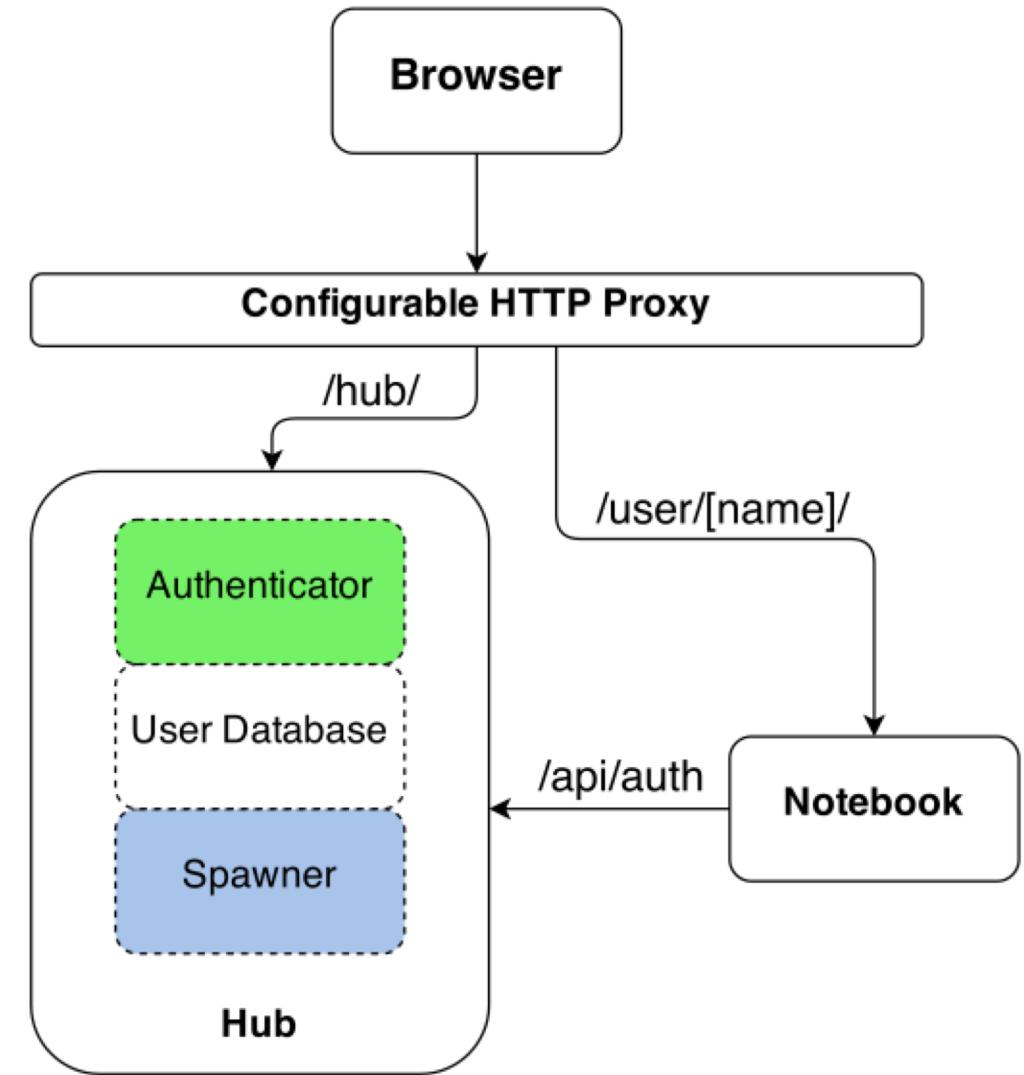
JupyterHub

- 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

The screenshot shows a web browser window for the JupyterHub service at <https://jupyter.cscs.ch/>. The page title is "JupyterHub". The header includes the CSCS logo, "Home", "Token", "Admin", "ETH zürich", and "Logout". A message states: "The JupyterHub service is under maintenance the first Wednesday of every month, 8:00-12:00." The main section is titled "Spawner Options". It contains the following configuration fields:

- Piz Daint node type: gpu (dropdown)
- Queue: JupyterHub dedicated queue (single node only) (dropdown)
- Training course reservation: [empty input field]
- Number of nodes: 1 (dropdown)
- Job duration: 1 hour (dropdown)
- Account (leave empty for default): [empty input field]
- Start IPyParallel automatically with MPI?: No (dropdown)
- If yes, how many processes per node? (default: one process per virtual core): [empty input field]
- Start Dask.distributed automatically?: No (dropdown)
- If yes, how many tasks per node? (default: one task per node): 1 (input field)
- NB: the number of threads = ncores / npprocesses

A large red button at the bottom is labeled "Spawn".

At the bottom of the page, there is a footer with links: "© CSCS 2018 | Imprint & Disclaimer | Terms & Privacy Policy". Below the footer is a search bar with options: "Search", "Highlight All", "Match Case", and "Whole Words".



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETHzürich

Demo of JupyterLab interface and exercises



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETHzürich

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
```

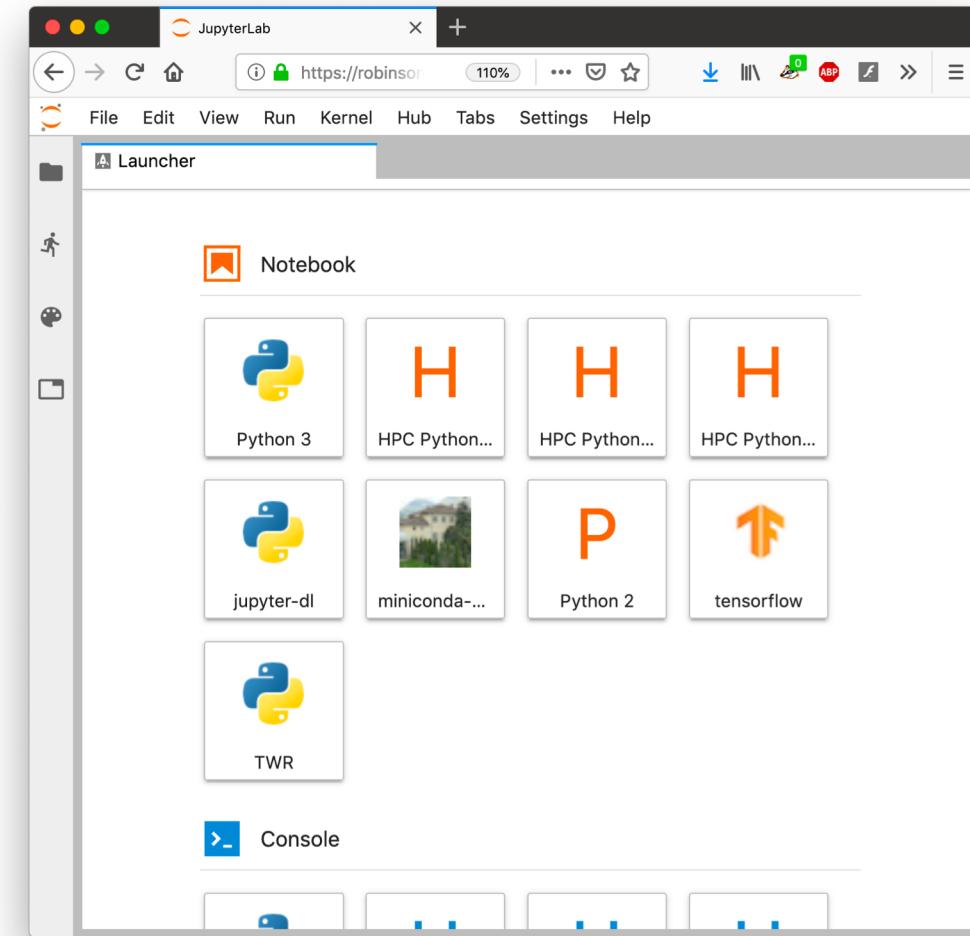
- This has created and installed a “kernel spec” file (`kernel.json`) at the path described in the output

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

ETHzürich

Tutorial on Python for HPC



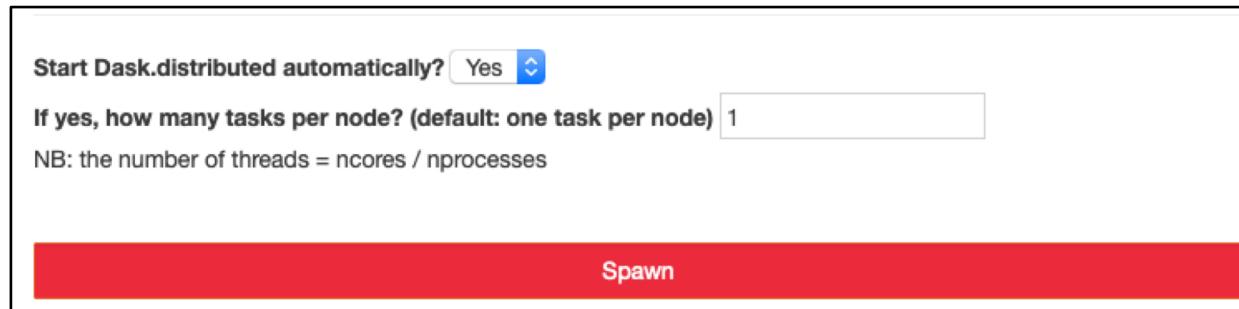
CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETHzürich

Distributed Dask

- 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

