HPC and cloud computing

IBU HPC Infrastructure

Interfaculty Bioinformatics Unit(IBU)

HPC = High Performance Computing

Using large amounts of power

- over a short time(hours)(HPC): weather forecast, genetic diagnostic
- over a long time(months)(High Throughput Computing, HTC): Astrophysics, climate research
- grid computing: Particle Physics at CERN

History

- Cray-1, 1976, 160 MFLOPS | Smartphone, 2013: 1GFLOPS
- IBM BlueGene/P, 2007, 23 TFLOPS, 65’537 CPUs
- Cray, XC50, 2017, 27 PFLOPS, 133’716 CPUs(Piz Daint, CSCS)
- Ubelix, 6300 CPUs
- IBU Cluster, 1888 CPUs
- My Laptop, 8 CPUs

Features

Operating System Operating systems used on top 500 supercomputers(wikipedia): gradually turn from Unix to Linux, very rare K.A./Ver., BSD, Windows, Mac.

Queuing System

- Concurrency on resources(CPUs, RAM) for users and job
- Optimal usage of resources

Storage

- Large capacites
  - 1 Hard Disk: 16TB
  - Piz Dint: 8’000 TB
  - Ubelix: 3’000 TB
  - IBU: 1’000 TB
- High number of files
  - typically: 100’s of millions of files

Network

- Nodes Interconnect
  - Typical: 10-56 Gbit/s
  - Network type: TCP/IP or infiniband
- Outbound connection
  - Typical: 10 GBit/s

Internal Network

- IBU 40GBit/s switch

Chanllenges

- Electrical Power
  - Piz Daint: 3MW
  - IBU: 15kW
  - My Laptop: 60W
  - City of Bern: 114MW
- Cooling
- Data flow
  - IBU Cluster: 1PB Data
  - Uplink: 10 GBit/s (10-50days to transfer)

Services

Rschiny, Sequenceserver, BugFRI, openBIS, Galaxy, Gitlab, Rstudio, IBU Cloud, openProjects, Proxmox VMs

IBU HPC Linux Cluster

- **Head node = entry point**
  - ssh binfservms01.unibe.ch
- Cent OS 7
  - 2 * 6 cores
  - 64 GB RAM
  - 1 TB/home
  - 10 Gbit/s Network uplink

Data Storage

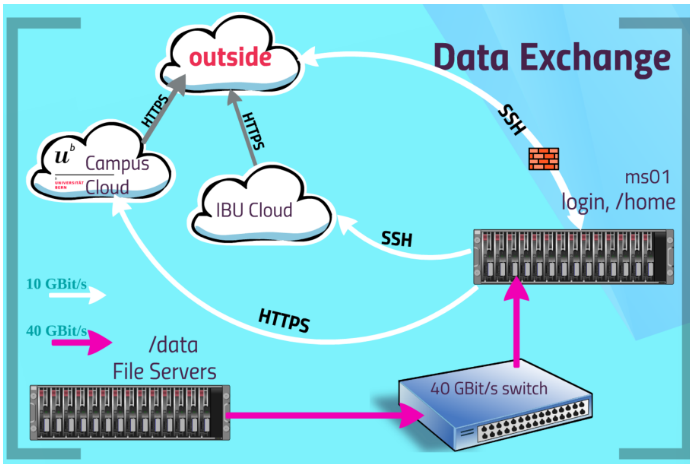
- Total active ~ 400 TB
  - /home/*username* -> /home 1 TB ms01
  - /data/projects/pnnn_abcd -> /data 600 TB fs07
  - /data/users/username -> /data 600 TB fs07
  - /scratch
    - directory local to each node
    - during job excution: `$SCRATCH`
    - /scratch/172007
    - deleted after job completion

Backup

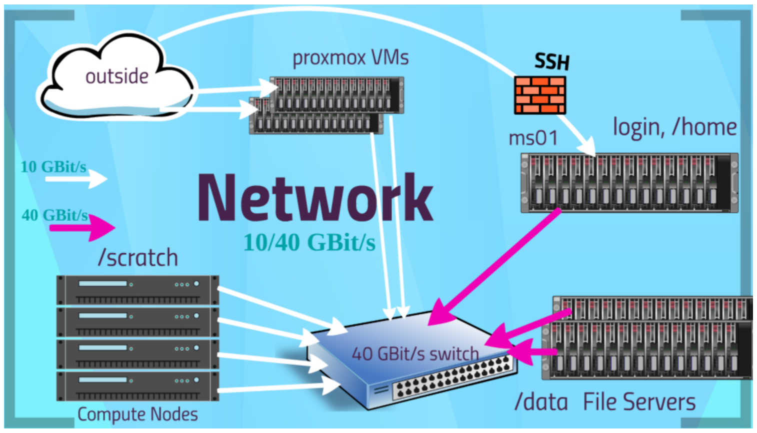
outside <-> von Roll <-> Vetsuisse

- von Roll
  - IBU HPC Cluster
  - Ubelix
  - Research storage
- Vetsuisse
  - Sequencers
  - Ingestion servers
  - Backup server

Data Exchange



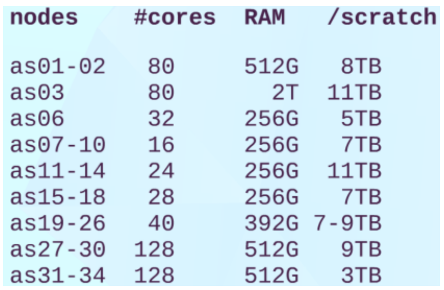
Network



Compute nodes

binfservas[01-34]: 32 servers, 2048 cores

- `clusterstate.sh`



```
nodes     #cores  RAM   /scratch

as01-02    80     512G    8TB
as03       80       2T   11TB
as06       32     256G    5TB
as07-10    16     256G    7TB
as11-14    24     256G   11TB
as15-18    28     256G    7TB
as19-26    40     392G  7-9TB
as27-30   128     512G    9TB
as31-34   128     512G    3TB
```

SSH

Secure channel over an unsercured network

clinet <-> internet <-> server

- confidentiality

- intergrity
- authentication

Cryptography

Symmetric cryptography Goal: establish a secured channel => confidentiality + integrity

Needs a Shared Secret: key => needs a Key Exchange Algorithm

Key Exchange Algorithm Diffe-Hellman

Asymmetric cryptography public/private keys pair User authentication

Server authentication: same principle, reverse sides

SSH Uses

- interactive sessions (shell)
- commands execution on server
- data transfer (scp, sftp)
- port forwarding

Take home

- protect your ssh private key (passphrase)
- use /scratch whenever possible
- beware of small files on /projects, /home
- organize backups
- ibu-best-practices

Introduction to SLURM and modules

1. SLURM: introduction

Limited resources

- Cluster has many users wanting to run jobs, which limits: **1. CPU 2. Working memory 3. Time**
- How to assign which resources to which job?

Job scheduling

- Job(computing) In **computing**, a **job** is a unit of work or unit of execution(that performs said work).
- Job scheduler: A **job scheduler** is a computer application for controlling unattended background program execution of jobs.

SLURM

- Simple Linux Utility for Resource Management
- Job scheduler on: UBELIX, IBU cluster, and many more

Resource allocation commands

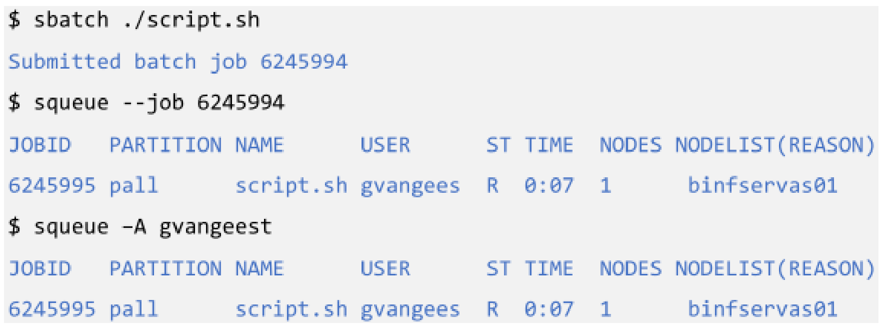
- `sbatch`, `srun`, `salloc`
- **sbatch [options] script**
- `$ sbatch --cpus-per-task=32 --mem-per-cpu=4G ./script.sh`

```
#!/usr/bin/env bash
my_program \
--cpu 32 \
--memory 128G
```

- `$ sbatch ./script.sh`

```
#!/usr/bin/env bash
#SBATCH --cpus-per-task=32
#SBATCH --mem-per-cpu=4G

my_program \
--cpu 32 \
--memory 128G
```

  – 

```
$ sbatch ./script.sh
Submitted batch job 6245994
$ squeue --job 6245994
JOBID   PARTITION NAME      USER      ST TIME  NODES NODELIST(REASON)
6245995 pall      script.sh gvangees  R  0:07  1      binfservas01
$ squeue -A gvangeest
JOBID   PARTITION NAME      USER      ST TIME  NODES NODELIST(REASON)
6245995 pall      script.sh gvangees  R  0:07  1      binfservas01
```

2. Frequently used sbatch options

2.1 Required resources

- CPU `--cpus-per-task=2`
- Working memory `--mem-per-cpu=4G`
- Time (days-hours:minutes:seconds) `--time=1-05:00:00`
- *Low values could cause your job to start earlier* **But**: *job will fail if resources are overrequested!*

2.2 user specific

- Job name: `--job-name=my_job_name`
- e-mail
  - `--mail-user=user@students.unibe.ch`
  - `--mail-type=begin,end,fail`

2.3 output & error

- `--output=existing/path/output_%j.o`
- `--error=existing/path/error_%j.e`
- **Path should exist! Job will fail otherwise (without error message)**

3. Interactive jobs

Why submit interactive job?

- Interactive job: allocated resources that are approachable with shell
- Head (login) node is not for computation
- Debugging and testing can be much more convenient if interactive

srun

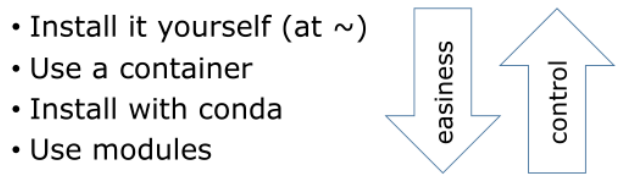
- Versatile command
- Used for job steps within sbatch (not treated in this course)
- Also for allocation of interactive job with pty (pseudo-terminal mode)

```
$ srun --cpus-per-task=1 --mem-per-cpu=4000 \
> --time=00:05:00 --pty bash
```

- Exit the interactive job with **exit**

4. Modules

Software



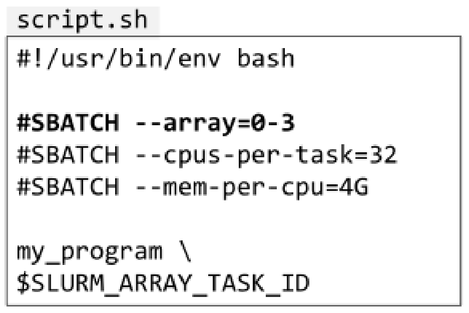
Modules

- Check for available modules: `module avail`
- Add a module to environment: `module add`
- Unload a module: `module rm`
- Available modules: https://www.vital-it.ch/services

5. Job arrays

5.1 Jobs in parallel

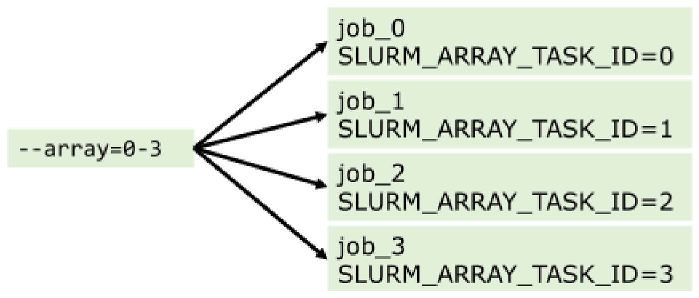
- Run similar command with different parameters: parameter sweep
- E.g. alignment (e.g. with minimap2) on several files



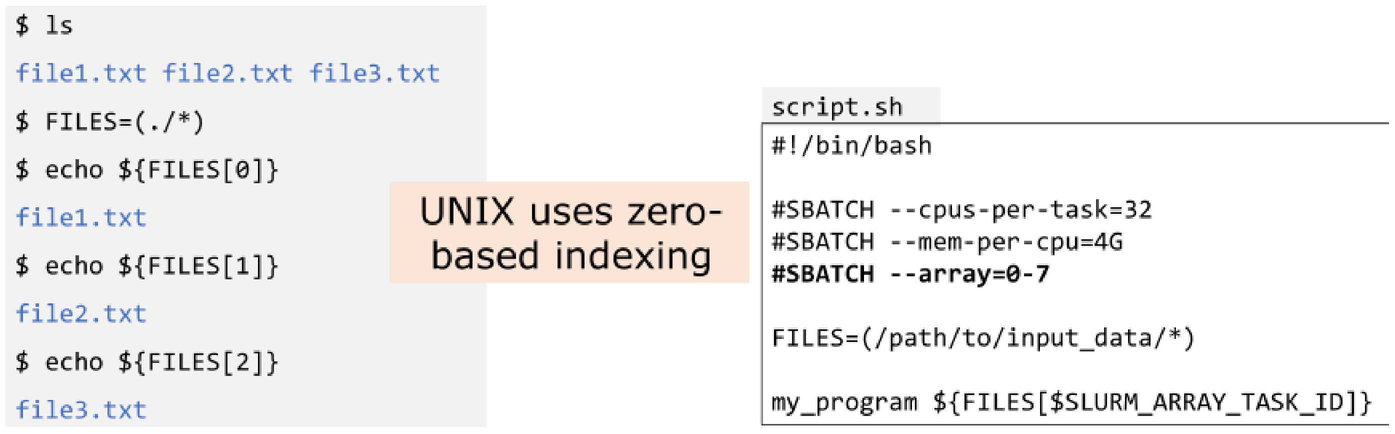
script.sh
```
#!/usr/bin/env bash

#SBATCH --array=0-3
#SBATCH --cpus-per-task=32
#SBATCH --mem-per-cpu=4G

my_program \
$SLURM_ARRAY_TASK_ID
```



5.2 Using UNIX arrays



```
$ ls
file1.txt file2.txt file3.txt
$ FILES=(./*)
$ echo ${FILES[0]}
file1.txt
$ echo ${FILES[1]}
file2.txt
$ echo ${FILES[2]}
file3.txt
```

UNIX uses zero-based indexing

script.sh

```
#!/bin/bash

#SBATCH --cpus-per-task=32
#SBATCH --mem-per-cpu=4G
#SBATCH --array=0-7

FILES=(/path/to/input_data/*)

my_program ${FILES[$SLURM_ARRAY_TASK_ID]}
```

Interactive and reproducible computing with Jupyter and friends

Introduction

Jupyter notebooks are interactive computing documents especially popular in data intensive fields (data science).

By their nature they are a great tool for:

- Easy **design** of analysis workflows
- **Documenting** code / workflows and increasing their reproducibility
- Exploiting **cloud computing** resources

Used in academic research and in companies

Bloomberg, PANGEO is the first and foremost a community promoting open, reproductible and scalable science.

Course content

1. Interactive computing - Jupyter
2. Resources - Run jupyter on HPC/cloud
3. Publish - Github and Zenodo to publish code
4. Reproducible code - Renku/Binder

Why reproducible code?

Levels of reproducibility: minimal

Computations only described. Maybe possible to reconstruct at great pain.

Impossible to verify.

Levels of reproducibility: upon request

Computations only described.

In principle possible to reconstruct.

- The Science Journal policy (must be available)
- Answers one gets from authors (not prefer sharing)

Levels of reproducibility: complete code

Possible to reconstruct.

Impossible to reproduce exactly (e.g. package versions missing)

Levels of reproducibility: reproducible code

Possible to reproduce exactly (via container technology)

The future: eLife example

Jupyter Notebooks

Interactive computing with Jupyter

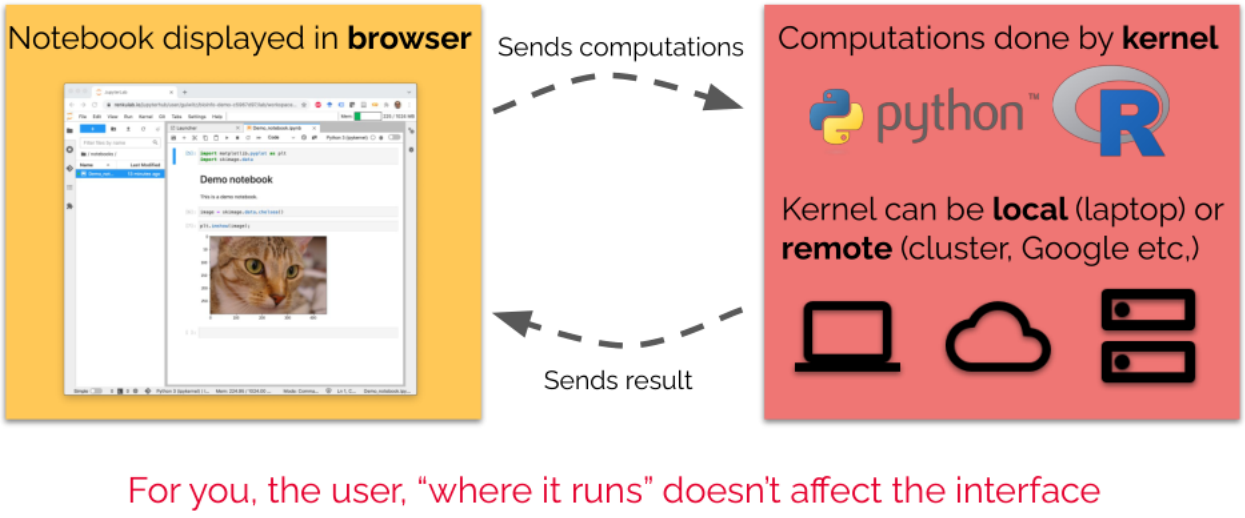
- Write and execute code
- Display images and plots
- Document every step with formatted text (Markdwon)
- Excute code step by step
- Call other software from notebooks (e.g. ilastik)

What is a Jupyter notebook?

A **text** file (easily sent around) Rendered by Jupyter in the **browser** Split into sections called **cells**

Cells can contain: - Code - Formatted text - Rich output

How and where does a notebook compute


Notebook displayed in **browser**
Demo notebook
Sends computations
Sends result
Computations done by **kernel**
python
R
Kernel can be **local** (laptop) or **remote** (cluster, Google etc.)
For you, the user, "where it runs" doesn't affect the interface

Right side kernel could be **binder**, **Colab**, **Renku**

Renku notebook

Notebook cells

- Code broken into chunks: cells
- Variables defined for whole notebook
- Only the order of cell excution matters
- **Good practive: Top-down order**
- The cell type can be switched from Code to Text(Markdown)
- Possible to run all cells are part of a notebook

- **Toolbar** can be used to:
  - Copy/paste cells
  - Add new cells
  - Run a cell or stop execution
  - Change cell type from code to markdown
  - More options in menus

Notebook handling Right-click on notebook in panel to: Rename, Download, Shut down, Copy, Duplicate, etc.

The notebook kernel Notebook content does not depend on kernel.

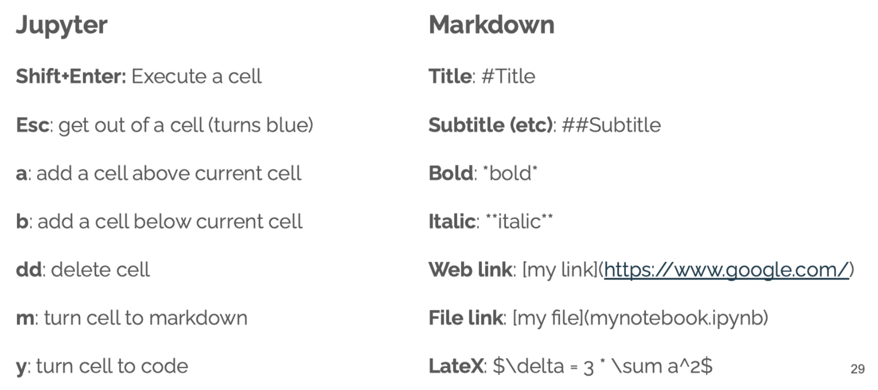
Variables conserved as long as kernel is ON (green dot).

Kernel can be restarted.

- Interrupt long calculation
- Re-initialize variables

Good practice: periodically restart kernel to avoid "*strange*" states

Jupyter cheat sheet



Jupyter	Markdown
Shift+Enter: Execute a cell	**Title**: #Title
Esc: get out of a cell (turns blue)	**Subtitle (etc)**: ##Subtitle
a: add a cell above current cell	**Bold**: *bold*
b: add a cell below current cell	**Italic**: **italic**
dd: delete cell	**Web link**: [my link](https://www.google.com/)
m: turn cell to markdown	**File link**: [my file](mynotebook.ipynb)
y: turn cell to code	**LateX**: $\delta = 3 * \sum a^2$

29

Mixing languages in Jupyter: command line

- Exclamation mark: `!pwd`
- Use "magic" commands:

```
%%bash
cd myfolder
ls
```

Beyond notebooks

- Interactive features with ipywidgets:
- Creating interactive web-apps with voilà:
- Create interactive online books with Jupyter:
- Running a multi-user Jupyter with JupyterHub (e.g. The Littlest JupyterHub)

Other public Jupyter resources

- With switch-AAI login (same as Ilias login): EPFL: https://noto.epfl.ch/ Jupyter running on EPFL servers, fully customizable environments With switch-edu or GitHub:

- Swiss Data Science Center: https://renkulab.io/ Powerful combination of Jupyter, GitHub and data repository
- With Kaggle (ML competition site): https://www.kaggle.com/
- Access to interesting datasets, GPU etc.

Run Jupyter locally: Docker

There are images to run Jupyter e.g.

- To install: `docker pull jupyter/datascience-notebook`
- Or directly install and run:

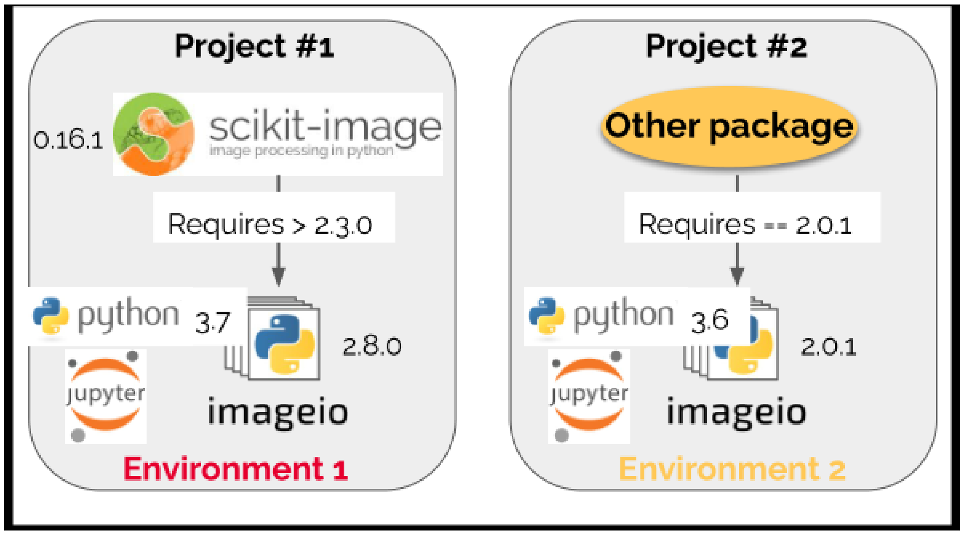
  ```
  docker run -p 8885:8888
  jupyter/datascience-notebook
  ```
- Browse to http://localhost:8885/lab
- Alternatively, after: `docker pull jupyter/datascience-noteboo` run from Docker Desktop and set the port to 8885. Open a command line, and recover token with: `jupyter notebook list` Browse to http://localhost:8885/lab

How do I install Jupyter?

Easiest solution is conda based: * Install Anaconda: access Jupyter via simple click in a user interface * Install miniconda and start from command line with: jupyter lab * Both install a minimal set of tools (Jupyter, Numpy etc.)

Installations: why environments?

- Why not simply open a terminal and run e.g. pip to install all necessary packages? We can enclose each project into an **environment** !
- Each environment contains all necessary tools, including python
- With conda, dependencies are "resolved" by conda



Jupyter on cluster

We recommend installing all necessary components via conda:

1. You can load conda as a module: `module load Conda/miniconda/latest`
2. You can make it easy to use conda by typing: `conda init`

3. Exit the ssh session (e.g. type exit) and ssh-login again. You should see (base) now at the start of the line.
4. Create an environment in which you install jupyter (and matplotlib for demo): `conda create -n myenv jupyterlab matplotlib` And wait…

In principle to run Jupyter, you now need to 1) activate the environment and 2) run Jupyter:

```
conda activate myenv
jupyter lab
```

However:

1. You now need to access Jupyter via ssh and not over regular web
2. You need to run Jupyter within a SLURM Job, and not on the login node

Jupyter on cluster: ssh tunneling

1. Start jupyter like this: `jupyter lab --no-browser --ip=0.0.0.0 --port=8889`
2. Tunnel Jupyter from the cluster port 8889 to you local port 8889: `ssh -N -f -L 8889:binfservms01:8889 your_username@binfservms01.unibe.ch`
3. Open your local browser and go to: `localhost:8889`
4. Enter the token that appeared in the first terminal

Jupyter on cluster: interactive jobs

1. Use srun to start an interactive job as a bash shell: `srun --mem-per-cpu=1G --cpus-per-task=1 --time=01:00:00 --pty bash`

- You should see that your node has changed e.g. to binfservas01

2. Activate your environment and start jupyter:

```
conda activate myenv
jupyter lab --no-browser --ip=0.0.0.0 --port=8889
```

3. Establish again an SSH tunnel but change the compute node! `ssh -N -f -L 8889:binfservas01:8889 your_username@binfservms01.unibe.ch`

Jupyter in the cloud: Google Colab

Google's version of Jupyter

- Same basic principles, different layout
- Kernels run on Google infrastructure for free
- GPUs available
- Opens any notebook on Github
- R is still experimental, create notebook with https://colab.research.google.com/notebook#create=true&language=r

Colab sessions

- Sessions time-out after max 12h
- Data accessible through Google Drive
- Common packages pre-installed
- Additional packages need to be installed in each notebook

Upload your notebook to Colab If you have a Google Account, you can use the service for free. Go here: https://colab.research.google.com Choose Upload. Try to run your notebook.

Example of Colab usage

- **kallisto | bustools** is a workflow for pre-processing single-cell RNA-seq data:
- **ZeroCostDL4Mic** Simplifying usage of deep learning for image processing in biology, Usage of Colab specific features like forms

Jupyter in the "real" cloud: Google Compute Engine

- Same type of offers from *Google Cloud Compute, Amazon EC2, Microsoft Azure, Digital Ocean*
- For Swiss academics, use **Switch Engines**
- Access to "unlimited" resources
- No queuing
- Not free, potentially very expensive (GPU)

Jupyter on Google Compute Engine Simple example script to run on a VM to set-up Jupyter and a few packages and access to it in your browser.

```
## Create a VM on Google Compute engine and add a firewall policy allowing
## for tcp access on port 8887. Check your machine's IP address XXX.XX.XX.XXX
## Execute the following lines after sshing into the machine with Google's
in-browser SSH terminal

## update linux
sudo apt-get update
sudo apt-get upgrade
## install a compiler
sudo apt-get install g++

## install conda
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh

## create a conda environment and install jupyter + dask
conda create -n myenv pip jupyter jupyterlab numpy matplotlib scikit-image

## Now start Jupyter
conda activate myenv
jupyter notebook --no-browser --port=8887 --ip='0.0.0.0'

## Now in your browser you can go to http://XXX.XX.XX.XXX:8887

## To simplify the starting of Jupyter you can edit the configuration file

## Create a jupyter notebook configuration file

#jupyter notebook --generate-config

## Add these lines in the jupyter_config.py file

#c = get_config()
#c.NotebookApp.ip = '0.0.0.0'
#c.NotebookApp.open_browser = False
#c.NotebookApp.port = 8887
```

Code repository and management: GitHub

Sharing notebooks: GitHub

- GitHub is a repository for code based on git, a software to keep track of changes in software
- Projects are "folders" called a repositories
- Public repositories are browsable: https://github.com/guiwitz/hpc_cloud - GitHub is **many** more things than just a repository

Much more with GitHub

- Fork (“copy”) other people’s repositories
- Create software releases
- **GitHub Actions**: Automatically execute workflows on a repository upon events like a push. E.g. run tests on multiple OS’s (see e.g. https://github.com/guiwitz/hpc_cloud)
- **GitHub Pages**: Automatically generate a static website e.g. for documentation.

Sharing files: from static to dynamic

Sharing notebooks statically

- Notebooks are rendered **statically** on GitHub
- **Nbviewer** offers a more reliable rendering

MyBinder

Making notebooks interactive with binder

- Creates a remote Jupyter instance and copies Github repository
- Opens Jupyter in the browser and **works exactly like a local Jupyter**
- Add a repository address e.g. https://github.com/guiwitz/hpc_cloud
- Use a specific commit / branch
- Copy the text for the markdown badge

MyBinder badge To add a badge, copy the Markdown link (previous slide). Something like:

```
[![Binder](https://mybinder.org/badge_logo.svg)](https://mybinder.org/v2/gh/guiwitz/hpc_cloud/main)
```

Add it to your `README.md` file (edit, copy/paste link, commit change).

MyBinder sessions

- Short sessions: stop after a few minutes inactivity
- Download your modified notebook OR Save/load it to/from browser storage
- Does not work with “external” software (e.g. external interactive windows)

Making notebooks interactive with Colab

Add a Badge like for binder. We can add to our README.md something like this:

```
[![Open In Colab](https://colab.research.google.com/assets/colab-badge.svg)](https://colab.
research.google.com/github/guiwitz/hpc_cloud/blob/main/Demo_notebook.ipynb)
```

Notebooks on Renku

Renku combines container, notebook and repository technology

1. Choose environment: Jupyter, RStudio, packages etc.
2. All settings and files are kept and updated in a repository
3. A Docker image for that environment is created and updates at every change
4. The image can be run on Swiss Data Science Center infrastructure

Preserve and cite: Zenode

A repository for multiple data types

- Reports that are not published but should be citable
- Datasets, often related to an article

- Software either "professional" or custom script e.g. accompanying an article

What Zenodo offers

- Security: GitHub can suspend your account without notice. Zenodo is publicly funded and guarantees your "artefacts" are available
- "Zenodo does not impose any requirements on format, size, access restrictions or licence"
- Upload data over time and add an embargo e.g. until a publication
- Offers a DOI, digital object identifier, a unique id that can be used to reference a software, dataset etc. E.g. https://doi.org/10.7554/eLife.49305

Connecting GitHub and Zenodo

To avoid adding "test-repositories" to Zenodo, we use today the Zenodo Sandbox. It's identical to Zenodo but it's content can be purged. Some functionalities are only "for show" and do not work normally.

Connecting GitHub and Zenodo

Select Repositories to "synchronize"

- Select repositories to synchronize
- Upon creating a release, Zeonod copies the repository and assigns a DOI
- You can add a badge to the repository to show how to reference it

Create a GitHub release

Get DOI and add badge

Using GitHub+Zenodo+DOI in real life

Coming full circle: Mybinder + Zenodo DOI Use the Zenodo DOI to start an interactive session directly on MyBinder! The Sandbox DOI does not work. To test this use e.g.: https://zenodo.org/record/3240495#.X5NLFUL7Tlw

UBELIX submit.unibe.ch

What is a Node?

- Individual "computers" that compose a cluster
  - Similar components
- Different type of nodes
  - Login
  - CPU compute
  - GPU compute
  - Service nodes

Login Nodes (submitXX)

Which commands are tasks for login nodes:

1. ~~python physics_sim.py~~
2. make
3. create_directories.sh
4. ~~molecular_dynamics_2~~
5. tar -xzf R-3.3.0.tar.gz

! no computation on login nodes

UBELIX software

Module environment

- multiple versions of various packages
- Preventing unintended interference
- managed via environment variables
- `$ module avail`
  - List available modules
- `$ module load <module>`
  - Load specific module
- `$ module list`
  - List loaded modules
- `$ module purge`
  - Unload all modules

BioInf Software

Vital-IT provides software for bioinformatics

- `$ module load vital-it`
- `$ module avail`

SLURM

- SLURM kills jobs, which exceed
  - memory or time limits!
  - Smaller and shorter jobs shedule faster
  - Appropriate values must be determined iteratively:

1. Make an elaborate guess for the first run
2. Check runtime and memory usage (mail, sacct)
3. Adapt values for future invocations of the same job.

- To be on the save side, add a (few hours) time and (2G) memory

Q: Where is the Output?

- Default: slurm-.out
- stderr and stdout can be split and file names be specified

Threads vs Tasks

- Threads: (***OpenMP***)
  - Single executable instance
  - Spawn and merge threads
  - **Shared memory**
  - `#SBATCH --cpus-per-task=<nr>`
- Tasks: (*MPI*)
  - Multiple executable instances
  - Communicate with each other
  - Distributed memory/multiple nodes
  - `#SBATCH --ntasks=<nr>`

Transferring Files to/from the cluster

- wget - Download files from the Internet
- Transfer files/folders from/to local desktop/laptop

```
# best practice to pack and compress first:
$ tar -zcf <tar_file> <data>
# scp [option] source destination
[user@local ~] $ scp -r myproject/ <user>@submit.unibe.ch:~/
[user@local ~] $ scp <user>@submit.unibe.ch:~/myproject/result.txt .
```

- `$ rsync -avz <source> <destination>`
- sftp client (GUI), e.g. FileZilla, MobaXterm, …
- Different line endings
  - UNIX: `\n` (newline)
  - Windows: `\r\n` (carriage return + newline)
- This might cause a problem!
- Convert Windows text files to UNIX format
  - `$ dos2unix <file>`
- Convert UNIX text files to Windows format
  - `$ unix2dos <file>`