Using Jupyter

- Why not IPython?
- Installing IPython and Jupyter
  - Load a compiler and python using modules
  - Create a virtual environment
  - Activate the virtual environment
  - Install Jupyter and ipyparallel
  - MPI4PY
- Running IPython
- Running Jupyter
  - Customize sbatch options

Jupyter is not an HPC application and no support beyond this documentation will be provided by SCITAS.

The Jupyter documentation can be found at [http://jupyter.org/documentation](http://jupyter.org/documentation)

Why not IPython?

IPython is the interactive python kernel used by Jupyter: [https://ipython.org](https://ipython.org)

Using IPython without Jupyter greatly simplifies the launch process and is much more suitable for use on a HPC cluster.

Installing IPython and Jupyter

The key to successfully installing the tools is using python virtual environments. Please note that you will need to install for each architecture (the $SYS_TYPE variable used below).

Load a compiler and python using modules

```bash
$ module load gcc
$ module load python
```

or

```bash
$ module load intel
$ module load python
```

Create a virtual environment

Shown here for GCC but the same applies for the Intel compiler.

```bash
$ virtualenv --system-site-packages opt/$SYS_TYPE/venv-gcc
```

Using base prefix '/ssoft/spack/cornalin/v1/opt/spack/linux-rhel7-x86_E5v4_Mellanox/gcc-5.4.0/python-3.6.1-moc7d7wihdugzp5fi4ew7za5muzwvrn6'

New python executable in /home/user/opt/x86_E5v4_Mellanox/venv-gcc/bin/python3.6
Also creating executable in /home/user/opt/x86_E5v4_Mellanox/venv-gcc/bin/python
Installing setuptools, pip, wheel...done.

Activate the virtual environment

```bash
$ source opt/$SYS_TYPE/venv-gcc/bin/activate
```

(venv-gcc) [user@fidis ~]$
Install Jupyter and ipyparallel

```bash
(venv-gcc) [user@fidis ~]$ pip install jupyter ipyparallel
Collecting jupyter
  ..
  ..
```

MPI4PY

At this point you may wish to install MPI4PY - see the documentation for how to do this.

Running IPython

The easiest way to use IPython is via the Sinteract tool which gives interactive access to a compute node:

```bash
[user@fidis ~]$ /usr/bin/Sinteract -p debug -n 1 -c 28 -t 01:00:00 -m 120G
Cores: 28
Tasks: 1
Time: 01:00:00
Memory: 120G
Partition: debug
Account: mylab
Jobname: interact
Resource:
QOS: normal
Reservation:
salloc: Granted job allocation 439044
srun: Job step created

[user@f210 ~]$ source opt/$SYS_TYPE/venv-gcc/bin/activate
(venv-gcc) [user@f210 ~]$ ipython
Python 3.6.1 (default, Jul 4 2017, 17:46:54)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.
In [1]:
```

You now have all the power of IPython on a dedicated compute node.

Running Jupyter

Running Jupyter is a lot more complicated as we need to launch the server on the login node but ensure that the computational tasks are executed on a compute node. This is why we installed ipyparallel alongside Jupyter.

Here we show how to launch multiple tasks in parallel - it is an HPC cluster after all!

After loading the virtual environment we use ipcluster to start worker engines on the node(s)

```bash
ipcluster start --init --profile=default --ip=*** -n=<ntasks> --engines=Slurm --SlurmEngineSetLauncher.
timelimit=<timelimit> --SlurmEngineSetLauncher.queue=<partition> --SlurmEngineSetLauncher.account=<account>
```

Note:
To find your account, type the following command:
sacctmgr -Pn show assoc where user=$USER format=account

Make sure that you choose meaningful values for ntasks, timelimit, partition and account

Behind the scenes ipcluster submits a batch job to SLURM so the usual caveats about batch jobs apply. The Job ID is given when you launch so the normal squeue/scontrol commands can be used to see what's going on.

```
(venv-gcc) [user@fidis ~]$ ipcluster start --init --profile=default --ip="*" --engines=Slurm --SlurmEngineSetLauncher.timelimit=01:00:00 --SlurmEngineSetLauncher.queue=debug --SlurmEngineSetLauncher.account=myaccount &
[1] 11222
2018-03-26 12:24:34.206 [IPClusterStart] Starting ipcluster with [daemon=False]
2018-03-26 12:24:34.207 [IPClusterStart] Starting Controller with LocalControllerLauncher
2018-03-26 12:24:35.212 [IPClusterStart] Starting 4 Engines with Slurm
2018-03-26 12:25:05.226 [IPClusterStart] Engines appear to have started successfully
```

(venv-gcc) [user@fidis ~]$

We now launch jupyter on the login node:

```
jupyter notebook --ip="$(hostname -s).epfl.ch"
```

this will run a jupyter notebook and provide you with an url that you can open on your local machine

```
(venv-gcc) [user@fidis ~]$ jupyter notebook --ip="$(hostname -s).epfl.ch"
[I 12:27:43.105 NotebookApp] Writing notebook server cookie secret to /run/user/141633/jupyter/notebook_cookie_secret
[I 12:27:43.722 NotebookApp] Serving notebooks from local directory: /home/user
[I 12:27:43.722 NotebookApp] 0 active kernels
[I 12:27:43.722 NotebookApp] The Jupyter Notebook is running at:
[I 12:27:43.722 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 12:27:43.723 NotebookApp]
Copy/paste this URL into your browser when you connect for the first time, to login with a token:
http://fidis.epfl.ch:8888/?token=9b1a0819019e55a8d3fce78d1e8467fefc9a7bb20389ed1d
```

Create a notebook on your local machine by opening the URL and try:

```
import ipyparallel as ipp
c = ipp.Client(profile="default")
```

This will connect to the engines on the node(s). It may fail if the engines are not spawned yet.

```
%%px
import os
import platform
print("{0} on {1}".format(os.getpid(), platform.node()))
```

The %%px magic expresses that the full cell in the notebook should be run in parallel.
Once you have finished don't forget to stop the ipcluster server!

```bash
(venv-gcc) [user@fidis ~]$ ipcluster stop
```

Now you have a working parallel setup it's possible to make it a bit more user friendly

The file `~/.ipython/profile_default/ipcluster_config.py` that contains most of the configurations was created - you can edit it and add the options passed to the ipcluster command.

```bash
jupyter nbextension enable --user --py ipyparallel
```

If you rerun jupyter you should see a IPython Clusters tab and in there a line with your default profile. You can choose the number of cores to use

**Customize sbatch options**

If you need to add options to sbatch like memory constraints, you can create a sbatch template that you provide to the ipcluster command.

Create a `sbatch_jupyter.sh` file with additional `#SBATCH --arguments`

```bash
#!/bin/sh
#SBATCH --job-name=ipy-controller-{cluster_id}
#SBATCH --ntasks=1
#SBATCH --mem 8G
%s --profile-dir="{profile_dir}" --cluster-id="{cluster_id}"
```

Run the ipcluster command with the `--SlurmEngineSetLauncher.batch_template_file` argument
ipcluster start --init --profile=default --ip=*** -n=<ntasks> \
  --engines=Slurm \
  --SlurmEngineSetLauncher.timelimit=<timelimit> \
  --SlurmEngineSetLauncher.queue=<partition> \
  --SlurmEngineSetLauncher.account=<account> \
  --SlurmEngineSetLauncher.batch_template_file=./sbatch_jupyter.sh