JupyterLab with C++ on the curnagl cluster

JupyterLab can be run on the curnagl cluster for testing purposes, only as an intermediate step in the porting of applications from regular workstations to curnagl.

This tutorial intends to setup JupyterLab on the cluster together with the support for the C++ programming language, through the xeus-cling kernel. Besides the IPyKernel kernel for the python language, which is natively supported, we will also provide the option to install support for the following kernels: IRKernel (**R**), IJulia (**julia**), MATLAB kernel (**matlab**), IOctave (**octave**), stata_kernel (**stata**) and sas_kernel (**sas**).

These instructions are hence related to the JupyterLab on the curnagl cluster tutorial, but the implementation is very different because a JIT compiler is necessary in order to interactively process C++ code. Instead of using a python virtual environment in order to isolate and install JupyterLab, the kernels and the corresponding dependencies, we use micromamba.

Setup of the micromamba virtual environment

First create/choose a folder ${WORK} under the **/scratch** or the **/work** filesystems under your project (ex. WORK=*/work/FAC/.../my_project*). The following needs to be run only once on the cluster (preferably on an interactive computing node):

```
module load gcc python
export MAMBA_ROOT=/dcsrsoft/spack/external/micromamba
export MAMBA_ROOT_PREFIX="${WORK}/micromamba"
eval "$(${MAMBA_ROOT}/micromamba shell hook --shell=bash)"
micromamba create -y --prefix ${WORK}/jlab_menv python==3.9.13 jupyterlab ipykernel numpy
matplotlib xeus-cling -c conda-forge
```

The IPyKernel and the xeus-cling kernel for handling C++ are now available. The other kernels need to be installed according to your needs.

Installing the optional kernels

Each time you start a new session on the cluster, remember to define the variable ${WORK} according to the path you chose when creating the virtual environment.

IRKernel

```
module load gcc r
export R_LIBS_USER=${WORK}/jlab_menv/lib/Rlibs
mkdir ${R_LIBS_USER}
echo "install.packages('IRkernel', repos='https://stat.ethz.ch/CRAN/',
lib=Sys.getenv('R_LIBS_USER'))" | R --no-save
export MAMBA_ROOT=/dcsrsoft/spack/external/micromamba
export MAMBA_ROOT_PREFIX="${WORK}/micromamba"
eval "$(${MAMBA_ROOT}/micromamba shell hook --shell=bash)"
echo "IRkernel::installspec()" | micromamba run --prefix ${WORK}/jlab_menv R --no-save
```

IJulia

```
module load gcc julia
export JULIA_DEPOT_PATH=${WORK}/jlab_menv/lib/Jlibs
julia -e 'using Pkg; Pkg.add("IJulia")'
```

MATLAB kernel

```
${WORK}/jlab_menv/bin/pip install matlab_kernel matlabengine==9.11.19
```

IOctave

```
${WORK}/jlab_menv/bin/pip install octave_kernel
echo "c.OctaveKernel.plot_settings = dict(backend='gnuplot')" >
~/.jupyter/octave_kernel_config.py
```

stata_kernel

```
module load stata-se
${WORK}/jlab_menv/bin/pip install stata_kernel
${WORK}/jlab_menv/bin/python -m stata_kernel.install
sed -i "s/^stata_path = None/stata_path = $(echo ${STATA_SE_ROOT} | sed 's/\//\\\//g')\/stata-
```

```
se/" ~/.stata_kernel.conf
sed -i 's/stata_path = \(.*\)stata-mp/stata_path = \1stata-se/' ~/.stata_kernel.conf
```

sas_kernel

```
module load sas
${WORK}/jlab_menv/bin/pip install sas_kernel
sed -i "s/'\/opt\/sasinside\/SASHome/'$(echo ${SAS_ROOT} | sed 's/\//\\\//g')/g"
${WORK}/jlab_venv/lib64/python3.9/site-packages/saspy/sascfg.py
```

Running JupyterLab

Before running JupyterLab, you need to start an interactive session!

```
Sinteractive
```

Take note of the name of the running node, that you will later need. On curnagl, you can type:

```
hostname
```

If you didn't install all of the kernels, the corresponding lines should be ignored in the commands below. **The execution order is important, in the sense that loading the gcc module should always be done before activating virtual environments.**

```
# Load python and setup the environment for micromamba to work
module load gcc python
export MAMBA_ROOT=/dcsrsoft/spack/external/micromamba
export MAMBA_ROOT_PREFIX="${WORK}/micromamba"
eval "$(${MAMBA_ROOT}/micromamba shell hook --shell=bash)"

# IOctave (optional)
module load octave gnuplot

# IRKernel (optional)
export R_LIBS_USER=${WORK}/jlab_menv/lib/Rlibs

# IJulia (optional)
export JULIA_DEPOT_PATH=${WORK}/jlab_menv/lib/Jlibs

# Launch JupyterLab (on the shell a link that can be copied on the browser will appear)
```

```
cd ${WORK}
micromamba run --prefix ${WORK}/jlab_menv jupyter-lab
```

Before you can copy and paste the link into your favorite browser, you will need to establish an SSH tunnel to the interactive node. From a UNIX-like workstation, you can establish the SSH tunnel to the curnagl node with the following command (replace <username> with your user name, and <hostname> with the name of the node you obtained above, and the <port> number is obtained from the link, it is typically 8888):

```
ssh -n -N -J <username>@curnagl.dcsr.unil.ch -L <port>:localhost:<port> <hostname>
```

You will be prompted for your password. When you have finished, you can close the tunnel with Ctrl-C.

Note on Python/R/Julia modules and packages

The modules you install manually from JupyterLab in Python, R or Julia end up inside the JupyterLab virtual environment (${WORK}/jlab_menv). They are hence isolated and independent from your Python/R/Julia instances outside of the virtual environment.

Créé 2023-02-27 18:07:18 UTC par Flavio Calvo
Mis à jour 2023-05-23 16:02:31 UTC par Flavio Calvo