`jupyter.kwarc.info` Documentation

Jonas Betzendahl
`jonas.betzendahl@fau.de`

2022-12-09

1 Setup

1.1 Server

The server everything hereafter is based on is a virtual machine (with 4GB of RAM and around 50GB of disk space for a course of roughly 30-50 students) running Debian Linux 10 (code name *Buster*). The VM runs on university hardware and is monitored and maintained (with regular, scheduled backups) by the sysadmin of our work group. This includes automated email notifications should the Server run low on disk space (which so far has not happened) or RAM (which happens very rarely, usually in the week leading up to exams).

The Debian server runs an Apache webserver wich directs all traffic toward the JupyterHub Server (see Section 1.3). For TLS, we use certificates from "Let's Encrypt!" [LE16] which turned out to be fairly hassle-free to setup.

1.2 Python

In our course, we aim to teach current and relevant technologies, so we decided to use Python 3.8 as the canonical version that everyone would use. Since version 3.8 is not available in the standard repositories we had to build this from sources[1].

It proofed important nevertheless to teach the students about the difference between Python 2 and Python 3 since the normal `python` comman on Debian Buster will still give a Python 2 version and hence they had to make sure to use the `python3` command when outside a notebook environment.

There are several libraries that students would need over the course of IWGS, first and foremost the webserver library `bottle` and the image manipulation library `Pillow`. We decided against *pre-installing* these for everyone (although that is easily doable) so that students would eventually have to *install* them (locally, only for their account) and get at least a passing familiarity with `pip3` and the process of library management.

Upon request from the students, we also decided to *install* (this time for everyone) some standard industry libraries (such as `SciPy`, `pandas`, ...) for students to "play around" with after the more curious ones invariably come into contact with them during their studies.

1.3 Jupyter

We use `jupyterLab` and `jupyterHub` (which are both part of project Jupyter [Jup]), largely as they come out of the box (though there are numerous extensions available). A full explanation of the intricacies of these softwares and how they play together is beyond the scope of this document, so we assume the reader to be familiar with the basics[2].

`jupyterHub` needs a software component for authenticating users plugged in. There are multiple so-called authenticators available for different use-cases and setups[3]. We dediced to use `nativeAuthenticator` [NA] since that allows a lot of control with little infrastucture.

Theoretically, any user can sign up via `nativeAuthenticator` after the server goes online. To avoid flooding we decided to forego open signup and have every user be confirmed by an administrator (which is the default, see Figure 1). The administrator must also create a user account on the Debian machine by the same name. All files the user creates will be stored in that account's home directory[4].

[1] Which we did using these instructions: `https://linuxize.com/post/how-to-install-python-3-8-on-debian-10/`

[2] If you need a refresher: `https://jupyterhub.readthedocs.io/slides/stable/index.html`

[3] See a fairly complete list at `https://github.com/jupyterhub/jupyterhub/wiki/Authenticators`

[4] It is wise to spend some thought on UNIX groups and default access rights so one student can't just copy this week's homework from their fellow student's directory

`jupyterLab` gives access to Jupyter Notebooks as well as Python REPLs and a *shell* on the underlying Debian machine (in case students need to start a webserver or another standalone application, which is diffcult from inside a notebook). We found this quite sufficient, but it can be extended to other languages (even compiled ones like `Haskell` and `C++`).

1.4 Exercise Workflow

During lectures, the weekly routine would be that a tutor develops a Jupyter notebook (which easily mixes markdown and executable code segments, which we found to be both intuitive and helpful since documentation is not as far removed from the place where the work happens) with exercises for the students to complete.

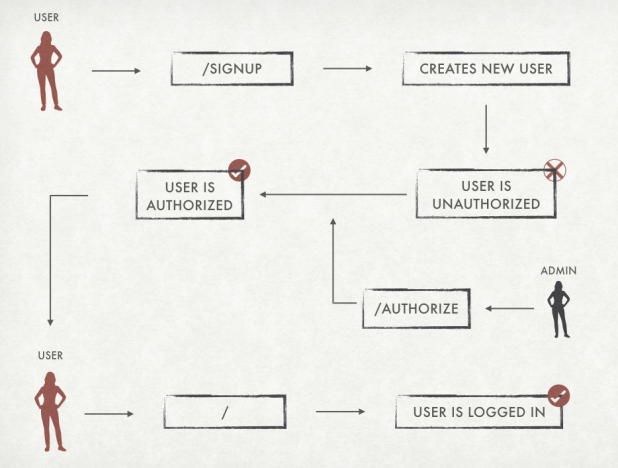

Figure 1: The normal signup flow for registering a new user with `nativeAuthenticator`

These notebooks (and potentially also the example solutions to last week's exercises) would then be placed in a directory on the account of the tutor. This directory would be synchronised to all other accounts at the appropriate time via *shell script* (which can be found in Appendix A) so that the students would find the latest exercises already available to them when logging in to Jupyter.

2 Known Issues

2.1 Crashes

For as-of-yet unknown reasons we observed that on our setup the `jupyterHub` server tends to crash roughly once a day (though without a lot of predictability; sometimes it would run for a week, sometimes only for an hour). This means all Jupyter Services were unreachable for users until the server was manually restarted.

To remedy this situation and allow for a semblance of stability, we added a `cron` job for the superuser that runs a *shell script* every minute. The full script can be found in Appendix B. This has as a consequence that users might momentarily loose connection and need to refresh their session, which can be inconvenient. We hope to find the cause and a cure for this problem in the future.

3 Further Explored Ideas

We were at one point considering to switch off `nativeAuthenticator` in favour of the university's Single-Sign-on (SSO) mechanism, to reduce the number of accounts new students had to sign up for and keep track of.

However, this idea was abandoned for multiple reasons. Integrating with the SSO would mean non-trivial engineering work at the interface ob JupyterHub and the university's infrastructure, since no such systems already exist.

It would also make it possible to associate the data of every account on our Jupyter server with a concrete person (in a way in which the current system does not, since we allow and encourage pseudonyminous account names), which raises GDPR compliance concerns.

Lastly, we plan on removing one year's students' accounts and data befor the next batch signs up. A SSO connection, however, would make the accounts permanent even though the students finished the course, leading to a buildup of abandoned accounts over the years.

References

[Jup] *Project Jupyter*. URL: `http://www.jupyter.org` (visited on 08/28/2020).

[LE16] Internet Security Research Group. *Let's Encrypt*. 2016. URL: `https://letsencrypt.org/`.

[NA] Leticia Portella. *NativeAuthenticator for JupyterHub*. URL: `https://github.com/jupyterhub/nativeauthenticator`.

A Shell script: sync_ex.sh

```
#!/bin/bash
exsource="/path/to/your/source/directory/here"
echo "Removing .ipynb_checkpoints"
find $exsource -type d -exec rm -rf {}/.ipynb_checkpoints \;
while read theuser
do
    # Do not sync to source
    if [ $theuser != "youradmin" ] ; then
        echo "Synching exercises for $theuser"
        rsync -h -v -r -og --chown=$theuser:all_user
              --ignore-existing $exsource /home/$theuser
        echo "----------"
    fi
done < /root/resources/accounts.txt
```

B Shell script: jptrhb.sh

```
#!/bin/bash
# We want this to be executable via cron, so we have to make sure
# the environment is there.
. /etc/profile
. /root/.bashrc
URL="https://jupyter.kwarc.info"
# Wellness-check
curl -L -I -X GET $URL 2>/dev/null 1>/root/logs/http.log
WELLNESS=""
# Checks if file is empty. If it is, server is probably down.
if [ -s /root/logs/http.log ]; then
while read line; do
if [[ $line == "HTTP/1.1 5"* ]]; then
# If any line in the curl response indicates a 5xx respond code,
# we assume server died.
WELLNESS="HTTP response code of 500"
fi
done </root/logs/http.log
else
WELLNESS="Server down"
fi

if [[ $WELLNESS == "" ]]; then
echo "Everything seems fine to me."
else
echo "There was something wrong!"
echo "Restarting JupyterHub and logging timestamp."

# Delete old logs.
find /root/logs -mtime +5 -type f -delete

# Kill current jupyterhub
pkill -f jupyterhub

# Restart with new log

# used to have a --debug after the config, if it misbehaves again,
# consider putting back in (will make logs quite large).
jupyterhub -f /etc/jupyterhub/jupyterhub_config.py --debug 2>1| tee -a /root/logs/jupyterhub-(date-Im).logLogtimestampofrestartecho(date -Im)":  "WELLNESS >> /root/logs/restarts.logfi
```

References

[Jup] *Project Jupyter*. URL: http://www.jupyter.org (visited on 08/28/2020).

[LE16] Internet Security Research Group. *Let's Encrypt*. 2016. URL: https://letsencrypt.org/.

[NA] Leticia Portella. *NativeAuthenticator for JupyterHub*. URL: https://github.com/jupyterhub/nativeauthenticator.