# SoC Renku Tutorial for Researchers v5

The School of Computing Renku platform has been upgraded to Renku v2.0.

Renku is a scalable open-source framework that facilitates collaborative, reproducible, and reusable data science - skills essential for success in today's professional environments. The platform provides an easy to use interactive environment with integrated version control, and storage capabilities. Renku fosters critical skills for researchers.

**A complete Renku tutorial is available here:**
https://renku.notion.site/Documentation-db396cfc9a664cd2b161e4c6068a5ec9

## 1. Login

Go to soc-gpu.computing.dcu.ie

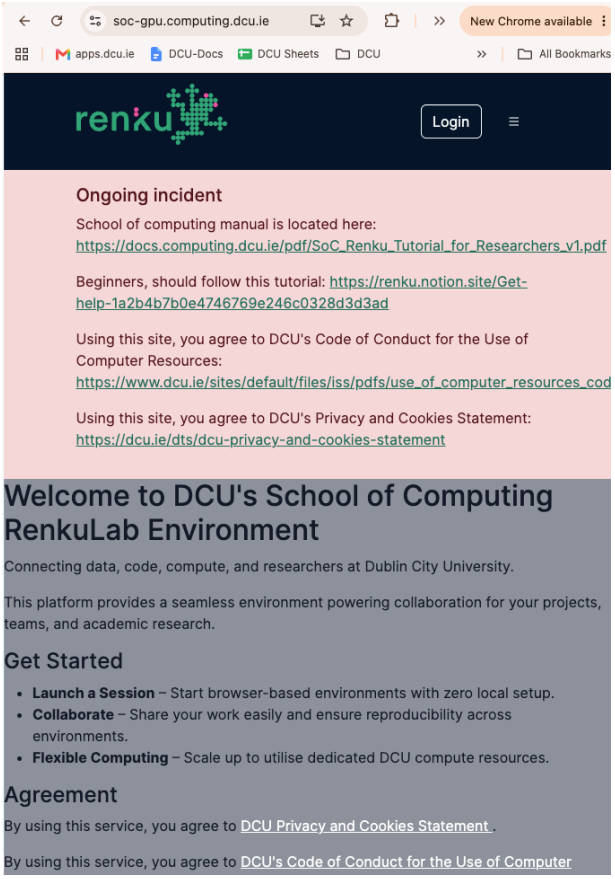



Figure 1: soc-gpu.computing.dcu.ie website

Click the Login option and register.

# 2. Create a project, Select a Template, and launch Jupyter Labs Environment.

After successfully logging in and connecting to GitLab, you will see a screen similar to Figure 2. Click the “+” to create a new Project

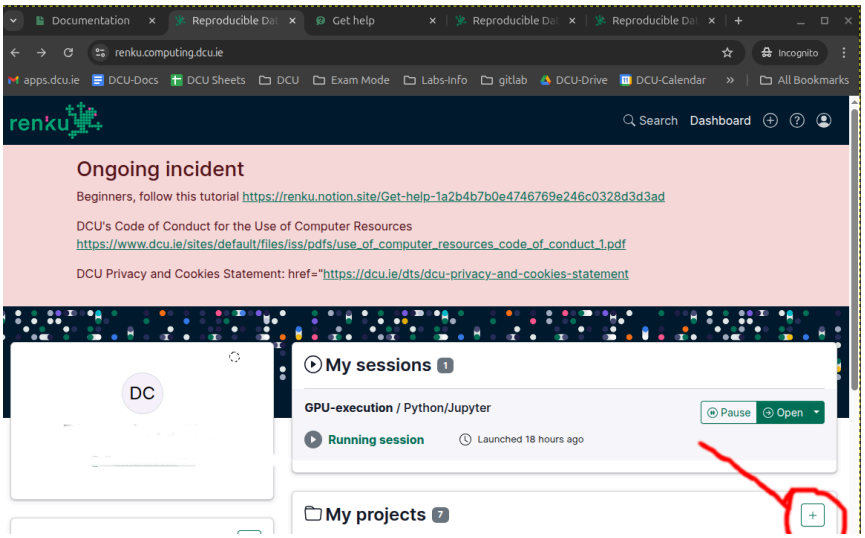



Figure 2: Successfully logged in

Choose the project visibility as "*Private*".

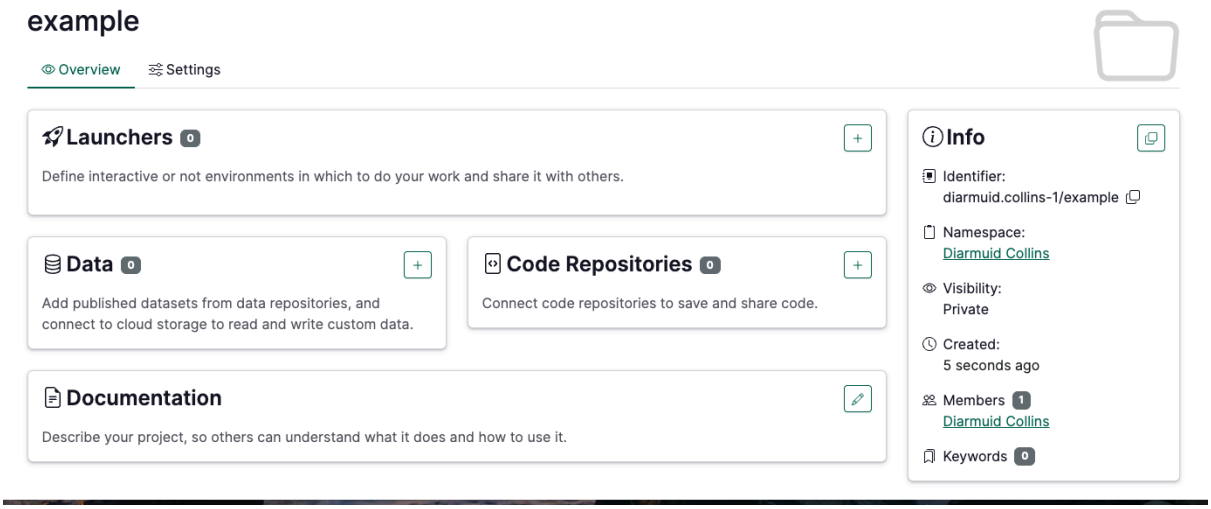



Figure 3: New Project Screen

# 3. How to connect your Renku account to your School of Computing GitLab account

1. In the User menu in the top right corner, click on Integrations.

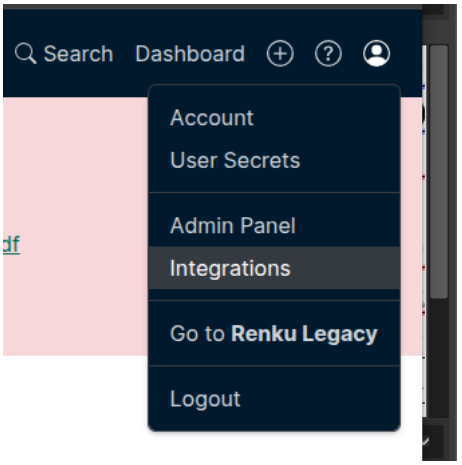



Figure 4: Integrations Menu Option

2. For the desired services, click on **Connect.**
    a. Approve the steps. After that, the service status will turn into connected.

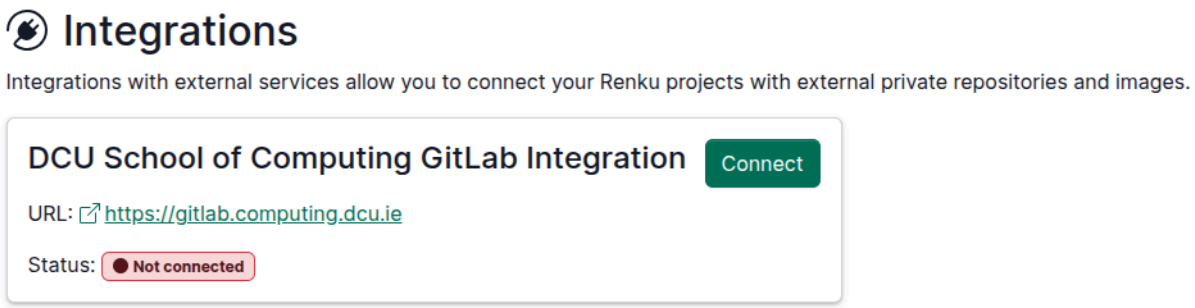



Figure 5: Click “Connect” to connect to the DCU SoC GitLab instance

3. Follow the prompts to install the Renku app in the namespace(s) where you want to use repositories in Renku.

Now you can [add code repositories to your projects] and read & write them in sessions!

For more information, follow these instructions:
https://renku.notion.site/Renku-How-To-Guides-900f417fc205439789a9fbdc5cadcec8?p=3d394998d82b44b4a053e9461949119a&pm=s

Back on the soc-gpu.computing.dcu.ie site, select the project, and add the HTTPS code repo created

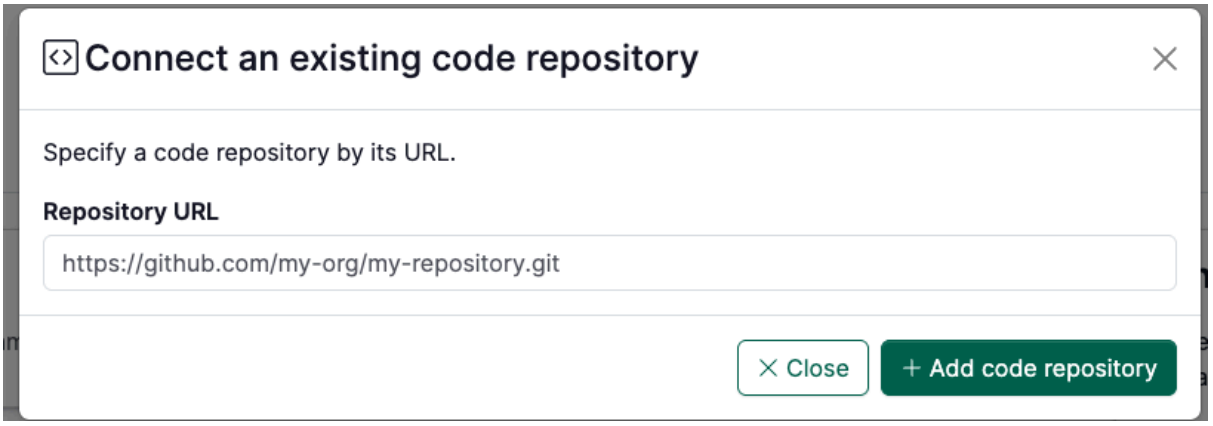



Figure 6: Code Repo HTTPS URL

It should connect easily, as shown in Figure 7

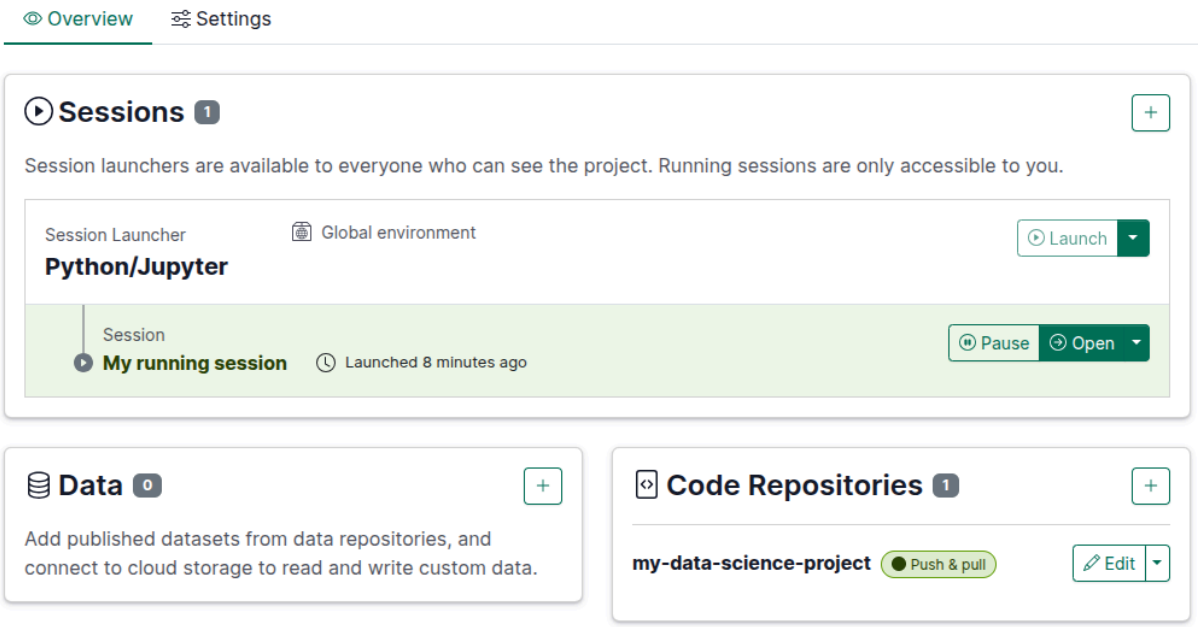



Figure 7: GitLab Code Repo Connected

## 4. Define and Launch a container session

Next we explain how to define a container session, and launch it.

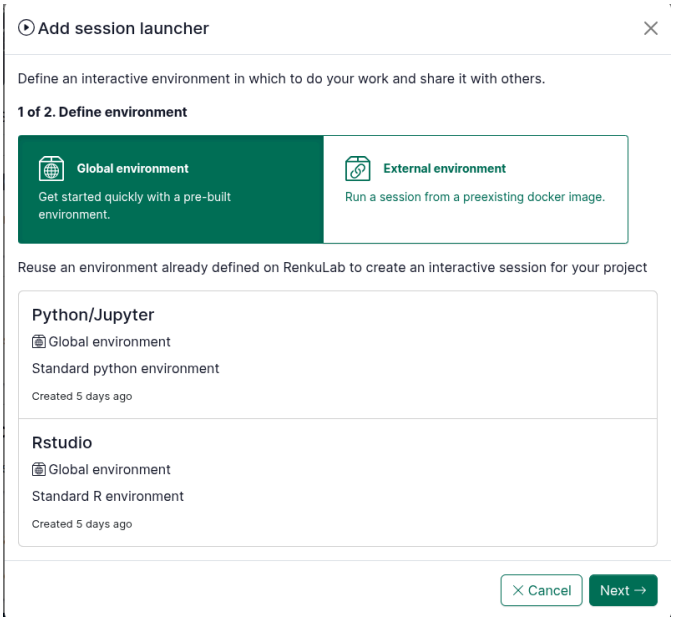



Figure 8: Add a Session Launcher

Define the project information. Select Python/Jupyter or Rstudio. Then click Next.

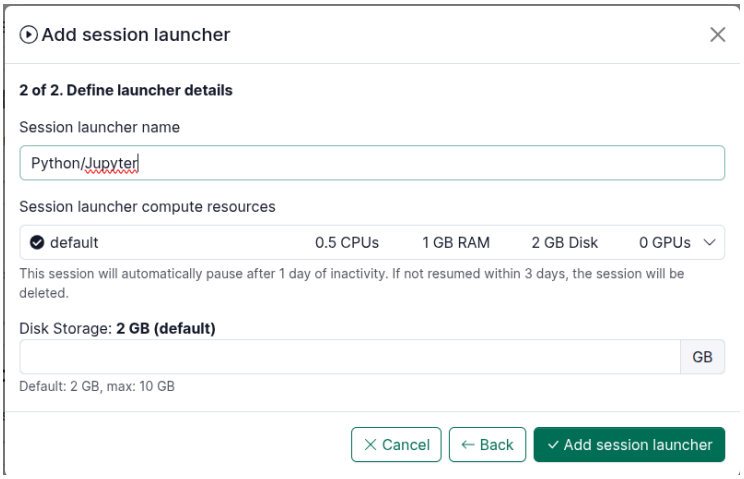



Figure 9: *"Add Session Launcher"* allows researcher to select session flavors

Default settings look like Figure 9. These settings are configurable by the administrator.

Click "*Add Session Launcher*"

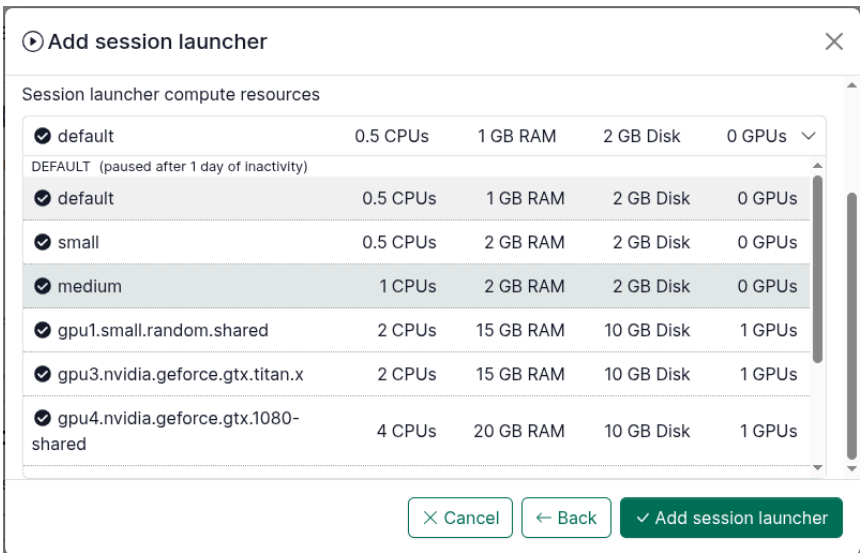



Figure 10: Default Session.

There are additional session (or flavor) options that can be assigned to projects, as shown in Figure 10. Users may have access to different sessions based on the Groups they belong to. Specific resources, such as a dedicated set of GPUs, can be allocated to different groups.

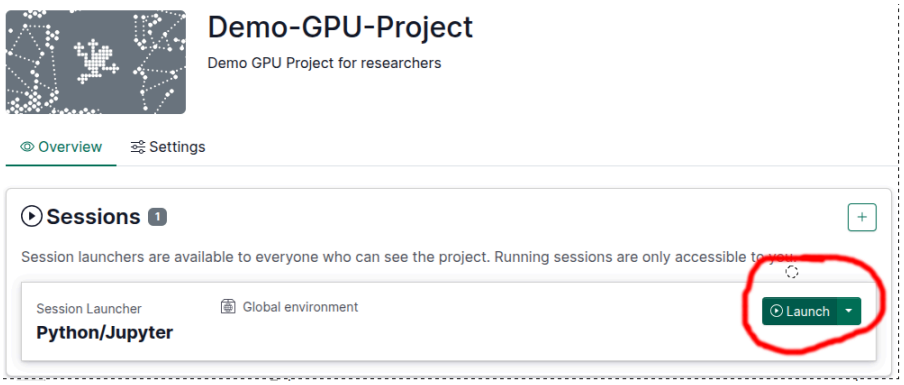



Figure 11: Next Select "*Launch*"

Assuming the docker image is available, click on ***Launch***, see Figure 11. If the container is not available, then you will see the following window in Figure 12, and it might take several minutes for the container to compile.

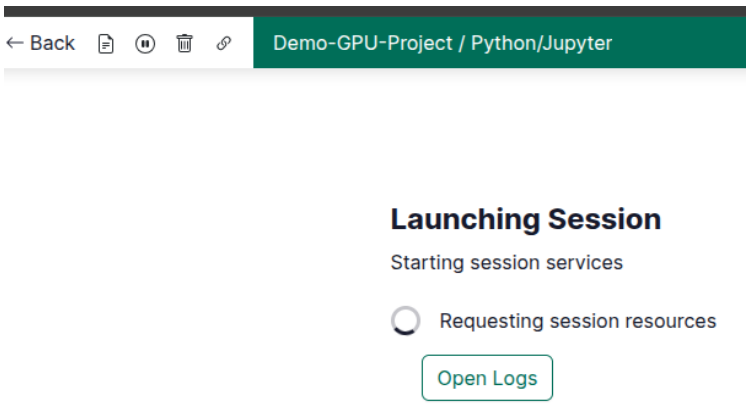



Figure 12: If the container is not compiled, the researcher could be stuck on Initialization for several minutes.

Note, if you have any issues starting the container, can you please send all of the logs in the above screenshot to support@computing.dcu.ie. This includes the logs in *Amalthea Session*, *Oauth2 Proxy*, *Authproxy,* and *Init Certificates*. This will help us debug the problem.

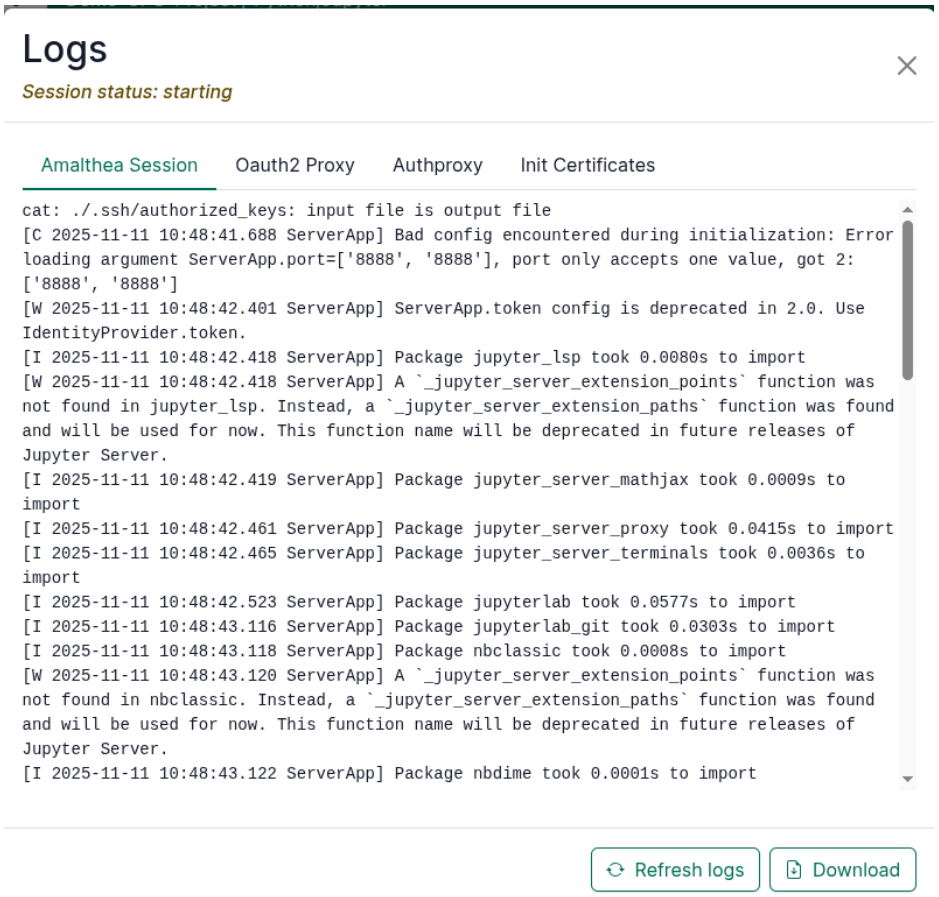



Figure 13: Capture the Container Logs if there are any issues launching.

When the pod launches, you should see the GitLab code repository that you connected to in Section 3.

```shell
cd /home/jovyan/work/my-data-science-project
```

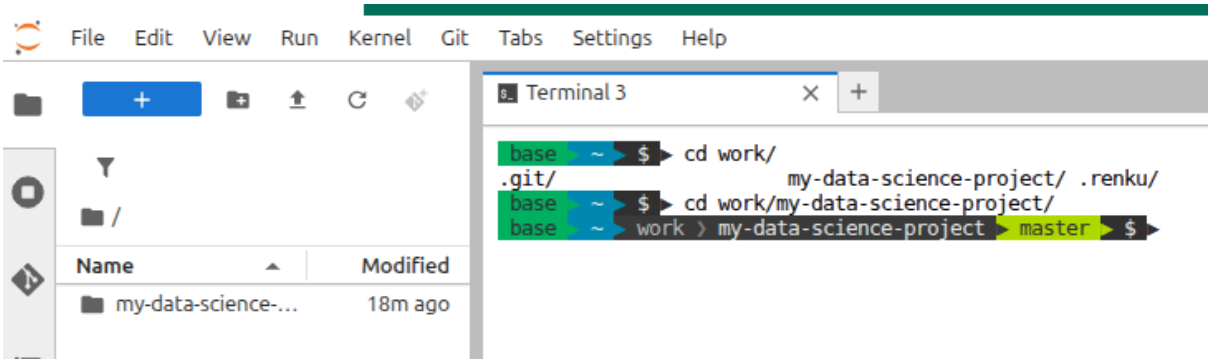



Figure 14: my-data-science-project Gitlab Repo

If you open a Terminal window, you will be able to see the GPU attached to the container.

- `nvidia-smi`: Included with the NVIDIA driver package.
- `cuda` and `nvcc` (if you're running CUDA workloads).

You can verify installation and GPU availability by running the following command in the container:

```shell
nvidia-smi
```

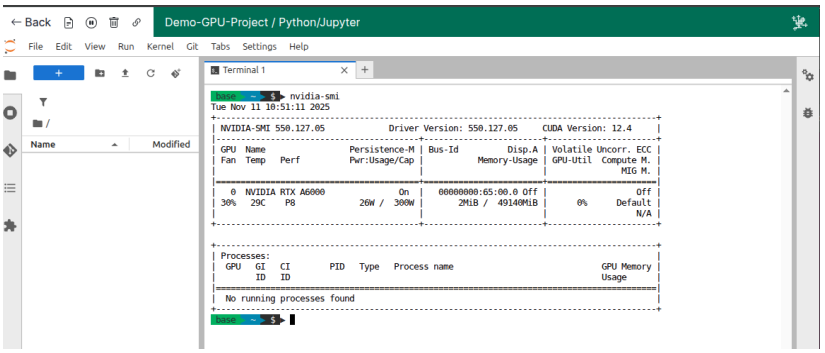



Figure 15: nvidia-smi command to show the GPU in the container

For example, creating a python file.

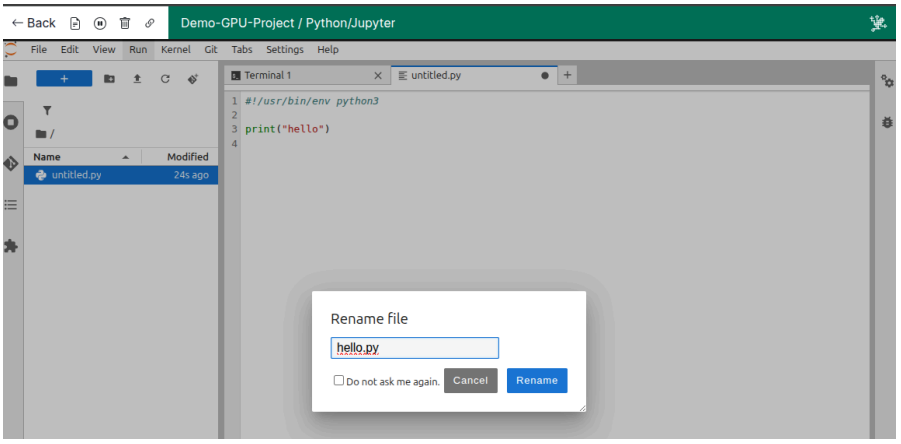



Figure 16: Creating a sample python file

To execute the script, open a terminal window in Renku, and run the following command

```
Shell
python3 hello.py
```

For Example,

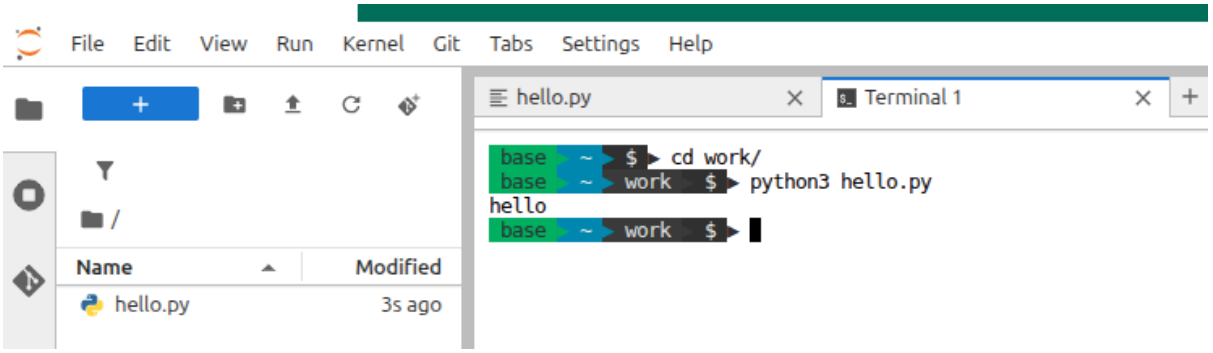



Figure 17: Executing the python file

# 5. JupyterLab Renku Template Environments

At this point, the JupyterLab environment should launch.

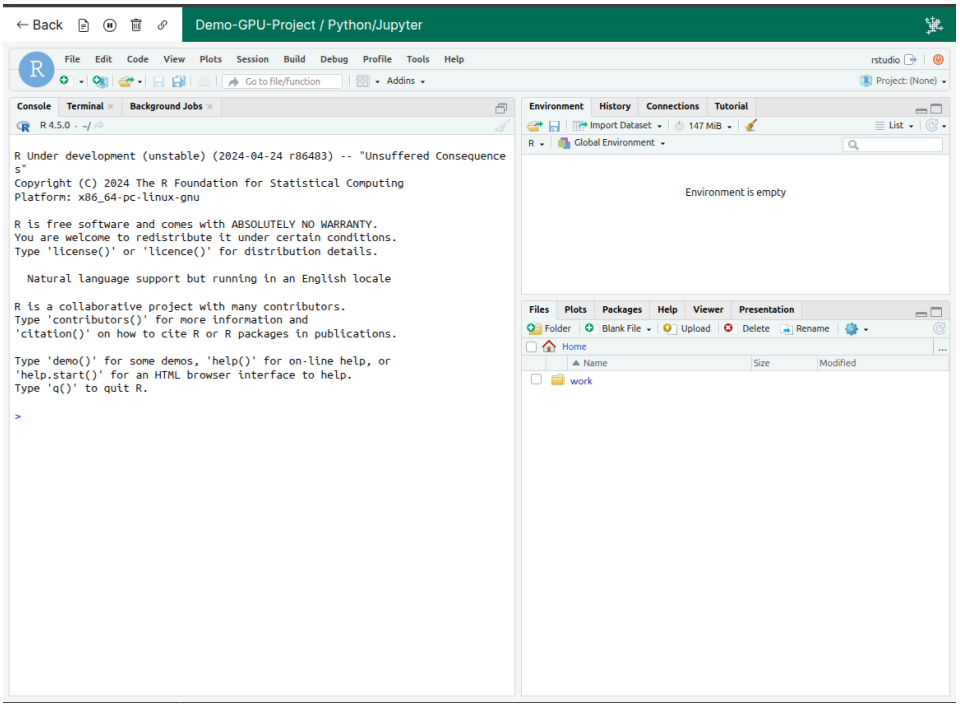



Figure 18: Example Console Renku Template

Figure 18 shows an example of the Console Renku Template displayed to the researcher.

# 6. Example: Create a Simple Deep Learning Workload

Follow these steps to create a sample GPU workload in Renku.

1. Create a new project.
    a. Make Visibility "**Private**".
    b. Click Next

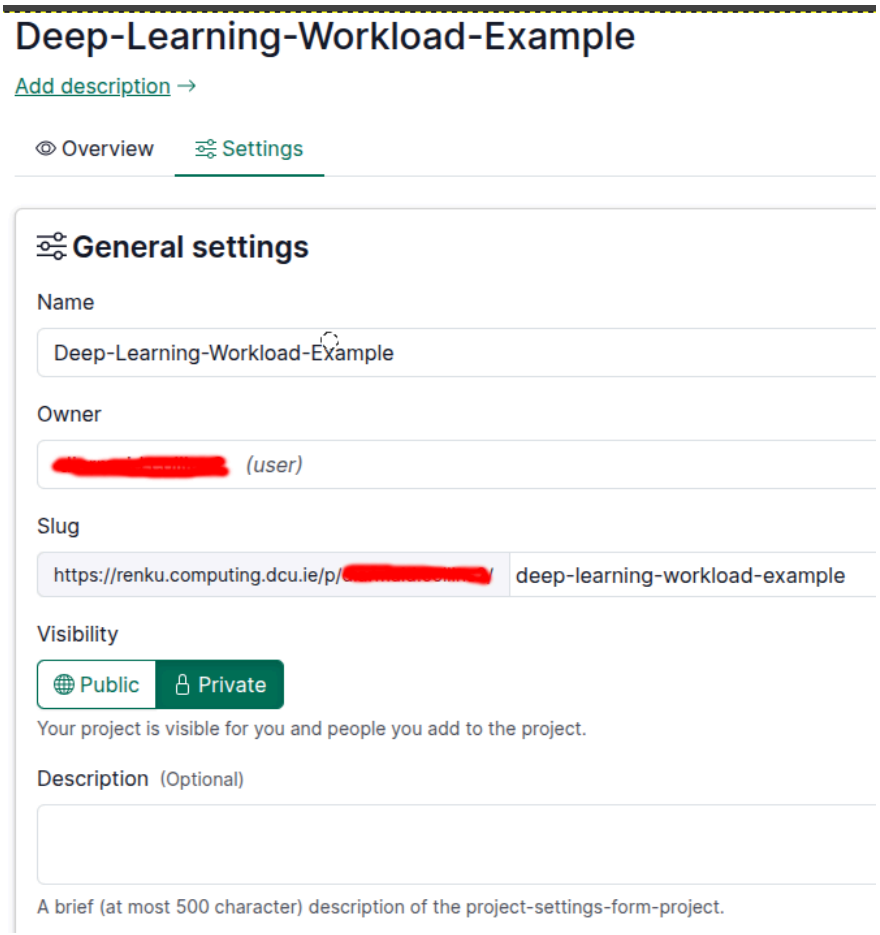



Figure 19: Create a Python (3.10.11) Project

2. Click "*Add Session Launcher*", Select "*Python/Jupyter"* and click "*Next*"

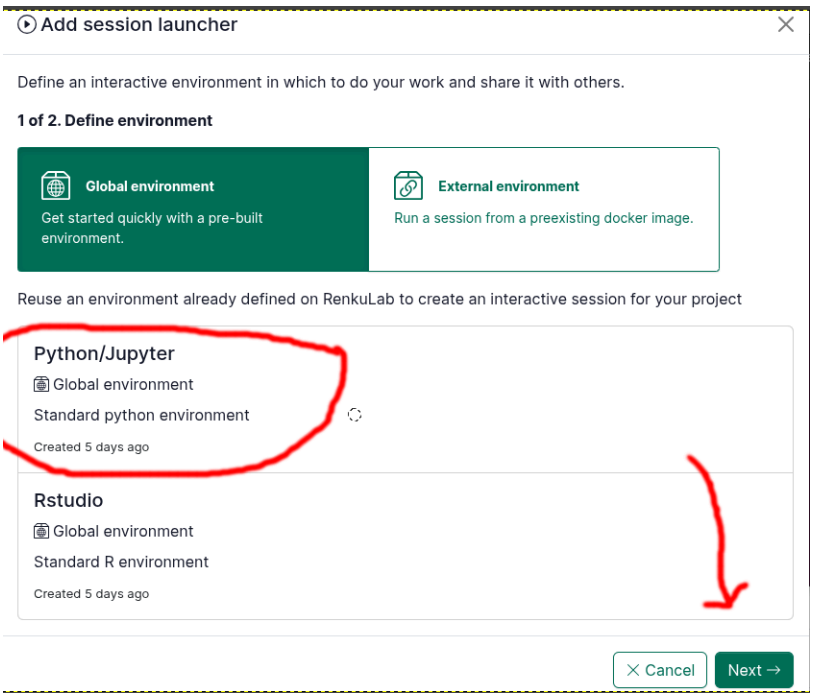



Figure 20: Click “Next”

3. Select Session Class: ***gpu6.nvidia.rtx.a6000-shared***. Increase the Storage to ***100GB***. Click ***Add Session Launcher***

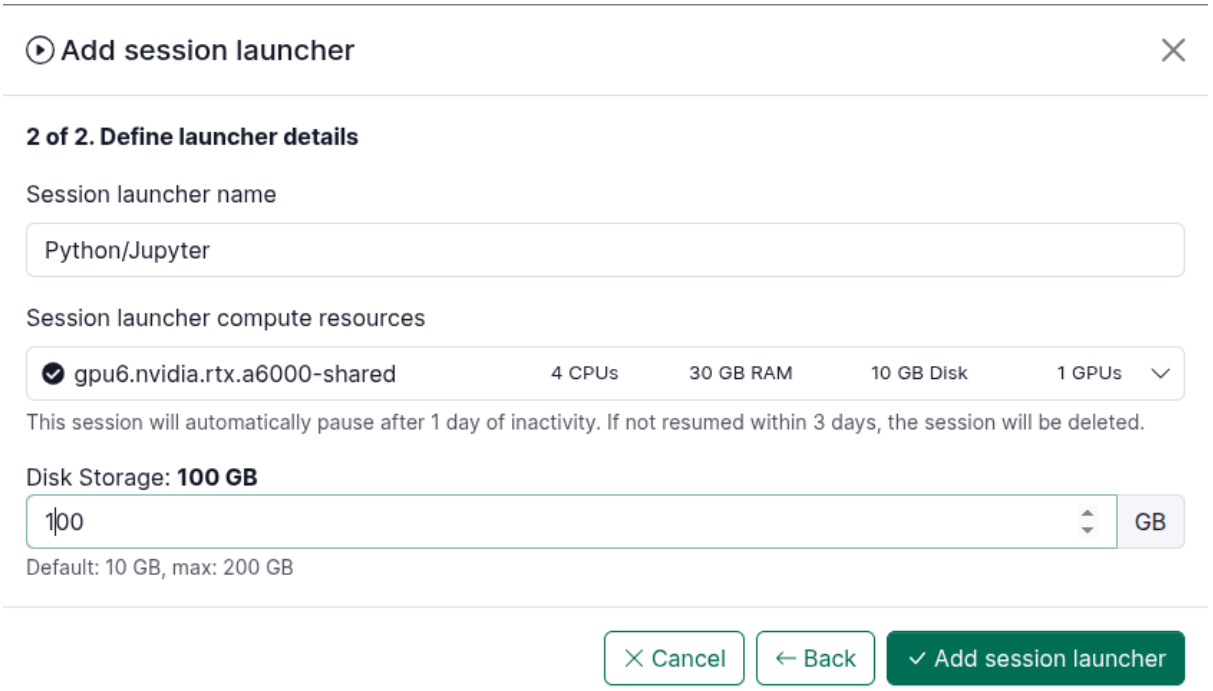



Figure 21: Define Session Launcher Details

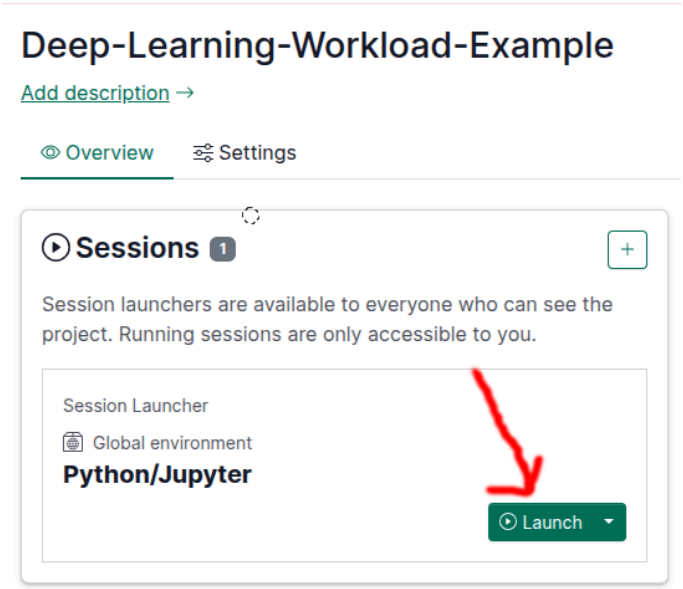



Figure 22: Select Launch to start the container Session

4. When the Session is created, in the Python/JupyterLab environment, select Terminal. See Figure 23

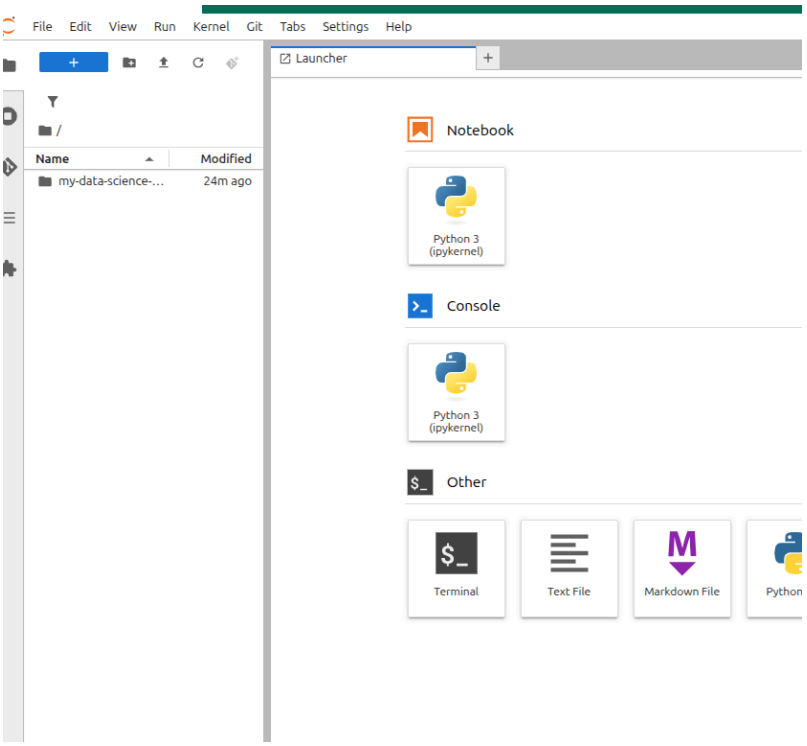



Figure 23: JupyterLab environment environment for Template Python Project

In Renku, open a terminal window, and type *renku -h* to get a list of renku commands

```
Shell
renku -h
```

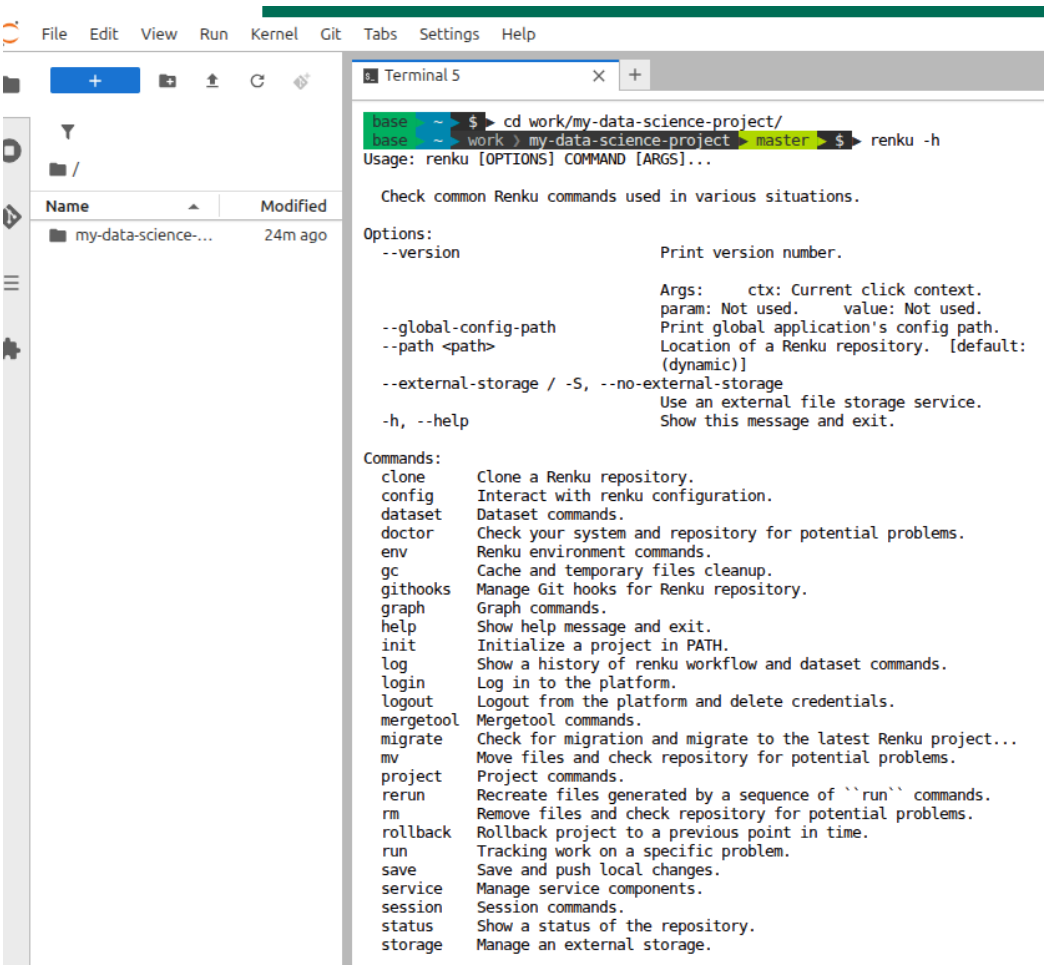



Figure 24: Renku workflow commands

cd into the cloned repo `my-data-science-project`

Shell

```
cd work/my-data-science-project/
# then run renku init command
renku init
# and select
1  python-minimal
```

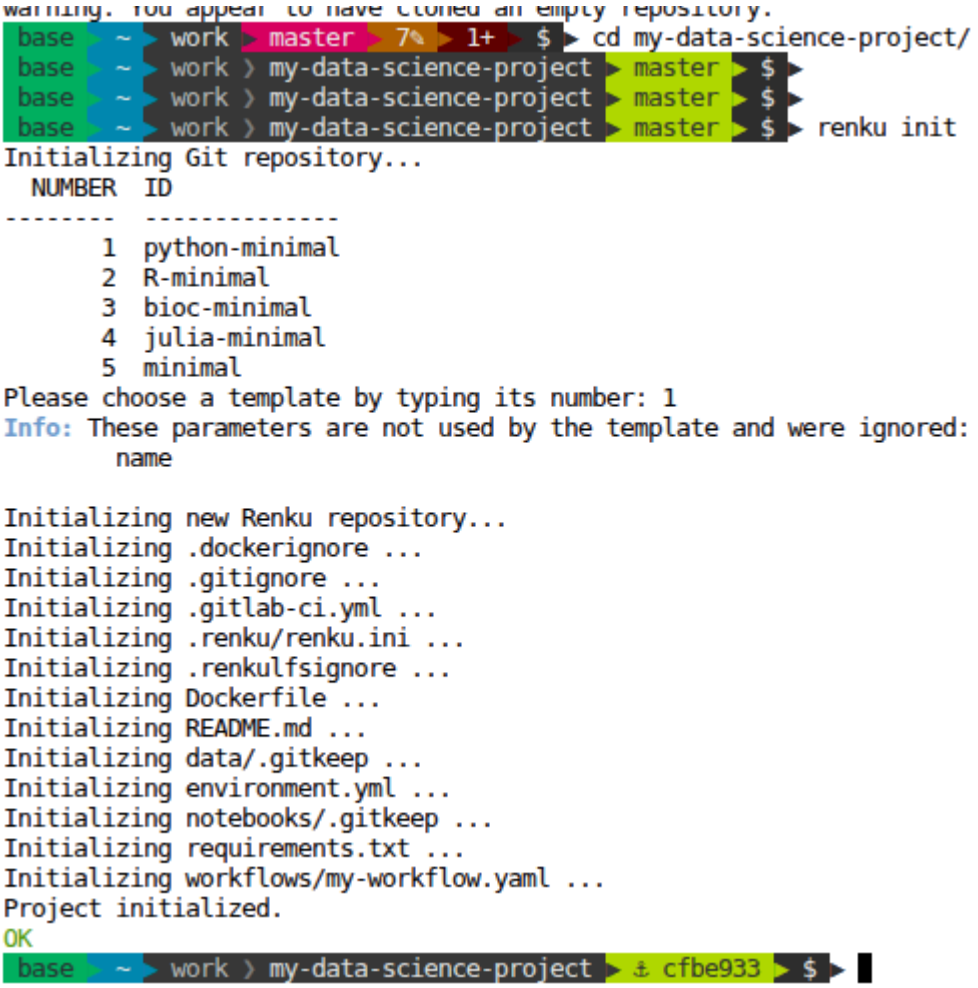



Figure 25: Initialise the renku project.

Type the following, replacing your name and email address:

```
Shell

 base ▸ ~ ▸ master ▸ 11+ ▸ $ ▸renku init
git config --global --add user.name "John Doe"
git config --global --add user.email "john.doe@example.com"
```

Run renku init command again, and select number 1 (python-minimal):

```
Shell
base ▸ ~ ▸ master ▸ 11+ ▸ $ ▸ renku init
Initializing Git repository...
  NUMBER  ID
--------  --------------
       1  python-minimal
       2  R-minimal
       3  bioc-minimal
```

```
       4  julia-minimal
       5  minimal
Please choose a template by typing its number: 1
Info: These parameters are not used by the template and were ignored:
        name

Initializing new Renku repository...
Initializing .dockerignore ...
Initializing .gitignore ...
Initializing .gitlab-ci.yml ...
Initializing .renku/renku.ini ...
Initializing .renkulfsignore ...
Initializing Dockerfile ...
Initializing README.md ...
Initializing data/.gitkeep ...
Initializing environment.yml ...
Initializing notebooks/.gitkeep ...
Initializing requirements.txt ...
Initializing workflows/my-workflow.yaml ...
Project initialized.
OK
 base ▸ ~ ▸ master ▸ 9+ ▸ $ ▸
```

According to the renku workflow, if you type *renku save*, it should update the gitlab repo.

```
Shell
renku save
```

**Note, SAVING your code is very important. The storage available on renku is ephemeral and will be DELETED. Always save your work to a git repository.**

For example:

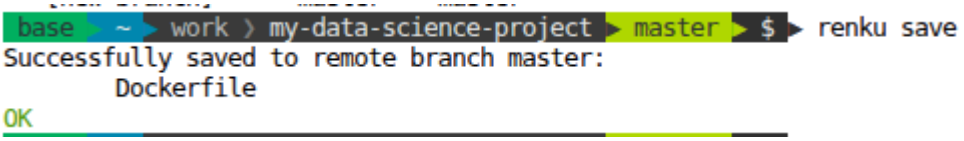



Figure 26: Renku save command, data stored on GitLab.computing.dcu.ie

# GPU Workloads

To test GPU usage, you need to run a workload that utilizes the GPU. For instance, Using TensorFlow or PyTorch (Deep Learning Workload). To Install TensorFlow to support the Nvidia A6000 GPU, install the tensorflow torch torchvision pip libraries using the following command:

```
None
# This command will take some time to install:
pip install tensorflow torch torchvision
```

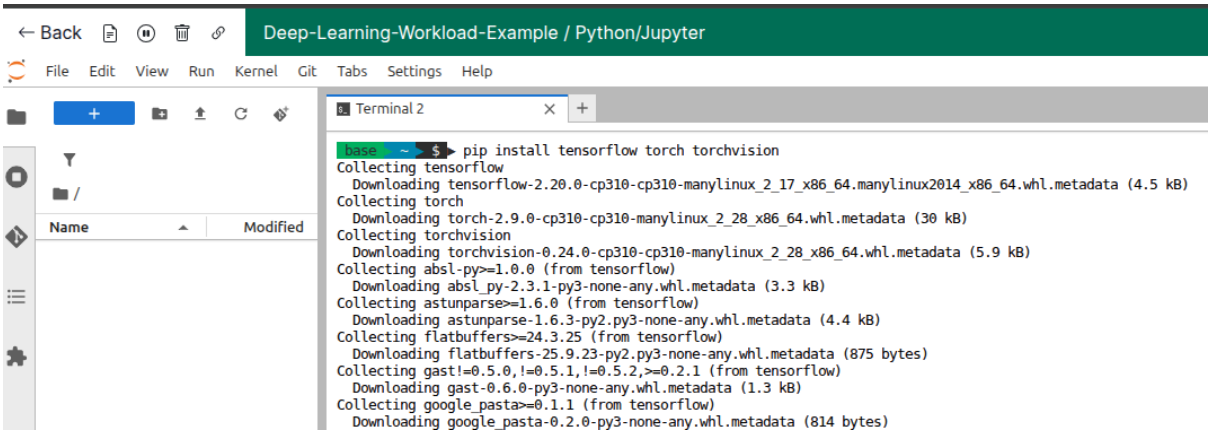



Figure 27: tensorflow torch torchvision installation using pip

For example, Figure 27 shows tensorflow torch torchvision installation using pip.

Or add *tensorflow* *torch* *torchvision* to the requirements.txt file

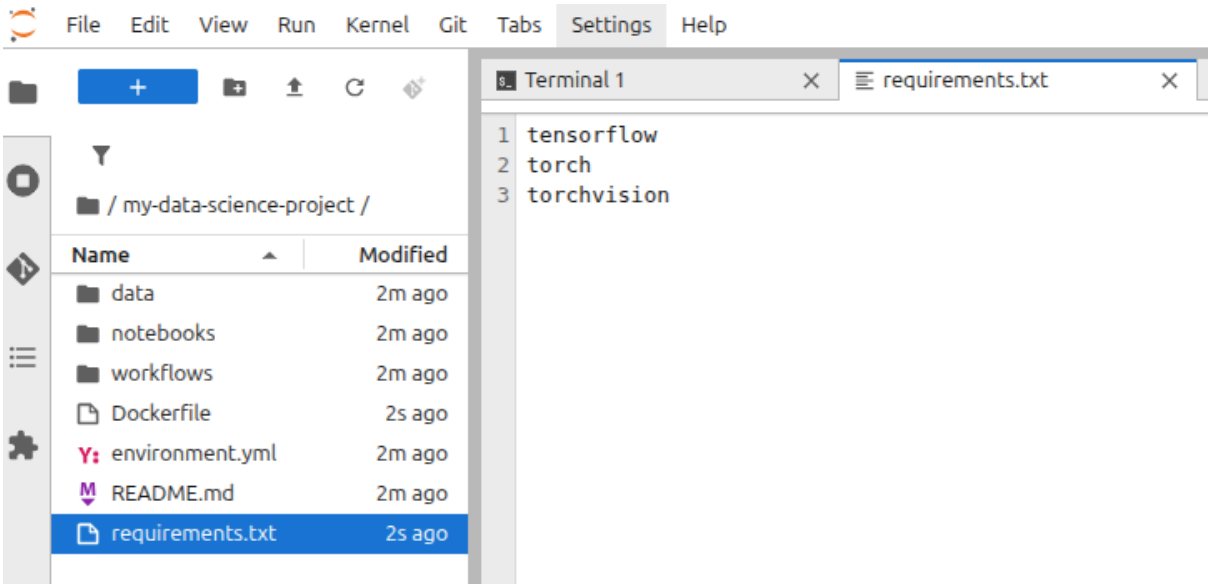



Figure 28: Using the requirements.txt file stored on GitLab.

```
Shell
pip install -r requirements.txt

# Remember to run renku save on the command line to update the gitlab
# project
renku save
```

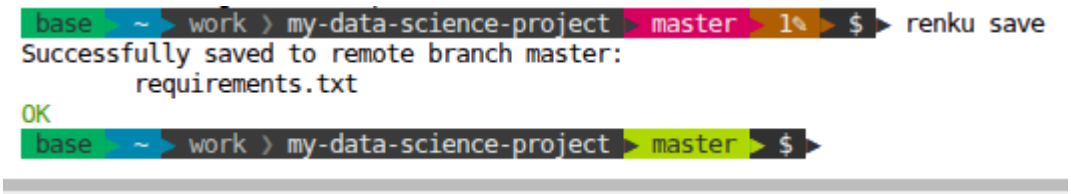



Figure 29: Remember to save all data to git instance

## Create the following python file *matrix_multiplication.py* to test the GPU

None

```
import torch
import time

# Allocate a tensor on the GPU
x = torch.rand((10000, 10000)).cuda()

# Record the start time
start_time = time.time()

# Initialize a counter
counter = 0

# Loop for 2 minutes
while time.time() - start_time < 120:  # 120 seconds = 2 minutes
    y = torch.matmul(x, x)  # Perform a matrix multiplication
    torch.cuda.synchronize()  # Ensure all GPU computations finish
    counter += 1  # Increment the counter
    print(f"Iteration {counter}: Matrix multiplication completed")

print(f"Finished running for 2 minutes. Total iterations: {counter}.")
```

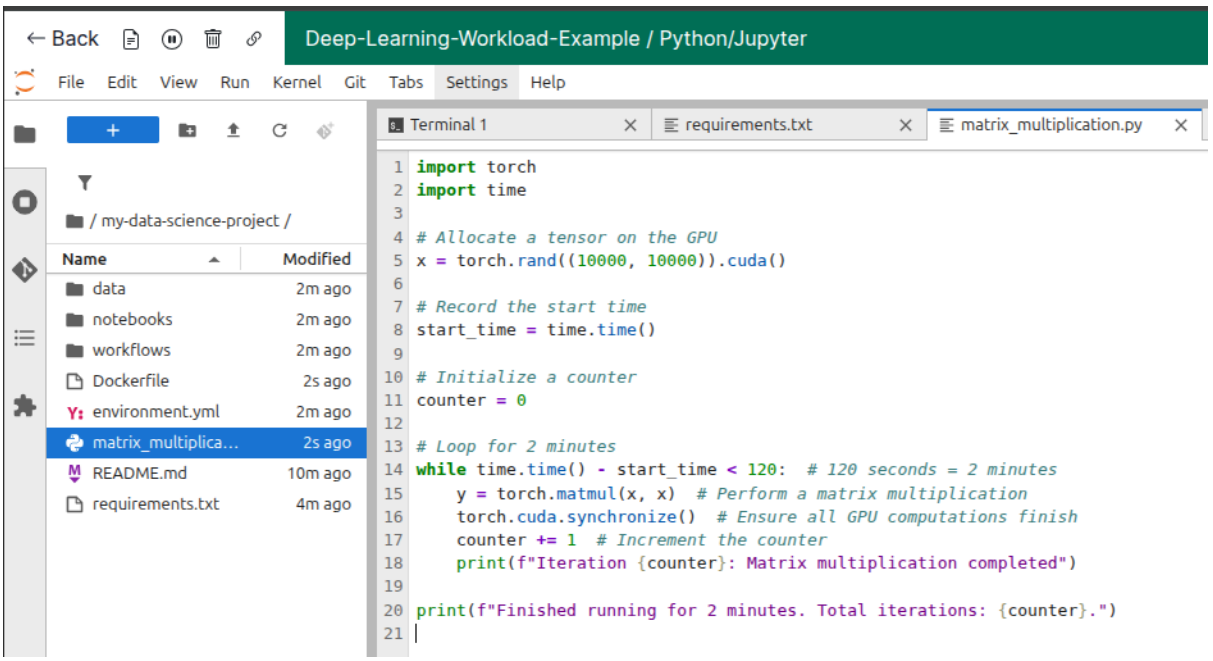



Figure 30: matrix_multiplication.py example python file

# How2 Monitor GPU utilisation

To test and monitor the percentage of GPU utilization on an NVIDIA GPU, follow these steps.

1. Open a new tab, and open a another terminal:

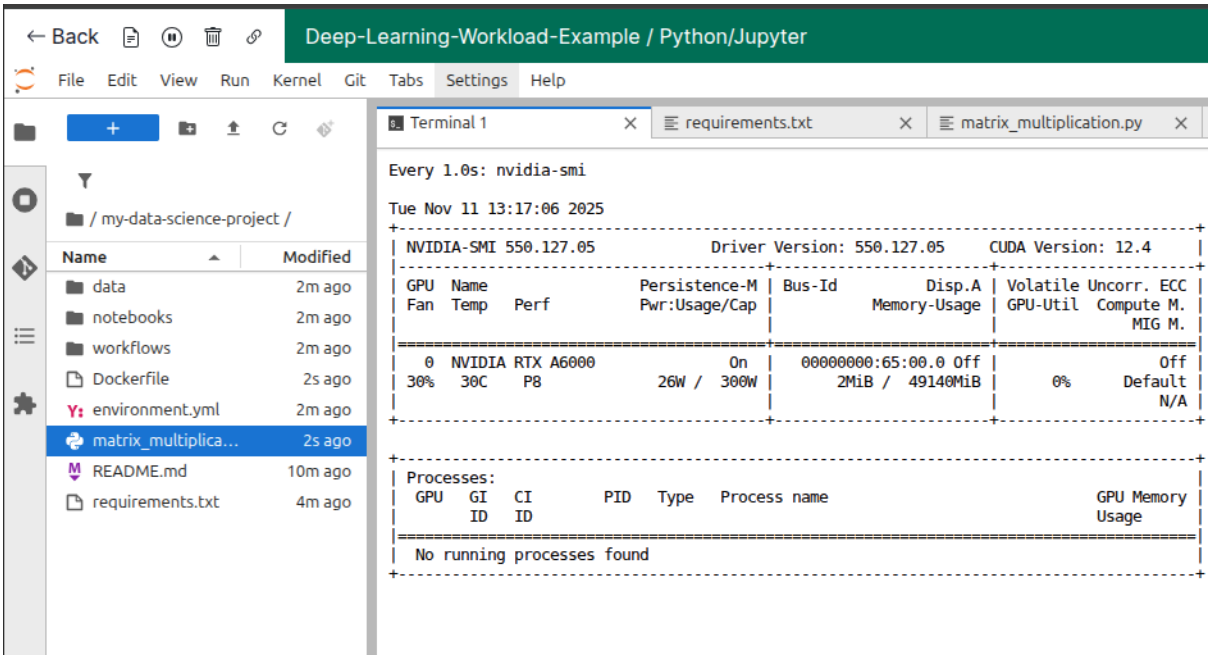



Figure 31: Launch a second Terminal to monitor GPU utilization

In the new terminal, run the following command to monitor the GPU utilisation:

```
None
watch -n 1 nvidia-smi
```

Run the `matrix_multiplication.py` script in the first terminal, and simultaneously monitor GPU usage with `nvidia-smi` command in the second terminal. Look at the **`% Util`** column under the **GPU** section to see the percentage utilization.

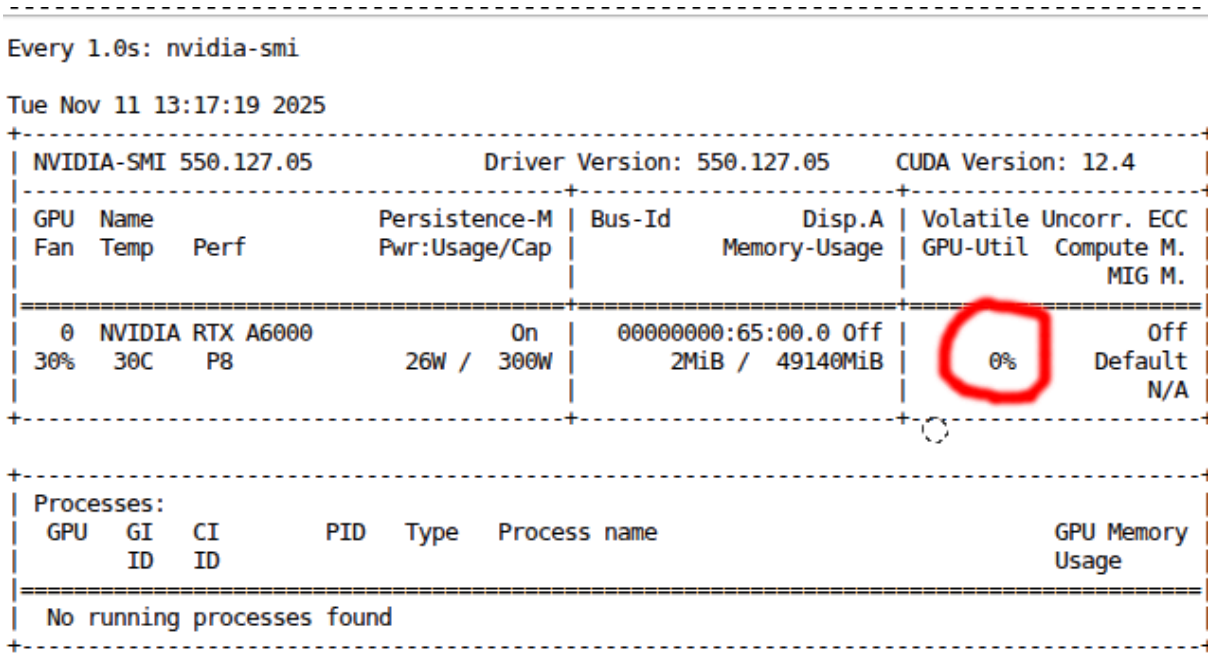



Figure 32: No application running - 0% GPU Utilization

Run the Matrix Multiplication Script with the following command:

```
None
python3 matrix_multiplication.py
```

Output from the command is shown in Figure 33.

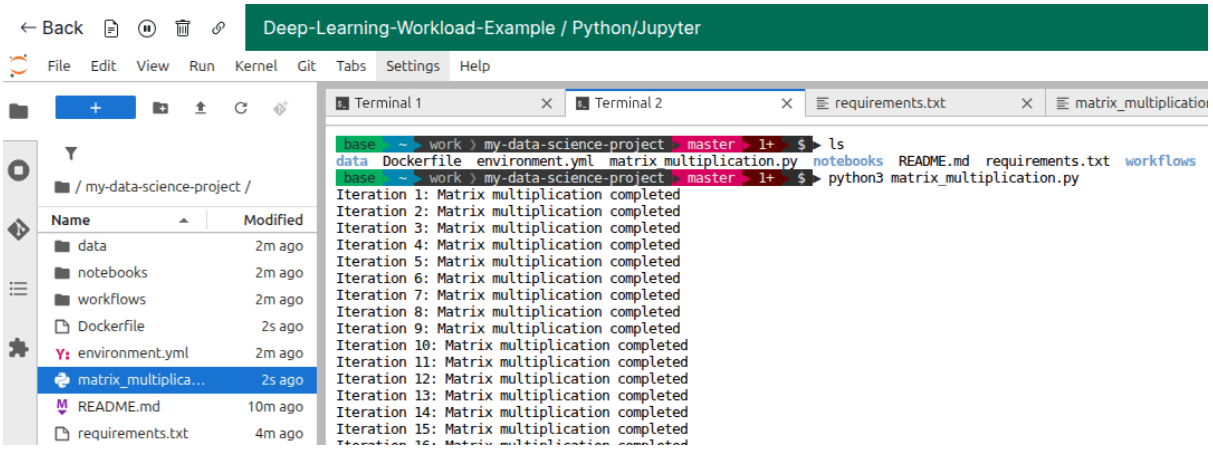



Figure 33: Output from python3 matrix_multiplication.py command

Switch to the second Terminal, and you should see that the GPU is at 100% Utilization while the matrix calculations are running. Note it should be at 100%

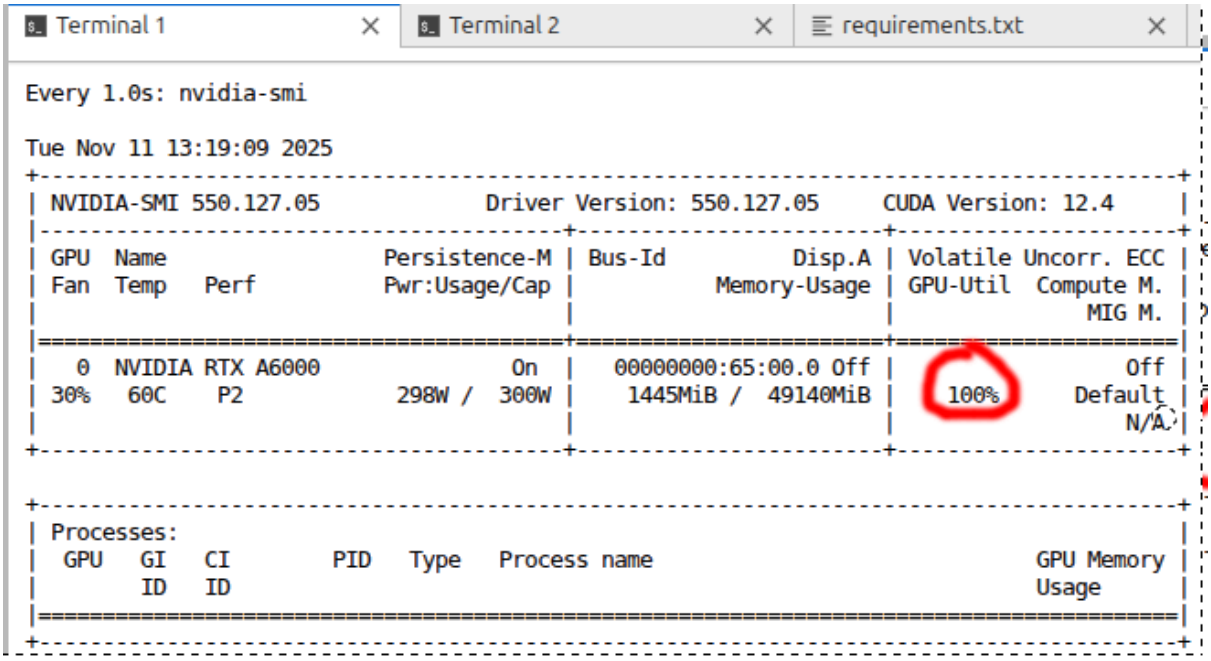



Figure 34: Shows GPU Utilization is at 100%

# 6. Using Renku to produce a reproducible workflow

Renku provides a comprehensive tutorial on the basics of using renku to create a reproducible, reusable “workflow”.

Figure 35 shows the Renku Cheat Sheet.

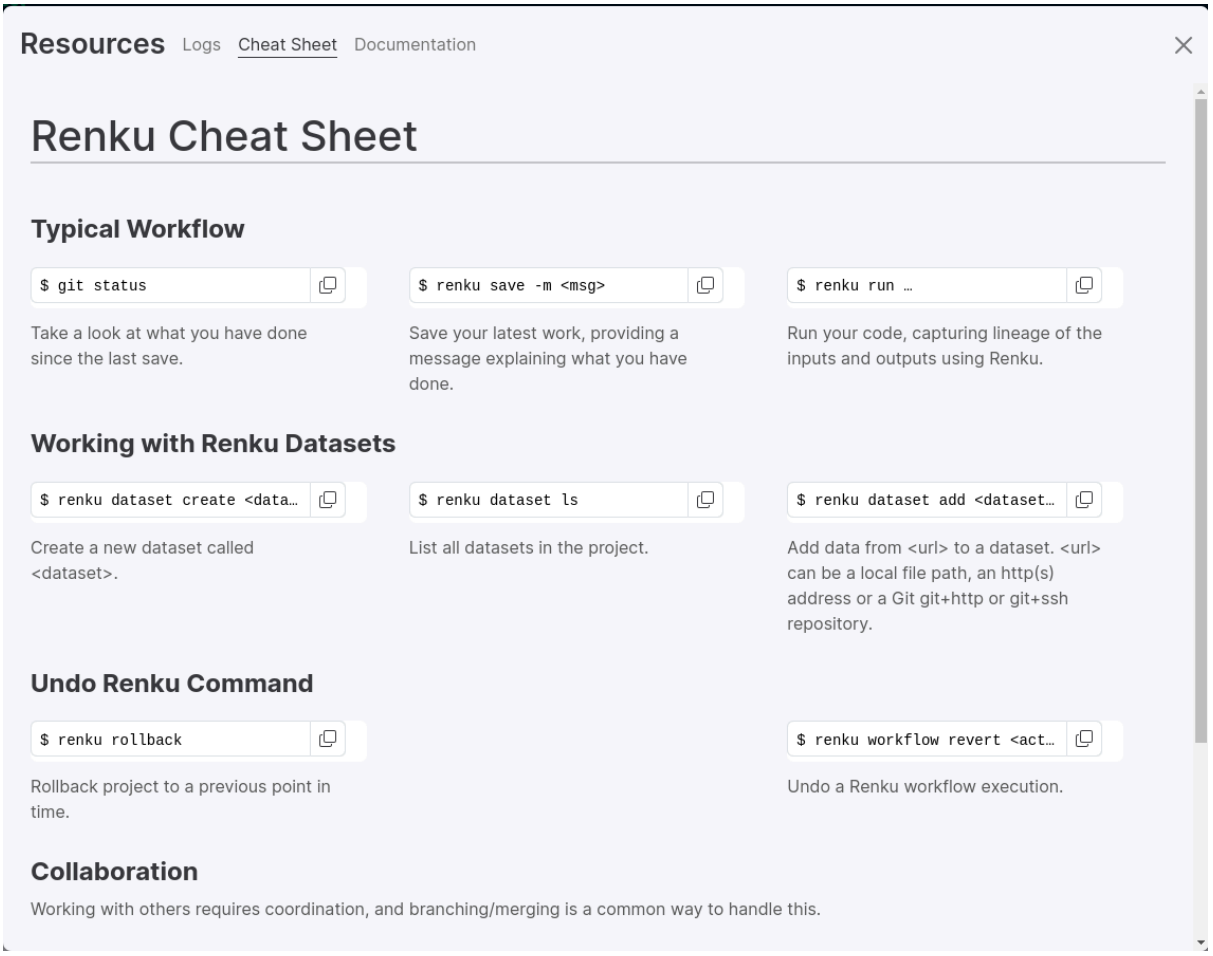



Figure 35: Renku Cheat Sheet

Typing the following command in the terminal will show the renku commands available:

```
None
renku -h
```

The response is shown in Figure 36.

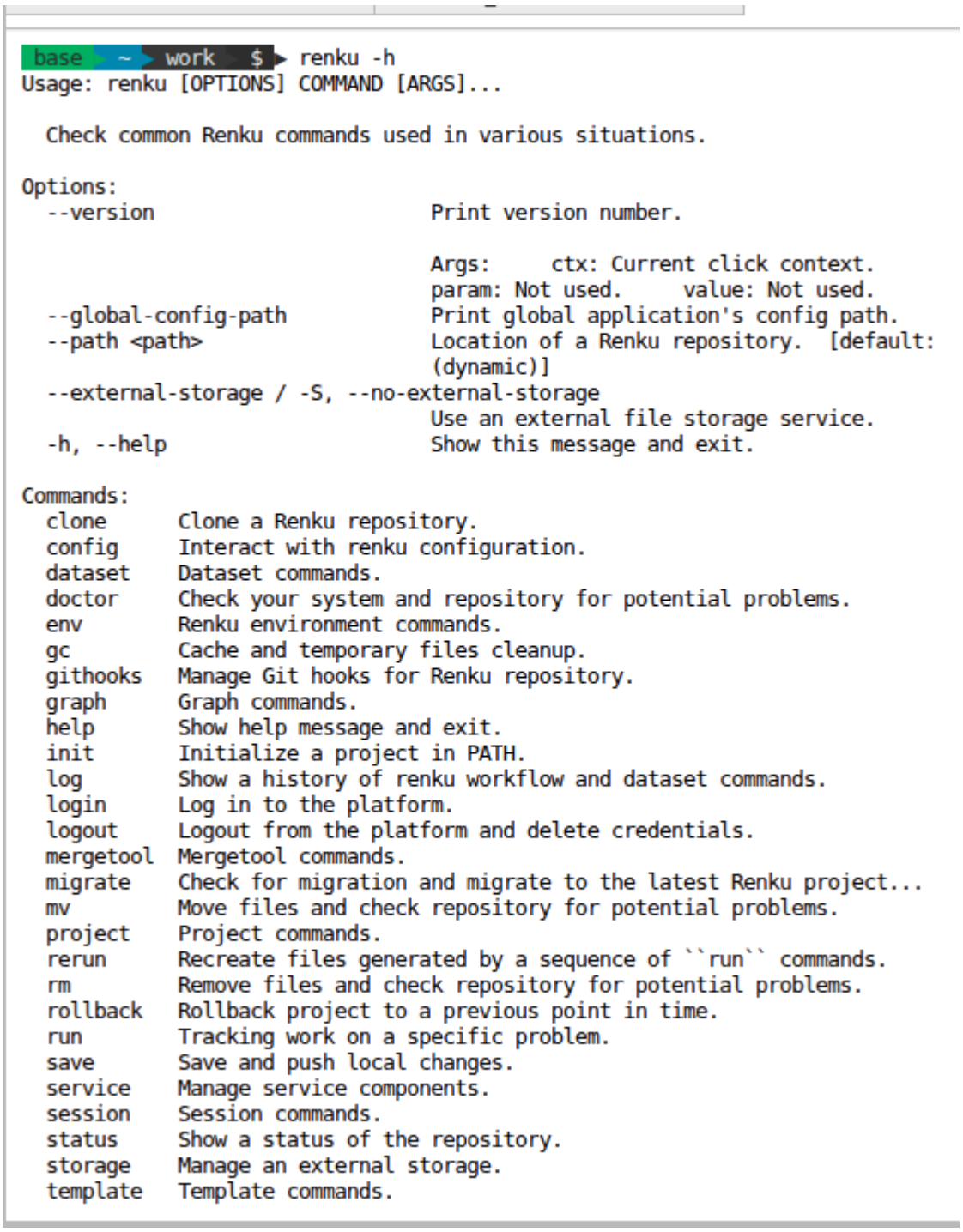



Figure 36: Output from Renku help command

Typing the following command in the terminal to save the changes using renku [see Figure 37]:

```
None
renku save -m "add matrix_multiplication.py file"
```

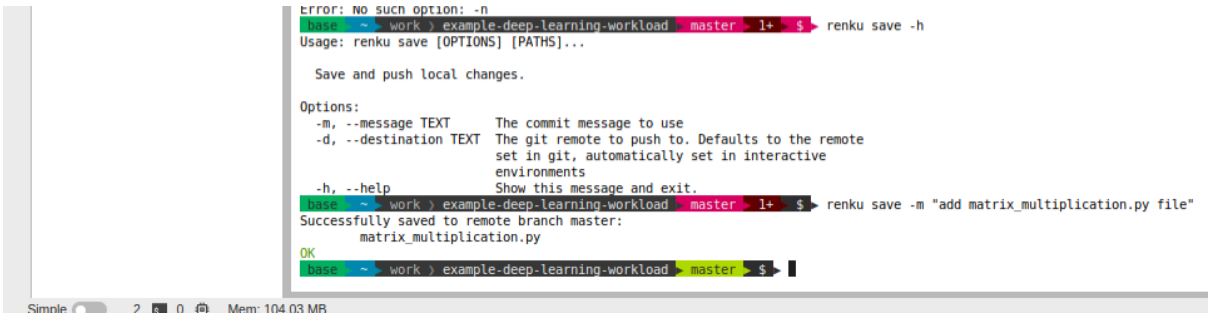



Figure 37: Shows renku save -m “a message” command

If you now check the project in Gitlab, it will show the file matrix_multiplication.py with comment “add matrix_multiplication.py file” was submitted 10 minutes ago. See Figure 38.

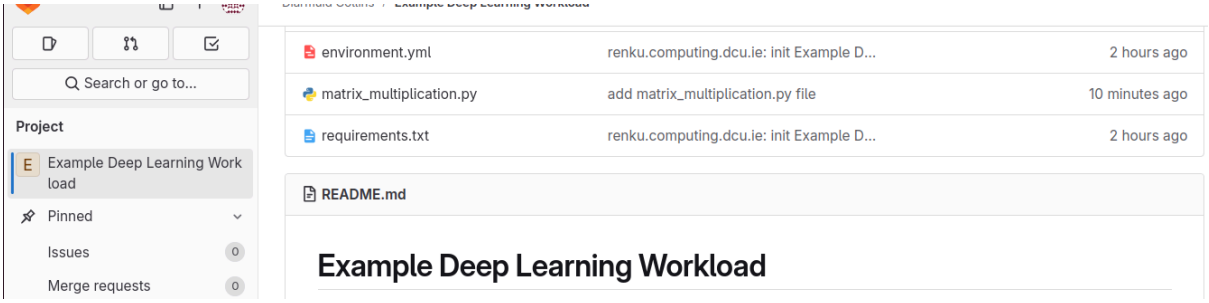



Figure 38: Gitlab

# 7. A Data Science Example with Renku

In this section, we are roughly following the Renku First Steps tutorial. Create a project called Flights-Tutorial. Start a terminal by clicking the Terminal icon (1) on the bottom right of the Launcher page.

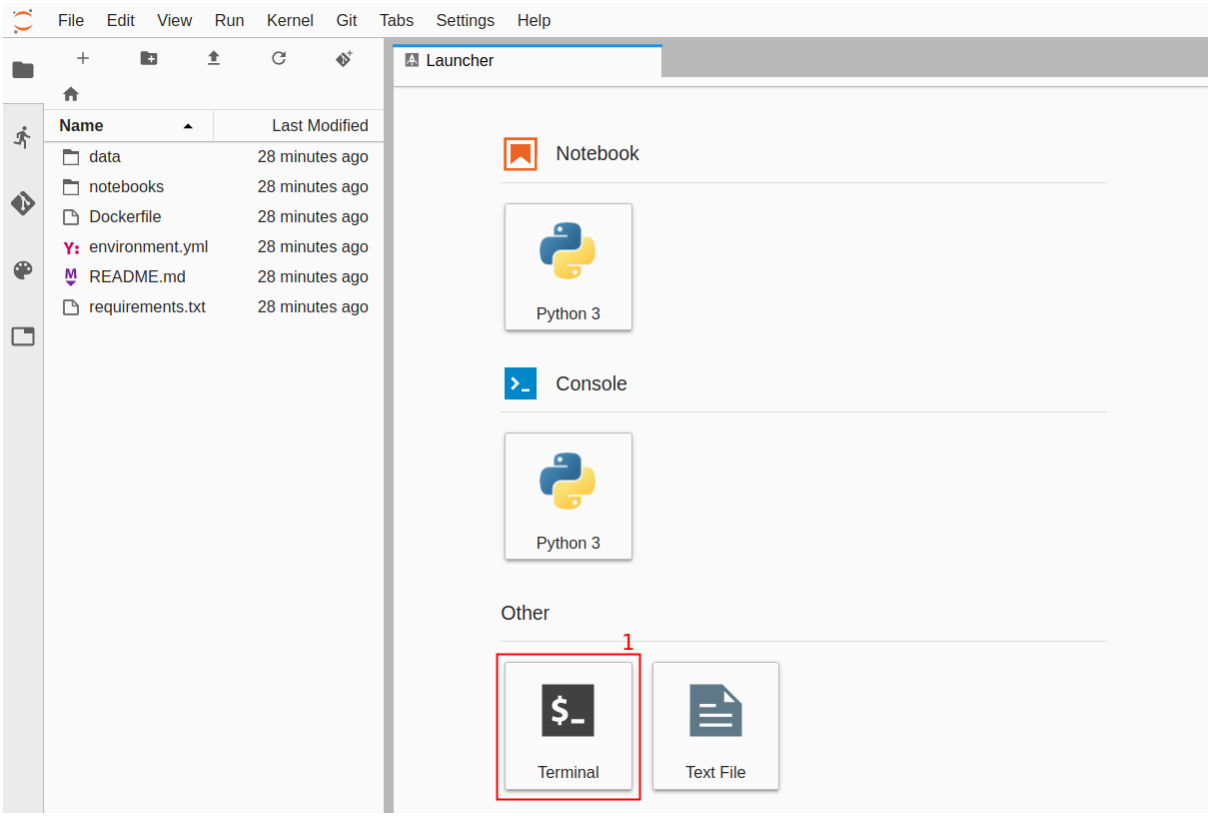



Figure 39: Start a terminal by clicking the Terminal icon (1)

Renku can create datasets containing data from a variety of sources. It supports adding data from the local file system or a URL. Renku can also import data from a data repository like the Dataverse or Zenodo.

## Adding data from a remote repository

The advantage of importing data from a data repository is that it can contain metadata that can be used to help interpret it. Another advantage is that data repositories assign DOIs to data which can be used to succinctly identify it and guarantee that the data will be findable and accessible for a longer period of time (usually at least 20 years).

The DOI for the dataset we want to import is https://www.doi.org/10.7910/DVN/WTZS4K

Execute the following line and when prompted if you really want to download the data, answer yes.

```
None
renku dataset import --name flight-data https://www.doi.org/10.7910/DVN/WTZS4K
Warning: The '--short-name/--name' flags are deprecated. Use '-s/--slug'
instead.
```

```
CHECKSUM    NAME                      SIZE    TYPE
----------  ------------------------  ------  ---------------
            2019-01-flights.csv.zip   7.9 MB  application/zip
Warning: Do you wish to download this version? [y/N]: y
Info: Adding these files to Git LFS:
        data/flight-data/2019-01-flights.csv.zip
To disable this message in the future, run:
        renku config set show_lfs_message false
OK
```

Here, we can see the method of referencing a dataset in a data repository by DOI. By doing so, we capture a reference to the source of the data in the metadata of the project.

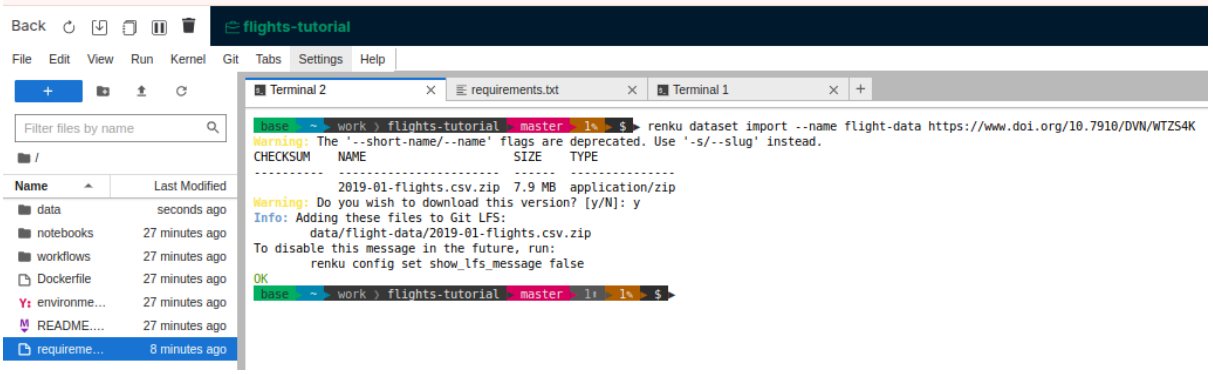



Figure 40: Importing a dataset into renku using a DOI

You can list the datasets in a project by running the following:

```
None
renku dataset ls
```

For example:

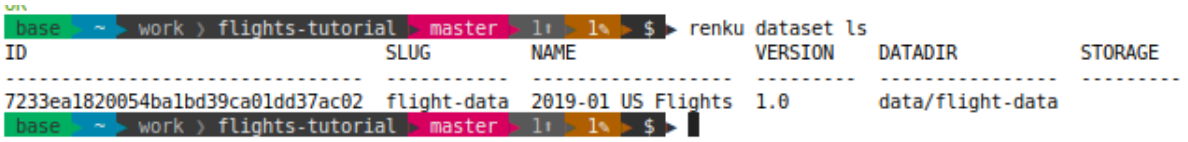



Figure 41: list renku data sets in the project

## Installing and managing python packages

Defining your runtime environment is critical if you would like others to be able to reuse your work. To achieve this, it is necessary to manage the software libraries that are needed for your code to execute. In Renku, we rely on existing conventions for specifying the execution environment.

For example:

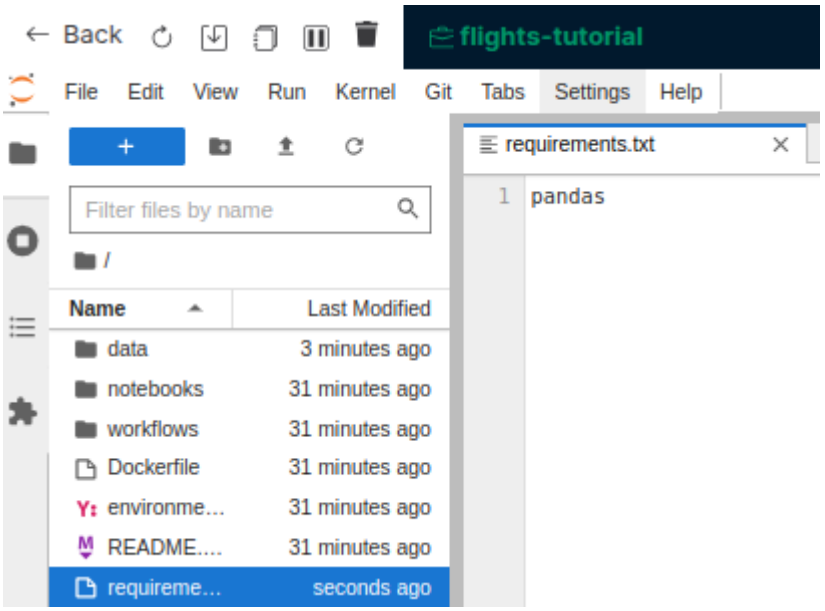



Figure 42: Import Python libraries to support reproducibility

Going back to the same terminal session as before, we can now install these packages using the following pip command:

```
None
pip install -r requirements.txt
```

For example:

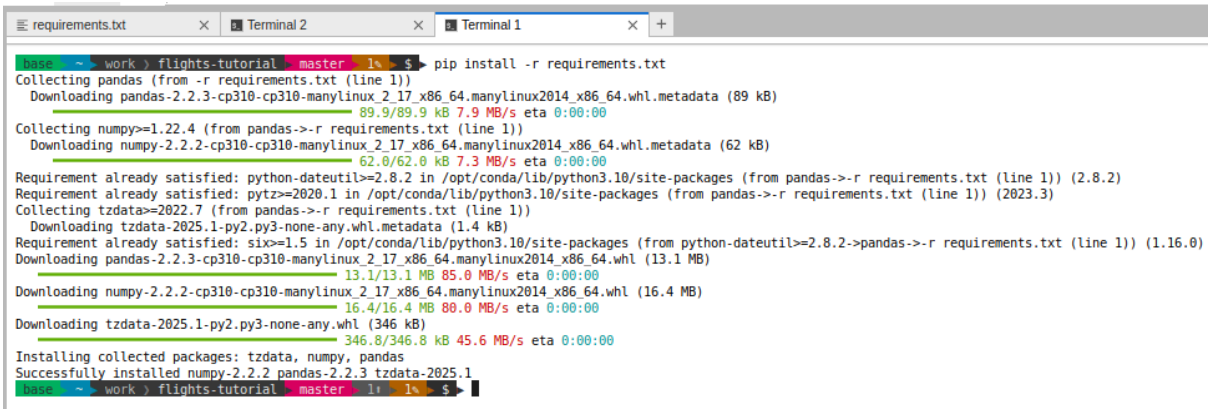



Figure 43: Installation of Python libraries

## Saving package additions

Make sure to save these additions to the repository. Its important to regularly save data. This can be done with the *renku save* command from the Terminal:

```
Successfully installed numpy-2.2.2 pandas-2.2.3 tzdata-2025.1
base ~ work ❯ flights-tutorial master 1↑ 1% $ renku save -m 'updated packages'
Successfully saved to remote branch master:
        requirements.txt
OK
base ~ work ❯ flights-tutorial master $
```

Figure 44: Push data to Gitlab using renku command line framework

Or using git:

```
base ~ work ❯ flights-tutorial master 1! 1% $ git add data/flight-data/2019-01-flights.csv.zip
base ~ work ❯ flights-tutorial master $ git commit -m "flight data"
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
base ~ work ❯ flights-tutorial master $ git push
Everything up-to-date
base ~ work ❯ flights-tutorial master $
```

Figure 45: Push data to Gitlab using git

## How to do data science

In the JupyterLab interface, use the file navigator in the left sidebar to go to the *Files* tab. Here, you see a listing of files and folders from your project. To create a new notebook, first double-click on the *notebooks folder*, then on the *'+' button*. Select *Python 3 (ipykernel)* to create a new notebook.

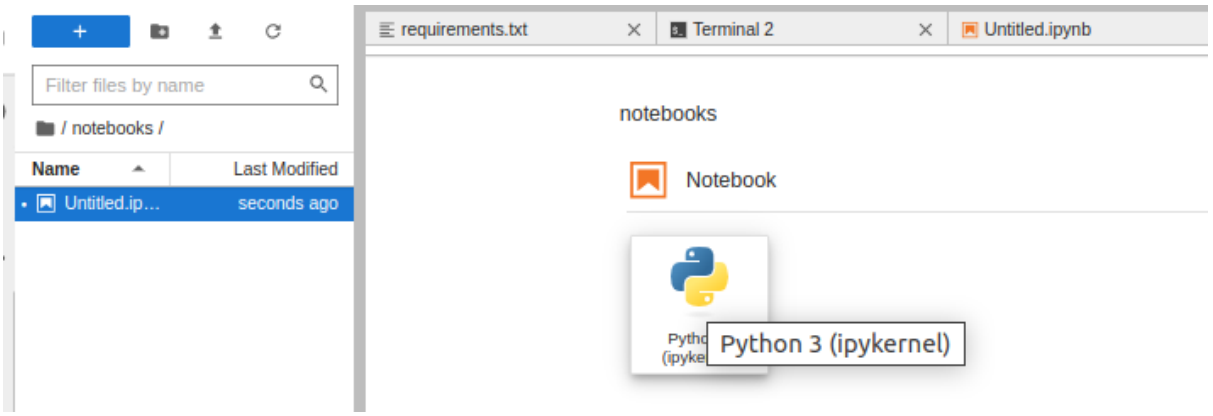



Figure 46: Create a new notebook using JupyterLab interface

To rename the notebook, right-click on its name (Untitled.ipynb) and select rename to *new-notebook.ipynb*. To quickly save your work, the easiest is to use the renku save command from the terminal - this will commit any uncommitted files and sync the changes with the git server. You can specify a custom commit message with -m.

```
base ~ work ❯ flights-tutorial master 1+ $ renku save -m "new notebook"
Successfully saved to remote branch master:
        notebooks/new-notebook.ipynb
OK
base ~ work ❯ flights-tutorial master $
```

Figure 47: Push data to Gitlab using “renku save” command

## Interactively explore the flights data

At this point in a data-science project, you would normally start by looking at the data, trying to understand its structure, and see how to go about answering our question: *how many*

*flights had Austin, TX as their destination*. Copy and paste the following code into the *new-nodebook.ipynb* file and press play.

```
None
import pandas as pd
input_path = "../data/flight-data/2019-01-flights.csv.zip"
output_path = "../data/output/2019-01-flights-filtered.csv"
df = pd.read_csv(input_path)
df.head()
df = df[df['DEST'] == 'DFW']
df.head()
```

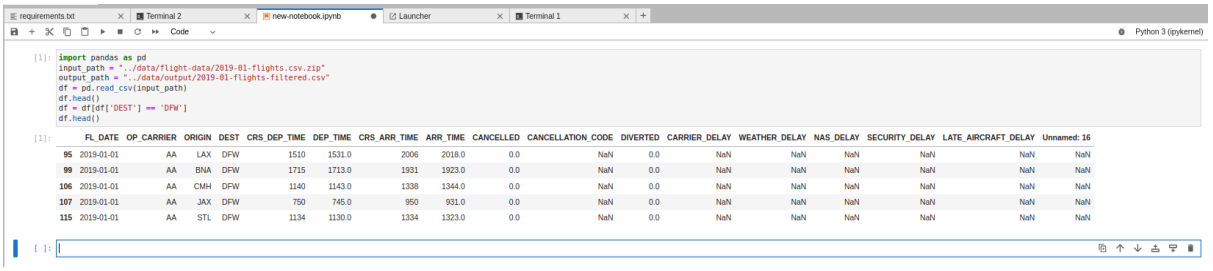



Figure 48: Result from new-nodebook.ipynb file to determine how many flights had Austin, TX as their destination

Update the code to count how many flights into Austin, Texas

```
None
import pandas as pd
input_path = "../data/flight-data/2019-01-flights.csv.zip"
output_path = "../data/output/2019-01-flights-count.txt"
#Count flights
df = pd.read_csv(input_path)
df.head()
flights_count = len(df)
result = "There were {} flights to Austin, TX in Jan 2019.\n".format(flights_count)
print(result)
```

```
[7]: import pandas as pd
     input_path = "../data/flight-data/2019-01-flights.csv.zip"
     output_path = "../data/output/2019-01-flights-count.txt"
     #Count flights
     df = pd.read_csv(input_path)
     df.head()
     flights_count = len(df)
     result = "There were {} flights to Austin, TX in Jan 2019.\n".format(flights_count)
     print(result)

     There were 583985 flights to Austin, TX in Jan 2019.

[ ]:
```

Figure 49: Update the code to count how many flights into Austin, Texas

## Renku Workflows

A renku workflow consists of a series of steps, each of which may consume some input files, execute code, and produce output files. The outputs of one step are frequently the inputs of another — this creates a connection between the code execution and results. When workflows become more complex, the bookkeeping can be tedious. That is where Renku comes in — it is designed to keep track of these dependencies for you.

More information on renku workflows can be found here:
https://renku.notion.site/Renku-Community-Portal-2a154d7d30b24ab8a5968c60c2592d87

## Collaborate on RenkuLab

Beyond building and maintaining your own data science workflows, there are many ways to use Renku to share your project with collaborators.

- Find & access public projects
- Fork a project and take it in your own direction
- Add collaborators to the conversation through Issues
- In Gitlab, add or be added as a developer on a project

More information here about collaboration in renku can be found here:
https://renku.notion.site/Renku-Tutorials-1460df2efafc80c2b27acd221aa34a24?p=1460df2efafc806faffee5d715e625f4&pm=s

# 8. Installing Packages in docker container

We might install additional Linux (Ubuntu) software packages, such as apt-utils and vim, by uncommenting and adapting the Dockerfile's RUN command (with each package on its own line and using \ to continue the command across lines), to ensure that our environment has all necessary tools and dependencies required for our R or Python packages to function properly.

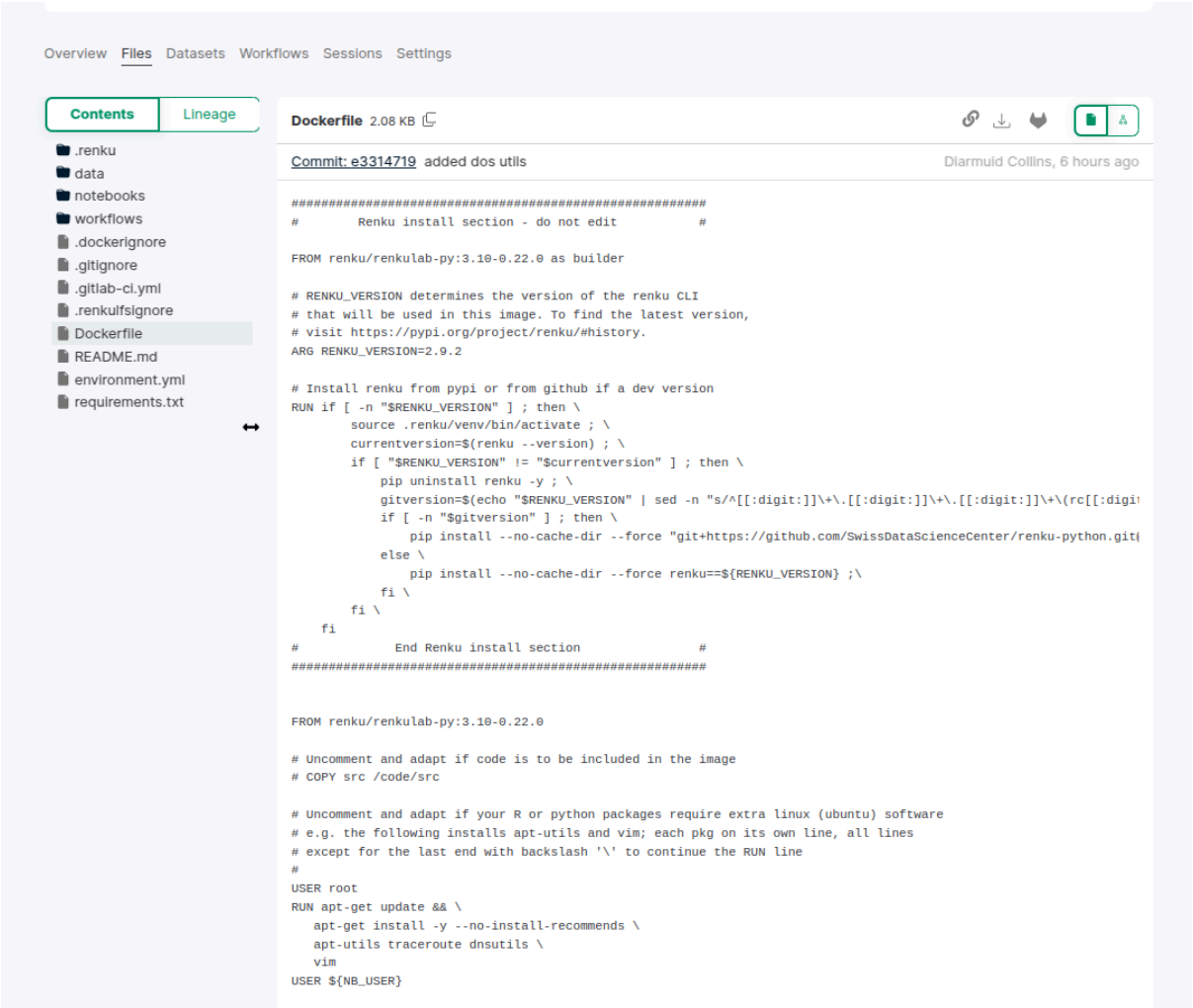



Figure 50: Update the Dockerfile to install packages in docker container

This step ensures the base operating system has all required utilities and libraries installed to support the software and packages used in data analysis or development tasks.

```
None
USER root
RUN apt-get update && \
   apt-get install -y --no-install-recommends \
   apt-utils traceroute dnsutils \
   vim
USER ${NB_USER}
```

# 9. Integrating Ubuntu with AWS S3 buckets

To handle large datasets efficiently, it may be necessary to integrate Renku sessions with external storage solutions like AWS S3 buckets or similar platforms. For instance, integrating a Renku session with AWS S3 requires a few key steps, such as installing the AWS CLI within the Docker container, setting up AWS credentials in the renku sessions, and utilizing the CLI to manage data transfers and interactions with the S3 buckets.

### 1. Install AWS CLI on Ubuntu

- Open the Dockerfile
- Uncomment the following command, and add in the ***awscli***.

```
None
# Uncomment and adapt if your R or python packages require extra linux
# (ubuntu) software e.g. the following installs apt-utils and vim; each pkg #
on its own line, all lines except for the last end with backslash '\' to
# continue the RUN line

USER root
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
    apt-utils \
    awscli \
    vim
USER ${NB_USER}
```

For example:

```
33 # Uncomment and adapt if your R or python packages require extra linux (ubuntu) software
34 # e.g. the following installs apt-utils and vim; each pkg on its own line, all lines
35 # except for the last end with backslash '\' to continue the RUN line
36 #
37 USER root
38 RUN apt-get update && \
39     apt-get install -y --no-install-recommends \
40     apt-utils \
41     awscli \
42     vim
43 USER ${NB_USER}
44
```

Figure 51: Installing awscli package in a docker container

- Save the configuration using the renku command

```
None
renku save -m "added awscli to docker file"
```

- Then recompile the container by deleting the current session, and restarting it. This will force the container to be recompiled and the *awscli* will be installed.

### 2. Configure AWS Credentials

Once the new container has been compiled and the JupyterLab interface has been restarted, run the following command to configure your AWS credentials:

```
None
aws configure
```

You will be prompted to enter:

- AWS Access Key ID: Your AWS access key.
- AWS Secret Access Key: Your AWS secret key.
- Default region name: The AWS region (e.g., us-east-1).
- Default output format: Optional (e.g., json).

These credentials are stored in ~/.aws/credentials and ~/.aws/config.

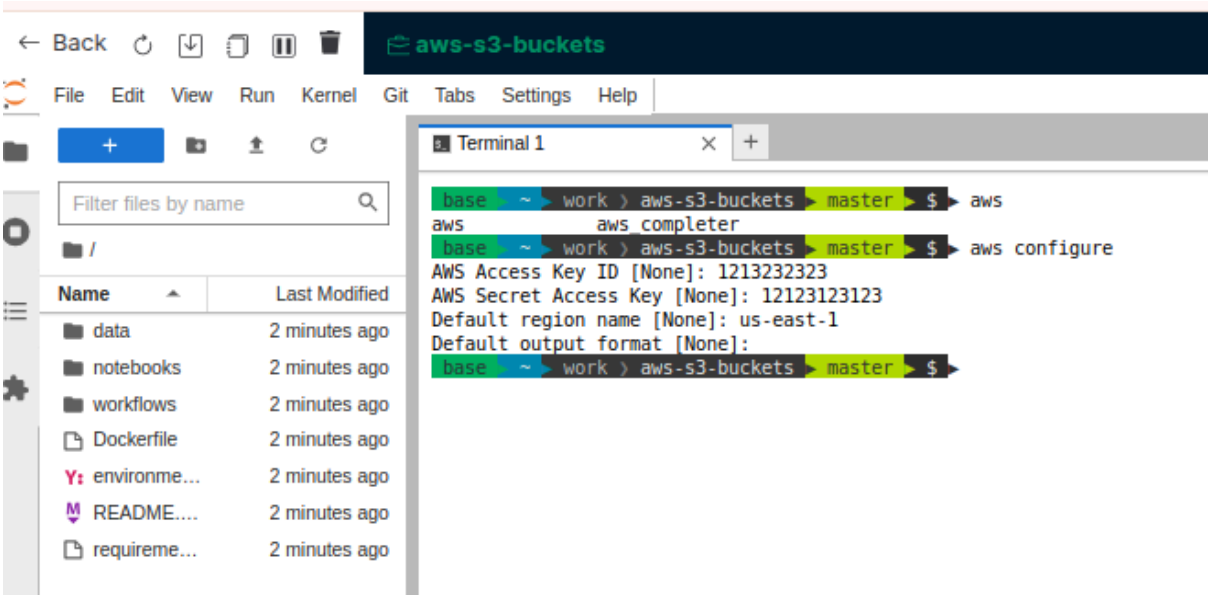



Figure 52: Configuring AWS credentials on renku container

**3. List S3 Buckets**

To list all S3 buckets in your account:

```
None
aws s3 ls
```

**4. Create an S3 Bucket**

To create a new S3 bucket:

```
None
aws s3 mb s3://your-bucket-name
```

Replace your-bucket-name with a unique name.

**5. Upload Files to S3 Bucket**

To upload a file to your S3 bucket:

```
None
aws s3 cp /path/to/your/file s3://your-bucket-name/
```

**6. Download Files from S3 Bucket**
To download a file from your S3 bucket:

```
None
aws s3 cp s3://your-bucket-name/your-file /path/to/destination/
```

By following these steps, you can effectively integrate Ubuntu with AWS S3 buckets and manage your cloud storage efficiently.

# 10. Pause / Save renku session

In this version of Renku, its possible to pause a session and come back to it later. Essentially, if a session remains inactive for a few days, it will be automatically deleted. If you attempt to pause a session, the operation will not succeed. You should always back up your code to a git repository (Gitlab.computing.dcu.ie or other).

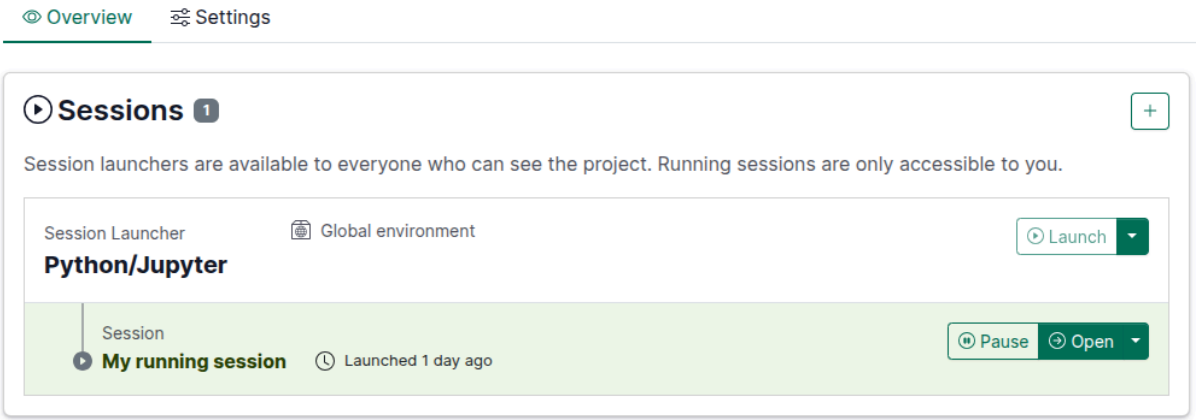



Figure 58: Pausing a Session in Renku.

# 11. Use a custom docker container in Renku

This information is adapted for DCU from this URL:

```
Shell
Documentation: How to use your own docker image for a Renku session
https://renku.notion.site/How-to-use-your-own-docker-image-for-a-session-launcher-11f0df2efafc80af848ffcaf9ccff31c
```

For more advanced setups, please refer to this URL.

Below we are only focusing on Jupyter notebooks.

Note, the image must meet **all** of the conditions below in order to work on DCU RenkuLab:

1. **The image needs a front end**
   The image you use needs a user interface web frontend to run so that you can access the session via the browser. Examples of these include JupyterLab, VSCode or RStudio.

2. **The image will be run as non-root**
   For security reasons, sessions are started with a non-root user.

# A. Clone Repo: renku-example-deep-learning-workload

The simplest approach to get going on DCU renku is to clone or fork this repo:

```
Shell
https://gitlab.computing.dcu.ie/dcollins/renku-example-deep-learning-workload
```

This repo is already configured to work with DCU renku.

At this point, if you require specific libraries or packages to be in the docker container, you can add them to the Dockerfile.

For example:

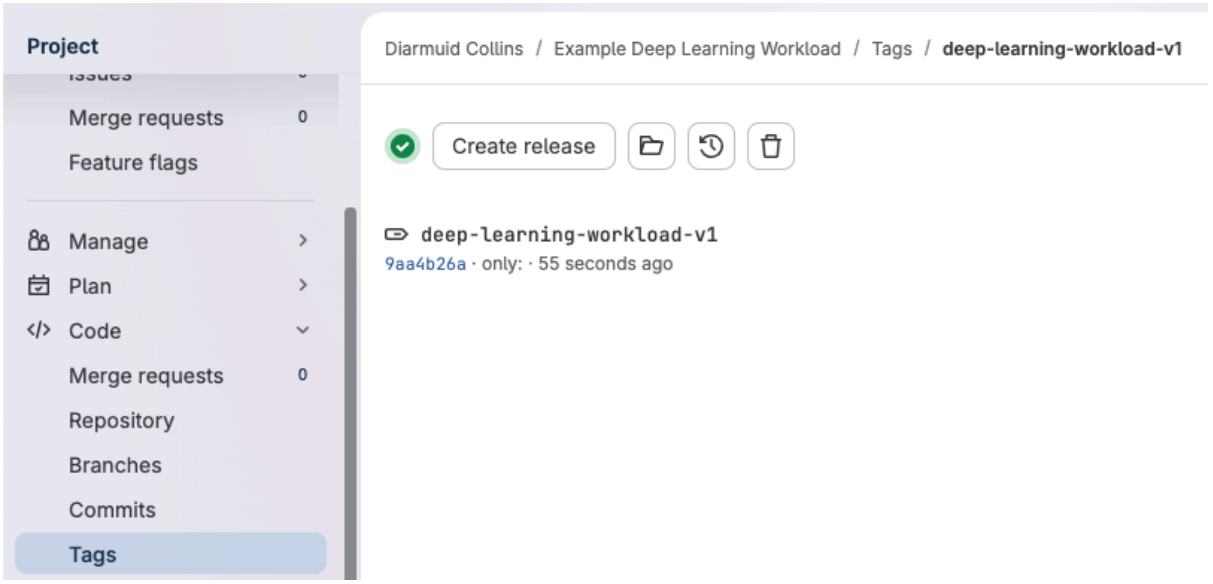
**Dockerfile** 1.62 KiB

```
1  FROM python:3.10-slim
2
3  ENV DEBIAN_FRONTEND=noninteractive
4  ENV HOME=/home/jovyan
5
6  # Install required system packages including tmux
7  RUN apt-get update && apt-get install -y \
8      bash-completion \
9      git \
10     build-essential \
11     wget \
12     gcc \
13     make \
14     perl \
15     libssl-dev \
```

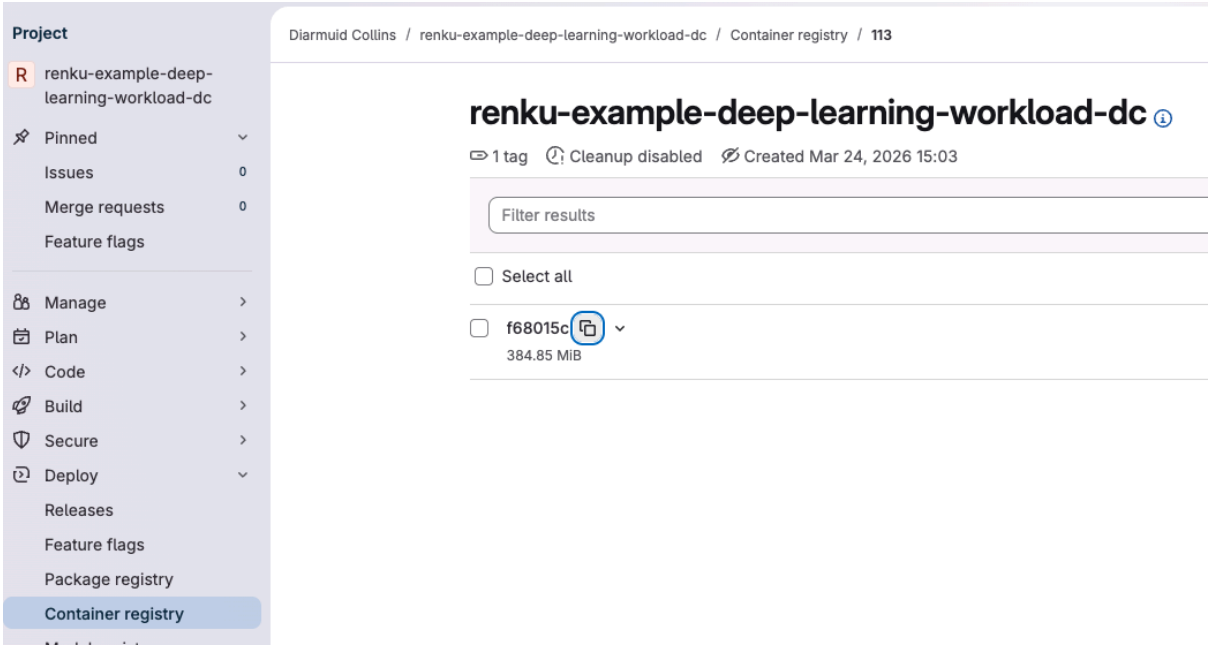
# B. Trigger the Build

In GitLab, creating a **Git Tag** does not automatically trigger a **Docker Build** unless you have a CI/CD pipeline configured to do so.

Create a new tag (e.g., `v1.0.1`) under the code menu option, and then select "*Create Release*". GitLab will see the new tag and release, and will run the pipeline. You'll see the image appear in **Deploy > Container Registry** after a few minutes.




Once the container has been compiled, go to Deploy > Container Registry and copy the container image path.



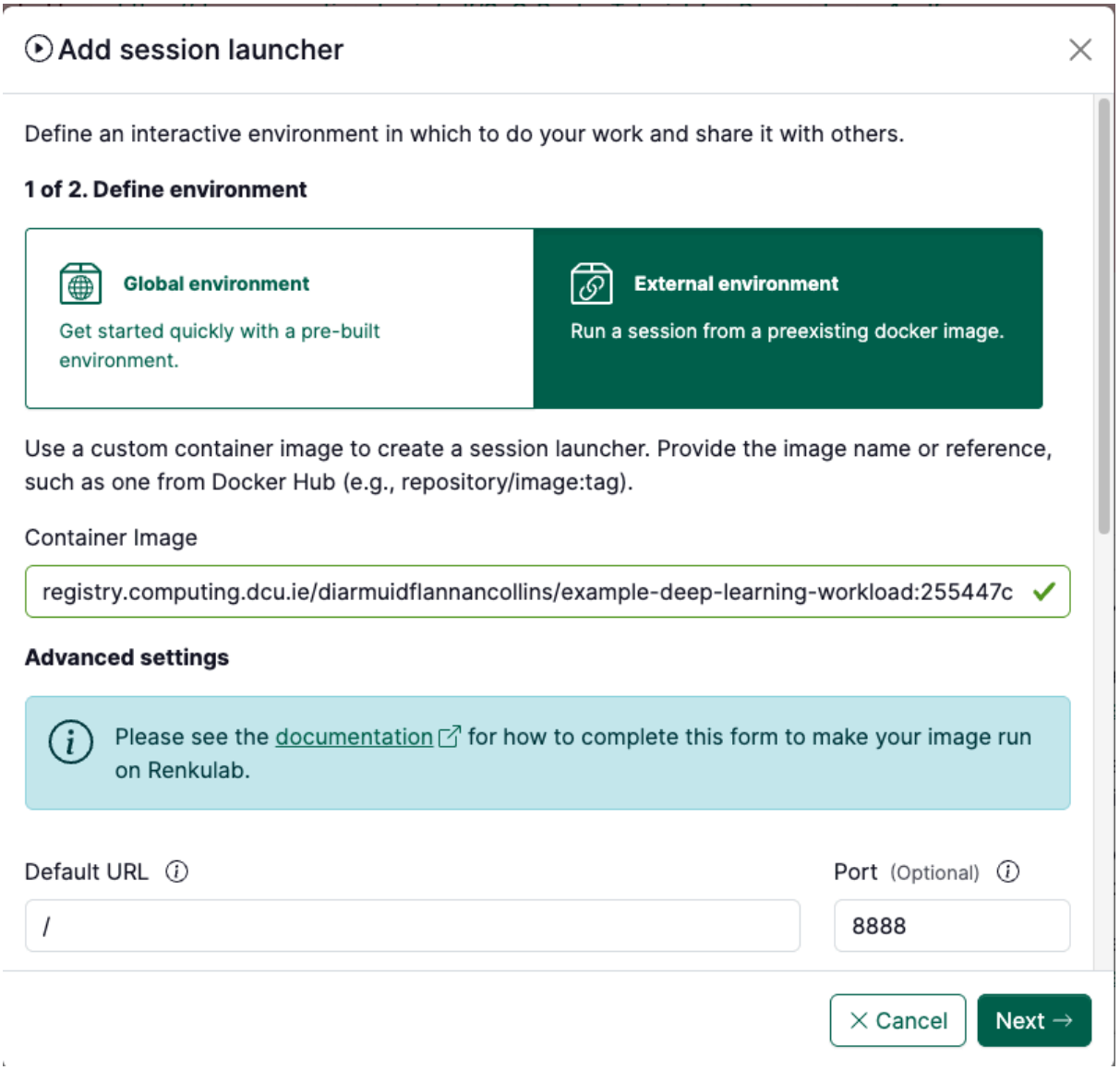

The container image path should resemble this:

Shell

```
registry.computing.dcu.ie/username/renku-example-deep-learning-workload-dc:f68015c
```

# C. In Renku

Login into renku. Click create project. Then Add a Session. You will see the following screen. Configure Renku Launcher.




## In Advanced Settings:

### Container Image

In the *container image* field, copy in the container image path from above:

```
registry.computing.dcu.ie/username/renku-example-deep-learning-workload-dc:f68015c
```

## Port

8888

## Default URL

/lab

---

## Command ENTRYPOINT

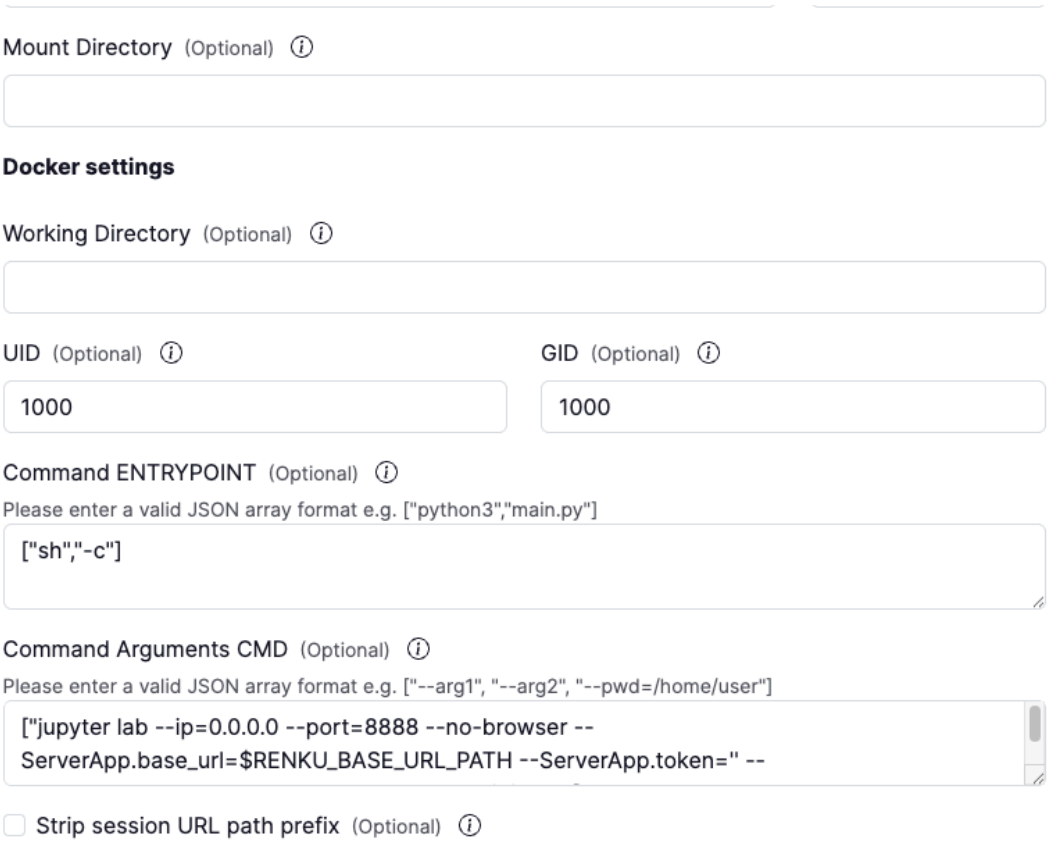
["sh", "-c"]

---

## Command Arguments

["jupyter lab --ip=0.0.0.0 --port=8888 --no-browser
--ServerApp.base_url=$RENKU_BASE_URL_PATH --ServerApp.token=''
--ServerApp.password='' --ServerApp.allow_origin='*'"]

For example:



# D. Troubleshooting Checklist

If the image still isn't showing up, check these three things:

1. **Is the Registry Enabled?** Go to **Settings > General > Visibility, project features and permissions** and make sure "Container Registry" is toggled **ON**.
2. **Runner Availability:** Ensure your project has a Runner available (check **Settings > CI/CD > Runners**). If you are using GitLab.com, this is usually handled for you.
3. **The "Deploy" Menu:** Remember that the registry isn't on the main project page. You have to click **Deploy** on the left sidebar, then **Container Registry**.

# E. Working Dockerfile for Renku (Jupyter)

Shell

```
FROM python:3.10-slim

ENV DEBIAN_FRONTEND=noninteractive
ENV HOME=/home/jovyan

# Install required system packages including tmux
RUN apt-get update && apt-get install -y \
    bash-completion \
    git \
    build-essential \
    wget \
    gcc \
    make \
    perl \
    libssl-dev \
    zlib1g-dev \
    libbz2-dev \
    libgmp-dev \
    pkg-config \
    python3-dev \
    vim \
    nano \
    tmux \
    rclone \
    fuse3 \
    ocl-icd-libopencl1 \
    ocl-icd-opencl-dev \
    opencl-headers \
    clinfo \
    && rm -rf /var/lib/apt/lists/*


# Add NVIDIA OpenCL ICD
```

```
RUN mkdir -p /etc/OpenCL/vendors && \
    echo "/usr/lib/x86_64-linux-gnu/libnvidia-opencl.so.1" >
/etc/OpenCL/vendors/nvidia.icd

# Create Renku-compatible user
RUN useradd -ms /bin/bash -u 1000 jovyan

# Create Renku working directory
RUN mkdir -p /home/jovyan/work

# Install Jupyter
RUN pip install --no-cache-dir \
    jupyterlab \
    notebook

# Clone a repo
#WORKDIR /home/jovyan
#RUN git clone https://github.com/<repo>

# Compile repo
#WORKDIR /home/jovyan/repo/src
#RUN ./configure && make -s clean && make -sj$(nproc)
#RUN ./configure --enable-opencl && make -s clean && make -sj$(nproc)

#To support Mounting Google drive
#RUN sed -i 's/^#user_allow_other/user_allow_other/' /etc/fuse.conf

# Needed for FUSE inside container
#RUN mkdir -p /mnt/gdrive

# Enable bash completion for jovyan
RUN echo 'if [ -f /etc/bash_completion ]; then . /etc/bash_completion; fi' >>
/home/jovyan/.bashrc && \
    echo "export PS1='\\u@\\h:\\w$ '" >> /home/jovyan/.bashrc

# Fix permissions
RUN chown -R jovyan:jovyan /home/jovyan

USER jovyan
WORKDIR /home/jovyan/work

EXPOSE 8888
```

# 12. How2 Access Google Drive data Using Rclone on Renku

Before completing this section, you will need a custom container configuration. Please complete Section *11. Use a custom docker container in Renku*

To Get the OAuth token JSON, you can:

1. Do a one-time manual auth on one machine with rclone config, and copy the resulting ~/.config/rclone/rclone.conf file to Renku.
2. Or use rclone authorize on one machine and capture the token.

Note, this step cannot be completed on the Renku platform. You will need to install rclone locally on your machine/laptop

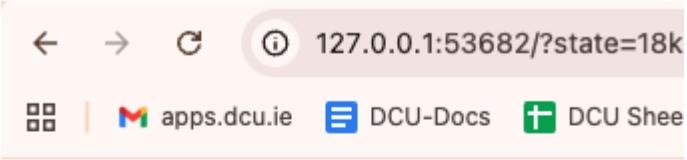
**Example:**

```shell
rclone authorize "drive"
```

This will open a browser, authenticate with Google, and return a token like:

```shell
{
 "access_token": "...",
 "token_type": "Bearer",
 "refresh_token": "...",
 "expiry": "..."
}
```

Once sign in is complete, you will see the following message in your browser.




**Success!**

---

All done. Please go back to rclone.

Copy token information to Renku

Once a token is generated, place this in the following folder on Renku:

```
Shell
~/.config/rclone/rclone.conf
```

For example:

```
jovyan@diarmuid-col-d9df571e4b30-0:~/work$ ls -la ~/.config/rclone/rclone.conf
-rw-r--r-- 1 jovyan jovyan 504 Mar 24 11:38 /home/jovyan/.config/rclone/rclone.conf
jovyan@diarmuid-col-d9df571e4b30-0:~/work$
```

Structure for ~/.config/rclone/rclone.conf:

```
Shell
[gdrive]
type = drive
scope = drive
token = {"access_token":"...","refresh_token":"...","expiry":"..."}
```

## Check Configuration from CLI

After setup, test it on renku command line with the following command:

```
Shell
rclone lsd gdrive:
```

For example, you should see something like this:

```
jovyan@diarmuid-col-d9df571e4b30-0:~/work$ vi /home/jovyan/.config/rclone/rclone.conf
jovyan@diarmuid-col-d9df571e4b30-0:~/work$ rclone lsd gdrive:
          -1 2024-03-01 09:16:23        -1 1. DCU Notes
```

## Best Approach — Sync Only a Specific Folder with Research Data

Instead of syncing the entire Google drive, just sync the folder with the required dataset:

```
Shell
rclone sync gdrive:RenkuDataset ~/RenkuDataset
```

This copies **Google Drive → to renku local** and makes the local folder a mirror.

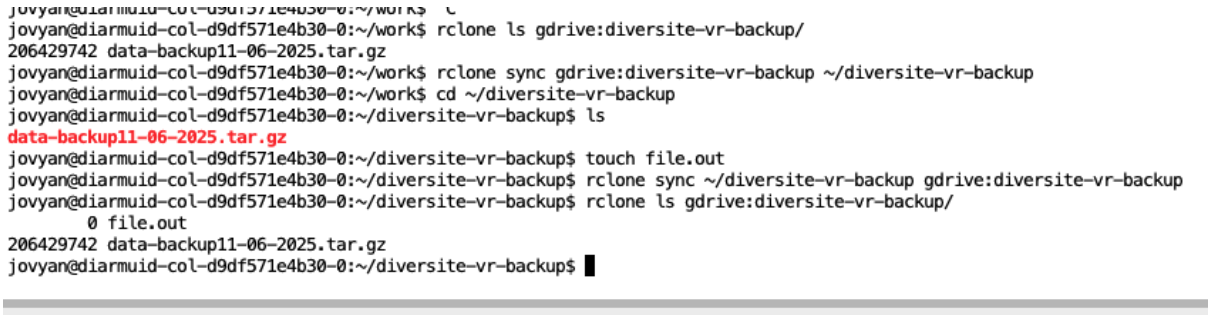
## Now, Sync Local Changes Back to Google Drive

Now you edited / added files locally and want to push them back.

You just run the **sync in the opposite direction**:

```shell
rclone sync ~/RenkuDataset gdrive:RenkuDataset
```

The following screenshot shows all the steps outlined above on Renku.



# 13. Steam data into a Kubernetes Docker Container

To stream data into a Kubernetes Docker container on a particular port, you need to set up a Kubernetes service and deployment that exposes the container's port. Here's a step-by-step guide:

1. Ensure your Docker container is running an application that listens on a specific port. For example, if your application listens on port 5000, your Dockerfile might look like Figure 54 below. Add the following to the Dockerfile:

```
None
EXPOSE 5000
```

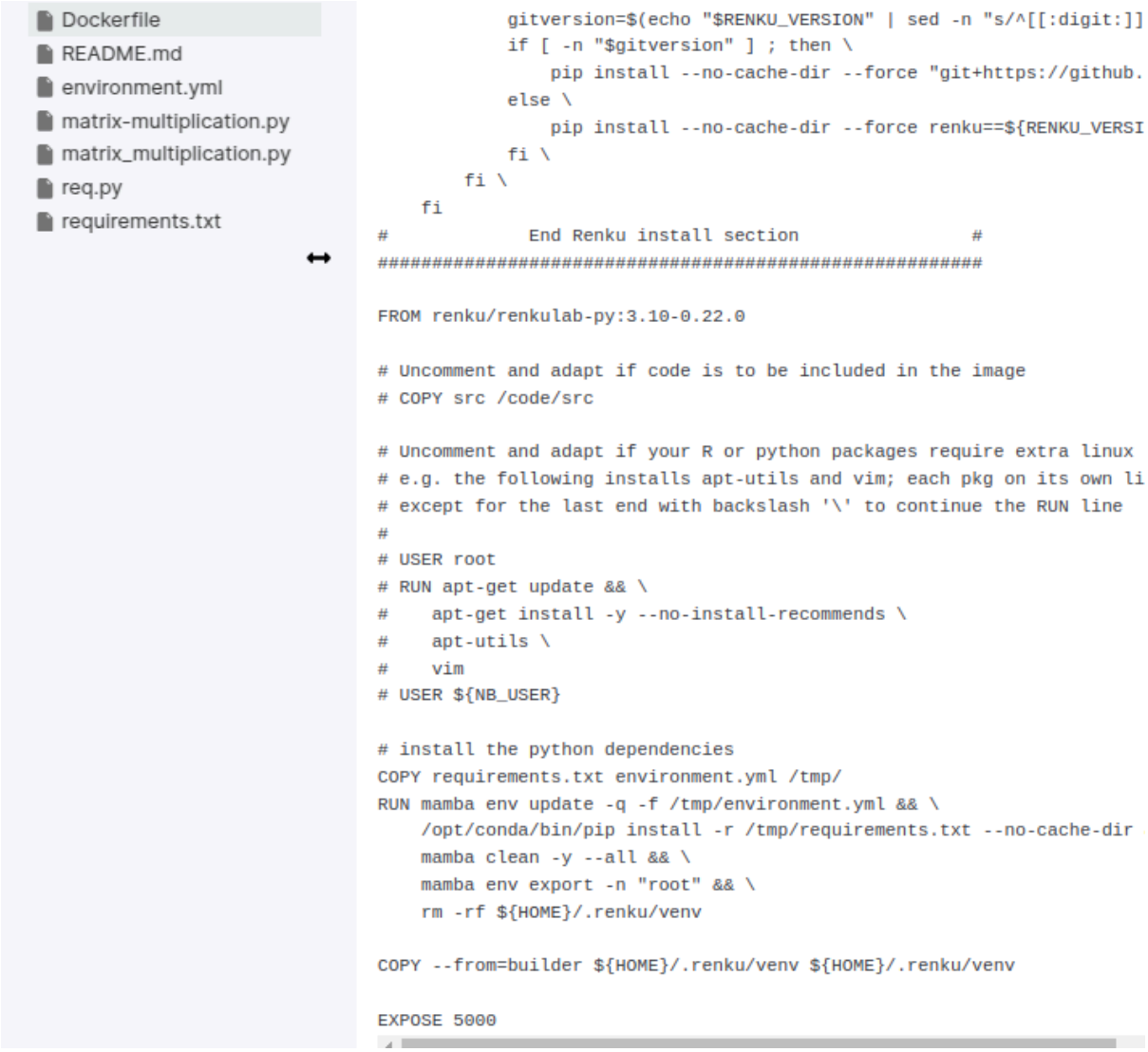



Figure 54: Note the EXPOSE 5000 in the Dockerfile

Expose the Deployment via a service. This needs to be emailed to support@computing.dcu.ie

```
None
apiVersion: v1
kind: Service
metadata:
  name: renku-service
spec:
  selector:
    app: renku
  ports:
    - protocol: TCP
      port: 5000
      targetPort: 5000
  type: ClusterIP
```

To be run on the kubernetes cluster:

```
None
kubectl apply -f service.yaml
```

Your container should be accessible within the Kubernetes cluster at port 5000.

**Expose the Service via an Ingress (Optional)**

If an ingress via a domain name to the cluster is required from external sources, the following needs to be configured.

If you have an Ingress controller, you can expose the service externally:

```
None
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: renku-ingress
spec:
  rules:
  - host: soc-gpu.computing.dcu.ie
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: renku-service
            port:
              number: 5000
```

**This needs to be emailed to support@computing.dcu.ie and they will set it up.**

# 14. Support

If researchers experience issues, they should contact the IT Support Team on *support@computing.dcu.ie* for further assistance. When submitting a support request, it is important to provide as much detail as possible to help resolve the issue efficiently. Researchers should include the following information:

- a clear description of the problem and any error messages they encounter (including the exact text of the error),
- the name of the Renku / Gitlab project or application they are working on.
- providing screenshots of the issue can be incredibly helpful in diagnosing the problem.
- If applicable, they should also note the time and date when the issue occurred, as well as any recent changes they may have made to their account or system settings.

This information will enable the support team to assist them more effectively and ensure a quicker resolution.

# 15. Troubleshooting Issues

## Issues 1: Permission denied: '/home/jovyan/.cache/torch'

```
None
"PermissionError: [Errno 13] Permission denied: '/home/jovyan/.cache/torch'"
How to change .cache location in torch?
```

You can change the cache directory for PyTorch by setting the `TORCH_HOME` environment variable to a different location. For example, if you want to set it to `/tmp/torch_cache`, you can do this:

### Temporary (for current session only)

```
None
export TORCH_HOME=/tmp/torch_cache
```

### Permanent (for all sessions)

Add the following line to your shell configuration file (e.g., `~/.bashrc`, `~/.zshrc`, or `/etc/environment` if system-wide):

```
None
export TORCH_HOME=/tmp/torch_cache
```

Then apply the changes:

```
None
source ~/.bashrc
```

### Python-only (without changing environment variables)

You can set this directly in your Python script:

```
None
import os
os.environ['TORCH_HOME'] = '/tmp/torch_cache'

import torch
print(torch.__version__)  # Just to confirm PyTorch loads with the new cache
```

This should redirect the cache directory from `/home/jovyan/.cache/torch` to `/tmp/torch_cache`.

## Issues 2: The session could not start because no image is available

If you receive the below error message, just recompile the image. Due to the volume of people using renku, we aggressively remove compiled images (after 7 days currently)

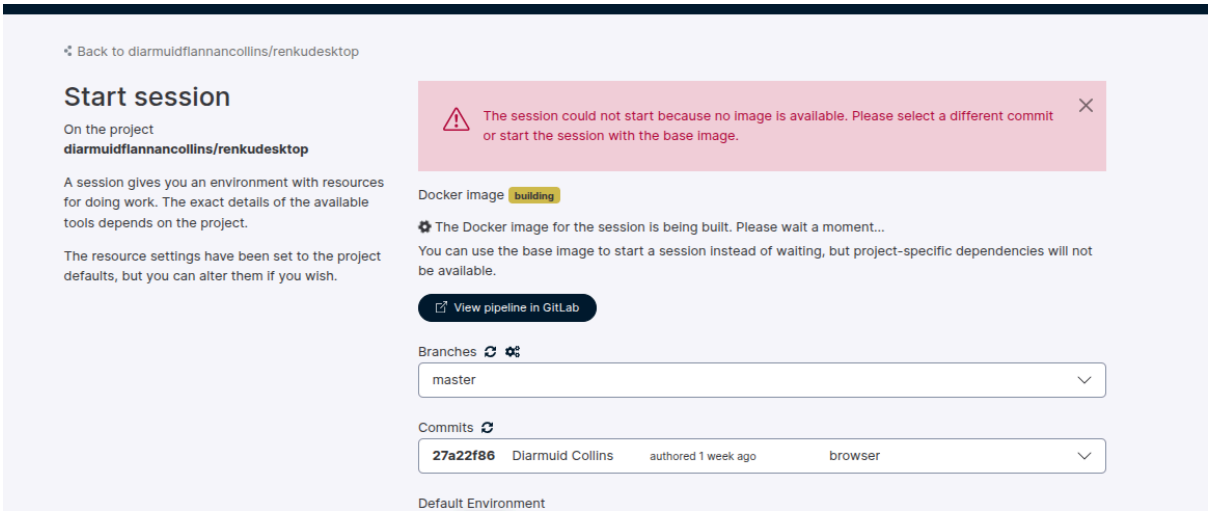
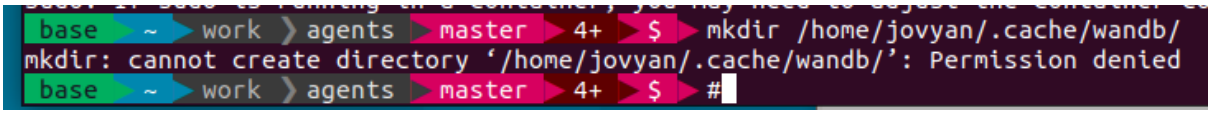



Figure 55: Recompile the image

## Issues 3: Getting “permission denied” when using /home/jovyan/.cache directory




In the Dockerfile for the project, you need to uncomment the following line after the OS installation packages has taken place:

```
None
USER ${NB_USER}
```

For example:

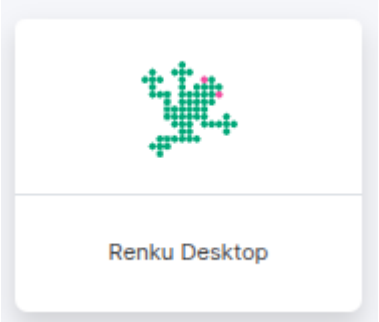
```
RUN apt-get update && apt-get install -y wget curl && \
    curl -fsSL https://ollama.com/install.sh | sh && \
    apt-get clean && rm -rf /var/lib/apt/lists/*

USER ${NB_USER}

# install the python dependencies
```

## Issues 4: I need a browser in the Docker Container

When creating a project, select the Renku Desktop option




Then, in the Docker file, insert the following code:

```
None
# Install Google Chrome and Ollama

USER root
#RUN mkdir -p /home/jovyan/.cache && chown -R jovyan:users
/home/jovyan/.cache

# Install necessary dependencies
RUN apt-get update && apt-get install -y \
    wget \
    gnupg \
    software-properties-common

# Add Microsoft Edge repository
RUN wget -q -O -
https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor
-o /usr/share/keyrings/microsoft-edge.gpg \
    && echo "deb
[signed-by=/usr/share/keyrings/microsoft-edge.gpg]
https://packages.microsoft.com/repos/edge stable main" >
/etc/apt/sources.list.d/microsoft-edge.list

# Install Microsoft Edge
RUN apt-get update && apt-get install -y microsoft-edge-stable

# Clean up
```

```
RUN apt-get clean && rm -rf /var/lib/apt/lists/*

# Set Edge as the default browser (optional)
ENV BROWSER=/usr/bin/microsoft-edge

# Run Microsoft Edge
CMD ["microsoft-edge"]

RUN apt-get update && apt-get install -y wget curl && \
    curl -fsSL https://ollama.com/install.sh | sh && \
    apt-get clean && rm -rf /var/lib/apt/lists/*

USER ${NB_USER}
```

Note, the USER ${NB_USER} directive in a Dockerfile switches the user context to ${NB_USER} for subsequent commands. ${NB_USER} is typically an environment variable that holds the username of the non-root user who will be using the container. This is a common practice in Dockerfiles to avoid running as the root user for security reasons. This is required to ensure you continue to have permissions to access files and folders in the home directory ~/

Full example file here:

None

```
# For finding latest versions of the base image see
# https://github.com/SwissDataScienceCenter/renkulab-docker
FROM renku/renkulab-vnc:3.11-0.24.0

# Uncomment and adapt if code is to be included in the
image
# COPY src /code/src

# Uncomment and adapt if your R or python packages require
extra linux (ubuntu) software
# e.g. the following installs apt-utils and vim; each pkg
on its own line, all lines
# except for the last end with backslash '\' to continue
the RUN line
```

```
#
# USER root
# RUN apt-get update && \
#    apt-get install -y --no-install-recommends \
#    apt-utils \
#    vim
#

# Install Edge Browser and Ollama

USER root

# Install necessary dependencies
RUN apt-get update && apt-get install -y \
    wget \
    gnupg \
    software-properties-common

# Add Microsoft Edge repository
RUN wget -q -O - https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor -o /usr/share/keyrings/microsoft-edge.gpg \
    && echo "deb [signed-by=/usr/share/keyrings/microsoft-edge.gpg] https://packages.microsoft.com/repos/edge stable main" > /etc/apt/sources.list.d/microsoft-edge.list

# Install Microsoft Edge
RUN apt-get update && apt-get install -y microsoft-edge-stable

# Clean up
RUN apt-get clean && rm -rf /var/lib/apt/lists/*

# Set Edge as the default browser (optional)
ENV BROWSER=/usr/bin/microsoft-edge
```

```
# Run Microsoft Edge
CMD ["microsoft-edge"]

RUN apt-get update && apt-get install -y wget curl && \
    curl -fsSL https://ollama.com/install.sh | sh && \
    apt-get clean && rm -rf /var/lib/apt/lists/*

USER ${NB_USER}

# install the python dependencies
COPY requirements.txt environment.yml /tmp/
RUN mamba env update -q -f /tmp/environment.yml && \
     /opt/conda/bin/pip install -r /tmp/requirements.txt &&
\
    conda clean -y --all && \
    conda env export -n "root"

# RENKU_VERSION determines the version of the renku CLI
#  that  will  be  used  in  this  image.  To  find  the  latest
version,
# visit https://pypi.org/project/renku/#history.
ARG RENKU_VERSION=2.9.2

############################################################
# Do not edit this section and do not add anything below

#  Install  renku  from  pypi  or  from  github  if  it's  a  dev
version
RUN if [ -n "$RENKU_VERSION" ] ; then \
        source .renku/venv/bin/activate ; \
        currentversion=$(renku --version) ; \
         if [ "$RENKU_VERSION" != "$currentversion" ] ; then
\
            pip uninstall renku ; \
```

```
                gitversion=$(echo "$RENKU_VERSION" | sed -n
"s/^[[:digit:]]\+\.[[:digit:]]\+\.[[:digit:]]\+\(\.dev[[:di
git:]]\+\)*\(+g\([a-f0-9]\+\)\)*\(+dirty\)*$/\3/p") ; \
            if [ -n "$gitversion" ] ; then \
                                        pip  install  --force
"git+https://github.com/SwissDataScienceCenter/renku-python
.git@$gitversion" ;\
            else \
                 pip install --force renku==${RENKU_VERSION}
;\
            fi \
        fi \
    fi

############################################################
```