NeuroLibre

Release v0.1

Jul 19, 2023

Author guidelines

Beta Release

Please be advised that our service is currently in its beta release of development. As a beta user, your feedback and suggestions are highly valuable in helping us identify and address any issues. We kindly request your patience and understanding as we work diligently to enhance the service based on your input.

As a registered preprint publisher, NeuroLibre goes beyond the traditional boundaries of research dissemination by offering NeuroLibre Reproducible Preprints (NRPs).

Code, data, and computational runtime are not supplementary but rather integral components of published research.

Embracing this principle, NRPs are built by seamlessly combining the outputs of your preprint's executable content with the scientific prose, all within the same execution runtime required for your analyses.

Moving from static PDFs with code and data availability statements to NRPs is the quantum leap that modern research yearns for. With NeuroLibre, we are dedicated to make that leap as easy as it gets.

Explore a published NRP

https://doi.org/10.55458/neurolibre.00004

For further details on why moving beyond static text and illustrations is a central challenge for scientific publishing in the 21st century, see the following perspective article by the NeuroLibre team (DuPre et al. 2022):

https://doi.org/10.1371/journal.pcbi.1009651

CHAPTER 1

Bird's eye view of the NRP publication workflow

NRPs are dynamic

Enabling you not only to execute NRPs directly in your web browser and regenerate the figures from the preprint, but also providing support for interactive figures and even dashboards!

This interactivity allows for a more immersive exploration of the preprint, whether by reviewers upon peer-reviewed journal submission, or by researchers who are interested in your work.

To **submit** an NRP you need to provide the following:

1. A **public code repository** that has a single or a collection of Jupyter Notebooks and/or MyST markdown files.
2. A **public data repository** needed to generate the outputs (typically figures) from the executable part of your content.
3. **Reproducible runtime configurations** recognized by BinderHub.
4. A bibtex formatted **bibliography** (paper.bib) and **author information** (paper.md).

Using your ORCID, you can login to NeuroLibre's submission portal and fill out a simple form. After **content moderation**, NeuroLibre starts a **technical screening process**, which takes place on GitHub using NeuroLibre's one-of-a-kind editorial workflow, powered by the OpenJournals.

NRPs are FAIR

NeuroLibre archives all the necessary reproducibility assets required to successfully build an NRP at the time of submission.

NeuroLibre's production BinderHub, container registry, data storage, and web servers are reserved for the published NRPs, ensuring the long-term functionality and accessibility.

This makes NRPs Findable, Accessible, Interoperable, and Reusable.

During the technical screening process, our editorial bot RoboNeuro and a screener works with you to ensure a successful build of your NRP on NeuroLibre test servers.

After a successful build, the following reproducibility assets are transferred from our preview server (public) to our production servers (reserved for published NRRs only) and archived individually on Zenodo:

1. Docker image
2. Dataset (unless already archived)
3. Repository (version cut at the latest successful build)
4. Built NRP (HTML pages of the executable book)

Each of these reproducibility assets is attributed to every author on the NRP, and they are assigned a DOI (Digital Object Identifier).

Once the archival process is complete, a summary PDF is generated. This PDF is necessary to officially register NRPs as preprints.

More than another Binder/Jupyter instance

In addition to providing a seamless publication workflow for reproducible preprints and officially registering them for discoverability of scholarly content, NeuroLibre overcomes an important reproducibility roadblock. Because absent the bundling of reproducibility assets within a dedicated publication framework:

- Public container registries wipe out images.
- Future attempts to build the same images (like Binder) frequently stumble over version clashes, resulting in failures.
- Unless cached on the same server where JupyterHub runs and archived, data often slips into the realm of inaccessibility.

As a next-generation publisher, NeuroLibre ensures that your preprint retains its reproducibility prowess. With our robust framework, we preserve and safeguard all necessary assets, leaving no room for disappearing images, version woes, or elusive data. Rest easy, knowing your preprint remains reproducible and readily available to all.

All the archived reproducibility assets, cited references, and the link to the reproducible preprint are resource linked to the DOI assigned by NeuroLibre upon publication (DOI: 10.55458/neurolibre).

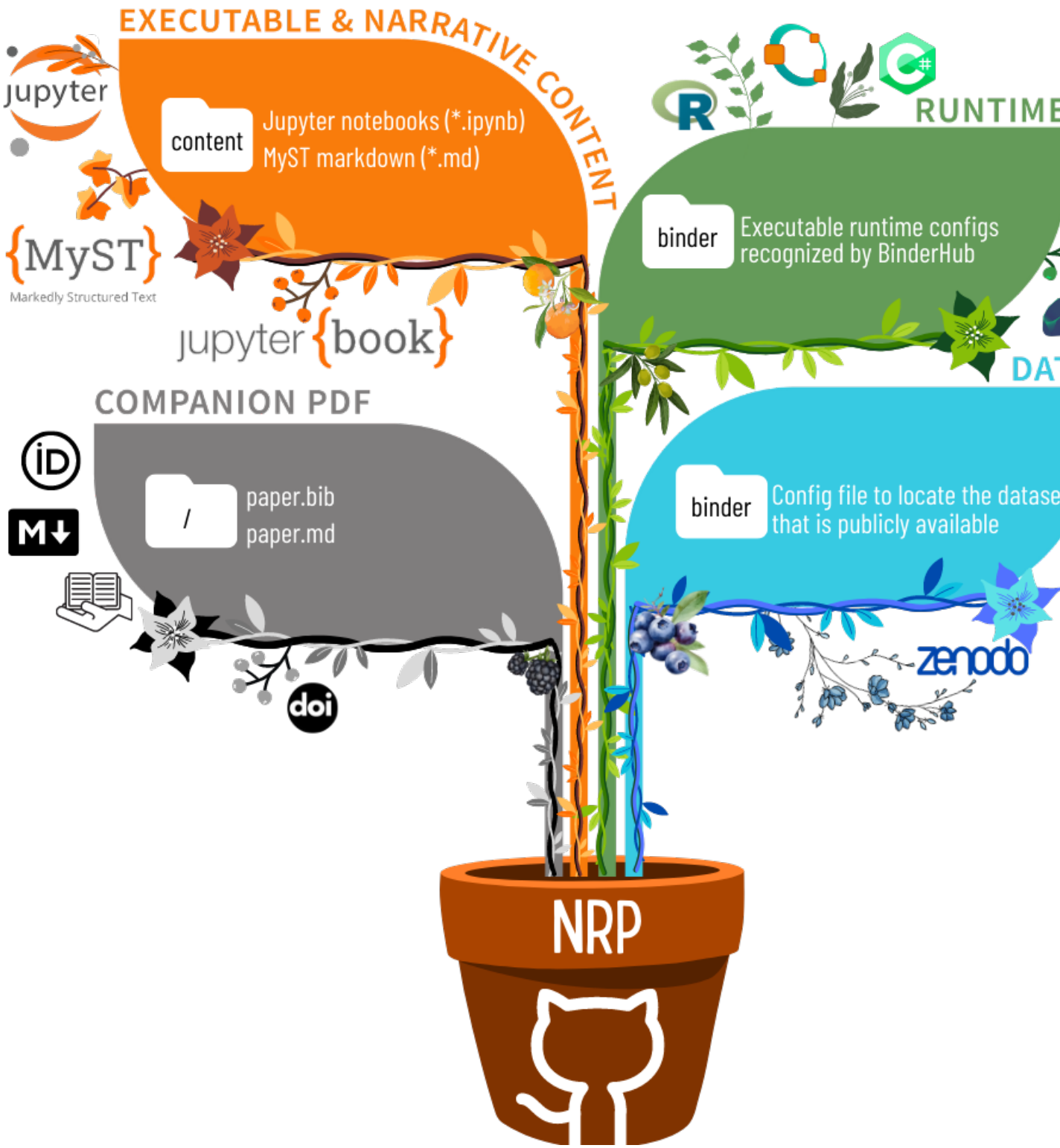
Similar to that in traditional preprint repositories (e.g. arXiv), NeuroLibre updates metadata relationship to an Author Accepted Manuscript (AAM) or Version of Record (VoR) after your article has been accepted for publication by a journal, following the peer review process and any revisions requested by the reviewers or editors.

1.1 Contributions are welcome!

NeuroLibre is fully open-source and draws its strength from community-developed tools such as BinderHub and Open Journals. You can find more information under our github organization.

1.1.1 Structure your NRP repository

Scholarly publishing has evolved from the clunky days of typewriters and snail mail, to the digital age of electronic word documents. The next step of the evolution takes root from a **GitHub repository** behold the NeuroLibre Reproducible Preprints (NRP)!



The illustration above is a concise overview of the key components required to bring an NRP to life from a public GitHub repository.

Prepare your NRP

The following sections provide details on the expected layout of an NRP repository that lives on GitHub.

The content folder

To provide a powerful, flexible, and interactive way to create your preprint, NRPs are based on the Jupyter Book.

When building the Jupyter Book for an NRP (which is a compact website), NeuroLibre expects locating your **Jupyter Notebooks** and/or **MyST Markdown** files within a folder named content.

Reference documentations

Inside the content directory, you have the freedom to organize the `SOURCE` files as per your preference:

```
root/
├─ content/
│   ├─ _toc.yml                  [REQUIRED]
│   ├─ _config.yml               [REQUIRED]
│   ├─ _neurolibre.yml           [OPTIONAL]
│   ├─ my_notebook.ipynb         [SOURCE]
│   ├─ my_myst.md                [SOURCE]
│   ├─ MY FOLDER
│   │   ├─ another_notebook.ipynb [SOURCE]
```

The relationship between the source files and the table of contents of your NRP must be defined in the `content/_toc.yml` file, as it is a `REQUIRED` component.

Another `REQUIRED` component is the `content/_config.yml` to customize the appearance and behavior of your Jupyter Book.

Supported programming languages

NRPs, being part of the Jupyter ecosystem, offer the flexibility to utilize a wide range of programming languages, provided they do not require a license (e.g., MATLAB is not supported yet, but you can use Octave).

You can take advantage of any language that has a compatible kernel listed in the Jupyter kernels for writing the executable content of your NRP.

Another important consideration is to ensure that BinderHub configurations support the language of your choice, or you know how to create a Dockerfile to establish a reproducible runtime environment. Further detail on this matter is provided in the following (green) section.

Managing citations and bibliography in your reproducible preprint

To cite articles in your reproducible preprint, include your bibtex formatted bibliographic entries in a `paper.bib` file located at the root of your repository, which is the same bibliography used by the companion PDF.

To point the Jupyter Book build to the relevant bibliography, add the following to the `content/_config.yml` file:

```
bibtex_bibfiles:
   - ../paper.bib
sphinx:
   config:
      bibtex_reference_style: author_year
```

For further details regarding the management of citations and bibliography in Jupyter Book, please see the reference docs.

Reproducible preprint in disguise (traditional article layout)

If you prefer your reproducible preprint to have a layout resembling a traditional article — single page and without sidebars — you can achieve this by creating your content in a single Notebook or MyST markdown file. Additionally, include a `content/_neurolibre.yml` file with the following content:

```
book_layout: traditional
single_page: index.ipynb
```

See an example of an NRP that combines the appearance of a traditional article with the powerful features of a Jupyter Book.

Make the most of your NRP with interactive visualizations

We strongly recommend incorporating interactive visualizations, such as those offered by plotly, to enhance the value of your NRP.

By utilizing interactive visualizations, you can fully leverage the potential of your figures and present your data in a more engaging and insightful manner.

You can visit the reference JupyterBook documentation to have your interactive outputs rendered in your NRP.

The binder folder (runtime)

One of the essential features of NRPs is the provision of dedicated BinderHub instances for the published preprints. This empowers readers and researchers to interactively explore and reproduce the findings presented in the NRP through a web browser, without installing anything to their computers.

By leveraging NeuroLibre's BinderHub, each NRP receives its isolated computing environment, ensuring that the code, data, and interactive elements remain fully functional and accessible.

The NRP repository's binder folder contains all the essential runtime descriptions to tailor such isolated computing environments for each reproducible preprint.

How to setup your runtime

To specify your runtime and set up the necessary configuration files for your runtime environment, please refer to the binderhub configuration files documentation.

To implement this in your NRP repository, create a binder folder and place the appropriate configuration files inside it according to your runtime requirements. These configuration files will define the environment in which your preprint's code and interactive elements will run when accessed through NeuroLibre's BinderHub.

NeuroLibre specific dependencies

As we build a Jupyter Book for your NRP in the exact same runtime you defined, we need the following Python dependencies to be present. For example, in a `binder/_requirements.txt` file:

```
repo2data>=2.6.0
jupyter-book==0.14.0
```

We recommend not using `jupyter-book` versions newer than `0.14.0` as of July 2023.

Currently, we are using `repo2data` to download the dataset needed to run your executable content. For details, please see the following (blue) section.

Example runtime environments

You can explore the binder-examples GitHub organization to find useful examples of configuration files.

Moreover, for additional insights and inspiration, you can visit the roboneurolibre GitHub organization to explore various NRP repositories. Observe how each preprint defines its runtimes and customizes the Binder environment to suit their research needs.

Ensuring reproducibility and resource allocation in NRPs

As of July 2023, each NRP Jupyter Book build is allocated the following resources:

- 8 hours of execution time
- 1 or 2 CPUs at 3GHz
- 6GB of RAM

Please note that the Jupyter Book build (`book build`) occurs only after a successful `runtime build` (BinderHub). The resource allocations mentioned above apply specifically to the `book build`.

Understanding the distinction between the `runtime build` and `book build` is crucial for adhering to reproducible practices.

It is strongly advised NOT to download external dependencies during the book build, as NeuroLibre cannot guarantee their long-term preservation. As a best practice, all runtime dependencies should be handled during the `runtime build` using the BinderHub configuration files.

The binder folder (data)

NeuroLibre Reproducible Preprints (NRPs) aim to distill your analysis into reproducible insights. One of the core requirements for achieving this goal is to have access to the dataset used in the analysis.

Currently, we utilize a work-in-progress tool called `repo2data` to facilitate the downloading of your dataset to our servers and to associate it with the NRP you are building. To locate the necessary information, NeuroLibre searches for the `binder/data_requirement.json` file.

Content of the data_requirement.json

Currently, repo2data is compatible with public download URIs from the following providers:

- Google Drive
- Amazon S3
- OSF
- Zenodo
- Datalad

Data will not be downloaded if the URL is not from one of the providers above.

```
{ "src": "https://download/url/of/the/dataset",
 "dst": "/location/of/the/data/relative/to/the/binder/folder",
 "projectName": "unique_project_name"}
```

The `dst` field above is not considered when your data is downloaded to the NeuroLibre servers. On the server-side, data is set to be available at the `data/unique_project_name` directory, where the `data` folder is (read-only) mounted to the root of your repository, i.e. next to the `binder` and `content` folders.

Therefore, the `dst` key is only important when you are testing your notebook locally. For example, if your `data_requirement.json` is the following

```
{ "src": "https://...",
 "dst": "../../",
 "projectName": "my_nrp_data"}
```

then repo2data will download the data in a folder named `data/my_nrp_data` that is next to the folder that contains your repository, as two upper directories correspond to that location.

Nevertheless, you don't have to manually identify the folder location. Instead, you can use the following pattern in Python:

```
from repo2data.repo2data import Repo2Data
import os
data_req_path = os.path.join("..","..", "binder", "data_requirement.json") # Change␣
↪with respect to the location of your notebook
repo2data = Repo2Data(data_req_path)
data_path = repo2data.install()[0]
my_data = os.path.join(data_path,'my_data.nii.gz')
```

In the example above, the notebook that uses repo2data is under the `content/00/my_notebook.ipybn`. Consequently, the `data_requirement.json` was located in two directories above.

After being downloaded to the server, any subsequent attempts to re-download the data will be disregarded unless modifications are made to the `data_requirement.json` file.

Data allocation

As of July 2023, each NRP is allowed to:

- use up to 10GB of data (to be downloaded from a trusted source)
- around 8GB of runtime storage (derivatives generated after executing your book)

If you are sharing a compressed file archive (e.g., zip)

Please ensure that the parent directory is not included in the archive. Otherwise, when the data is automatically extracted, repo2data will not be able to locate the actual content.

To achieve this on osx:

```
cd /to/your/data/directory
zip -r my_data.zip . -x ".*" -x "__MACOSX"
```

Similarly on Ubuntu:

```
cd /to/your/data/directory && zip -r ../my_data.zip .
```

Using Google Drive to share your NRP data

Please make sure that your Drive folder (or the zip file) is publicly available, then locate your project ID (a complex array of 33 characters that you can find on the url).

Then you can construct the download url with that ID: `https://drive.google.com/uc?id=${PROJECT_ID}`

Example `data_requirement.json`:

```
{ "src": "https://drive.google.com/uc?id=1_zeJqQP8umrTk-evSAt3wCLxAkTKo0lC",
"dst": "./data",
"projectName": "my_data_in_gdrive"}
```

Using Datalad to share your NRP data

If the `src` is provided with a URI that ends in `.git`, Repo2Data will then use the `datalad` to download the data.

```
{ "src": "https://github.com/OpenNeuroDatasets/ds000005.git",
"dst": "./data",
"projectName": "repo2data_datalad"}
```

Using S3 to share your NRP data

If the `src` url starts with `s3://`, Repo2Data will use `aws s3 sync --no-sign-request` to download your data.

```
{ "src": "s3://openneuro.org/ds000005",
"dst": "./data",
"projectName": "repo2data_s3"}
```

Using OSF to share your NRP data

Repo2Data uses `osfclient osf -p PROJECT_ID clone` command. You will need to provide the link to the public project containing your data https://osf.io/.../:

```
{ "src": "https://osf.io/fuqsk/",
"dst": "./data",
"projectName": "repo2data_osf"}
```

If you need to download subsets from a larger a project, you can achieve this using the remote_filepath field which runs `osf -p PROJECT_ID fetch -f file` command. For example:

```
{ "src": "https://osf.io/fuqsk/",
"remote_filepath": ["hello.txt", "test-subfolder/hello-from-subfolder.txt"],
"dst": "./data",
"projectName": "repo2data_osf_multiple"}
```

Using Zenodo to share your NRP data

We also support the use of the public data repository Zenodo through "zenodo_get." Ensure that your project is public and has a DOI with the form "10.5281/zenodo.XXXXXXX."

```
{ "src": "10.5281/zenodo.6482995",
"dst": "./data",
"projectName": "repo2data_zenodo"}
```

If this is the case, please indicate dutring the submission that you already have a DOI for your dataset, so that the NeuroLibre publication workflow skips the data archival step.

The companion PDF

To publish your NRP as a preprint, a PDF is necessary. Our PDF template integrates all the reproducibility assets created at the end of a successful book build as part of the publication.

To create a PDF, two files are required: `paper.md` and `paper.bib` at the root of your NRP repository.

Authors and affiliations

The front matter of `paper.md` is used to collect meta-information about your preprint:

```
---
title: 'White matter integrity of developing brain in everlasting childhood'
tags:
   - Tag1
   - Tag2
authors:
   - name: Peter Pan
     orcid: 0000-0000-0000-0000
     affiliation: "1, 2"
   - name: Tinker Bell
     affiliation: 2
affiliations:
- name: Fairy dust research lab, Everyoung state university, Nevermind, Neverland
   index: 1
- name: Captain Hook's lantern, Pirate academy, Nevermind, Neverland
   index: 2
date: 08 September 1991
```

(continues on next page)

(continued from previous page)

```
bibliography: paper.bib
---
```

The corpus of this static document (`paper.md`) is intended for a big picture summary of the preprint generated by the executable and narrative content you provided (in the `content`) folder. You can include citations to this document from an accompanying BibTex bibliography file `paper.bib`.

Testing the PDF compilation

Add a file `.github/workflows/draft-pdf.yml` to your repository:

```
on: [push]

jobs:
paper:
   runs-on: ubuntu-latest
   name: Paper Draft
   steps:
      - name: Checkout
      uses: actions/checkout@v3
      - name: Build draft PDF
      uses: neurolibre/neurolibre-draft-action@master
      with:
         journal: neurolibre
         # This should be the path to the paper within your repo.
         paper-path: paper.md
      - name: Upload
      uses: actions/upload-artifact@v1
      with:
         name: paper
         # This is the output path where Pandoc will write the compiled
         # PDF. Note, this should be the same directory as the input
         # paper.md
         path: paper.pdf
```

Beta Release

Please be advised that our service is currently in its beta release of development. As a beta user, your feedback and suggestions are highly valuable in helping us identify and address any issues. We kindly request your patience and understanding as we work diligently to enhance the service based on your input.

1.1.2 Test your NRP

It is really important to first test your submission locally to alleviate further issues when deploying on Neurolibre server. You need to make sure that:

- All the notebooks run locally with the hardware requirements from computation and data section.
- The jupyter book builds fine locally (make sure that you are not using cache files).

Test locally

Assuming that:

- you already installed all the dependencies to develop your notebooks locally
- your preprint repository follows the NeuroLibre SUBMISSION_STRUCTURE.

You can easily test your preprint build locally.

1. Install Jupyter Book

```
pip install jupyter-book
```

2. Manage your data

Given the following minimalistic repository structure:

```
.
├── binder
│   ├── requirements.txt
│   └── data_requirement.json
├── content
│   ├── _build
│   ├── notebook.ipynb
│   ├── _config.ym
│   └── _toc.yml
└── README.md
```

Create a directory `data` at the root of the repository. Install Repo2Data and configure the `dst` from the requirement file so it points to the `data` folder.

```
pip install repo2data
```

Run `repo2data` inside your notebook and get the path to the data.

```
# install the data if running locally, or points to cached data if running on␣
↪neurolibre
data_req_path = os.path.join("..", "binder", "data_requirement.json")
# download data
repo2data = Repo2Data(data_req_path)
data_path = repo2data.install()
```

See also:

Check this example for running repo2data, agnostic to server data path.

3. Book build

- Navigate to the repository location in a terminal

```
cd /your/repo/directory
```

- Trigger a jupyter book build

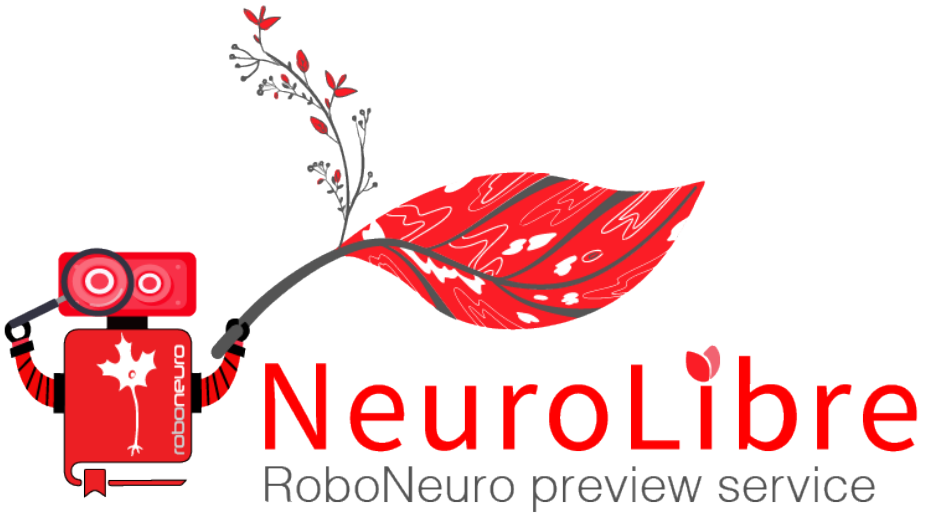
```
jupyter-book build ./content
```

See also:

Please visit reference documentation on executing and caching your outputs during a book build.

Testing on NeuroLibre servers

Meet `RoboNeuro`! Our preprint submission bot is at your service 24/7 to help you create a NeuroLibre preprint.



We would like to ensure that all the submissions we receive meet certain requirements. To that end, we created the RoboNeuro preview service, where you point RoboNeuro to a public GitHub repository, enter your email address, then sit back and wait for the results.

Note: RoboNeuro book build process has two stages. First, **it creates a virtual environment** based on your runtime descriptions. If this stage is successful, then it proceeds to build a Jupyter Book by **re-executing your code** in that environment.

- On a successful book build, we will return you a preprint that is beyond PDF!
- If the build fails, we will send two log files for your inspection.

Warning: A successful book build on RoboNeuro preview service is a prerequisite for submission.

Please note that RoboNeuro book preview is provided as a public service with limited computational resources. Therefore we encourage you to build your book locally before requesting our online service. Instructions are available in LOCAL_TESTING.

Debugging for long NeuroLibre submission

As for `mybinder`, we also provide a binder submission page so you can play with your notebooks on our servers. Our binder submission page is available here: https://test.conp.cloud.

When this process is really usefull for debugging your submission live, it can be verry long to get it. Indeed, a jupyter book build will always occur under the hood, and as part of the build process it will try to execute everything within your submission. This can make the build process very long (especially if you have a lot of long-running notebooks), and so you will end up waiting forever to get the binder instance.

If you are in a case where the jupyter book build fails on Neurolibre for whatever reason but works locally, you can bypass the jupyter book build to get the interactive session almost instantly.

Note: For example if you have "out of memory" errors on Neurolibre, you can reduce the RAM requirements on the interactive session, and try to re-run the jupyter book build directly on the fly.

Just add `--neurolibre-debug` in your latest commit message to bypass the jupyter book build (as in this git commit). Now if you register your repository on https://test.conp.cloud, you will have your binder instance almost instantly. You should be able to open a terminal session or play with the notebooks from there.

Note: This setup requires a previous valid binder build. If you are not able to build your binder, then you don't have a choice to fix the installation locally on your PC.

Warning: Please remember to remove the flag `--neurolibre-debug` when you are ready to submit, since NeuroLibre needs to build the jupyter book.

Beta Release

Please be advised that our service is currently in its beta release of development. As a beta user, your feedback and suggestions are highly valuable in helping us identify and address any issues. We kindly request your patience and understanding as we work diligently to enhance the service based on your input.

1.1.3 Submit your NRP to NeuroLibre

Before you submit

Before submitting your NRP, please make sure that your GitHub repository adheres to the *expected structure*. It is `RECOMMENDED` for the authors to test the functionality of their NRPs locally and using the roboneuro test service.

To submit your NRP:

- Login to the submission portal on https://neurolibre.org by using your **ORCID** (required).
- Click the submit button (either on the top bar or on the banner)

Submission form includes the following fields:

- `Title` Please provide the same title provided in your `content/_config.yml`.
- `Repository` Please provide the GitHub URL to your NRP repository.
- `Branch` We recommend leaving this field empty, which defaults to the `main` branch of your NRP repository, which is expected to be the most up to date before submission.
- `Software version`: If you have a release tag corresponding to the version of your submission, please indicate. Put `v1.0` otherwise.

- `Main subject of the paper`: Select `mri`, `fmri` (the list will be extended).
- `Type of the submission`: Select `New submission`
- `Preprint data DOI`: Please provide if your dataset has already been given a DOI.
- `Message to the editors`: Briefly inform our content moderators about your submission in a few sentences, please keep it short.

After your submission, the managing editor will initiate a pre-review issue in the NeuroLibre reviews repository, provided that the content moderation is successful. During the pre-review, a **technical screener** will be assigned to your submission and the "review" will be started, which is a GitHub issue on the reviews repository.

Techical screening vs peer review

Technical screening is conducted to verify the functionality of your NRP. As a preprint publisher, NeuroLibre does not assess the scientific content of the preprint.

1.1.4 Reader guidelines

This will help you navigate through a NeuroLibre prprint!

(talk about jb-book interface, chapters, binder icon etc...) (inder instance specific stuff like launching notebook if markdown, create new terminal, navigate files)

1.1.5 Reviewer guidelines

As a NeuroLibre reviewer, you are responsible for the technical quality of the resources available for our community. Neurolibre welcome submissions along two tracks of neuroscience-related material: (1) tutorials, (2) paper companions. Prior to review, an editor establishes that the submission qualifies in principles, and an administrator has made the resource available for the neurolibre binder, so you can review the material directly on our portal (the link is at the top of the `README.md` file). Now your role is to ensure the submitted materials take full advantage of the notebook format, prior to final publication. Specific criteria for review are listed below.

Technical review Criteria

Examples of high quality tutorials can be found in the scikit-learn documentation, for example this one on cross-validation. Examples of high quality article companions can be found as collab links in the article building blocks of interpretability. Specific areas for review include:

- Is the text clear and easy to read? In particular, are the sentences free of jargon?
- Are the figures properly annotated and help understand the flow of the notebook?
- Are the notebooks of appropriate lengths?
- Are the notebooks split into logical sections? Could the sections be split or merged between notebooks?
- For paper companions, is it possible to link each section of the notebook to a figure, or a section of the paper?
- Are the code cells short and readable?
- Should portions of the code be refactored into a library?

Code review

Note that you are not expected to review code libraries shipped with the notebooks. This work is better suited for other publication venues, such as the Journal of Open Source Software. Minimal feedback is encouraged in the following areas:

- is the code organized into logical folder structure?
- is the code documented?
- are there automated tests implemented?

Scientific review

Your are not expected to review the scientific soundness of the work. This step is typically handled by traditional peer-review in scientific journals. However, if a work appears to be of obvious unsufficient quality, we encourage you to contact the editors privately and suggest that the submission be withdrawn.

How to interact with authors

We encourage you to open as many issues as necessary to reach a high quality for the submission. For this purpose, you will use the github issue tracking system on the repository associated with the submission. Please assign the issues to the lead author of the submission, who will submit a pull request in order to address your comments. Review the pull request and merge it if you think it is appropriate. You can also submit a pull request yourself and ask the author to approve the changes. Please remain courteous and constructive in your feedback, and follow our code of conduct.

When you have completed your review, please leave a comment in the review issue saying so. You can include in your review links to any new issues that you the reviewer believe to be impeding the acceptance of the repository.

How to interact with editors and NeuroLibre

You can tag the editors in any of your issues. If you need to communicate privately with an editor, you can use direct messages on the mattermost brainhack forum. You can also post your questions in the ~neurolibre-reviewers channel, if you want the entire NeuroLibre community to help. Just be mindful that authors of the submission have potentially access to this public channel.

Conflict of interest

The definition of a conflict of Interest in peer review is a circumstance that makes you "unable to make an impartial scientific judgment or evaluation." (PNAS Conflict of Interest Policy). NeuroLibre is concerned with avoiding any actual conflicts of interest, and being sufficiently transparent that we avoid the appearance of conflicts of interest as well.

As a reviewer, conflict of interests are your present or previous association with any authors of a submission: recent (past four years) collaborators in funded research or work that is published; and lifetime for the family members, business partners, and thesis student/advisor or mentor. In addition, your recent (past year) association with the same organization of a submitter is a COI, for example, being employed at the same institution.

If you have a conflict of interest with a submission, you should disclose the specific reason to the submissions' editor. This may lead to you not being able to review the submission, but some conflicts may be recorded and then waived, and if you think you are able to make an impartial assessment of the work, you should request that the conflict be waived. For example, if you and a submitter were two of 2000 authors of a high energy physics paper but did not actually collaborate. Or if you and a submitter worked together 6 years ago, but due to delays in the publishing industry, a

paper from that collaboration with both of you as authors was published 2 year ago. Or if you and a submitter are both employed by the same very large organization but in different units without any knowledge of each other.

Declaring actual, perceived, and potential conflicts of interest is required under professional ethics. If in doubt: ask the editors.

Attribution

Some material in this section was adapted from the "Journal of Open Source Software" reviewing guidelines, released under an MIT license.

1.1.6 Infrastructure overview

At the bottom of our infrastructure, we rely on openstack which spawns our multiple VMs (what we will reffer later as instance) and virtual volumes. After successfull spawning of the instance, it is assigned a floating IP used to connect to it from the outside world. The cloudflare DNS then properly configure the choosen domain name under `*.conp.cloud` automatically pointing to the assigned floating IP. When the network has been properly setup, the installation can continue with kubernetes and finishes with BinderHub.

We want to share our experience with the community, hence all our installation scripts are open-source available under neurolibre/kubeadm-boostrap and neurolibre/terraform-binderhub.

Warning: NeuroLibre is still at an alpha stage of development, the github repositories will change frequently so be carefull if you use them.

You can find more details on the installation at *Bare-metal to BinderHub*.

1.1.7 Bare-metal to BinderHub

Installation of the BinderHub from bare-metal is fully automatic and reproducible through terraform configuration runned using this Docker container.

The following is intended for neurolibre backend developpers, but can be read by anyone interrested in our process. It assumes that you have basic knowledge on using the command line on a remote server (bash, ssh authentification..).

The sections *Pre-setup* and *Docker-specific preparations* should be done just the first time. Once it is done, you can directly go to the section *Spawn a BinderHub instance using Docker*.

Pre-setup

You first need to prepare the necessary files that will be used later to install and ssh to the newly spawned BinderHub instance.

We are using git-crypt to encrypt our password files for the whole process, these can be uncrypted with the appropriate `gitcrypt-key`. For the ssh authentication on the BinderHub server, you have two choices : i) use neurolibre's key (recommended) or ii) use your own ssh key.

Note: You can request the `gitcrypt-key`, `neurolibre's ssh key`, `cloudflare` and `arbutus API keys` to any infrastructure admin if authorized.

Warning: You should never share the aformentioned file to anyone.

1. Create a folder on your local machine, which is later to be mounted to the Docker container for securely using your keys during spawning a BinderHub instance. Here, we will call it `my-keys` for convenience:

```
cd /home/$USER
mkdir /my-keys
```

2. Option (i), use neurolibre's key (recommended):
   a. Simply copy the public `id_rsa.pub` and private key `id_rsa` to `/home/$USER/my-keys/`

```
cp id_rsa* /home/$USER/my-keys/
```

3. Option (ii), use your own local key:
   a. Make sure your public key and private are under `/home/$USER/.ssh` an copy it to `/home/$USER/my-keys`.

```
cp /home/$USER/.ssh/id_rsa* /home/$USER/my-keys/
```

   b. If not already associated, add your local's key to your GitHub account:
      - You can check and add new keys on your GitHub settings.
      - Test your ssh connection to your GitHub account by following these steps.
4. Finally, copy the key `gitcrypt-key` in `/home/$USER/my-keys/`.

Docker-specific preparations

You will install a trusted Docker image that will later be used to spawn the BinderHub instance.

1. Install Docker and log in to the dockerhub with your credentials.

```
sudo docker login
```

2. Pull the Docker image that encapsulates the barebones environment to spawn a BinderHub instance with our provider (compute canada as of late 2019). You can check the different tags available under our dockerhub user.

```
sudo docker pull conpdev/neurolibre-instance:v1.3
```

Spawn a BinderHub instance using Docker

To achieve this, you will instantiate a container (from the image you just pulled) mounted with specific volumes from your computer. You will be mounting two directories into the container: `/my_keys` containing the files from *Pre-setup*, and `/instance_name` containing the terraform recipe, artifacts and API keys.

Warning: The Docker container that you will run contain sensitive information (i.e. your ssh keys, passwords, etc), so never share it with anyone else. If you need to share information to another developer, share the Dockerfile and/or these instructions.

Note: The Docker image itself has no knowledge of the sensitive files since they are used just at runtime (through entrypoint command).

1. Place a `main.tf` file (see *Appendix A* for details) into a new folder `/instance-name`, which describes the terraform recipe for spawning a BinderHub instance on the cloud provider. For convenience, we suggest that you use the actual name of the instance (value of the `project_name` field in `main.tf`).

```
mkdir /home/$USER/instance-name
vim /home/$USER/instance-name/main.tf
```

Note: If you choose not to copy `main.tf` file to this directory, you will be asked to fill out one manually during container runtime.

2. Now you can copy the cloudflare `keys_cc.sh` and computecanada/arbutus `*openrc.sh` API keys.

```
cp PATH/TO/keys_cc.sh /home/$USER/instance-name/
cp PATH/TO/*openrc.sh /home/$USER/instance-name/
```

3. Start the Docker container which is going to spawn the BinderHub instance:

```
sudo docker run -v /home/$USER/my_keys:/tmp/.ssh -v /home/$USER/instance-name:/
↪terraform-artifacts -it neurolibre-instance:v1.2
```

4. Take a coffee and wait! The instance should be ready in 5~10 minutes.
5. For security measure, stop and delete the container that you used to span the instance:

```
sudo docker stop conpdev/neurolibre-instance:v1.3
sudo docker rm conpdev/neurolibre-instance:v1.3
```

If you need more information about this docker, check the neurolibre repository.

Appendix A

Here we describe the default terraform recipe that can be used to spawn a BinderHub instance, it is also available online. There are three different modules used by our terraform scripts, all run consecutively and only if the previous one succeeded.

1. `provider` populates terraform with the variables related to our cloud provider (compute canada as of late 2019):
    - `project_name`: name of the instances (will be `project_name_master` and `project_name_nodei`)
    - `nb_nodes`: number of k8s nodes **excluding** the master node
    - `instance_volume_size`: main volume size of the instances in GB **including** the master node
    - `ssh_authorized_keys`: list of the public ssh keys that will be allowed on the server
    - `os_flavor_master`: hardware configuration of the k8s master instance in the form `c{n_cpus}-{ram}gb-{optionnal_vol_in_gb}`
    - `os_flavor_node`: hardware configuration of the k8s node instances
    - `image_name`: OS image name used by the instance

- `docker_registry`: domain for the Docker registry, if empty it uses `Docker.io` by default
- `docker_id`: user id credential to connect to the Docker registry
- `docker_password`: password credential to connect to the Docker registry

Warning: The flavors and image name are not fully customizable and should be set accordingly to the provider's list. You can check them through openstack API using `openstack flavor list && openstack image list` or using the horizon dashboard.

2. `dns` related to cloudflare DNS configuration:
   - `domain`: domain name to access your BinderHub environment, it will automatically point to the k8s master floating IP
3. `binderhub` specific to binderhub configuration:
   - `binder_version`: you can check the current BinderHub version releases here
   - `TLS_email`: this email will be used by Let's Encrypt to request a TLS certificate
   - `TLS_name`: TLS certificate name should be the same as the domain but with dashes - instead of points .
   - `mem_alloc_gb`: Amount of RAM (in GB) used by each user of your BinderHub
   - `cpu_alloc`: Number of CPU cores (Intel® Xeon® Gold 6130 for compute canada) used by each user of your BinderHub

```
module "provider" {
source = "git::ssh://git@github.com/neurolibre/terraform-binderhub.git//terraform-
↪modules/providers/openstack"

project_name         = "instance-name"
nb_nodes             = 1
instance_volume_size = 100
ssh_authorized_keys  = ["<redacted>"]
os_flavor_master     = "c4-30gb-83"
os_flavor_node       = "c16-60gb-392"
image_name           = "Ubuntu-18.04.3-Bionic-x64-2020-01"
is_computecanada     = true
docker_registry      = "binder-registry.conp.cloud"
docker_id            = "<redacted>"
docker_password      = "<redacted>"
}

module "dns" {
source = "git::ssh://git@github.com/neurolibre/terraform-binderhub.git//terraform-
↪modules/dns/cloudflare"

domain    = "instance-name.conp.cloud"
public_ip = "${module.provider.public_ip}"
}

module "binderhub" {
source = "git::ssh://git@github.com/neurolibre/terraform-binderhub.git//terraform-
↪modules/binderhub"

ip               = "${module.provider.public_ip}"
domain           = "${module.dns.domain}"
```

(continues on next page)

(continued from previous page)

```
admin_user        = "${module.provider.admin_user}"
binder_version    = "v0.2.0-n121.h6d936d7"
TLS_email         = "<redacted>"
TLS_name          = "instance-name-conp-cloud"
mem_alloc_gb      = 4
cpu_alloc         = 1
docker_registry   = "${module.provider.docker_registry}"
docker_id         = "${module.provider.docker_id}"
docker_password   = "${module.provider.docker_password}"
}
```

1.1.8 Bare-metal to local Docker registry and volumes

Internet speed is the top-priority for our server. We already experienced in the past slow internet speed on Arbutus that caused us a lot of issues, specifically on the environment building phase. The binderhub was stuck at the building phase, trying in vain to pull images from `docker.io` to our server.

Note: When the notebook was successfully created, slow internet is not an issue anymore because the interaction between the user and the binder instance is not demanding.

Among many ideas, one of them that came up pretty quickly was to simply create our own local docker registry on arbutus. This would allow for low latency when pulling the images from the registry (connected to the local network where the binderhub resides).

The following documentation explains how we built our own docker registry on Arbutus, it is intended for developpers who want to spawn a new `Binderhub` on another `openstack` host. It contains also instructions on how to create volumes on `openstack` (for the `Repo2Data` databases) and attach them to the docker registry.

Note: It is still not the case, but in the future we expect the docker registry spawning to be part of the terrafrom configurations.

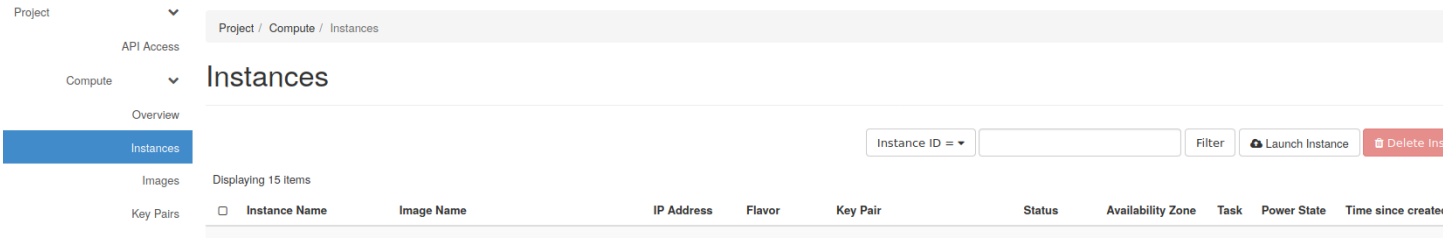
Instance spawning

The first thing to do is to create a new instance on Arbutus using `openstack`. It provides a graphical interface to interract with our openstack project from computecanada.

You will first need to log-in into the openstack dashboard.

Note: You can request the password to any infrastructure admin if authorized.

Now you can spawn a new instance under `Compute/Instances` with the `Launch Instance` button.



A new window will appear where you can describe the instance you want, the following fields are mandatories:

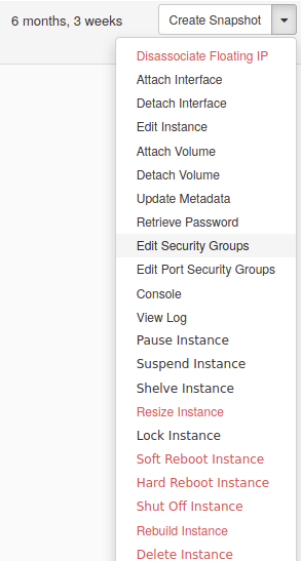
- `Instance Name`: name of the instance, choose whatever you want
- `Source`: OS image name used by the instance, select `*Bionic-x64*`
- `Flavor`: hardware configuration of the instance, `c8-30gb-186` is more than enough
- `Key Pair`: list of the public ssh keys that will be allowed on the server, find the one that match the binderhub you created in *Bare-metal to BinderHub*

Click on `Launch instance` at the bottom when you finished.

External floating IP

To access the instance from the outside, we need a `public floating IP` pointing to the instance. If you don't already have one, you can allocate a new IP under `Network/Floating IPs` and by clicking to `Allocate IP To Project`.

When it is done, click on the right of the instance under `Compute/Instances` to associate this new floating IP.



Warning: You have a limited amount of floating IPs, so be carefull before using one.

Firewall

Firewall rules will help you protect the instance against intruders and can be created on `openstack` via `Security Groups`.

1. Create a new `Security Group` under `Network/Security Groups`.
2. Click on `Manage rules` on the right and create an `IPV4` rule for all `IP Protocol` and `Port Range`, with a `Remote CIDR` from your local network.

   For example, if the internal `IP address` from your instances is in the range `192.167.70.XX`, the `Remote CIDR` would be `192.167.70.0/24`.

> **Note:** Using a `Remote CIDR` instead of `Security Group` could be considered as unsafe. But in our case it is the easiest way to allow access, since all our binderhub instances uses the same private network.

3. Enable also the ports `22 (SSH)`, `80 (HTTP)` and `443 (HTTPS)`.
4. Update the `Security Group` under `Compute/Instances`, and click on the right to select `Edit Security Groups`.

You should now have `ssh` access for the `ubuntu` user on the instance

```
ssh ubuntu@<floating_ip>
```

> **Warning:** If you cannot access the instance at this time, you should double check the public key and/or the firewall rules. It is also possible you hit some limit rate from compute canada, so retry later.

DNS specific considerations

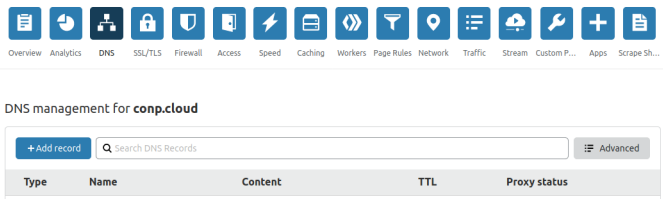
We will need to secure the Docker registry through `HTTPS` to use it with `Binderhub`, it is not possible otherwise.

The Cloudflare DNS will defined the registry domain and provide the `TLS` certificate for us.

1. Log-in to cloudflare

> **Note:** You can request the password to any infrastructure admin if authorized.

2. Under the `DNS` tab, you have the option to create a new record



3. Create an `A` record with a custom sub-domain, and the `IPV4` address pointing to the floating IP from *External floating IP*.

Volumes creation

One feature of `Neurolibre` is to provide database access to the users of the `Binderhub`, through user predefined Repo2Data requirement file. These databases are stored into a specific volume on the Docker registry instance.

In the same time, another specific volume contains all the docker images for the registry.

These volumes will be created through `openstack`.

1. Go under `Volumes/Volumes` tab
2. Click on `Create a Volume` and define the name of the volume and its storage size
3. Attach this volume to the Docker registry instance by clicking on the right of the instance under `Compute/Instances`
4. Repeat the process from (1) to (3) to create the Docker registry image volume

Once the volumes are created on `openstack`, we can `ssh` to the registry instance and mount the volumes:

1. Check that the volume(s) are indeed attached to the instance (should be `/dev/vdc`):

```
sudo fdisk -l
```

2. Now we can configure the disk to use it,

```
sudo parted /dev/vdc
mklabel gpt
mkpart
(enter)
ext3
0%
100%
quit
```

3. Check that the partition appears (should be `/dev/vdc1`):

```
sudo fdisk -l
```

4. Format the partition,

```
sudo mkfs.ext3 /dev/vdc1
```

5. Create a directory and mount the partition on it:

```
sudo mkdir /DATA
sudo chmod a+rwx /DATA
sudo mount /dev/vdc1 /DATA
```

6. Check if `/dev/vdc1` is mounted on `/DATA`
7. Repeat all the steps from (1) to (6) for the Docker registry volume (name of directory would be `/docker-registry`).

Docker registry setup

After `ssh` to the instance, install Docker on the machine by following the official documentation.

We will now secure the registry with a password. Create a directory `auth` and a new `user` and `password`:

```
mkdir auth
sudo docker run --entrypoint htpasswd registry:2.7.0 -Bbn user password > auth/
↪htpasswd
```

Create also a folder that hold the registry content (for easier backup):

```
sudo mkdir /docker-registry
```

After that you can launch the registry,

```
sudo docker run -d -p 80:80 --restart=always --name registry \
-v /docker-registry:/var/lib/registry \
-v /home/ubuntu/auth:/auth -e "REGISTRY_AUTH=htpasswd" \
-e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" \
-e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd \
```

(continues on next page)

(continued from previous page)

```
-e REGISTRY_HTTP_ADDR=0.0.0.0:80 \
-e REGISTRY_STORAGE_DELETE_ENABLED=true \
registry:2.7.0
```

Warning: `/docker-registry` is the Docker registry volume that we configured in *Volumes creation*.

Now the registry should be running, follow this documentation to test it.

You can try it on your machine (or another instance). You would first need to log-in to the Docker registry using the domain name you configure `my-binder-registry.conp.cloud` in *DNS specific considerations*:

```
sudo docker login my-binder-registry.conp.cloud --username user --password password
sudo docker pull ubuntu:16.04
sudo docker tag ubuntu:16.04 my-binder-registry.conp.cloud/my-ubuntu
sudo docker push my-binder-registry.conp.cloud/my-ubuntu
```

Note: The Docker registry can be accessed through its HTTP api. This is how you can delete images from the registry for example.

BinderHub considerations

On each k8s node (including the worker), you will also need to log-in. You may also need to add the docker config to the `kubelet` lib, so the docker registry is properly configured on you kubernetes cluster.

```
sudo docker login my-binder-registry.conp.cloud --username user --password password
cp /home/${admin_user}/.docker/config.json /var/lib/kubelet/
```

1.1.9 BinderHub test mode

This document explains how to contribute to BinderHub from a bare-metal server. If you are a Neurolibre dev, you don't need to follow *First time setup* section, just jump directly to *Code integration* section.

First time setup

Create an instance with openstack using bionic image, don't forget to assign a floating IP. After, you can ssh to this instance.

Note: You can find detailed instructions on how to create an openstack instance in *Bare-metal to local Docker registry and volumes*.

All the following should be run as root :

```
sudo su - root
```

Now install docker.

Install npm and other dependencies :

```
apt-get install libssl-dev libcurl4-openssl-dev python-dev python3 python3-pip curl␣
↪socat
curl -sL https://deb.nodesource.com/setup_13.x | sudo -E bash -
apt-get install -y nodejs
```

Install minikube for a bare-metal server.

Install kubectl.

Warning: Don't forget to let `kubectl` run commands as your own user: `sudo chown -R $USER $HOME/.kube $HOME/.minikube`.

Install binderhub repo:

```
git clone https://github.com/jupyterhub/binderhub
cd binderhub
```

You can now follow the contribution guide from step 3.

Note: Since you are in a bare-metal environment like, you don't need to use `eval $(minikube docker-env)`

You can now connect and verify the binderhub installation by accessing http://localhost:7777/.

Code integration

To make changes to the K8s integration of BinderHub, such as injecting *repo2data* specific *labels* to a *build pod*, we need to bring up a BinderHub for development.

The following guidelines are inhereted from the original BinderHub docs. This documentation assumes that the development is to be done in a remote node via `ssh` access.

1. `ssh` into the previously configured node

Note: Ask any infrastructure admin for the current binderhub debug instance, if authorized.

2. Launch shell as the root user:

   ```
   sudo su - root
   ```

3. Make sure that the following `apt` packages are installed
   - `npm`
   - `git`
   - `curl`
   - `python3`
   - `python3-pip`
   - `socat`
4. Ensure that the `minikube` is installed, if not follow these instructions.
5. Clone the `BinderHub` repo and `cd` into it:

```
git clone https://github.com/jupyterhub/binderhub
cd binderhub
```

6. Start `minikube`:

```
minikube start
```

7. Install `helm` to the minikube cluster:

```
curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get | bash
```

8. Initialize `helm` in the minikube cluster:

```
helm init
```

9. Add `JupyterHub` to the helm charts:

```
helm repo add jupyterhub https://jupyterhub.github.io/helm-chart/
helm repo update
```

   The process is successfull if you see the `Hub is up` message.

10. Install BinderHub and its development requirements:

```
python3 -m pip install -e . -r dev-requirements.txt
```

11. Install JupyterHub in the minikube with helm:

```
./testing/minikube/install-hub
```

12. Make minikube use the host Docker daemon :

```
eval $(minikube docker-env)
```

    Expect `'none' driver does not support 'minikube docker-env' command` message. This is intended behavior.

13. Run `helm list` command to see if the JupytherHub is listed. It should look like:

```
binder-test-hub 1 DEPLOYED jupyterhub-0.9.0-beta.4 1.1.0
```

Now, you are ready to start BinderHub with a config file. As done in the reference doc, start the binderhub with the config in the `testing` directory:

```
python3 -m binderhub -f testing/minikube/binderhub_config.py
```

Note: You are starting `BinderHub` with module name. This is possible thanks to the step-10 above. In that step, `-e` argument is passed to `pip` to point the local `../binderhub` directory as the project path via `.` value. This is why the changes you made in the `/binderhub` directory will take effect.

There are some details worth knowing in the `testing/minikube/binderhub_config.py` file, such as:

```
c.BinderHub.hub_url = 'http://{}:30123'.format(minikube_ip)
```

This means that upon a successful build, the BinderHub session will be exposed to `your_minikube_IP:30123`. To find out your minikube IP, you can Simply run *minikube ip* command.

The port number `30123` is described in `jupyterhub-helm-config.yaml`.

If everything went right, then you should be seeing the following message:

```
[I 200318 23:53:33 app:692] BinderHub starting on port 8585
```

Just leave this terminal window as is. Open a new terminal and do ssh forward the port `8585` to the port `4000` of your own computer by:

```
ssh -L 4000:127.0.0.1:8585 ubuntu@<floating-ip-to-the-node>
```

Open your web browser and visit `http://localhost:4000/`. BinderHub should be running here.

When you start a build project by pointing BinderHub to a GitHub repo, a pod will be associated with the process. You can see this pod by opening a *third* terminal in your computer. Do not login shell as root in the second terminal, which is used for `ssh 8585-->4000` port forwarding.

In the 3rd terminal, do the steps 1 and 2 (above), then:

```
kubectl get pods -n binder-test
```

If you injected some metadata, label etc. to a pod, you can see by:

```
kubectl get describe -n binder-test <pod_name>
```

It is expected that you'll receive a 404 response after a successful Binder build. This is because the user is automatically redirected from `8585` to the instance served at `your_minikube_IP:30123`.

If you would like to interact with a built environment, you need to forward `your_minikube_IP:30123` to another port in your laptop using another terminal.

Finally, Docker images created by Binder builds in the minikube host can be seen simply by `docker images`. If you'd like to switch docker environment back to the default user, run `eval $(docker-env -u)`.

Terminate the BinderHub running on port `8585` by simply *ctrl+c*.

To delete the JupyterHub running on minikube, first `helm list`, then `helm delete --purge <whatever_the_name_is>`.

Further tips such as using a local `repo2docker` installation instead of the one comes in a container, enabling debug logging (really useful) and more, please visit the original resource.

To see how BinderHub automates building and publishing images for helm charts, please visit the chartpress.

1.1.10 Submission workflow backend

The submission workflow backend has several components that are still not part of the terraform installation. It is divided into multiple parts: * the data server which serves the jupyter books * the python API to communicate with the bind, communicate with the API, archive and deal with DOIs. * front-end for the website design, forked from JOSS

Data server and python API

You will find all instructions in the github repo: https://github.com/neurolibre/neurolibre-data-api It has two branches: `main` for the test server and `prod` for the production server.

1.1.11 FAQ - Frequently Asked Questions

How can I test a NeuroLibre submission ?

You can test your NeuroLibre submission using our RoboNeuro preview service. Make sure to follow *Test your NRP* if you need more details.

Can I submit a non-github repository ?

We don't accept non-github submission. Still if you need the interactive binder, you can use any of those providers.

What are the hardware limits on NeuroLibre ?

Running time should take less than 8 hours to execute uncached (that includes all notebooks and book build) with or 2 CPU@3GHz. You need use less than ~7.5GB of RAM, take less than 10GB runtime storage and no more than 5GB of data.

I want to contribute, how can I do that ?

It would be a pleasure to include external people on our project, please reach out to us via our mattermost brainhack forum or #TODO:EMAIL! There are fundamentally two ways to contribute to NeuroLibre: as a reviewer or as a developper.

The reviewer team is in charge of checking if submissions execute ok on our servers, there are also exchanging with the author to help them improve it. Developer team works on the binderhub administration, backend workflows and frontend (including github integrations, and JOSS website template).

Which languages does NeuroLibre support ?

All languages supported by the jupyter ecosystem, check the following list.

What type of review are you doing ?

We are a preprint service, and so we stand for minimalistic reviews. This includes basic formatting, and checking if code executes.

How do I manage datasets with NeuroLibre ?

We use repo2data to manage input data, and cache. For more information please check the data related section.

Which version of `jupyter-book` and `repo2data` should I use ?

We always recommend latest versions for better compatibility. For `jupyter-book` you are free to use any version you like, since we rely only on the compiled artifcats. For `repo2data`, we highly advise to match latest version because this is what is used on the backend.

How can I cache my experiments ?

If you have some cache data, you can use `repo2data` to make the data available on NeuroLibre. Then follow the information about jupter-book caching here.

Can I use Dockerfile for my submission ?

As highlghted in our documentation, we don't recommend building with Dockerfiles. However if you don't have choice, you can check our section, and more specifically binder with Dockerfile instructions.