Author guidelines

1 Bird’s eye view of the NRP publication workflow
   1.1 Contributions are welcome! ................................................................. 4
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

**Author guidelines**
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.

1.1. Contributions are welcome!
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:

```yaml
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:

```yaml
book_layout: traditional
single_page: index.ipynb  # The notebook/MyST that creates the whole preprint.
```

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

Compatibility of this format with citations and bibliography has not been tested yet.

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

1.1. Contributions are welcome!
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:

```bash
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 (Binder-Hub). 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.

```json
{
    "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

```json
{
    "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:

```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)

1.1. Contributions are welcome!
• 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-evSAI3wCLxAkTKo01C",
  "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/...:
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:

```json
{
  "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.”

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

If this is the case, please indicate during 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:

```yaml
---

**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)
```

---

**1.1. Contributions are welcome!**
Please pay careful attention when editing this section as the author names, ORCID IDs, and affiliations specified here will be used for the archival of the reproducibility assets, as well as for the published preprint. Ensuring the accuracy of this information is crucial, as any inaccuracies may result in researchers not being properly attributed for their contributions.

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**

Contrary to what publishing giants may lead you to believe, creating a PDF is a relatively straightforward process. In fact, we have simplified it by utilizing GitHub actions to compile your draft PDF automatically within just a few minutes after each commit you push to your repository. This streamlined approach ensures that you can focus on your work while effortlessly generating the PDF without any manual hassle.

Add a file .github/workflows/draft-pdf.yml to your repository with the following content (also available in the template repository):

```yaml
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
```

After a successful build, the draft PDF will be available as an artifact Your Repo -> Actions -> Latest Run -> Artifacts -> paper.zip

---

**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

Prior to deployment on the Neurolibre server, it is crucial to thoroughly test your submission locally to prevent potential issues. Here are the essential steps to ensure a smooth deployment:

- **Verify Local Notebook Execution:**
  
  Test all the notebooks locally, ensuring they run seamlessly with the specified hardware requirements mentioned in the computation and data section.

- **Validate Jupyter Book Build:**
  
  Locally build the Jupyter book without relying on cache files to ensure it compiles successfully. By meticulously testing your submission locally, you can preemptively address any problems and guarantee a reliable deployment experience on the Neurolibre server. This proactive approach will save time and ensure a smoother integration into the platform.

- **Test your NRP on RoboNeuro**
  
  We provide a web service to test your NRP builds using the same infrastructure involved in the technical screening process.

#### Quick guide for local testing

With the necessary dependencies already installed for local notebook development and your preprint repository structured according to the *Structure your NRP repository*, conducting a local test of your preprint build becomes a straightforward process.

**Install Jupyter Book (prefer <= 0.14.0)**

```
pip install jupyter-book
```

**Describe your data dependencies**

Given the following minimalistic repository structure:

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>binder</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>content</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Create a folder named data at the root of the repository and don’t forget to add it to the .gitignore file to exclude it from version control:

Install Repo2Data:
pip install repo2data

Configure the dst from the requirement file (binder/data_requirement.json), so it points to the data folder you created:

```
{"src": "/url/to/the/online/source",
"dst": ".//data",
"projectName": "my_dataset"}
```

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

### Build your Jupyter Book

```
cd /your/repo/directory
jupyter-book build ./content
```

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

### See also: GitHub Actions to test book builds

You can take a look at this example GitHub Actions file to see how you can trigger a book build whenever a commit is pushed to the main branch of your NRP repository.

Please be aware that the GitHub Actions workflow’s necessary steps may vary depending on your specific requirements. It’s essential to adapt the workflow accordingly to suit your NRP’s needs.

### Testing on NeuroLibre servers

We are thrilled to present RoboNeuro, our dedicated preprint submission bot, designed to assist you in effortlessly creating NeuroLibre preprints anytime you need. With RoboNeuro at your service round the clock, preprint submission becomes a breeze!

This service allows you to evaluate whether your submission meets specific requirements before formal submission. It helps ensure that all contributions align with our guidelines and standards.
Using the RoboNeuro preview service is a straightforward process:

2. Provide the URL of your public GitHub repository.
3. Enter your email address and the GitHub commit SHA for the build.
4. Wait for the service to process your request and send you the results via email.

By leveraging the RoboNeuro preview service, we aim to enhance the quality and consistency of submitted preprints. It serves as a valuable tool for authors, allowing them to verify that their work aligns with the necessary criteria before proceeding with the formal submission process.

**Note:** The RoboNeuro book build process comprises two distinct stages:

- Firstly, it establishes a virtual environment based on your specified runtime descriptions *(runtime build)*.
- Upon successful completion of this stage, it proceeds to the second phase, where it builds the Jupyter Book by re-executing your code within that environment *(book build)*.

**On a successful book build,** we will return you a https://preview.neurolibre.org URL that serves your NRP at the provided commit SHA from which it was built.

**If the build fails,** we will send you an html file that contains the following logs:

- Binder build logs
- Book build logs
- Execution logs of individual notebooks

**A successful book build on RoboNeuro preview service makes technical screening easier.**

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 using https://robo.neurolibre.org.

**Debugging a long-running book build**

We offer https://test.conp.cloud, a Binder build page, where you can initiate a runtime build. By default, if the runtime build succeeds, it automatically proceeds to perform a book build.

However, if the execution of all your notebooks take considerably long time (say more than 20 minutes), you may want to bypass the book build phase to debug a particular notebook in the runtime environment built on our test BinderHub. This is particularly useful in cases where the Jupyter book build fails on Neurolibre but works locally.

To bypass the book build, simply include `--neurolibre-debug` in your latest commit message, as demonstrated in this git commit. Upon a successful runtime build, you can interact with your NeuroLibre Research Preprint (NRP) through the Jupyter Notebook interface in your web browser. This interface allows you to execute code cells in specific notebooks or run commands in the terminal as needed.

**Note:** If you run into 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 in the same session.

**Do not forget to remove the debug flag**

1.1. Contributions are welcome!
Once you have finished debugging your executable content on https://test.conp.cloud, make sure that your latest commit does not have the *--neurolibre-debug* in its commit message. Otherwise, following requests will not trigger a book build.

**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 *Structure your NRP repository*. It is **RECOMMENDED** for the authors to test the functionality of their NRPs locally, then using the RoboNeuro test service (https://robo.neurolibre.org).

To submit your NRP:

- Login to the submission portal on https://neurolibre.org by using your **ORCID ID** (REQUIRED).
- Click the submit button (the one on the top bar, or the fancier one 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.

**Technical 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 insufficient 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 Full-stack server components

NeuroLibre operates two servers dedicated to serving static files and API endpoints. These servers are designed to provide essential resources and functionality for various purposes, including testing, technical reviews of NeuroLibre Reproducible Preprints (NRPs), and post-publication access, where the resources are reserved for published NRPs only.
The purpose of this section is to offer an overview of each component within the NeuroLibre full-stack server. This overview will provide a solid understanding of its architecture before delving into hands-on development. It serves as a helpful guide to familiarize oneself with the system’s structure and functionalities, facilitating a smoother and more informed development process.

**Static files**

Static files are the reproducible preprint content (HTML, CSS, JS, etc.) that are generated in one of the following cases:

1. The user front-end of the RoboNeuro web application (https://roboneuro.herokuapp.com/)
2. The technical screening process on the NeuroLibre review repository (https://github.com/neurolibre/neurolibre-reviews/issues)
3. The finalized version of the reproducible preprint.

Cases 1-2 are handled on the preview server (on Compute Canada Arbutus to preview.conp.cloud), while case 3 is handled on the production server (on NeuroLibre’s own cloud to preprint.conp.cloud), both making the respective content available to the public internet.

Under the hood, we use NGINX to serve static content. To manage the DNS records for the domain conp.cloud over which NGINX serves the content, we are using Cloudflare. Cloudflare also provides SSL/TLS encryption and CDN (content delivery network, not Cotes-des-Neiges ;), a tiny Montrealer joke).

A good understanding of these concepts is essential for successfully deploying NeuroLibre’s reproducible preprints to production. Make sure you have a solid grasp of these concepts before proceeding with the deployment instructions.

**API endpoints**

An application programming interface (API) endpoint is a specific location within NeuroLibre server (e.g., preview.conp.cloud/api/books/all) that provides access to resources and functionality that are available (e.g., list reproducible preprints on this server):

- Some of the resources and functions available on the preview and production servers differ. For instance, only the production server is responsible for archiving the finalized preprint content on Zenodo, while JupyterBook builds are currently only executed on the preview server.

- On the other hand, there are common resources and functions shared between the preview and production servers, such as retrieving a reproducible preprint.

There is a need to reflect this separation between preview, production, and common tasks in the logic of how NeuroLibre API responds to the HTTP requests. To create such a framework, we are using Flask. Our Flask framework is defined by three python scripts:

```bash
full-stack-server/
  api/
    neurolibre_common.py
    neurolibre_preview.py
    neurolibre_production.py
```

Even though Flask includes a built-in web server that is suitable for development and testing, it is not designed to handle the high levels of traffic and concurrency that are typically encountered in a production environment.

Gunicorn, on the other hand, is a production-grade application server that is designed to handle large numbers of concurrent tasks. It acts as a web service gateway interface (WSGI) that knows how to talk Python to Flask. As you can infer by its name, it is an “interface” between Flask and something else that, unlike Gunicorn, knows how to handle web traffic.

1.1. Contributions are welcome!
That something else is a reverse proxy server, and you already know its name, NGINX! It is the gatekeeper of our full-stack web server. NGINX decides whether an HTTP request is made for static content or the application logic (encapsulated by Flask, served by Gunicorn).

I know you are bored to death, so I tried to make this last bit more fun:

This Flask + Gunicorn + NGINX trio plays the music we need for a production-level NeuroLibre full-stack web server. Of these 3, NGINX and Gunicorn always have to be all ears to the requests coming from the audience. In more computer sciency terms, they need to have their own daemons.

NGINX has its daemons, but we need a unix systemd (d for daemon) ritual to summon daemons upon Gunicorn. To do that, we need go to the /etc/ dungeon of our ubuntu virtual machine and drop a service spell (/systemd/neurolibre.service). This will open a portal (a unix socket) through which Gunicorn’s daemons can listen to the requests 24/7. We will tell NGINX where that socket is, so that we can guide right mortals to the right portals.

Let’s finish the introductory part of our full-stack web server with reference to this Imagine Dragons song:

```
When the kernels start to crash
And the servers all go down
I feel my heart begin to race
And I start to lose my crown

When you call my endpoint, look into systemd
It's where my daemons hide
It's where my daemons hide
Don't get too close, /etc is dark inside
It's where my daemons hide
It's where my daemons hide

I see the error messages flash
I feel the bugs crawling through my skin
I try to debug and fix the code
But the daemons won't let me win (you need sudo)
```

P.S. No chatGPT involved here, only my demons.

**Security**

On Cloudflare, we activate full(strict) encryption mode for handling SSL/TLS certification. In addition, we disable legacy TLS versions of 1.0 and 1.1 due to known vulnerabilities. With these configurations, we receive a solid SSL Server Rating of A from SSL Labs.

While implementing SSL is a fundamental necessity for the security of our server, it is not sufficient on its own. SSL only addresses the security of the communication channel between a website and its users, and does not address other potential security vulnerabilities. For example, any web server will be subjected to brute-force attacks typically coming from large botnets. To deal with this, we are using fail2ban, which is a tool that monitors our nginx log files and bans IP addresses that show malicious activity, such as repeated failed login attempts.

**What else? - Future considerations**

Another consideration is client-side certificate authorization. In this approach, clients (e.g., roboneuro) are required to present a digital certificate as part of the authentication process when they attempt to access a server or service. The server then verifies the certificate to determine whether the client is authorized to access the requested resource. This would require creating a client certificate on Cloudflare, then adding that to the server block:
Verification must be location-optional, as it works against serving static files. To achieve this only for api endpoints, the config would look like this:

```nginx
location /api/ {
    ...
    if ($ssl_client_verify != "SUCCESS") { return 403; }
    ...
}
```

This is currently NOT implemented due to potential issues on Heroku, where our web apps are hosted. Alternatively, Cloudflare provides API Shield for enterprise customers and mutual TLS for anyone.

**Performance**

For further details on tuning NGINX for performance, see these blog posts about optimizing nginx configuration and load balancing.

You can use GTMetrix to test the loading speed of individual NeuroLibre preprints. The loading speed of these pages mainly depends on the content of the static files they contain. For example, pages with interactive plots rendered using HTML may take longer to load because they encapsulate all the data points for various UI events.

### 1.1.7 Deploy and configure NeuroLibre servers

Clone this repository to home directory (typically /home/ubuntu):

```bash
cd ~
git clone https://github.com/neurolibre/full-stack-server.git
```

Be careful not to run these commands (or anything else in this section) as the root user. If you ssh’d into the VM as root, you can switch to ubuntu by executing the su ubuntu command in the remote terminal.

Throughout the rest of this section, `<type>` refers to either preview or preprint.

**Install Redis**

 Simply follow these instructions to install Redis on Ubuntu.

Our server will use Redis both as message broker and backend for Celery asynchronous task manager. What a weird sentence, is not it? I tried to explain above what these components are responsible for.

**Flask, Gunicorn, Celery, and other Python dependencies**

This documentation assumes that the server host is a Ubuntu VM. To install Python dependencies, we are going to use virtual environments.

Ensure that python3 (3.6.9 or later) is available:

```bash
which python3
```

Install virtualenv by:

1.1. Contributions are welcome!
Create a new folder (venv) under the home directory and inside that folder, create a virtual environment named neurolibre:

```
mkdir ~/venv
cd ~/venv
python3 -m venv neurolibre
```

Note: Please do not replace the virtual environment name above (neurolibre) with something else. You can take a look at the systemd/neurolibre-<type>.service configuration files as to why.

If successful, you should see ~/venv/neurolibre created. Now, activate this virtual environment to the install dependencies in the right place:

```
source ~/venv/neurolibre/bin/activate
```

If successful, the name of the environment should appear on bash, something like (neurolibre) ubuntu@neurolibre-sftp:~/venv$. Ensure that the (neurolibre) environment is activated when you are executing the following commands:

```
pip3 install --upgrade pip
pip3 install -r ~/full-stack-server/api/requirements.txt
```

You can confirm the packages/versions via pip3 freeze.

**Add working environment secret variables**

Based on the env.<type>.template file located at the /api/ folder under this repository (~/.full-stack-server/api). create a ~/.full-stack-server/api/.env file and fill out the respective variable values:

```
cp ~/full-stack-server/api/env.<type>.template ~/full-stack-server/api/.env
nano .env
```

Note, this file will be ignored by git as it MUST NOT be shared. Please ensure that the file name is correct (~/.full-stack-server/api/.env).

**Configure the server as a systemd service**

Depending on the server type [preview or preprint], copy the respective content from systemd folder in this repository into /etc/systemd/system:

```
sudo cp ~/full-stack-server/systemd/neurolibre-<type>.service /etc/systemd/system/
-neurolibre-<type>.service
```

If the python virtual environment and its dependencies are properly installed, you can start the service by:

```
sudo systemctl start neurolibre-<type>.service
```

You can check the status by

```
sudo systemctl status neurolibre-<type>.service
```
This should start multiple `gunicorn` workers, each one of them binding our flask application to a `unix socket` located at `~/full-stack-server/api/neurolibre_<type>_api.sock`. You can check the existence of the `*.sock` file at this directory. The presence of this socket file is of key importance as in the next step, we will register it to nginx as an upstream server!

Reminder: Replace the `<type>` in the commands above either with `preprint` or `preview` depending on the server (e.g., `neurolibre-preview.service`) you are configuring. Note that this is not only a naming convention, but also defines a functional separation between the roles of the two servers.

### Configure Celery as a systemd service

For Celery async task queue manager to work, there are two requirements:

1. Redis properly installed and running
2. `neurolibre-<type>.service` is up and running (see previous step)

### Preprint <-> Preview serve data sync configurations

After technical screening process, the final version of the Jupyter Book and respective data will be transferred from the preview (source) to the preprint (destination) server. At least as for the current convention. To achieve this, we preferred `rsync` that uses ssh for communication between the source and destination.

Whenever the public IP of either server changes and/or the VMs are re-spawned from scratch, please ensure that the following configuration is valid.

1. Create an ssh keypair on the destination (preprint) server `ssh-keygen -t rsa`
2. Add the public key (`*.pub`) to the `~/.ssh/authorized_keys` file in the source (preview) server. This will allow production server to pull files from the preview server.
3. Confirm that you can ssh into the source (preview) server from the destination (preprint) server `ssh -i ~/.ssh/key ubuntu@preview.server.ip`
4. Create an ssh configuration file `~/.ssh/config` on the destination (preprint) server to recognize preview (source) server as a host. The content of the configuration will be:

```ini
[Host neurolibre-preview]
  HostName xxx.xx.xx.xxx
  User ubuntu
```

Ensure that the you replaced `xxx.xx.xx.xxx` with the public IP address of the preview server. The first line of the configuration above declares the alias `neurolibre-preview`. If you change this name, you will need to make respective changes in the `neurolibre_preprint_api.py`.

1. Test file transfer. SSH into the destination (preprint) server and pull an example file from the source server:

```bash
rsync -avR neurolibre-preview:/DATA/foo.txt /
```

Provided that the `/DATA/foo.txt` exists on the source (preview) server and you successfully configured ssh, you should see the same file appearing at the same destination (directory syncing, see more here) on the destination (preprint) server.

### Cloud-level considerations

1. Contributions are welcome!
NGINX installation and configurations

To install and configure nginx:

```bash
sudo apt install nginx
```

Allow HTTP (80) and HTTPS (443) ports:

```bash
sudo ufw allow 80,443/tcp
```

Create the following folders:

```bash
sudo mkdir /etc/nginx/sites-available
sudo mkdir /etc/nginx/sites-enabled
```

**Update the nginx.conf and add neurolibre_params**

Replace the default nginx configuration file with the one from this repository:

```bash
sudo cp ~/full-stack-server/nginx/nginx.conf /etc/nginx/nginx.conf
```

Add proxy pass parameters for the upstream server that is gunicorn/flask:

```bash
sudo cp ~/full-stack-server/nginx/neurolibre_params /etc/nginx/neurolibre_params
```

**Add server-specific configuration files**

Depending on the server type [preprint or preview], copy `/nginx/neurolibre-<type>.conf` file to `/etc/nginx/sites-available`:

```bash
sudo cp ~/full-stack-server/nginx/neurolibre-<type>.conf /etc/nginx/sites-enabled/
```

Reminder: Replace the `<type>` in the commands above either with `preprint` or `preview` depending on the server (e.g., `neurolibre-preview.service`) you are configuring.

**Create SSL certificates**

- Login to the cloudflare account, got to the respective site domain (e.g. comp.cloud or neurolibre.org), under the SSL/TLS -> Origin Server -> Create Certificate.
- Use the default method (RSA 2048), leave the host names as is (or define a single subdomain, your call). Click create.
- This will create two long strings, one for certificate (first) and one for the private key. Create two files under `/etc/ssl` directory:
  - cd /etc/ssl
  - sudo nano /etc/ssl/comp.cloud.pem -> Copy the certificate key here and save
  - sudo nano /etc/ssl/comp.cloud.key -> Copy the key string here and save.
Note: `conp.cloud.pem` and `conp.cloud.key` can be changed with any alternative name, such as `neurolibre.pem` and `neurolibre.key` as long as the origin certificate content is accurate AND if your `nginx.conf` is configured to look for that new file name:

```bash
ssl_certificate /etc/ssl/conp.cloud.pem;
ssl_certificate_key /etc/ssl/conp.cloud.key;
```

Remember that the same directives also exist in the `/etc/nginx/sites-available/neurolibre-<type>.conf` configuration files, both for preview and preprint. If you decide to change the certificate name, you will need to update these configs as well.

### A tiny hack to serve swagger UI static assets over upstream

This is a bit tricky both because a funny `_` (what python gives) vs – (what nginx expects) mismatch, also because we will be serving the swagger UI over a convoluted path. When you run the flask app locally, it will know where to locate UI-related assets and serve the UI on your localhost. But when we attempt it from `https://<type>.neurolibre.org/documentation`, our NGINX server will not be able to locate them, so we help it:

```bash
sudo mkdir /etc/nginx/html/flask-apispec
```

This is required for both server types.

### Start the server

When you symlink the configuration file from `sites-available` to `sites-enabled`, it will take effect:

```bash
sudo ln -s /etc/nginx/sites-available/neurolibre-<type>.conf /etc/nginx/sites-enabled/neurolibre-<type>.conf
```

then

```bash
sudo systemctl restart nginx
```

That’s it! The server should be accessible at the domain you configured (e.g. `https://preview.neurolibre.org`)

This is required for both server types.

- Remember to use the correct name (`neurolibre-<type>.conf`) for the respective (preprint or preview) server you are configuring.
- Also, if your upstream server, i.e. the gunicorn socket, is not active, the webpage will not load. Ensure that `sudo systemctl status neurolibre-<type>.service` shows active status for the respective server.

### Newrelic installation

We will deploy New Relic Infrastructure (`newrelic-infra`) and the NGINX integration for New Relic (`nri-nginx`, source repo) to monitor the status of our host virtual machine (VM) and the NGINX server.

With these tools, we will be able to track the performance and availability of our host and server, and identify and troubleshoot any issues that may arise. By using New Relic and the NGINX integration, we can manage and optimize the performance of our system from a single location.
You need credentials to login to NewRelic portal. Otherwise you cannot proceed with the installation and monitoring.

Ssh into the VM (ssh -i ~/.ssh/your_key root@full-stack-server-ip-address) and follow these instructions:

1. Install new relic infrastructure agent

After logging into the newrelic portal, click + add data, then type ubuntu in the search box. Under the infrastructure & OS, click Linux. When you click the Begin installation button, the installation command with proper credentials will be generated. Simply copy/paste and execute that command on the VM terminal.

Alternatively, you can replace <NEWRELIC-API-KEY-HERE> and <NEWRELIC-ACCOUNT-ID-HERE> with the respective content below (please do not include the angle brackets).

```bash
curl -Ls https://download.newrelic.com/install/newrelic-cli/scripts/install.sh | bash
&& sudo NEW_RELIC_API_KEY=<NEWRELIC-API-KEY-HERE> NEW_RELIC_ACCOUNT_ID=<NEWRELIC-ACCOUNT-ID-HERE> /usr/local/bin/newrelic install
```

After successful installation, the newrelic agent should start running. Confirm its status by:

```
sudo systemctl status newrelic-infra.service
```

If the installer prompted you to add additional packages including NGINX, Golden Signal Alerts etc., you may skip the step 2. below. Nevertheless, go through the second bullet point (of step 2) to confirm successful nri-nginx installation.

1. Install new relic nginx integration

• Download the nri-nginx_*_amd64.deb from the assets of the latest (or a desired) nri-nginx release. You can get the download link by right clicking the respective release asset:

```
wget https://github.com/newrelic/nri-nginx/releases/download/v3.2.5/nri-nginx_3.2.5-1_amd64.deb -O ~/nri-nginx_amd64.deb
```

• Install the package

```
cd ~
sudo apt install ./nri-nginx_amd64.deb
```

• If the installation is successful, you should see nginx-config.yaml.sample upon:

```
ls /etc/newrelic-infra/integrations.d
```

For the next step, confirm that the stab_status of the nginx is properly exposed to 127.0.0.1/status by:

```
curl 127.0.0.1/status
```

The output should look like:

```
Active connections: 1
server accepts handled requests
126 126 125
Reading: 0 Writing: 1 Waiting: 0
```

1. Configure the nginx agent

We will use the default configuration provided in the sample configuration by copying it to a new file:
This action will start the nri-nginx integration. Run `sudo systemctl status newrelic-infra.service` to confirm successful. You should see the “Integration health check finished with success” message for `integration_name=nri-nginx`.

### Fail2ban installation and configuration

- Install

```bash
sudo apt-get install fail2ban
```

- Copy over fail2ban configurations from this repository to where they should be:

```bash
sudo cp -R ~/full-stack-server/fail2ban/* /etc/fail2ban
```

Note that these configurations assume that `/home/ubuntu/nginx-error.log` and `/home/ubuntu/nginx-access.log` are where they should be and configured as error/access logs for the nginx server.

- Activate NGINX jails:

```bash
sudo systemctl restart fail2ban.service
```

- Confirm that the service is ready:

```bash
sudo systemctl status fail2ban.service
```

- See the number and the list of jails set up:

```bash
sudo fail2ban-client status
```

- Be a responsible guardian and take a look at those jails. For example, to see the list of blocked IPs due to suspicious authentication retries:

```bash
sudo fail2ban-client status nginx-http-auth
```

You can check other jails (e.g., `nginx-noproxy`, `nginx-nonscript`, `sshd`).

- Use your get out of the jail card:

In case you trapped yourself while testing if the jail works:

```bash
sudo fail2ban-client set nginx-http-auth unbanip ip.address.goes.here
```

See this documentation for further details regarding the configurations.

### 1.1.8 Monitor, Debug, and Improve

#### Use Newrelic

Login to the NewRelic portal where you can take a look at all the entities from both `preview` and `preprint` server. These entities could be specific to NGINX or the hosts events. You can take a look at a variety of logs, and see if there are any errors or critical warnings thrown.
NeuroLibre, Release v0.1

NewRelic not only provides centralized monitoring of multiple resources, but also allows you to set alert conditions! You can install the mobile application to your iPhone/Android and get immediately notified when things are out of control.

**Know your logs**

We have several systemd services that are critical. You can use `journalctl` to take a look at what’s going on with each one of them. For example, if you need to take a look at the logs from gunicorn (through which Flask logs are forwarded):

```
journalctl -xn 20 -u neurolibre-preview.service
```

The above would help you understand what went wrong if the service failed to restart. Note that `sudo systemctl status neurolibre-preview.service` is not going to explain what went wrong at the level you expect.

Here, `-xn` is the number of last N lines of log with application context and `-u` is followed by the name of the service (e.g., `nginx.service`). For further details, see `journalctl` reference.

### 1.1.9 Hosting DashBoards with Dokku

**Create a Dokku instance**

- Create a Ubuntu VM and associate a floating IP (vm.floating.ip).
- Install Dokku to the VM by following debian installation instructions.
- Add an ssh key to the Dokku. To achieve this, you need root access (`sudo -i`). If an ssh keypair does not exist (`~/.ssh/id_rsa`), create one (`ssh-keygen`):

```
dokku ssh-keys:add <name> ~/.ssh/id_rsa
```

- Add global domain

```
dokku domains:add--global db.neurolibre.org
```

- On Cloudflare, add an A record for wildcard nested domain `*.db.neurolibre.org`. This will require total TLS to issue individual certificates for every proxied hostname (paid feature). Otherwise, SSL termination will fail.
- On Cloudflare:
  - Under the SSI/TLS -> Edge Certificates -> *.db.neurolibre.org (Type Advanced) should be active.
  - SSL/TLS encryption mode must be set to FULL
  - A record added for the floating IP address (`*.db.neurolibre.org 206.xxx.xx.xx`) must be proxied (orange cloud)
- Install letsencrypt plugin:

```
sudo dokku plugin:install https://github.com/dokku/dokku-letsencrypt.git
```

- Create a DNS edit token for zone neurolibre.org
  - Login to cloudflare
  - My Profile -> API Tokens -> Create Token
  - Next to Edit Zone DNS, click use this template
– First row: Zone - DNS - Edit
– Second row: Include - Specific zone - neurolibre.org
– Continue to summary, create token and note it down for the next step

Note: You can check supported lego providers for up-to-date environment variable names. Note that dokku letsencrypt plugin does not support _FILE suffixes to read values from a file. If you are living up to your parents’ societal expectations, your lego provider probably refers to a real logo store where you buy some interlocking plastic bricks to entertain your kids.

• Configure the plugin:
  – dokku letsencrypt:set --global email conp.dev@gmail.com
  – dokku letsencrypt:set --global dns-provider cloudflare
  – dokku letsencrypt:set --global dns-provider-CLOUDFLARE_DNS_API_TOKEN
    <add-dns-api-token-here>

Deploy a dashboard

• Now we are ready to deploy applications (PHEW). Clone a compatible repository, e.g., my-dashboard and cd into it:

    cd ~/my-dashboard
dokku apps:create my-dashboard

If your app needs, you’ll need to create service plugins at this step.

• Add git remote for your application:

    git remote add dokku dokku@[vm.floating.ip]:my-dashboard

• Deploy the application

    git push dokku <reference_branch>:master

Replace the <reference_branch> with the branch from which you want to deploy the app.

• The push command above will start the deployment. If the VHOST is not enabled by default, you may not see URL being printed at the end of the deployment. In either case, add domain for the app:

    dokku domains:add my-dashboard db.neurolibre.org

Confirm that it is enabled with:

    dokku domains:report my-dashboard

enable otherwise:

    dokku domains:enable my-dashboard

• Add certificates to the application:

    dokku letsencrypt:enable my-dashboard

• That’s it! If successful, the app should be live on https://my-dashboard.db.neurolibre.org
• Check all the resources to see if things are in order:

1.1. Contributions are welcome!
NeuroLibre, Release v0.1

```dokku report my-dashboard```

Needless to say, `my-dashboard` here is just an example name. The repository you’ll clone should have basic requirements (e.g., source code, a procfile to indicate what to execute and runtime dependency declarations such as requirements.txt, Gemfile, package.json, pom.xml, etc.) to deploy itself as an application to dokku.

**Troubleshooting**

- If the application appears to be running on the VM, but now accessible via web
  - Ensure that a proper **Security Group** is added to the instance to allow outside connections
  - Click on the instance -> Interfaces -> Under the Actions Tab -> Edit Security Groups
  - Move a suitable security group from the left panel (all) to the right panel (port)
- If there’s not a security group available
  - Network -> Security Groups -> Create

<table>
<thead>
<tr>
<th>Direction</th>
<th>Ether Type</th>
<th>Protocol</th>
<th>Port Range</th>
<th>Remote IP Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Egress</td>
<td>IPv4</td>
<td>ICMP</td>
<td>Any</td>
<td>192.168.73.30/32</td>
</tr>
<tr>
<td>Egress</td>
<td>IPv4</td>
<td>TCP</td>
<td>Any</td>
<td>192.168.73.30/32</td>
</tr>
<tr>
<td>Ingress</td>
<td>IPv4</td>
<td>TCP</td>
<td>22 (SSH)</td>
<td>0.0.0.0/0</td>
</tr>
<tr>
<td>Ingress</td>
<td>IPv4</td>
<td>TCP</td>
<td>80 (HTTP)</td>
<td>0.0.0.0/0</td>
</tr>
<tr>
<td>Ingress</td>
<td>IPv4</td>
<td>TCP</td>
<td>443 (HTTPS)</td>
<td>0.0.0.0/0</td>
</tr>
</tbody>
</table>

- Each application on Dokku will run in a container, named as “dynos” in Heroku. If you connect this VM to NewRelic (see instructions above), you can monitor each container/application/load and set alert conditions.
- **Permanent redirect from** *.dashboards.neurolibre.org** to** *.db.neurolibre.org*
  - Cloudflare -> Rules -> Page Rules
  - URL: *.dashboards.neurolibre.org/
  - Forwarding URL & 301 Permanent redirect
  - Destination URL: https://$1.db.neurolibre.org

### 1.1.10 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 successful 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 chosen domain name under *.comp. 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-bootstrap and neurolibre/terraform-binderhub.

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

You can find more details on the installation at [Bare-metal to BinderHub](#).
1.1.11 Bare-metal to BinderHub

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

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

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 unencrypted 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 aforementioned 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.

1.1. Contributions are welcome!
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 compdev/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 compdev/neurolibre-instance:v1.3
   sudo docker rm compdev/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_node1`)
   - **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

```terraform
module "provider" {
  source = "gitssh://git@github.com/neurolibre/terraform-binderhub.git//terraform-m" modules/providers/openstack"
  project_name = "instance-name"
  nb_nodes = 1
  instance_volume_size = 100
  ssh_authorized_keys = ["<redacted>"]
  os_flavor_master = "c4-30gb-83"
}
```

(continues on next page)

1.1. Contributions are welcome!
1.1.12 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 developers 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 terrafom configurations.
Instance spawning

The first thing to do is to create a new instance on Arbutus using openstack. It provides a graphical interface to interact 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.

1.1. Contributions are welcome!
Warning: You have a limited amount of floating IPs, so be careful before using one.

Firewall

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


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):

1.1. Contributions are welcome!
4. Format the partition,

```
sudo fdisk -l
```

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" \
-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 your Kubernetes cluster.

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

1.1.13 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
```

1.1. Contributions are welcome!
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 inherited 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:
10. Install BinderHub and its development requirements:

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

11. Install JupyterHub in the minikube with helm:

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

12. Make minikube use the host Docker daemon:

```bash
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 JupyterHub 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:

```bash
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:

```python
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:

```bash
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:

**1.1. Contributions are welcome!**
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.

### 1.1.14 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](https://github.com/neurolibre/neurolibre-data-api) It has two branches: `main` for the test server and `prod` for the production server.

### 1.1.15 FAQ - Frequently Asked Questions

- How can I test a NeuroLibre submission ?
- Can I submit a non-github repository ?
- What are the hardware limits on NeuroLibre ?
- I want to contribute, how can I do that ?
- Which languages does NeuroLibre support ?
- What type of review are you doing ?
- How do I manage datasets with NeuroLibre ?
- Which version of jupyter-book and repo2data should I use ?
- How can I cache my experiments ?
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 developer.

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 artifacts. For repo2data, we highly advise to match latest version because this is what is used on the backend.

1.1. Contributions are welcome!
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 jupyter-book caching here.

Can I use Dockerfile for my submission?

As highlighted 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.