Submission guidelines

1 For authors, it takes three steps: **prepare, test and submit**

2 For readers, it takes one click!

3 Contributions are welcome!
   3.1 Step 1 - Prepare submission ................................................. 8
   3.2 Step 2 - Test your submission ............................................... 10
   3.3 Step 3 - Submit ................................................................. 12
   3.4 Preprint repository structure ............................................... 12
   3.5 Local testing ................................................................. 22
   3.6 Reader guidelines ........................................................... 23
   3.7 Reviewer guidelines ......................................................... 24
   3.8 Infrastructure overview .................................................. 25
   3.9 Bare-metal to BinderHub ................................................... 26
   3.10 Bare-metal to local Docker registry and volumes ..................... 29
   3.11 BinderHub test mode ....................................................... 34
   3.12 Submission workflow backend .......................................... 37
   3.13 FAQ - Frequently Asked Questions ..................................... 37
**Warning:** NeuroLibre is at an alpha stage of development, and is not currently open for submissions.

**NeuroLibre**

*Planting the seeds of living publications*

*NeuroLibre is a preprint server for interactive data analyses.* It is tailored for publishing interactive neuroscience notebooks that can seamlessly integrate data, text, code and figures.
For authors, it takes three steps: prepare, test and submit

Our full-fledged (yet simple) submission workflow is designed to help researchers convert a collection of Jupyter Notebooks and Markdown files into publishable content.

Providing online-executable runtimes and a powerful data caching system, NeuroLibre makes a groundbreaking preprint server where you can plant the seeds of living publications.

As an author you should follow:

- **Step 1 - Prepare submission**
- **Step 2 - Test your submission**
- **Step 3 - Submit**
Chapter 1. For authors, it takes three steps: prepare, test and submit
For readers, it takes one click!

The interactive outputs (such as interactive figures) are readily available in NeuroLibre preprints. Yet, if you’d like to re-run the analyses and reproduce the outputs, our powerful cloud infrastructure is at the tip of your fingers.

See also:

To find out more about how to navigate a NeuroLibre preprint, you can visit Reader guidelines.
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.

See also:
If you are interested in contributing to NeuroLibre or to deploy one of your own, please visit developer documentation.
Warning: NeuroLibre is at an alpha stage of development, and is not currently open for submissions.

3.1 Step 1 - Prepare submission

Before submitting to Neurolibre there are several prerequisites that you must complete, those include:
3.1.1 Hosting

Github is our main hosting service to store, read and interact with any public submission. The submission, which is in fact a github repository, will need to be public.

Where our backend supports other git hosting services, our submission workflow doesn’t. Still if you are eligible, you may use any of those.

**Warning:** We are not supporting private repositories at the moment.

3.1.2 Coding language

Neurolibre is living under the jupyter notebook ecosystem, hence supporting a broad range of different coding languages. If the one you are using is inside the following list, then you should be ready to go. But please be mindful that the administrator and reviewer team mainly focuses on **python 3**, if you are using another language don’t expect fine support for debugging.

We only accept submissions which are in notebook `.ipynb` format. Though less recommended, we also support text notebook representation (like markdown) using **jupytex**.

Check the following minimalistic example of a jupyter notebook.

3.1.3 Repository layout

Our infrastructure heavily relies on binderhub for the interactive compute environment, and jupyter book for the publishing pdf artifact. The backend workflow make use of those dependencies to build your submission and make the data accessible.

We recommend you to use our template repository to give you a head-start. If you prefer to build the submission yourself, you will need to check the specific detailed section on the **Preprint repository structure**.

Either way you need to make sure that you have all the following files ready:

**See also:**

Take a look at our meta anlaysis on myelin submission which is publically available in this github repository.

3.1.4 Computation and data requirements

NeuroLibre strives to provides the best experience by giving access to lot of ressources, still we have limited hardware. Taking resources constraints into account, each submission should:

- not take more than 8 hours to execute uncached (that includes all notebooks and book build)
- run with 1 or 2 CPU@3GHz
- allocate strictly less than ~7.5GB of RAM
- have few dependencies to keep your installation light (idealy less than 1GB) and fast (<10min)
- take less than 10GB runtime storage (files generated by your scripts)
- use no more than 5GB of data (inputs, cached content), downloadable from a trusted source
**Warning:** A trusted web source is a well known dataset collection (like openneuro) or dataset fetching functions from libraries (for ex. nilearn.datasets.fetch*).

For additional information, our test server (to test and build submissions) uses Intel Xeon Gold 6248, while the production server (where the final submission lives) has Intel Xeon E5-2650.

**Warning:** NeuroLibre is at an alpha stage of development, and is not currently open for submissions.

### 3.2 Step 2 - Test your submission

Local testing is checking notebooks are running, binder and jupyter book is building. First locally, then second directly on neurolibre servers.

#### 3.2.1 Local testing

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).
- The interactive environment build correctly on [https://mybinder.org/](https://mybinder.org/).

To help you testing your book locally, you can follow the section on *Local testing*.

#### 3.2.2 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://binder.conp.cloud.

When this process is really useful for debugging your submission live, it can be very 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://binder.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.

**Warning:** NeuroLibre is at an alpha stage of development, and is not currently open for submissions.

3.2. Step 2 - Test your submission
3.3 Step 3 - Submit

**Warning:** A successful book build on RoboNeuro preview service is a prerequisite for submission. Your submission request will go through only if we can find a built preprint on our test server for your preprint repository.

Submission is as simple as:

- Filling in the short submission form at NeuroLibre web page
- Waiting for the managing editor to start a pre-review issue over in the NeuroLibre reviews repository: https://github.com/neurolibre/neurolibre-reviews

3.3.1 Technical screening

Our editorial team will start a technical screening process on GitHub to ensure the functionality of your preprint. We will contact you through email, so please check your email inbox!

**Warning:** NeuroLibre is at an alpha stage of development, and is not currently open for submissions.

3.4 Preprint repository structure

We expect to find all the submission material in a public GitHub repository that has the following structure:

```bash
├── binder
│   ├── requirements.txt
│   └── (data_requirement.json)
├── content
│   ├── _toc.yml
│   ├── _config.yml
│   ├── 01-simple_notebook.ipynb
│   └── (...)
│   └── paper.md
│       └── paper.bib
```

Our meta analysis on myelin submission which is publically available in this github repository should help you understand the layout.

**Warning:** If RoboNeuro does not see this file layout, it will fail to build the jupyter book build (but may be able to build the computing environment). Make sure that your file layout never change during runtime (especially if using a Dockerfile).
3.4.1 Quickstart: Preprint templates

To give you a head-start, we created preprint template repositories:

<table>
<thead>
<tr>
<th>Preprint</th>
<th>Programming language</th>
<th>GitHub repository</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link</td>
<td>Python</td>
<td>neurolibre/template</td>
</tr>
<tr>
<td>Link</td>
<td>C++</td>
<td>neurolibre/cpp</td>
</tr>
</tbody>
</table>

To use them, make sure to read the following steps:

1. Choose your template from the list below and create a new repository into your (or to an organization) account.
2. Follow the instructions in the README file.

The following section provides further detail about the structure of a NeuroLibre preprint repository.
3.4.2 1. The binder folder

Files in this folder specify two critical components of a NeuroLibre preprint:

1. **Runtime** dependencies (required)
2. **Dataset** needed for the analysis (optionally)

1.1 Runtime
1.1.1 Preprint-specific runtime dependencies

The execution runtime can be based on any of the (non-proprietary) programming languages supported by Jupyter. NeuroLibre looks at the binder folder to find some configuration files such as a requirements.txt (Python), R.install (R), Project.toml (Julia) or a Dockerfile.

See also:
The full list of supported configuration files is available here.

1.1.2 Environment configuration for NeuroLibre

You should try to make your environment clean and concise, that is why the preferred configuration file for NeuroLibre are the requirements.txt.

It should be small (to keep environment building and loading as short as possible), and versionnized (so your environment is fully reproducible, and cache-able).

For example this requirement is bad because it has lot of unnecessary dependencies:

```bash
numpy
scipy
jupyter
matplotlib
Pillow
scikit-learn
tensorflow
```

On the other hand, this one is concise, reproducible and will take much less time to build:

```bash
scikit-learn==0.16.1
tensorflow==2.4.0
```

**Warning:** Starting from pip 20.3, the package resolver changed its behaviour to reduce inconsistencies in software versions. As a consequence and if your submission has lot of interdependent dependencies, your build may take a while. This is typically the case if you see messages like this during the build:

```
INFO: pip is looking at multiple versions of linkify-it-py to determine which version is compatible with other requirements. This could take a while.
```

**Warning:** Make sure that your whole environment is not too big (<1GB of installed dependencies), and installation is fast (<10min). Large environments increase the binder spawn time, impact your computing performance, and takes a lot of space on our servers.

**Tip:** If your binder build fails with timeout errors, this is because your environment is too complex and slow to build. But thanks to Docker internal caching mechanism, you can still re-try to submit the same repository so it catch-up the build.
Best practices when using Dockerfiles

While Neurolibre can build a Dockerfile environment, we don’t recommend it as this can be a source of lot of errors during build. If you don’t have choice, please make sure to follow these specific instructions:

1. We recommend that you use our base image to help you build your Dockerfile for Neurolibre:

   ```
   FROM neurolibre/book:latest
   ...
   ```

2. Using a Dockerfile will tend to increase the size and complexity of your environment. Make sure to have layers (RUN command) that do not exceed 1GB to help the build and push process.

3. Keep the directory layout the same as your github repository. Modifying this layout in the Dockerfile is a high source of RoboNeuro build errors. For example, you should not:

   ```
   RUN git clone bad_layout && cd bad_layout
   WORKDIR bad_layout
   ```

4. DO NOT install and download data into the docker image, check the data section for that.

See also:

Read the Dockerfile instructions for binderhub for more information.

1.1.3 NeuroLibre dependencies

Our test server creates a virtual environment in which your content is re-executed to build a Jupyter Book. To enable this, we need some Python packages.

If you are using configuration files, we need latest version of jupyter-book in a requirements.txt file:

```
jupyter-book
jupytext
(if applicable) repo2data
```

1.2 Data

NeuroLibre offers generous data storage and caching to supercharge your preprint. If your executable content consumes input data, you need to read this section carefully. Indeed, we don’t allow data download other than through our method.

To download data, NeuroLibre looks for a repo2data configuration file: data_requirement.json. This file must point to a publicly available dataset, so it can be available during preprint runtime.

See also:

Repo2data can download data from several resources including OSF, datalad, zenodo or aws. For details, please visit the documentation.

Example preprint templates using repo2data for caching data on NeuroLibre servers:

<table>
<thead>
<tr>
<th>Download Resource</th>
<th>GitHub repository</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nilearn</td>
<td>neurolibre/repo2data-nilearn</td>
</tr>
<tr>
<td>OSF</td>
<td>neurolibre/repo2data-osf</td>
</tr>
</tbody>
</table>
**Warning:** RoboNeuro may fail downloading relatively large datasets (**exceeding 5GB**) or if the data server is too slow. This is because of some limitations, independent from us, in our software stack. If you face problems when downloading your data, please create an issue in your github repository so a Neurolibre admin can check it.

### Help RoboNeuro find your data during book build

Repo2Data downloads your data to a folder named `data`, which is created at the base of your repository.

**Note:** We suggest using repo2data locally before you request a RoboNeuro preview service. Matching this data loading convention will increase your chances of having a successful NeuroLibre preprint build, and will make your data dependency agnostic to computer.

Assuming you are running a notebook on NeuroLibre and have a requirement file as:

```json
{
    "src": "download_my_brain(data_dir=_dst);",
    "dataLayout": "neurolibre",
    "projectName": "PROJECT_NAME"
}
```

- A code cell in a `content/my_notebook.ipynb` would access data by:

  ```python
  import nibabel as nib
  import os
  img = nib.load(os.path.join('..', 'data', 'PROJECT_NAME', 'my_brain.nii.gz'))
  ```

- A code cell in a `content/01/my_01_notebook.ipynb` would access data by:

  ```python
  import nibabel as nib
  img = nib.load(os.path.join('..', '..', 'data', 'PROJECT_NAME', 'my_brain.nii.gz'))
  ```

If the data directories in your code cells are not following this convention, RoboNeuro will fail to re-execute your notebooks and interrupt the book build.

The best way to access data on Neurolibre servers is using the repo2data python api. This way all the data paths will be automatically recognized. For example if you have a notebook in `content/my_notebook.ipynb`:

```python
from repo2data.repo2data import Repo2Data
# 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()
```
3.4.3 1. The content folder

Executable & narrative content

+ Table of contents (_toc.yml)
+ Jupyter Book configuration (_config.yml)

The content folder
2.1 Executable & narrative content

NeuroLibre accepts the following file types to create a preprint that is beyond PDF:

- Jupyter Notebooks,
- MyST formatted markdown.
- Plain text markdown files.
- A mixture of all above

Warning: We don’t accept markdown files with narrative content only, that is not really beyond PDF :)

Note: You can organize your content in sub-folders.

2.1.1 Writing narrative content

Jupyter Book provides you with an arsenal of authoring tools to include citations, equations, figures, special content blocks and more into your notebooks or markdown files.

See also:
Please visit the corresponding Jupyter Book documentation page for guidelines.

2.1.2 Writing executable content

Based on the powerful Jupyter ecosystem, NeuroLibre preprints allow you to interleave computational material with your narrative. You can add some directives and metadata to your code cell blocks for Jupyter Book to determine the format and behavior of the outputs, such as interactive data visualization.

See also:
Please visit the corresponding Jupyter Book documentation page for guidelines.

There are two mandatory files that we look for in the content folder: _config.yml and _toc.yml. These files help RoboNeuro structure your book and configure some settings.

2.2 Table of contents

The _toc.yml file determines the structure of your NeuroLibre preprint. It is a simple configuration file specifying a table of content from all the executable & narrative content found in the content folder (and in subfolders).

See also:
The complete reference for the _toc.yml can be found here.

2.3 Book configuration

The _config.yml file governs all the configuration options for your Jupyter Book formatted preprint, such as adding a logo, enable/disable interactive buttons or control notebook execution and caching settings. Few important points:

3.4. Preprint repository structure
- Please ensure that the title and the list of authors matches those specified in the paper.md.

<table>
<thead>
<tr>
<th>title</th>
<th>&quot;NeuroLibre preprint template&quot; # Add your title</th>
</tr>
</thead>
<tbody>
<tr>
<td>author</td>
<td>John Doe, Jane Doe # Add author names</td>
</tr>
</tbody>
</table>

- Please ensure that the repository address is accurate.

| repository      | url: https://github.com/username/reponame # The URL to your repository |

- By default NeuroLibre force the notebook execution, still make sure you have it enabled.

| execute          | execute_notebooks: force |

See also:

The complete reference for the _config.yml can be found here.
3.4.4 1. Static summary

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'
---
```

The `paper.md` and `paper.bib`
The corpus of this static document 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`.

To check if your PDF compiles, visit RoboNeuro preprint preview page, select NeuroLibre PDF option and enter your repository address.

See also:

For more information on how to format your paper, please take a look at JOSS documentation.

---

**Warning:** NeuroLibre is at an alpha stage of development, and is not currently open for submissions.

### 3.5 Local testing

Assuming that:

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

You can easily test your preprint build locally.

#### 3.5.1 1. Install Jupyter Book

```
pip install jupyter-book
```

#### 3.5.2 2. Manage your data

Given the following minimalistic repository structure:
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.

```bash
pip install repo2data
```

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

```python
# 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.

### 3.5.3 3. Book build

- Navigate to the repository location in a terminal

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

- Trigger a jupyter book build

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

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

### 3.6 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)
3.7 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.

3.7.1 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?

3.7.2 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?

3.7.3 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.

3.7.4 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.

### 3.7.5 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.

### 3.7.6 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.

### 3.7.7 Attribution

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

### 3.8 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 *.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-boostrap and neurolibre/terraform-binderhub.
You can find more details on the installation at *Bare-metal to BinderHub.*

### 3.9 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 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.*

#### 3.9.1 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.

```
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/$USER/.ssh and copy it to /home/$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.
  ```
4. Finally, copy the key `gitcrypt-key` in `/home/$USER/my-keys/`.

### 3.9.2 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.

   ```sh
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.

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

### 3.9.3 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`).

   ```sh
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 `compute canada/arbutus *openrc.sh` API keys.

   ```sh
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:

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

### 3.9. Bare-metal to BinderHub
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:

```bash
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.

### 3.9.4 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
   - sshAuthorizedKeys: 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

28 Chapter 3. Contributions are welcome!
3.10 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 terrafrom configurations.

### 3.10.1 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 mandatory:

- **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.

### 3.10.2 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 careful before using one.

3.10.3 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 use 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.
3.10.4 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.

3.10.5 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).

### 3.10.6 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.comp.cloud in DNS specific considerations:

   sudo docker login my-binder-registry.comp.cloud --username user --password password
   sudo docker pull ubuntu:16.04
   sudo docker tag ubuntu:16.04 my-binder-registry.comp.cloud/my-ubuntu
   sudo docker push my-binder-registry.comp.cloud/my-ubuntu
3.10.7 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/
```

3.11 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.

3.11.1 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 -L 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/`.

### 3.11.2 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:

### 3.11. BinderHub test mode
The process is successful if you see the Hub is up message.

10. Install BinderHub and its development requirements:

```bash
git clone https://github.com/binderhub/binderhub
```}

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:

```bash
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:
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.

### 3.12 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

#### 3.12.1 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.

### 3.13 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?
3.13.1 How can I test a NeuroLibre submission?

You can test your NeuroLibre submission using our RoboNeuro preview service. Make sure to follow Step 2 - Test your submission if you need more details.

3.13.2 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.

3.13.3 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.

3.13.4 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).

3.13.5 Which languages does NeuroLibre support?

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

3.13.6 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.

3.13.7 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.
3.13.8 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.

3.13.9 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.

3.13.10 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.