NeuroLibre

Release v0.1

May 27, 2020

Contents:

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

NeuroLibre is a curated repository of interactive neuroscience notebooks that seamlessly integrates data, text, code and figures. Notebooks can be freely modified and re-executed through the web, offering a fully reproducible, “libre” path from data to figures. NeuroLibre is powered by the Binder project with computational resources provided by the Canadian Open Neuroscience Platform (CONP), CBRAIN, and Compute Canada.

CHAPTER 1

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.

1.1 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?

1.2 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?

1.3 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 unsufficient quality, we encourage you to contact the editors privately and suggest that the submission be withdrawn.

1.4 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.

1.5 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.

1.6 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.

1.7 Attribution

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

CHAPTER 2

Infrastructure overview

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

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

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

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

CHAPTER 3

Bare-metal to BinderHub

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

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

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

3.1 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` and `neurolibre's ssh key` to any infrastructure admin if authorized.

Warning: You should never share the `gitcrypt-key` and `neurolibre's ssh key` 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/`.

3.2 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.

```
sudo docker login
```

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

```
sudo docker pull conpdev/neurolibre-instance:v1.2
```

3.3 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 and artifacts.

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. 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
```

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

```
sudo docker stop conpdev/neurolibre-instance:v1.2
sudo docker rm conpdev/neurolibre-instance:v1.2
```

3.4 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_nodei`)
   - `nb_nodes`: number of k8s nodes **excluding** the master node
   - `instance_volume_size`: main volume size of the instances in GB **including** the master node
   - `ssh_authorized_keys`: list of the public ssh keys that will be allowed on the server
   - `os_flavor_master`: hardware configuration of the k8s master instance in the form `c{n_cpus}-{ram}gb-{optionnal_vol_in_gb}`
   - `os_flavor_node`: hardware configuration of the k8s node instances
   - `image_name`: OS image name used by the instance
   - `docker_registry`: domain for the Docker registry, if empty it uses `Docker.io` by default
   - `docker_id`: user id credential to connect to the Docker registry
   - `docker_password`: password credential to connect to the Docker registry

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

2. `dns` related to cloudflare DNS configuration:
   - `domain`: domain name to access your BinderHub environment, it will automatically point to the k8s master floating IP
3. `binderhub` specific to binderhub configuration:
   - `binder_version`: you can check the current BinderHub version releases here

- `TLS_email`: this email will be used by Let's Encrypt to request a TLS certificate
- `TLS_name`: TLS certificate name should be the same as the domain but with dashes - instead of points .
- `mem_alloc_gb`: Amount of RAM (in GB) used by each user of your BinderHub
- `cpu_alloc`: Number of CPU cores (Intel® Xeon® Gold 6130 for compute canada) used by each user of your BinderHub

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

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

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

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

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

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

CHAPTER 4

Bare-metal to local Docker registry and volumes

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

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

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

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

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

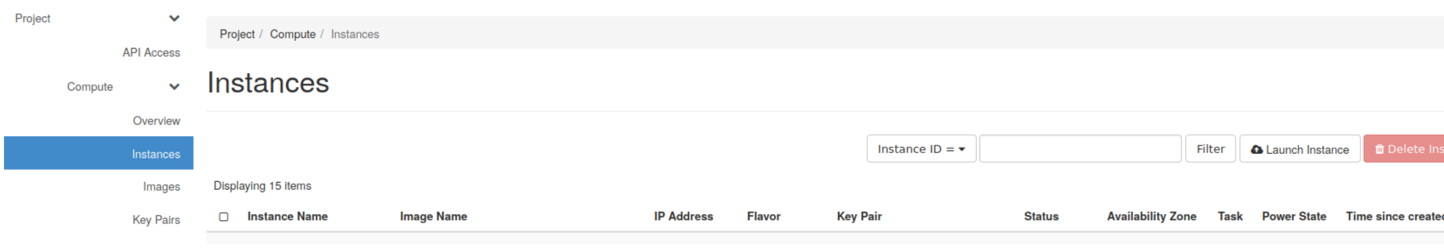
4.1 Instance spawning

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

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

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

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



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

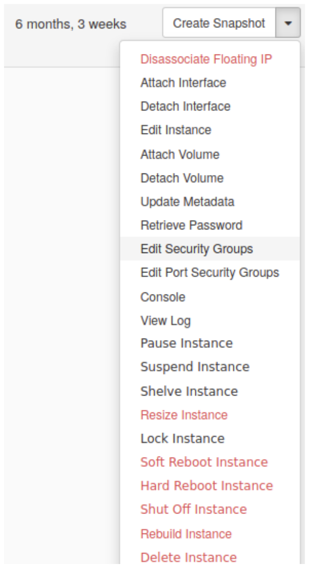
- `Instance Name`: name of the instance, choose whatever you want
- `Source`: OS image name used by the instance, select `*Bionic-x64*`
- `Flavor`: hardware configuration of the instance, `c8-30gb-186` is more than enough
- `Key Pair`: list of the public ssh keys that will be allowed on the server, find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

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

4.2 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 carefull before using one.

4.3 Firewall

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

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

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

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

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

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

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

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

4.4 DNS specific considerations

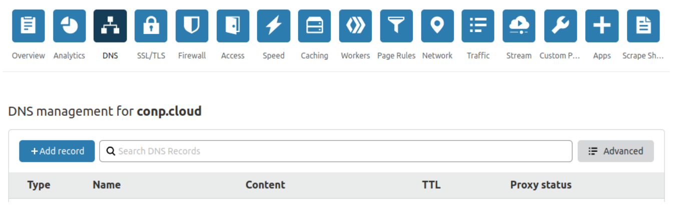
We will need to secure the Docker registry through `HTTPS` to use it with `Binderhub`, it is not possible otherwise.

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



DNS management for **conp.cloud**

Type	Name	Content	TTL	Proxy status

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

4.5 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`).

4.6 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 -Bbn user password > auth/htpasswd
```

After that you can launch the registry,

```
sudo docker run -d -p 80:80 --restart=always --name registry \
-v /certs:/certs \
-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 \
registry:2
```

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
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.

CHAPTER 5

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.

5.1 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 -sL https://deb.nodesource.com/setup_13.x | sudo -E bash -
apt-get install -y nodejs
```

Install minikube for a bare-metal server.

Install kubectl.

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

Install binderhub repo:

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

You can now follow the contribution guide from step 3.

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

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

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

1. `ssh` into the previously configured node

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

2. Launch shell as the root user:

   ```
   sudo su - root
   ```

3. Make sure that the following `apt` packages are installed

   - `npm`
   - `git`
   - `curl`
   - `python3`
   - `python3-pip`
   - `socat`

4. Ensure that the `minikube` is installed, if not follow these instructions.

5. Clone the `BinderHub` repo and `cd` into it:

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

6. Start `minikube`:

   ```
   minikube start
   ```

7. Install `helm` to the minikube cluster:

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

8. Initialize `helm` in the minikube cluster:

```
helm init
```

9. Add `JupyterHub` to the helm charts:

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

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

10. Install BinderHub and its development requirements:

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

11. Install JupyterHub in the minikube with helm:

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

12. Make minikube use the host Docker daemon :

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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