Plasma

Release v0.1

Jun 15, 2020

Contents

PLASMA

PLATEFORME D'E-LEARNING POUR L'ANALYSE DE DONNÉES SCIENTIFIQUES MASSIVES

POWERED BY jupyter

CHAPTER 1

Overview

Plasma is built with The Littlest JupyterHub (TLJH) and uses Docker containers to start the user servers.

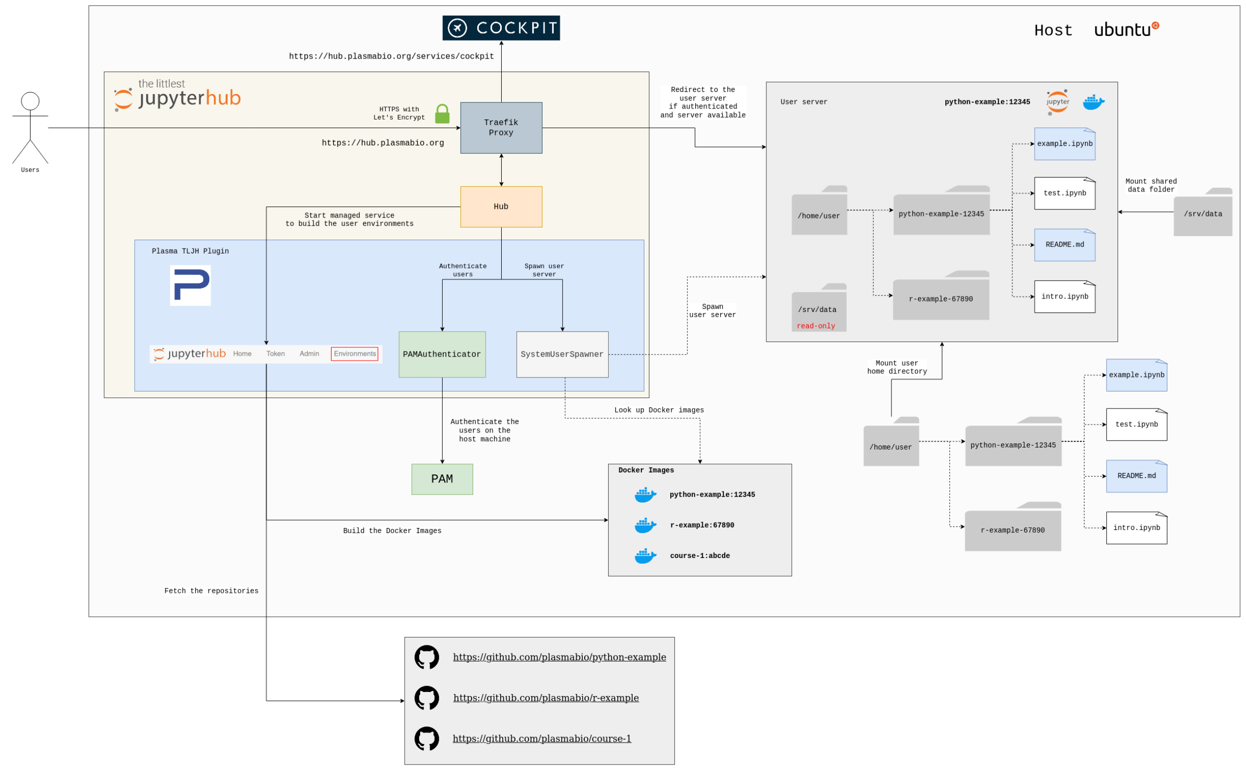
The project provides:

- A TLJH plugin with a predefined JupyterHub configuration
- Ansible playbooks to automate the deployment on a new server
- Documentation for the plugin and the Ansible playbooks

Plasma can be seen as an **opinionated TLJH distribution**:

- It gives admin users the possibility to configure multiple user environments backed by Docker images
- It provides an interface to build the user environments, accessible from the JupyterHub panel
- It uses PAM as the authenticator, and relies on system users for data persistence (home directories) and authentication
- It provides additional Ansible Playbooks to provision the server with extra monitoring tools

Here is an overview of all the different components and their interactions after Plasma has been deployed on a new server:



1.1 The JupyterHub Documentation

Since Plasma is built on top of JupyterHub and The Littlest JupyterHub distribution, it benefits from its community and high quality documentation.

For more information on these projects:

- JupyterHub Documentation
- The Littlest JupyterHub Documentation

CHAPTER 2

Installation

This guide will walk you through the steps to install Plasma on your own server.

2.1 Requirements

Before installing Plasma, you will need:

- A server running at least **Ubuntu 18.04**
- The public IP of the server
- SSH access to the machine
- A *priviledged user* on the remote machine that can issue commands using `sudo`

2.1.1 Adding the public SSH key to the server

To deploy Plasma, you need to be able to access the server via SSH.

This is typically done by copying the key to the remote server using the `ssh-copy-id` command, or by providing the key during the creation of the server (see section below).

To copy the SSH key to the server:

```
ssh-copy-id ubuntu@51.178.95.143
```

Alternatively, the SSH key can be copied from `~/.ssh/id_rsa.pub`, and looks like the following:

```
ssh-rsa␣
↪AAAAB3NzaC1yc2EAAAADAQABAAACAQCeeTSTvuZ4KzWBwUj2yIKNhX9Jw+LLdNfjOaVONfnYrlVYywRLexRcKJVcUOL8ofK/
↪RXW2xuRQzUu4Kpa0eKMM+iUPEKFF+RtLQGxn3aCVctvXprzrugm69unWot+rc2aBosX99j64U74KkEaLquuBZDd/
↪hmqxxbCr9DRYqb/aFIjfhBS8V0QdKVln1jPoy/nPCY6HMnovicExjB/E5s5lTj/
↪2qUoNXWF5r4zHQlXuc6CY0NN11F2/5n0KfSD3eunBd26zrhzpOJbcyftUV9YOICjJXWOLLOPFn2mqXsPa0k/
↪xRCjCiLv/aiU8xF5mJvYDEJ2jigqGihzfgPz4UEwH0bqQRsq9LrFYVcFLQprCknxxt9F2WgO6nv/
↪V5kgRSi3WOzRt12NcWjg1um/C2TTK9bSqFTEMXlPlsLxDa7Js/
↪kUMZh6N3rIzTsQpXuhKjQLxZ5TReUUdsGyAtU0eQv5rrJBr6ML02C9EMZ5NvduPs1w44+39WONCmoQoKBkiFIYfN0EV7Ps6kM6c
↪your_email@example.com
```

(continues on next page)

(continued from previous page)

It can then be manually added to `~/.ssh/authorized_keys` on the server.

For more information, checkout this tutorial on DigitalOcean to set up SSH Keys on Ubuntu 18.04.

2.1.2 Creating a new server (optional)

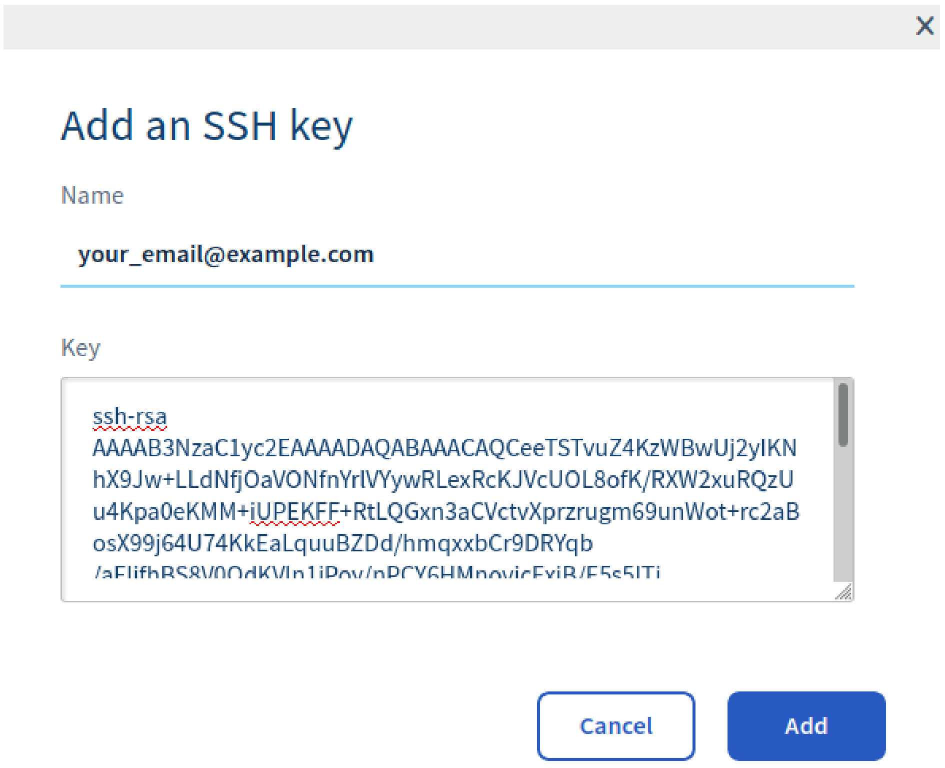
If you don't already have a server (or want to test the setup from scratch) you can create a new one using a cloud provider.

The Littlest JupyterHub documentation provides detailed guides for different cloud providers.

You can pick one of them, and stop at the point where the TLJH script (starting with `#!/bin/bash`) should be provided (this part is covered in the next section).

During the installation steps, you will be able to specify the SSH key to use to connect to the server.

The key must first be added to the list of available keys by using the cloud provider interface:



Add an SSH key

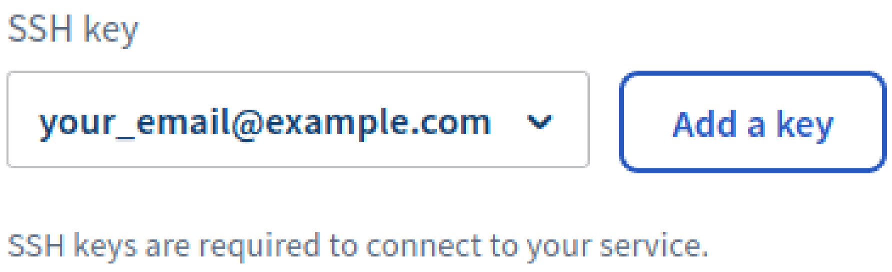
Name

your_email@example.com

Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQCeeTSTvuZ4KzWBwUj2yIKN
hX9Jw+LLdNfjOaVONfnYrlVYywRLexRcKJVcUOL8ofK/RXW2xuRQzU
u4Kpa0eKMM+iUPEKFF+RtLQGxn3aCVctvXprzrugm69unWot+rc2aB
osX99j64U74KkEaLquuBZDd/hmqxxbCr9DRYqb
/aFlifhRS8V0OdKVln1iPov/nPCY6HMnovicFxiB/E5s5lTi
```

When asked to choose an SSH key, select the one you just added:



2.1.3 Testing the connection

For a server with an `ubuntu` user, validate that you have access to it with:

```
ssh -t ubuntu@51.178.95.143 echo "test"
```

Which should output the following:

```
test
Connection to 51.178.95.143 closed.
```

2.1.4 Updating the local SSH config (optional)

Depending on the server used for the deployment, see *Creating a new server (optional)*, you might want to add the following to your local SSH config located in `~/.ssh/config`:

```
Host *
    ServerAliveInterval 60
    ServerAliveCountMax 10
```

These settings help keep the connection to server alive while the deployment is happening, or if you have an open SSH connection to the server.

2.2 Deploying with Ansible

2.2.1 What is Ansible?

Ansible is an open-source tool to automate the provisioning of servers, configuration management, and application deployment.

Playbooks can be used to define the list of tasks that should be executed and to declare the desired state of the server.

Check out the How Ansible Works guide on the Ansible official documentation website for more information.

2.2.2 Installing Ansible

Plasma comes with several *Ansible Playbooks* to automatically provision the machine with the system requirements, as well as installing Plasma and starting up the services.

Note: We recommend creating a new virtual environment to install Python packages.

Using the built-in `venv` module:

```
python -m venv .
source bin/activate
```

Using `conda`:

```
conda create -n plasma -c conda-forge python nodejs
conda activate plasma
```

Make sure Ansible is installed:

```
python -m pip install ansible>=2.9
```

Note: We recommend `ansible>=2.9` to discard the warning messages regarding the use of `aptitude`.

To verify the installation, run:

```
which ansible
```

This should return the path to the ansible CLI tool in the virtual environment. For example: `/home/myuser/miniconda/envs/plasma/bin/ansible`

2.2.3 Running the Playbooks

Check out the repository, and go to the `plasma/ansible/` directory:

```
git clone https://github.com/plasmabio/plasma
cd plasma/ansible
```

Create a `hosts` file with the following content:

```
[servers]
51.178.95.237

[servers:vars]
ansible_python_interpreter=/usr/bin/python3
```

Replace the IP corresponds to your server. If you already defined the hostname (see *HTTPS*), you can also specify the domain name:

```
[servers]
dev.plasmabio.org

[servers:vars]
ansible_python_interpreter=/usr/bin/python3
```

Then run the following command after replacing `<user>` by your user on the remote machine:

```
ansible-playbook site.yml -i hosts -u <user>
```

Many Ubuntu systems running on cloud virtual machines have the default `ubuntu` user. In this case, the command becomes:

```
ansible-playbook site.yml -i hosts -u ubuntu
```

Ansible will log the progress in the terminal, and will indicate which components have changed in the process of running the playbook:

```
PLAY [all]␣
↪****************************************************************************************************

TASK [Gathering Facts]␣
↪****************************************************************************************************
ok: [51.178.95.237]

TASK [Install aptitude using apt]␣
↪****************************************************************************************************
ok: [51.178.95.237]

TASK [Install required system packages]␣
↪****************************************************************************************************
ok: [51.178.95.237] => (item=apt-transport-https)
ok: [51.178.95.237] => (item=ca-certificates)
ok: [51.178.95.237] => (item=curl)
ok: [51.178.95.237] => (item=software-properties-common)
ok: [51.178.95.237] => (item=python3-pip)
ok: [51.178.95.237] => (item=virtualenv)
ok: [51.178.95.237] => (item=python3-setuptools)

TASK [Add Docker GPG apt Key]␣
↪****************************************************************************************************
ok: [51.178.95.237]

TASK [Add Docker Repository]␣
↪****************************************************************************************************
ok: [51.178.95.237]

TASK [Update apt and install docker-ce]␣
↪****************************************************************************************************
ok: [51.178.95.237]

PLAY [all]␣
↪****************************************************************************************************

TASK [Gathering Facts]␣
↪****************************************************************************************************
ok: [51.178.95.237]

TASK [Add Test User]␣
↪****************************************************************************************************
ok: [51.178.95.237]

PLAY [all]␣
↪****************************************************************************************************

TASK [Gathering Facts]␣
↪****************************************************************************************************
ok: [51.178.95.237]

TASK [Install aptitude using apt]␣
↪****************************************************************************************************
```

(continues on next page)

(continued from previous page)

```
ok: [51.178.95.237]

TASK [Install required system packages]␣
↪***************************************************************************************************
ok: [51.178.95.237] => (item=curl)
ok: [51.178.95.237] => (item=python3)
ok: [51.178.95.237] => (item=python3-dev)
ok: [51.178.95.237] => (item=python3-pip)

TASK [Download the TLJH installer]␣
↪***************************************************************************************************
ok: [51.178.95.237]

TASK [Run the TLJH installer]␣
↪***************************************************************************************************
changed: [51.178.95.237]

TASK [Upgrade the tljh-plasma plugin]␣
↪***************************************************************************************************
changed: [51.178.95.237]

TASK [Restart JupyterHub]␣
↪***************************************************************************************************
changed: [51.178.95.237]

PLAY RECAP␣
↪***************************************************************************************************
51.178.95.237              : ok=15   changed=3    unreachable=0    failed=0    ␣
↪skipped=0    rescued=0    ignored=0
```

2.2.4 Running individual playbooks

The `site.yml` Ansible playbook includes all the playbooks and will process them in order.

It is however possible to run the playbooks individually. For example to run the `tljh.yml` playbook only (to install and update The Littlest JupyterHub):

```
ansible-playbook tljh.yml -i hosts -u ubuntu
```

For more in-depth details about the Ansible playbook, check out the official documentation.

2.2.5 List of available playbooks

The Ansible playbooks are located in the `ansible/` directory:

- `docker.yml`: install Docker CE on the host
- `utils.yml`: install extra system packages useful for debugging and system administration
- `users.yml`: create the tests users on the host
- `quotas.yml`: enable quotas on the host to limit disk usage
- `cockpit.yml`: install Cockpit on the host as a monitoring tool
- `tljh.yml`: install TLJH and the Plasma TLJH plugin

- `admins.yml`: add admin users to JupyterHub
- `https.yml`: enable HTTPS for TLJH
- `uninstall.yml`: uninstall TLJH only
- `site.yml`: the main playbook that references some of the other playbooks

2.3 HTTPS

Warning: HTTPS is **not** enabled by default.

We do not recommend deploying JupyterHub without HTTPS for production use.

However in some situations it can be handy to do so, for example when testing the setup.

2.3.1 Enable HTTPS

Support for HTTPS is handled automatically thanks to Let's Encrypt, which also handles the renewal of the certificates when they are about to expire.

In your `hosts` file, add the `name_server` and `letsencrypt_email` variables on the same line as the server IP:

```
[servers]
51.178.95.237 name_server=dev.plasmabio.org letsencrypt_email=contact@plasmabio.org

[servers:vars]
ansible_python_interpreter=/usr/bin/python3
```

If you have multiple servers, the `hosts` file will look like the following:

```
[servers]
51.178.95.237 name_server=dev1.plasmabio.org letsencrypt_email=contact@plasmabio.org
51.178.95.238 name_server=dev2.plasmabio.org letsencrypt_email=contact@plasmabio.org

[servers:vars]
ansible_python_interpreter=/usr/bin/python3
```

Modify these values to the ones you want to use.

Then, run the `https.yml` playbook:

```
ansible-playbook https.yml -i hosts -u ubuntu
```

This will reload the proxy to take the changes into account.

It might take a few minutes for the certificates to be setup and the changes to take effect.

2.3.2 How to make the domain point to the IP of the server

The domain used in the playbook variables (for example `dev.plasmabio.org`), should also point to the IP of the server running JupyterHub.

This is typically done by logging in to the registrar website and adding a new entry to the DNS records.

You can refer to the documentation for The Littlest JupyterHub on how to enable HTTPS for more details.

2.3.3 Manual HTTPS

To use an existing SSL key and certificate, you can refer to the Manual HTTPS with existing key and certificate documentation for TLJH.

This can also be integrated in the `https.yml` playbook by replacing the `tljh-config` commands to the ones mentioned in the documentation.

2.4 Creating Users on the host

Note: By default the `site.yml` playbook does not create any users on the host machine.

This step is optional because in some scenarios users might already exist on the host machine and don't need to be created.

2.4.1 Using the users playbook

The `ansible/` directory contains a `users.yml` playbook that makes it easier to create new users on the host in batches.

First you need to create a new `users-config.yml` with the following content:

```yaml
users:
  - name: foo
    password: PLAIN_TEXT_PASSWORD

  - name: bar
    password: PLAIN_TEXT_PASSWORD
```

Replace the `name` and `password` entries by the real values.

`password` should correspond to the plain text value of the user password.

For more info about password hashing, please refer to the Ansible Documentation to learn how to generate the encrypted passwords.

When the user file is ready, execute the `users.yml` playbook with the following command:

```
ansible-playbook users.yml -i hosts -u ubuntu -e @users-config.yml
```

2.4.2 Handling secrets

Warning: Passwords are sensitive data. The `users.yml` playbook mentioned in the previous section automatically encrypts the password from a plain text file.

For production use, you should consider protecting the passwords using the Ansible Vault.

This `users.yml` playbook is mostly provided as a convenience script to quickly bootstrap the host machine with a predefined set of users.

You are free to choose a different approach for managing users that suits your needs.

2.4.3 Set Disk Quotas

Users can save their files on the host machine in their home directroy. More details in *User Data*.

If you would like to enable quotas for users to limit how much disk space they can use, you can use the `quotas.yml` Ansible playbook.

The playbook is heavily inspired by the excellent DigitalOcean tutorial on user quotas. Check it out for more info on user and group quotas.

Warning: It is recommended to do the initial quota setup **before** letting users connect to the hub.

Finding the source device

As mentioned in the tutorial, the first step is to find the device to apply quotas to.

To do so, SSH into the machine (*Requirements*) and execute the following command:

```
cat /etc/fstab
```

The output will be similar to:

```
LABEL=cloudimg-rootfs   /        ext4   defaults        0 0
LABEL=UEFI      /boot/efi       vfat    defaults        0 0
```

The source device for `/` might be different than `LABEL=cloudimg-rootfs`. If this is the case, copy the value somewhere so it can be used in the next step with the playbook.

Using the quotas playbook

To enable quotas on the machine, execute the `quotas.yml` playbook with the source device found in the previous section (if different):

```
# if the device is also named LABEL=cloudimg-rootfs
ansible-playbook quotas.yml -i hosts -u ubuntu

# if the source device is different (replace with the real value)
ansible-playbook quotas.yml -i hosts -u ubuntu -e "device=UUID=aaef63c7-8c31-4329-
↪8b7f-b90085ecccd4"
```

Setting the user quotas

The `users.yml` playbook can also be used to set the user quotas. In `users-config.yml` you can define quotas as follows:

```
# default quotas for all users
quota:
  soft: 10G
  hard: 12G

users:
  - name: foo
    password: foo
```

(continues on next page)

(continued from previous page)

```
    # override quota for a specific user
    quota:
      soft: 512M
      hard: 1G

  - name: bar
    password: bar
```

Then re-run the `users.yml` playbook as mentioned in *Using the users playbook*.

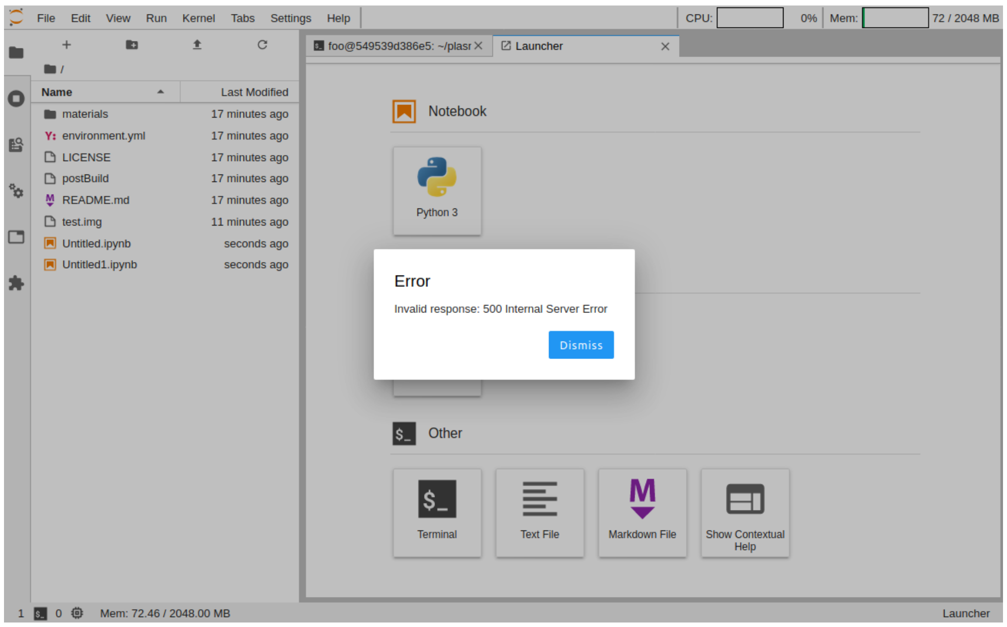
For example, if a user exceeds their quota when creating a file from the terminal inside the container, they will be shown the following message:

```
foo@549539d386e5:~/plasmabio-template-python-master$ fallocate -l 1G test.img
fallocate: fallocate failed: Disk quota exceeded
```

On the host machine, a user can check their quota by running the following command:

```
foo@test-server:~$ quota -vs
Disk quotas for user foo (uid 1001):
     Filesystem   space   quota   limit   grace   files   quota   limit   grace
      /dev/sda1   1024M*   512M   1024M   6days   33910       0       0
```

If the quota is exceeded and the user tries to create a new notebook from the interface, they will be shown an error dialog:



Error

Invalid response: 500 Internal Server Error

2.5 Adding Admin Users to JupyterHub

By default the `site.yml` playbook does not add admin users to JupyterHub.

New admin users can be added by running the `admins.yml` playbook:

```
ansible-playbook admins.yml -i hosts -u ubuntu --extra-vars '{"admins": ["foo", "bar
↪"]}'
```

(continues on next page)

(continued from previous page)

This playbook processes the list of users specified via the `--extra-vars` command and add them as admin one at a time.

Warning: The list passed via the `--extra-vars` parameter overrides the existing list of admins.

Alternatively it is also possible to use the `tljh-config` command on the server directly. Please refer to the Littlest JupyterHub documentation for more info.

2.6 Upgrading

2.6.1 Backup

Before performing an upgrade, you might want to back up some components of the stack.

Database

JupyterHub keeps the state in a sqlite database, with information such as the last login and whether a user is an admin or not.

TLJH keeps the database in the `/opt/tljh/state` directory on the server. The full path to the database is `/opt/tljh/state/jupyterhub.sqlite`.

To know more about backing up the database please refer to:

- The JupyterHub documentation
- The TLJH documentation on the state files

For more info on where TLJH is installed: What does the installer do?

Plasma TLJH Plugin

This TLJH plugin is a regular Python package.

It is installed in `/opt/tljh/hub/lib/python3.6/site-packages/tljh_plasma`, and doesn't need to be backed up as it doesn't hold any state.

User Environments

The user environments correspond to Docker images on the host. There is no need to back them up as they will stay untouched if not removed manually.

User Data

It is generally recommended to have a backup strategy for important data such as user data.

This can be achieved by setting up tools that for example sync the user home directories to another machine on a regular basis.

Check out the *User Data* section to know more about user data.

2.6.2 Running the playbook

To perform an upgrade of the setup, you can re-run the playbooks as explained in *Deploying with Ansible*.

Re-running the playbooks will:

- Update the TLJH Plasma plugin
- Update TLJH
- Restart JupyterHub and the Proxy to take new changes into account

However, performing an upgrade does not:

- Stop user servers
- Remove user environments (Docker images)
- Delete user data

In most cases, it is enough to only run the `tljh.yml` playbook to perform the upgrade.

Refer to *Running individual playbooks* for more info.

2.7 Uninstalling

If you want to uninstall The Littlest JupyterHub from the machine, you can:

- Destroy the VM: this is the recommended way as it is easier to start fresh
- Run the `uninstall.yml` Ansible playbook if destroying the VM is not an option

To run the playbook:

```
ansible-playbook uninstall.yml -i hosts -u ubuntu
```

Note: The playbook will **only** uninstall TLJH from the server.

It will **not**:

- delete user data
- remove environments and Docker images

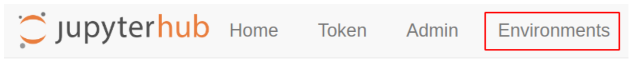
CHAPTER 3

User Environments

User environments are built as immutable Docker images. The Docker images bundle the dependencies, extensions, and predefined notebooks that should be available to all users.

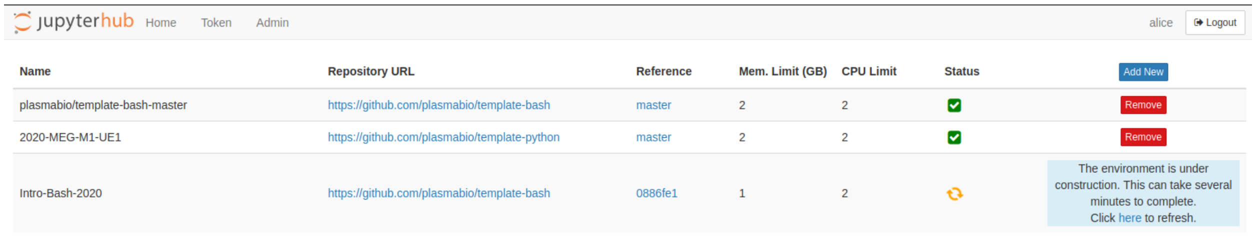
Plasma relies on the tljh-repo2docker plugin to manage environments. The `tljh-repo2docker` uses jupyter-repo2docker to build the Docker images.

Environments can be managed by admin users by clicking on `Environments` in the navigation bar:



Note: The user must be an **admin** to be able to access and manage the list of environments.

The page will show the list of environments currently available:



Name	Repository URL	Reference	Mem. Limit (GB)	CPU Limit	Status	Add New
plasmabio/template-bash-master	https://github.com/plasmabio/template-bash	master	2	2	✓	Remove
2020-MEG-M1-UE1	https://github.com/plasmabio/template-python	master	2	2	✓	Remove
Intro-Bash-2020	https://github.com/plasmabio/template-bash	0886fe1	1	2	⟳	The environment is under construction. This can take several minutes to complete. Click here to refresh.

After a fresh install, this list will be empty.

3.1 Managing User Environments

3.1.1 Preparing the environment

An *environment* is defined as an immutable set of dependencies and files.

Since Plasma uses jupyter-repo2docker, it relies on the same set of rules and patterns as `repo2docker` to create the environments.

Create a new repository

Plasma fetches the environments from publicly accessible Git repositories from code sharing platforms such as GitHub.

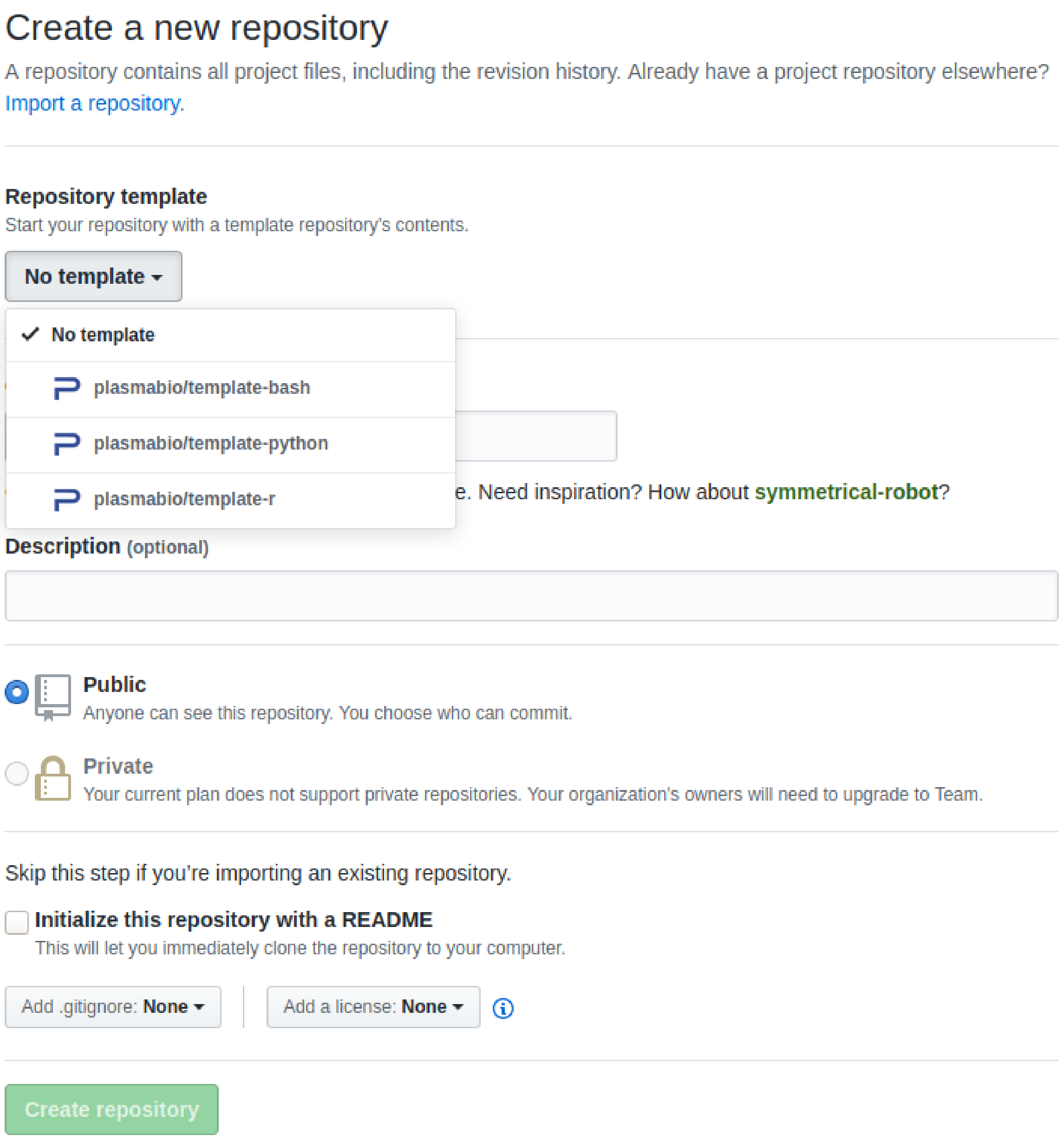
To create a new environment with its own set of dependencies, it is recommended to create a new repository on GitHub.

The plasmabio organization defines a couple of template repositories that can be used to bootstrap new ones:

- For Python: https://github.com/plasmabio/template-python
- For R: https://github.com/plasmabio/template-r
- For Bash: https://github.com/plasmabio/template-bash

To create a new repository using one of these templates, go to the organization and click on `New`.

Then select the template from the `Repository Template` dropdown:



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Repository template

Start your repository with a template repository's contents.

No template ▾

✓ No template
plasmabio/template-bash
plasmabio/template-python
plasmabio/template-r

e. Need inspiration? How about symmetrical-robot?

Description (optional)

● Public
Anyone can see this repository. You choose who can commit.

○ Private
Your current plan does not support private repositories. Your organization's owners will need to upgrade to Team.

Skip this step if you're importing an existing repository.

☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer.

Add .gitignore: None ▾ | Add a license: None ▾

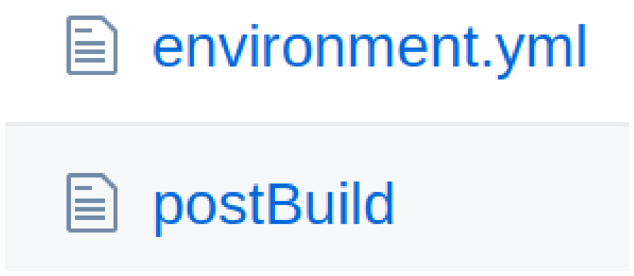
Create repository

How to specify the dependencies

`repo2docker` relies on a specific set of files to know which dependencies to install and how to build the Docker image.

These files are listed on the Configuration Files page in the documentation.

In the case of the Python Template, they consist of an `environment.yml` and `postBuild` files:



Testing on Binder

Since both Plasma and Binder use `repo2docker` to build the images, it is possible to try the environment on Binder first to make sure they are working correctly before adding theme to the JupyterHub server.

The template repository has a Binder button in the `README.md` file. This button will redirect to the public facing instance of BinderHub, mybinder.org, and will build a Binder using the configuration files in the repository.

You can use the same approach for the other environments, and update the Binder link to point to your repository.

Make sure to check out the documentation below for more details.

Extra documentation

To learn more about `repo2docker`, check out the Documentation.

To learn more about `Binder`, check out the Binder User Guide.

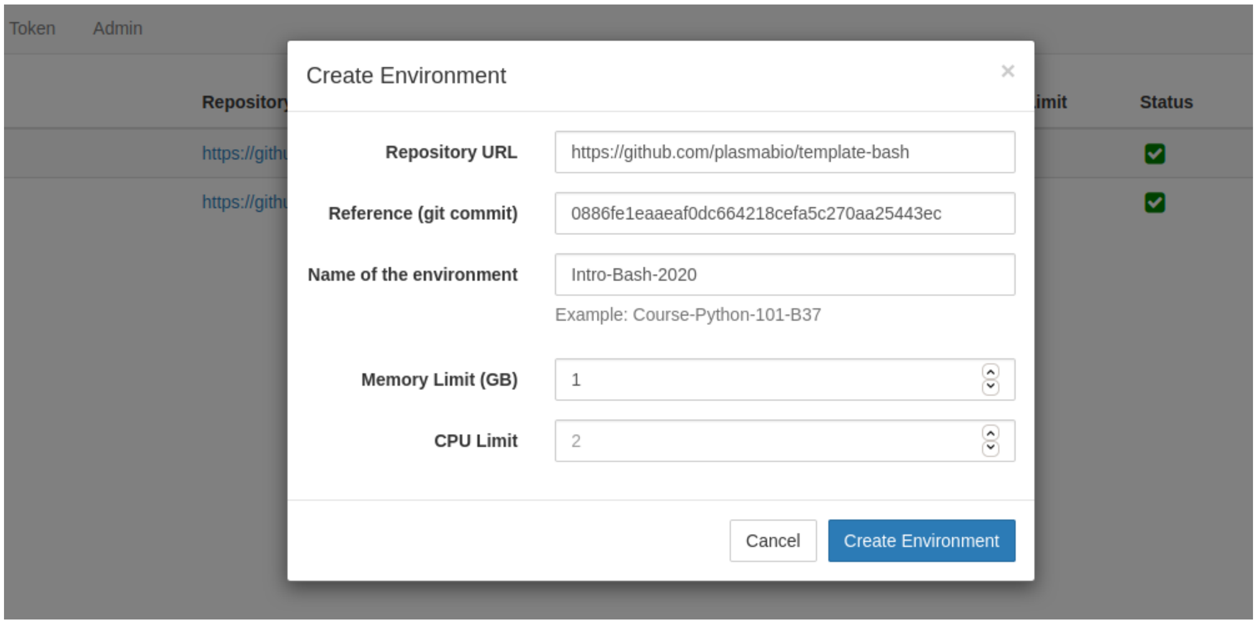
3.1.2 Adding a new environment

Now that the repository is ready, we can add it to the JupyterHub via the user interface.

To add the new user environment, click on the `Add New` button and provide the following information:

- `Repository URL`: the URL to the repository to build the environment from
- `Reference (git commit)`: the git commit hash to use
- `Name of the environment`: the display name of the environment. If left empty, it will be automatically generated from the repository URL.
- `Memory Limit (GB)`: the memory limit to apply to the user server. Float values are allowed (for example a value of `3.5` corresponds to a limit of 3.5GB)
- `CPU Limit`: the number of cpus the user server is allowed to used. See the JupyterHub documentation for more info.

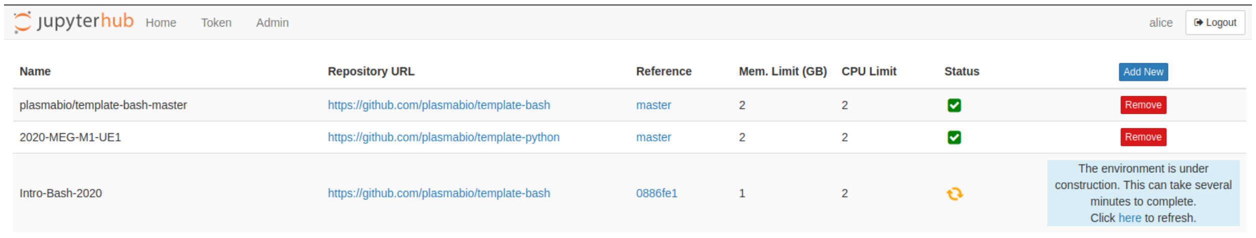
As an example:



Create Environment

Repository URL	https://github.com/plasmabio/template-bash
Reference (git commit)	0886fe1eaaeaf0dc664218cefa5c270aa25443ec
Name of the environment	Intro-Bash-2020
	Example: Course-Python-101-B37
Memory Limit (GB)	1
CPU Limit	2

After clicking on the `Add Image` button, the page will automatically reload and show the list of built environments, as well as the ones currently being built:

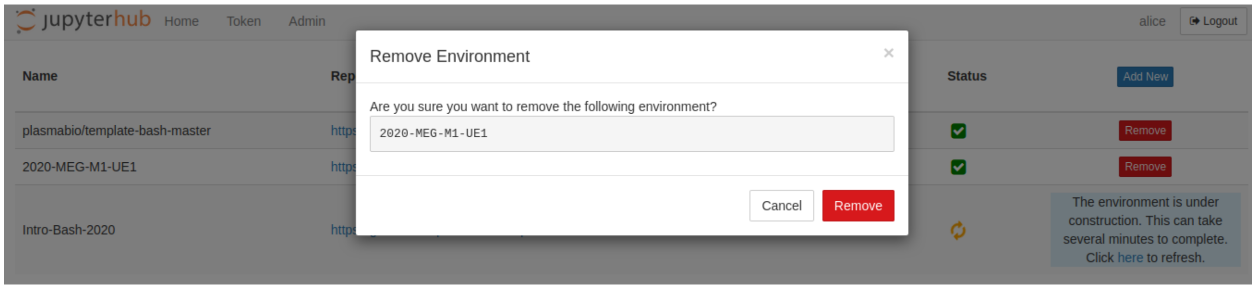


Name	Repository URL	Reference	Mem. Limit (GB)	CPU Limit	Status	Add New
plasmabio/template-bash-master	https://github.com/plasmabio/template-bash	master	2	2	✓	Remove
2020-MEG-M1-UE1	https://github.com/plasmabio/template-python	master	2	2	✓	Remove
Intro-Bash-2020	https://github.com/plasmabio/template-bash	0886fe1	1	2	⟳	The environment is under construction. This can take several minutes to complete. Click here to refresh.

Building a new environment can take a few minutes. You can reload the page to refresh the status.

3.1.3 Removing an environment

To remove an environment, click on the `Remove` button. This will bring the following confirmation dialog:

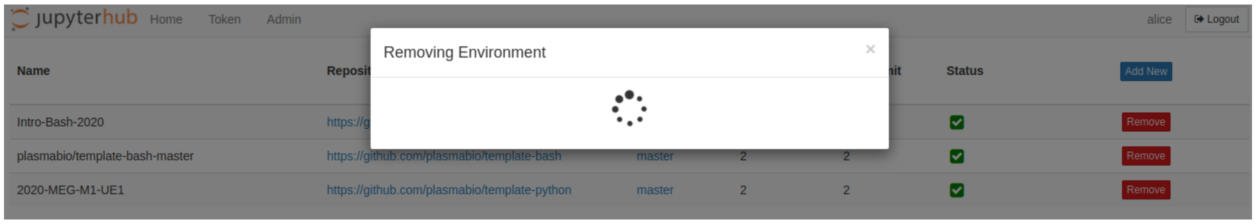

jupyterhub
Home
Token
Admin
alice
Logout
Name
Rep
Status
Add New
plasmabio/template-bash-master
2020-MEG-M1-UE1
Intro-Bash-2020
Remove
Remove
The environment is under construction. This can take several minutes to complete. Click here to refresh.

Remove Environment

Are you sure you want to remove the following environment?

```
2020-MEG-M1-UE1
```

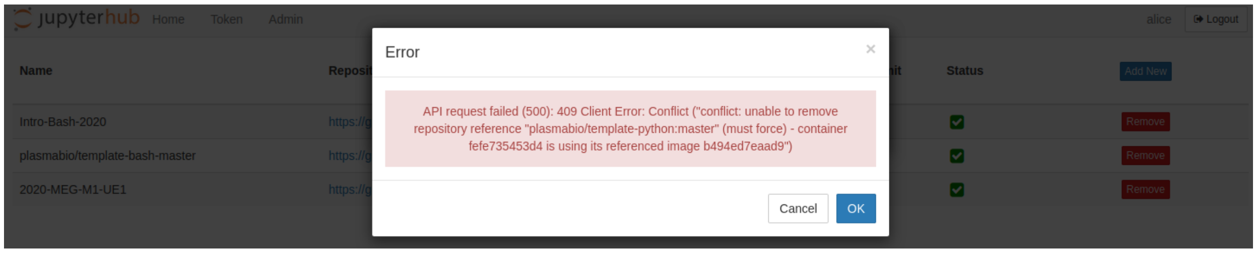
After clicking on `Remove`, a spinner will be shown and the page will reload shortly after:



Name	Reposit…				Status
Intro-Bash-2020	https://g…				✓
plasmabio/template-bash-master	https://github.com/plasmabio/template-bash	master	2	2	✓
2020-MEG-M1-UE1	https://github.com/plasmabio/template-python	master	2	2	✓

Removing an environment returns an error

It is possible that removing an environment returns an error such as the following:



API request failed (500): 409 Client Error: Conflict ("conflict: unable to remove repository reference "plasmabio/template-python:master" (must force) - container fefe735453d4 is using its referenced image b494ed7eaad9")

This is most likely because the environment is currently being used. We recommend asking the users to stop their server before attempting to remove the environment one more time.

The environment (image) that a user is currently using is also displayed in the Admin panel:

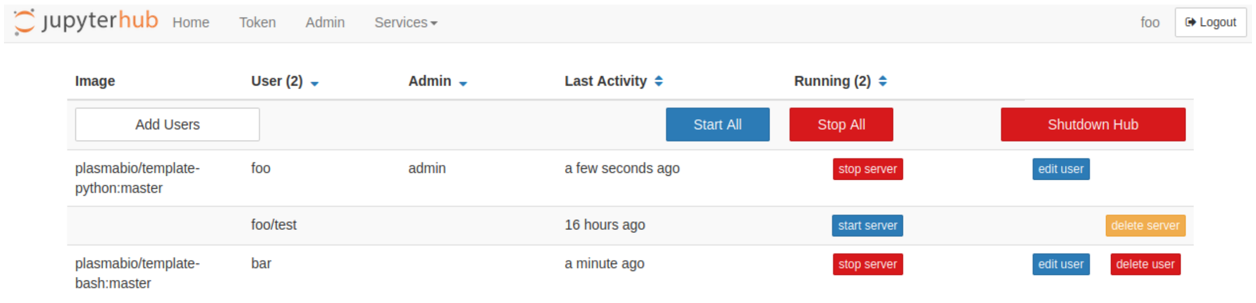


Image	User (2)	Admin	Last Activity	Running (2)		
plasmabio/template-python:master	foo	admin	a few seconds ago	stop server	edit user	
	foo/test		16 hours ago	start server		delete server
plasmabio/template-bash:master	bar		a minute ago	stop server	edit user	delete user

3.1.4 Updating an environment

Since the environments are built as Docker images, they are immutable.

Instead of updating an environment, it is recommended to:

1. Add a new one with the new `Reference`
2. Remove the previous one by clicking on the `Remove` button (see previous section)

CHAPTER 4

Configuration

4.1 Monitoring

Warning: HTTPS must be enabled to be able to access Cockpit. Refer to *HTTPS* for more info.

4.1.1 Installing Cockpit

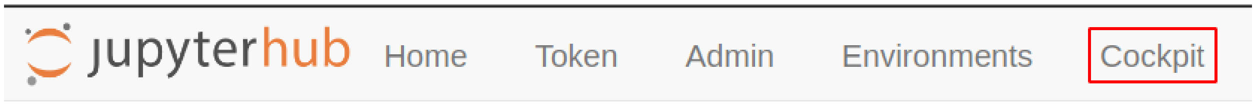
`cockpit` is not installed by default as a monitoring tool for the server.

First make sure HTTPS is enabled and the `name_server` variable is specified in the `hosts` file. See *HTTPS* for more info.

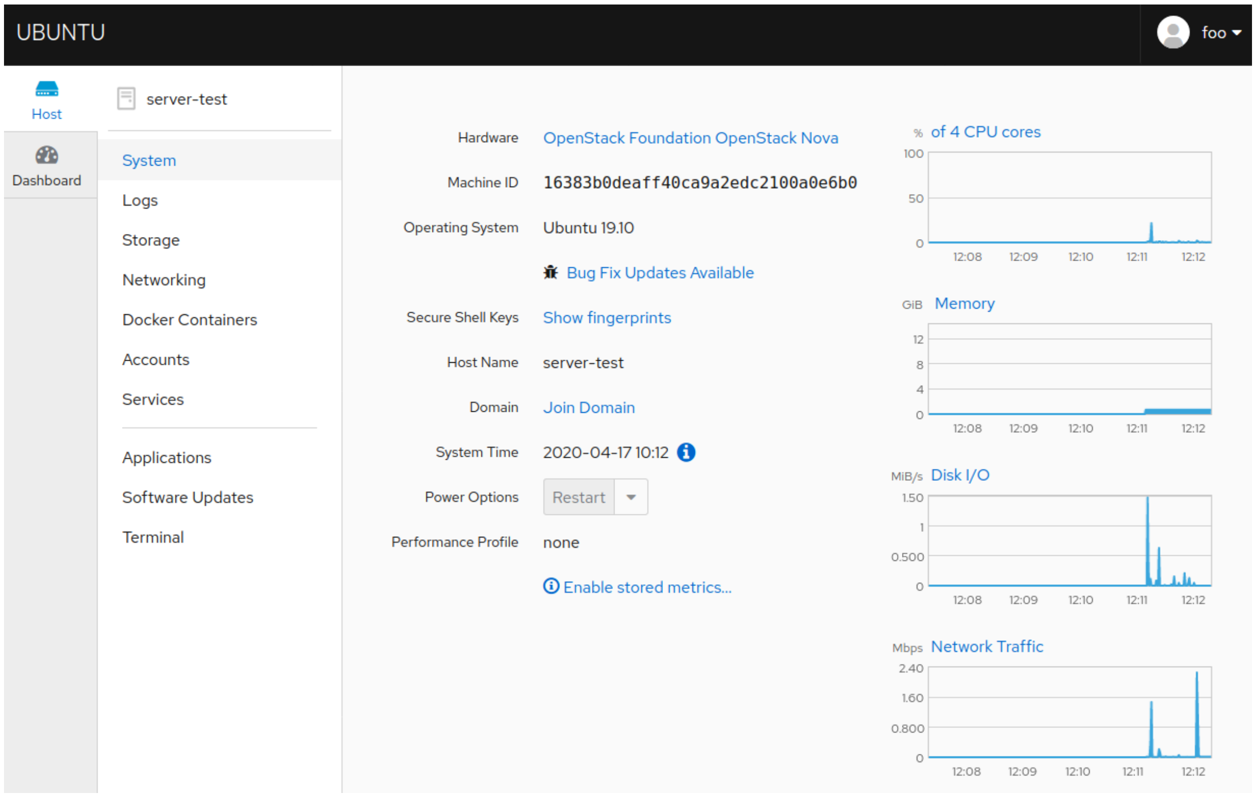
Then execute the `cockpit.yml` playbook:

```
ansible-playbook cockpit.yml -i hosts -u ubuntu
```

The Plasma TLJH plugin registers `cockpit` as a JupyterHub service. This means that Cockpit is accessible to JupyterHub admin users via the JupyterHub interface:



Users will be asked to login with their system credentials. They can then access the Cockpit dashboard:



Hardware OpenStack Foundation OpenStack Nova

Machine ID 16383b0deaff40ca9a2edc2100a0e6b0

Operating System Ubuntu 19.10

Bug Fix Updates Available

Secure Shell Keys Show fingerprints

Host Name server-test

Domain Join Domain

System Time 2020-04-17 10:12

Power Options Restart

Performance Profile none

Enable stored metrics...

4.1.2 Monitoring user servers with Cockpit

Note: Access to Docker Containers requires access to `docker`.

Make sure your user can access docker on the machine with:

```
sudo docker info
```

Your user should also be able to login with a password. If the user doesn't have a password yet, you can create a new one with:

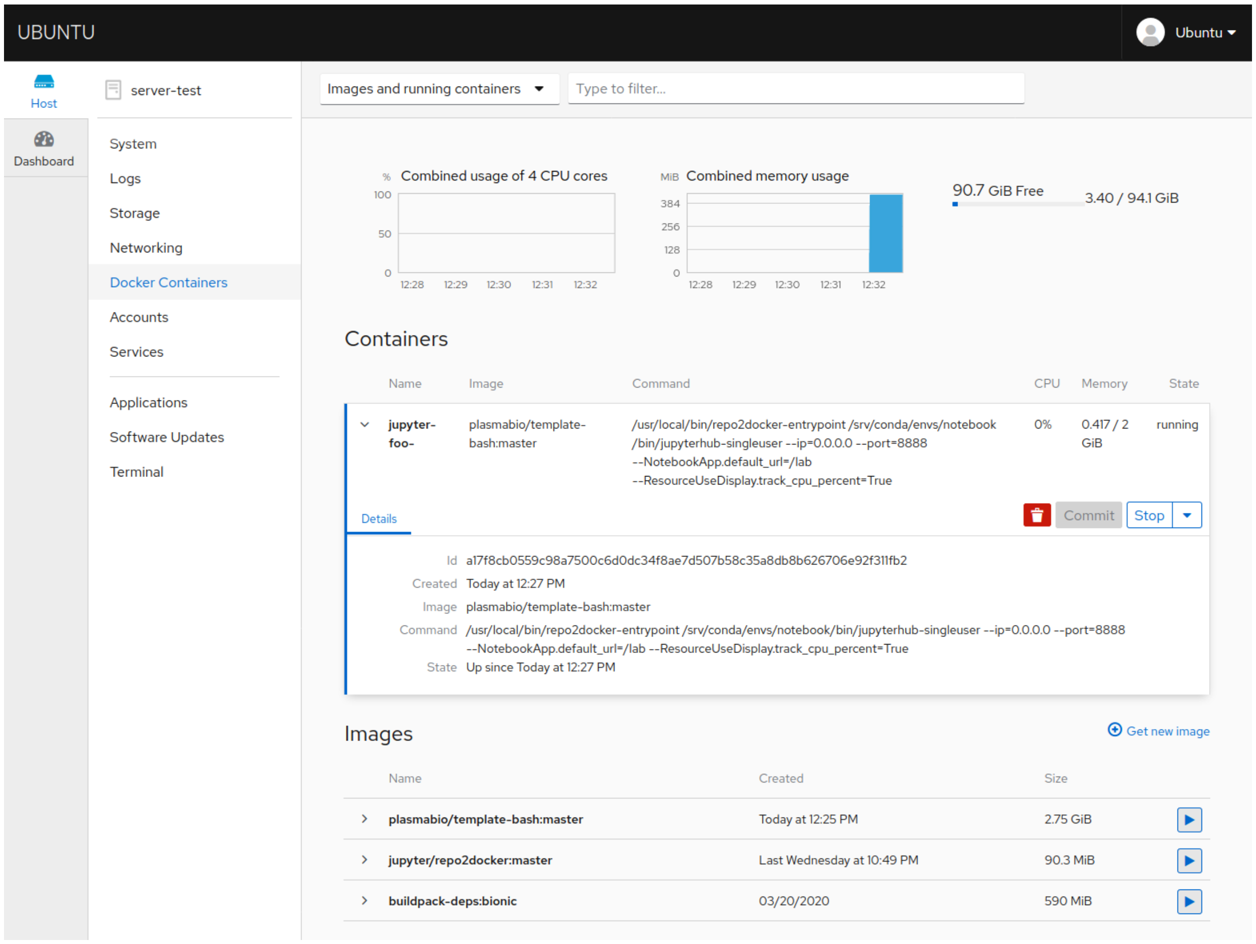
```
sudo passwd <username>
```

For example if your user is `ubuntu`:

```
sudo passwd ubuntu
```

To add more users as admin or change permissions from the Cockpit UI, see *Changing user permissions from the Cockpit UI*.

Since user servers are started as Docker containers, they will be displayed in the Cockpit interface in the `Docker Containers` section:



Containers

Name	Image	Command	CPU	Memory	State
jupyter-foo-	plasmabio/template-bash:master	/usr/local/bin/repo2docker-entrypoint /srv/conda/envs/notebook /bin/jupyterhub-singleuser --ip=0.0.0.0 --port=8888 --NotebookApp.default_url=/lab --ResourceUseDisplay.track_cpu_percent=True	0%	0.417 / 2 GiB	running

Id	a17f8cb0559c98a7500c6d0dc34f8ae7d507b58c35a8db8b626706e92f311fb2
Created	Today at 12:27 PM
Image	plasmabio/template-bash:master
Command	/usr/local/bin/repo2docker-entrypoint /srv/conda/envs/notebook/bin/jupyterhub-singleuser --ip=0.0.0.0 --port=8888 --NotebookApp.default_url=/lab --ResourceUseDisplay.track_cpu_percent=True
State	Up since Today at 12:27 PM

Images

Name	Created	Size
plasmabio/template-bash:master	Today at 12:25 PM	2.75 GiB
jupyter/repo2docker:master	Last Wednesday at 10:49 PM	90.3 MiB
buildpack-deps:bionic	03/20/2020	590 MiB

The Cockpit interface shows:

- The username as part of the name of the Docker container
- The resources they are currently using
- The environment currently in use

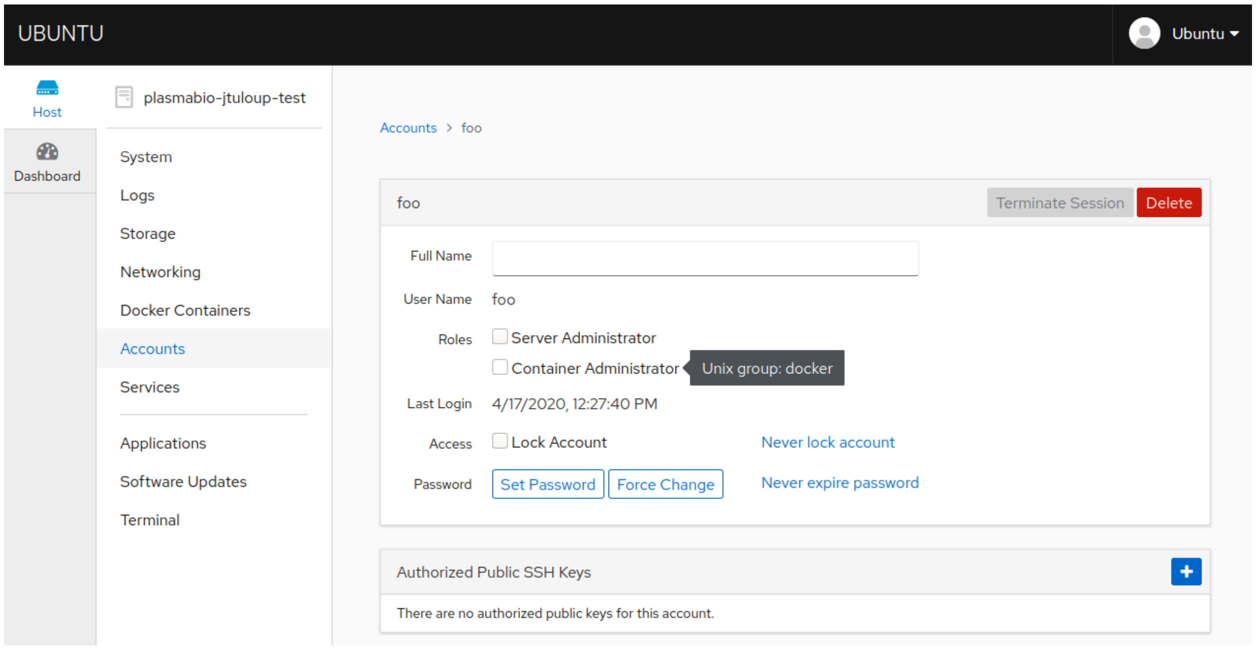
It is also possible to stop the user server by clicking on the “Stop” button.

4.1.3 Changing user permissions from the Cockpit UI

Note: You first need to be logged in with a user that has the `sudo` permission.

Cockpit makes it easy to add a specific user to a certain group.

For example a user can be given the “Container Administrator” role via the UI to be able to manage Docker containers and images on the machine:



foo

Full Name	
User Name	foo
Roles	Server Administrator
	Container Administrator
Last Login	4/17/2020, 12:27:40 PM
Access	Lock Account
Password	

Authorized Public SSH Keys

There are no authorized public keys for this account.

4.2 Data Persistence

4.2.1 User Data

The user servers are started using JupyterHub's SystemUserSpawner.

This spawner is based on the DockerSpawner, but makes it possible to use the host users to start the notebook servers.

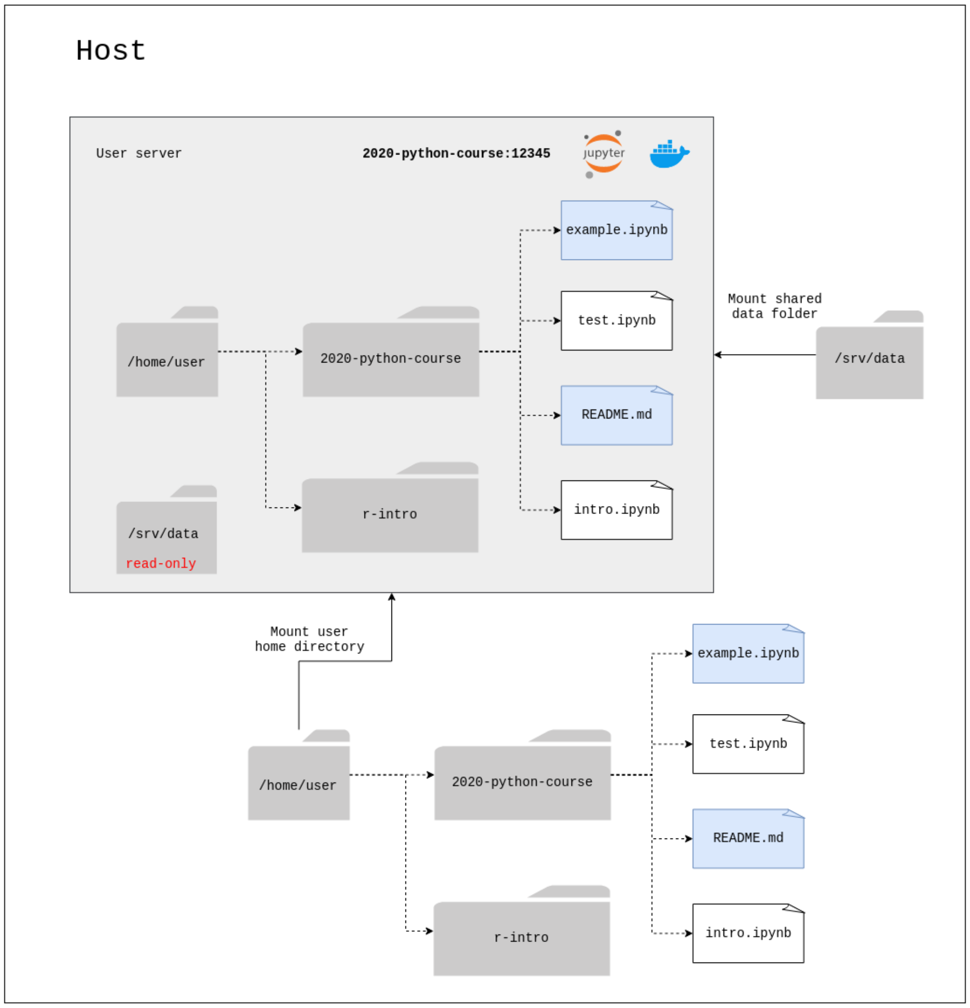
Concretely this means that the user inside the container corresponds to a real user that exists on the host. Processes will be started by that user, instead of the default `jovyan` user that is usually found in the regular Jupyter Docker images and on Binder.

For example when the user `foo` starts their server, the list of processes looks like the following:

```
foo@9cf23d669647:~$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  1.1  0.0  50944  3408 ?        Ss   11:17   0:00 su - foo -m -c "$0" "
↪$@" -- /srv/conda/envs/notebook/bin/jupyterhub-singleuser --ip=0.0.0.0 --port=8888 -
↪-NotebookApp.default_url=/lab --ResourceUseDisplay.track_cpu_percent=True
foo           32  5.4  0.8 399044 70528 ?        Ssl  11:17   0:01 /srv/conda/envs/
↪notebook/bin/python /srv/conda/envs/notebook/bin/jupyterhub-singleuser --ip=0.0.0.0␣
↪--port=8888 --NotebookApp.default_url=/lab --ResourceUseDisplay.track_cpu_
↪percent=True
foo           84  0.0  0.0  20312  4036 pts/0    Ss   11:17   0:00 /bin/bash -l
foo          112 29.0  0.5 458560 46448 ?        Ssl  11:17   0:00 /srv/conda/envs/
↪notebook/bin/python -m bash_kernel -f /home/foo/.local/share/jupyter/runtime/kernel-
↪9a7c8ad3-4ac2-4754-88cc-ef746d1be83e.json
foo          126  0.5  0.0  20180  3884 pts/1    Ss+  11:17   0:00 /bin/bash --rcfile /
↪srv/conda/envs/notebook/lib/python3.8/site-packages/pexpect/bashrc.sh
foo          140  0.0  0.0  36076  3368 pts/0    R+   11:17   0:00 ps aux
```

The following steps happen when a user starts their server:

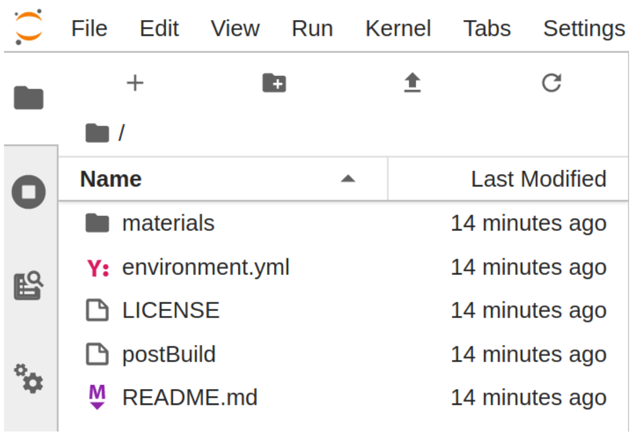
1. Mount the user home directory on the host into the container. This means that the file structure in the container reflects what is on the host.
2. A new directory is created in the user home directory for each new environment (i.e for each Docker image). For example if a user starts the `2020-python-course` environment, there will be a new folder created under `/home/user/2020-python-course`. This folder is then persisted to disk in the user home directory on the host. Any file and notebook created from the notebook interface are also persisted to disk.
3. On server startup, the entrypoint script copies the files from the base image that are initially in `/home/jovyan` to `/home/user/2020-python-course` in the container. They are then persisted in `/home/user/2020-python-course` on the host.



- The files highlighted in blue correspond to the files initially bundled in the environment. These files are copied to the environment subdirectory in the user home directory on startup.
- The other files are examples of files created by the user.

4.2.2 User server startup

The user server is started from the environment directory:



Name	Last Modified
materials	14 minutes ago
environment.yml	14 minutes ago
LICENSE	14 minutes ago
postBuild	14 minutes ago
README.md	14 minutes ago

The rest of the user files are mounted into the container, see *User Data*.

A user can for example open a terminal and access their files by typing `cd`.

They can then inspect their files:

```
foo@3e29b2297563:/home/foo$ ls -lisah
total 56K
 262882 4.0K drwxr-xr-x  9 foo  foo  4.0K Apr 21 16:53 .
6205024 4.0K drwxr-xr-x  1 root root 4.0K Apr 21 16:50 ..
 266730 4.0K -rw-------  1 foo  foo   228 Apr 21 14:41 .bash_history
 262927 4.0K -rw-r--r--  1 foo  foo   220 May  5  2019 .bash_logout
 262928 4.0K -rw-r--r--  1 foo  foo  3.7K May  5  2019 .bashrc
1043206 4.0K drwx------  3 foo  foo  4.0K Apr 21 09:26 .cache
 528378 4.0K drwx------  3 foo  foo  4.0K Apr 17 17:36 .gnupg
1565895 4.0K drwxrwxr-x  2 foo  foo  4.0K Apr 21 09:55 .ipynb_checkpoints
1565898 4.0K drwxr-xr-x  5 foo  foo  4.0K Apr 21 09:27 .ipython
1565880 4.0K drwxrwxr-x  3 foo  foo  4.0K Apr 21 09:26 .local
 262926 4.0K -rw-r--r--  1 foo  foo   807 May  5  2019 .profile
1050223 4.0K drwxrwxr-x 12 foo  foo  4.0K Apr 20 10:44 2020-python-course
1043222 4.0K drwxrwxr-x 13 foo  foo  4.0K Apr 20 17:07 r-intro
 258193 4.0K -rw-rw-r--  1 foo  foo   843 Apr 21 09:56 Untitled.ipynb
```

4.2.3 Shared Data

In addition to the user data, the plugin also mounts a shared data volume for all users.

The shared data is available under `/srv/data` inside the user server, as pictured in the diagram above.

On the host machine, the shared data should be placed under `/srv/data` as recommended in the TLJH documentation.

The shared data is **read-only**.

4.3 Resources

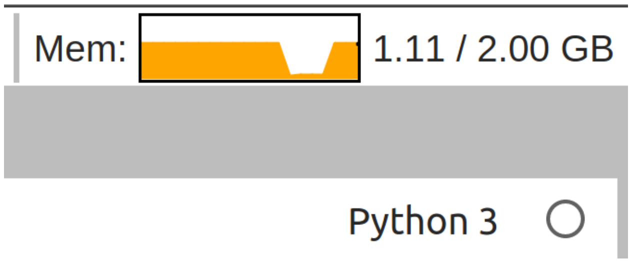
Plasma provides default values to limit the Memory and CPU usage.

4.3.1 Memory

By default Plasma sets a limit of `2GB` for each user server.

This limit is enforced by the operating system, which kills the process if the memory consumption goes aboved this threshold.

Users can monitor their memory usage using the indicator in the top bar area if the environment has these dependencies (see the *Displaying the indicators* section below).

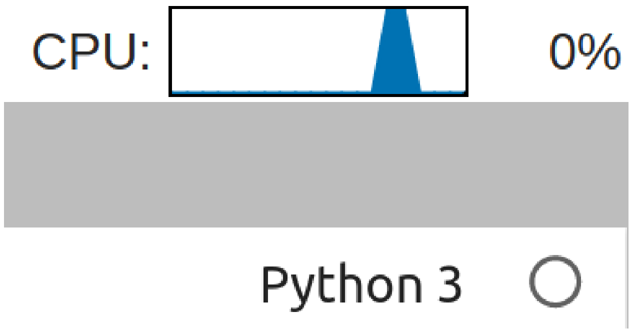


4.3.2 CPU

By default Plasma sets a limit of `2 cpus` for each user server.

This limit is enforced by the operating system, which throttles access to the CPU by the processes running in the Docker container.

Users can monitor their CPU usage using the indicator in the top bar area if the environment has these dependencies (see the *Displaying the indicators* section below).



4.3.3 Displaying the indicators

To enable the Memory and CPU indicators as shown above, the following dependencies must be added to the user environment:

- `nbresuse`
- `jupyterlab-topbar-extension`
- `jupyterlab-system-monitor`

As an example, checkout the following two links:

- Adding nbresuse
- Adding the JupyterLab extensions

4.4 Culling idle servers

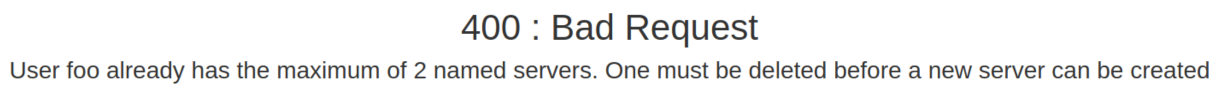
Plasma uses the same defaults as The Littlest JupyterHub for culling idle servers.

It overrides the `timeout` value to `3600`, which means that the user servers will be shut down if they have been idle for more than one hour.

4.5 Named Servers

By default users can run up to `2` simultaneous servers using the named servers functionality in JupyterHub.

If a user tries to start more servers, they will be shown the following message:



400 : Bad Request

User foo already has the maximum of 2 named servers. One must be deleted before a new server can be created

CHAPTER 5

Troubleshooting

Table of contents

5.1 How to SSH to the machine

First make sure your SSH key has been deployed to the server. See *Adding the public SSH key to the server* for more details.

Once the key is set up, connect to the machine over SSH using the following command:

```
ssh ubuntu@51.178.95.143
```

5.2 Looking at logs

See: The Littlest JupyterHub documentation.

5.3 Why is my environment not building?

If for some reasons an environment does not appear after *Adding a new environment*, it is possible that there are some issues building it and installing the dependencies.

We recommend building the environment either locally with `repo2docker` (next section) or on Binder.

See *Testing on Binder* and the repo2docker FAQ for more details.

5.4 Running the environments on my local machine

To run the same environments on a local machine, you can use `jupyter-repo2docker` with the following parameters:

```
jupyter-repo2docker --ref a4edf334c6b4b16be3a184d0d6e8196137ee1b06 https://github.com/
↪plasmabio/template-python
```

Update the parameters based on the image you would like to build.

This will create a Docker image and start it automatically once the build is complete.

Refer to the repo2docker documentation for more details.

5.5 My extension and / or dependency does not seem to be installed

See the two previous sections to investigate why they are missing.

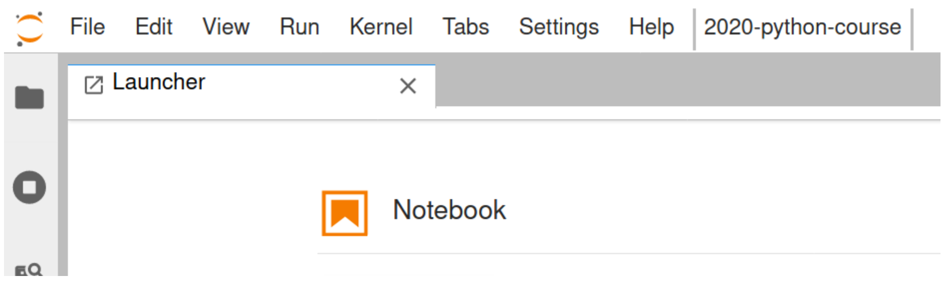
The logs might contain silent errors that did not cause the build to fail.

5.6 The name of the environment is not displayed in the top bar

This functionality requires the `jupyter-topbar-text` extension to be installed in the environment.

This extension must be added to the `postBuild` file of the repository. See this commit as an example.

The name of the environment will then be displayed as follows:



5.7 The environment is very slow to build

Since the environments are built as Docker images, they can leverage the Docker cache to make the builds faster.

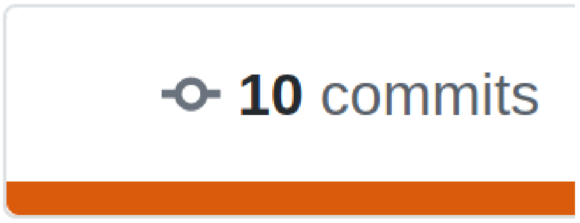
In some cases Docker will not be able to leverage the cache, for example when building a Python or R environment for the first time.

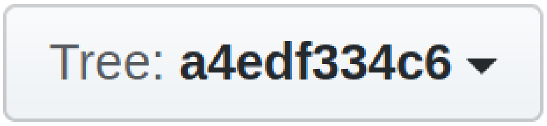
Another reason for the build to be slow could be the amount of dependencies specified in files such as `environment.yml` or `requirements.txt`.

Check out the previous section for more info on how to troubleshoot it.

5.8 Finding the source for an environment

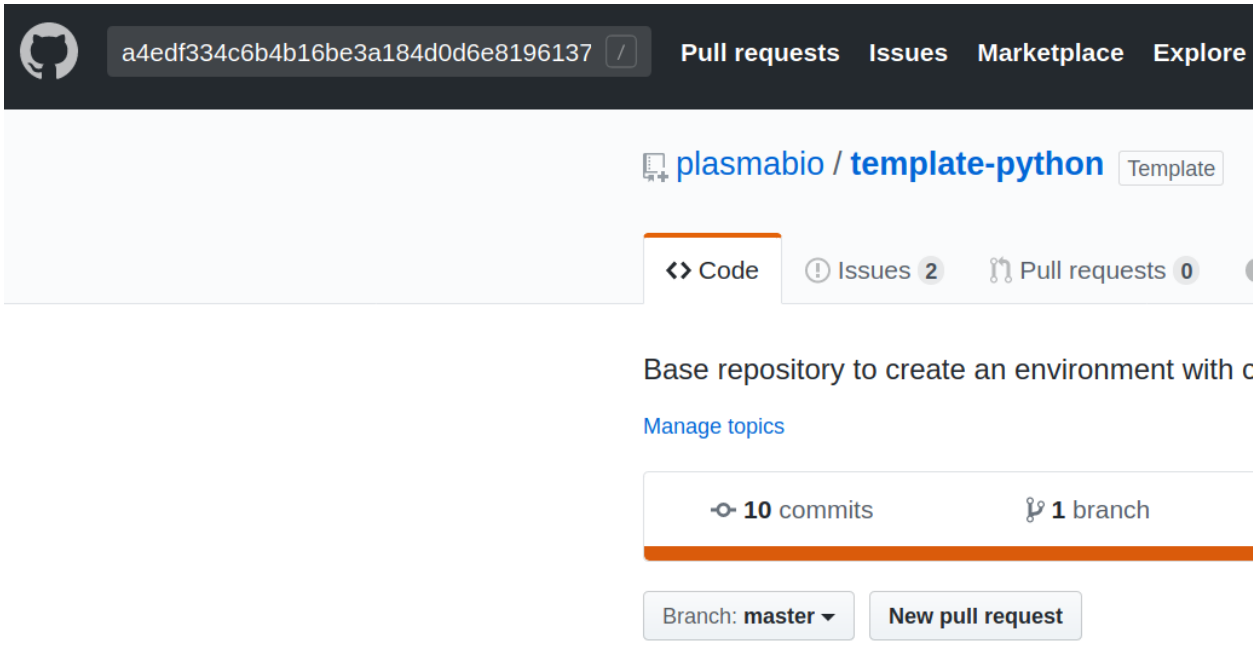
If you are managing the environments, you can click on the `Reference` link in the UI, which will open a new tab to the repository pointing the commit hash:





If you are using the environments, the name contains the information about the repository and the reference used to build the environment.

On the repository page, enter the reference in the search input box:



5.9 Removing an environment returns an error

See *Removing an environment returns an error* for more info.

CHAPTER 6

Contributing

Thanks for your interest in contributing to the project!

6.1 Writing Documentation

The documentation is available at docs.plasmabio.org and is written with Sphinx.

The Littlest JupyterHub has a good overview page on writing documentation, with externals links to the tools used to generate it.

First, create a new environment:

```
conda create -n plasma-docs -c conda-forge python
conda activate plasma-docs
```

In the `docs` folder, run:

```
python -m pip install -r requirements.txt
make html
```

Open `docs/_build/index.html` in a browser to start browsing the documentation.

Rerun `make html` after making any changes to the source.

6.2 Setting up a dev environment

It is possible to test the project locally without installing TLJH. Instead we use the `jupyterhub` Python package.

6.2.1 Requirements

`Docker` is used as a `Spawner` to start the user servers, and is then required to run the project locally.

Check out the official Docker documentation to know how to install Docker on your machine: https://docs.docker.com/install/linux/docker-ce/ubuntu/

6.2.2 Create a virtual environment

Using `conda`:

```
conda create -n plasma -c conda-forge python nodejs
conda activate plasma
```

Alternatively, with Python's built in `venv` module, you can create a virtual environment with:

```
python3 -m venv .
source bin/activate
```

6.2.3 Install the development requirements

```
pip install -r dev-requirements.txt

# dev install of the plasma package
pip install -e tljh-plasma

# Install (https://github.com/jupyterhub/configurable-http-proxy)
npm -g install configurable-http-proxy
```

6.2.4 Pull the repo2docker Docker image

User environments are built with `repo2docker` running in a Docker container. To pull the Docker image:

```
docker pull jupyter/repo2docker
```

6.2.5 Run

Finally, start `jupyterhub` with the config in `debug` mode:

```
python3 -m jupyterhub -f jupyterhub_config.py --debug
```

Open https://localhost:8000 in a web browser.