# The Littlest JupyterHub

***Release v0.1***

**Jun 09, 2020**

# Contents

A simple JupyterHub distribution for a small (0-100) number of users on a single server. We recommend reading *When to use The Littlest JupyterHub* to determine if this is the right tool for you.

# CHAPTER 1

## Development Status

This project is currently in **beta** state. Folks have been using installations of TLJH for more than a year now to great success. While we try hard not to, we might still make breaking changes that have no clear upgrade pathway.

# CHAPTER 2

# Installation

The Littlest JupyterHub (TLJH) can run on any server that is running at least **Ubuntu 18.04**. Earlier versions of Ubuntu are not supported. We have a bunch of tutorials to get you started.

- Tutorials to create a new server from scratch on a cloud provider & run TLJH on it. These are **recommended** if you do not have much experience setting up servers.

## 2.1 Installing

The Littlest JupyterHub (TLJH) can run on any server that is running at least **Ubuntu 18.04**. Earlier versions of Ubuntu are not supported. We have a bunch of tutorials to get you started.

Tutorials to create a new server from scratch on a cloud provider & run TLJH on it. These are **recommended** if you do not have much experience setting up servers.

### 2.1.1 Installing on Digital Ocean

#### Goal

By the end of this tutorial, you should have a JupyterHub with some admin users and a user environment with packages you want installed running on DigitalOcean.

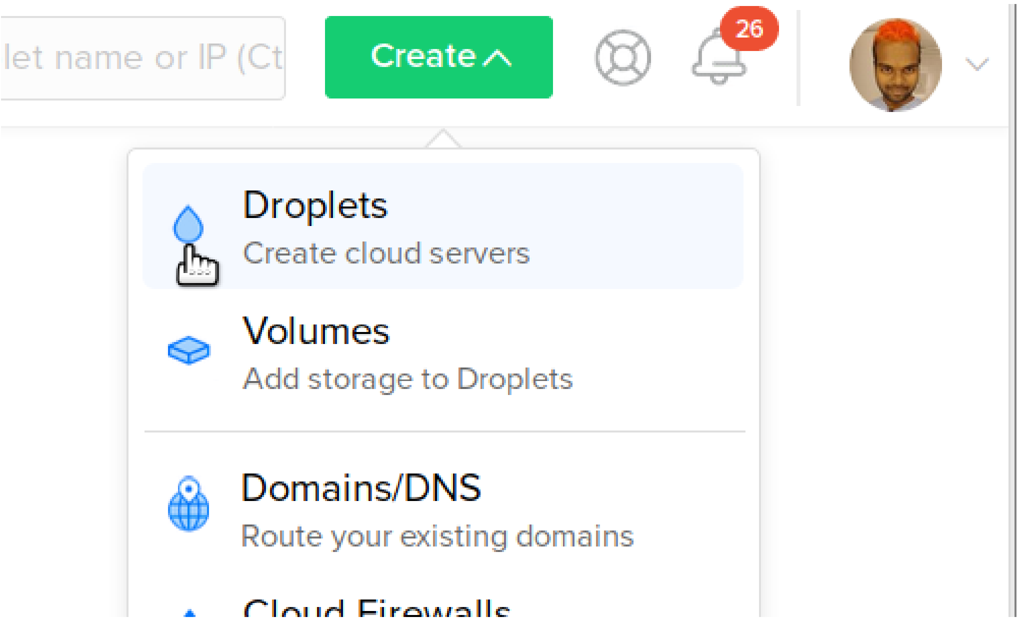
#### Pre-requisites

1. A DigitalOcean account with a payment method attached.

#### Step 1: Installing The Littlest JupyterHub

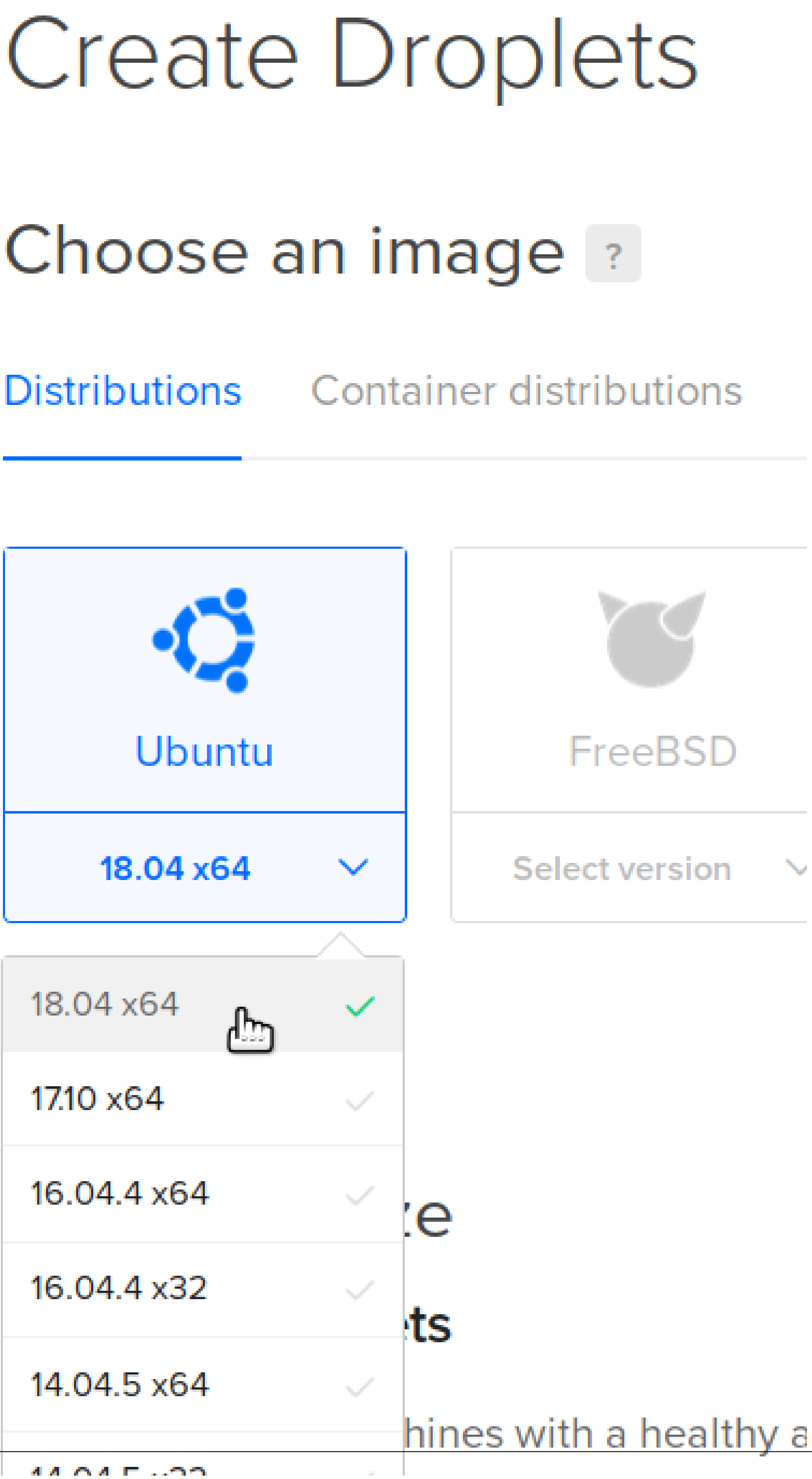
Let's create the server on which we can run JupyterHub.

1. Log in to DigitalOcean. You might need to attach a credit card or other payment method to your account before you can proceed with the tutorial.
2. Click the **Create** button on the top right, and select **Droplets** from the dropdown menu. DigitalOcean calls servers **droplets**.



This takes you to a page titled **Create Droplets** that lets you configure your server.

3. Under **Choose an image**, select **18.04 x64** under **Ubuntu**.



Create Droplets

Choose an image ?

Distributions | Container distributions

Ubuntu — 18.04 x64

FreeBSD — Select version

18.04 x64 ✓

17.10 x64

16.04.4 x64

16.04.4 x32

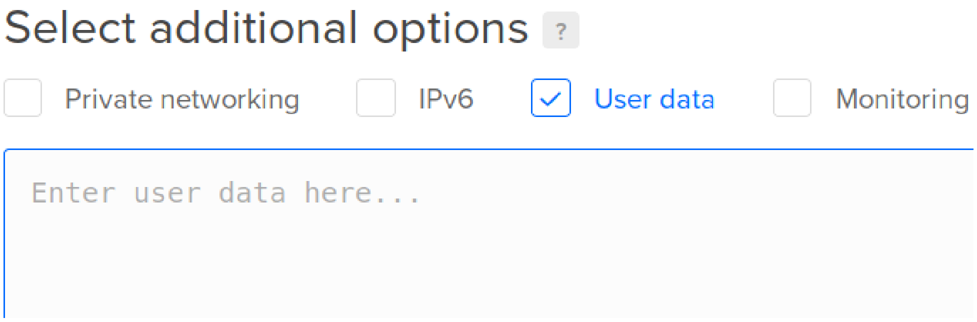
14.04.5 x64

14.04.5 x32

4. Under **Choose a size**, select the size of the server you want. The default (4GB RAM, 2CPUs, 20 USD / month) is not a bad start. You can resize your server later if you need.

   Check out our guide on How To *Estimate Memory / CPU / Disk needed* to help pick how much Memory, CPU & disk space your server needs.

5. Scroll down to **Select additional options**, and select **User data**.

   Select additional options ?

   □ Private networking □ IPv6 ☑ User data □ Monitoring

   Enter user data here...

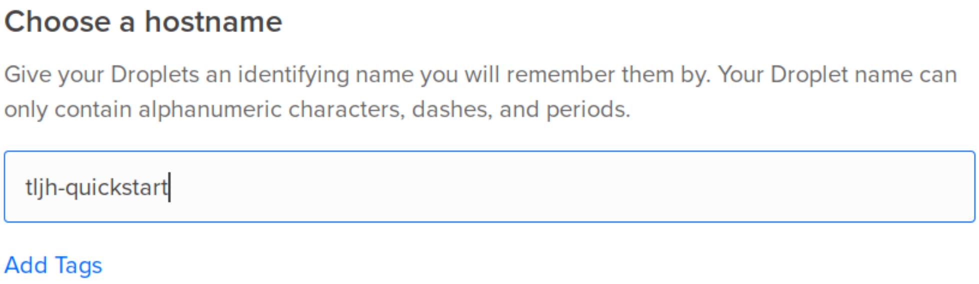
   This opens up a textbox where you can enter a script that will be run when the server is created. We will use this to set up The Littlest JupyterHub on this server.

6. Copy the text below, and paste it into the user data text box. Replace `<admin-user-name>` with the name of the first **admin user** for this JupyterHub. This admin user can log in after the JupyterHub is set up, and can configure it to their needs. **Remember to add your username**!

```
#!/bin/bash
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-
↪jupyterhub/master/bootstrap/bootstrap.py \
  | sudo python3 - \
    --admin <admin-user-name>
```

   **Note:** See *What does the installer do?* if you want to understand exactly what the installer is doing. *Customizing the Installer* documents other options that can be passed to the installer.

7. Under the **Finalize and create** section, enter a `hostname` that descriptively identifies this server for you.

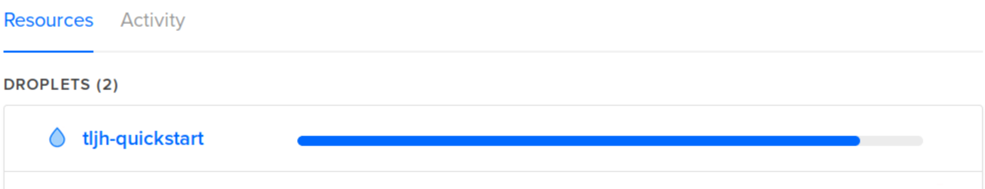
   

   Choose a hostname

   Give your Droplets an identifying name you will remember them by. Your Droplet name can only contain alphanumeric characters, dashes, and periods.

   tljh-quickstart

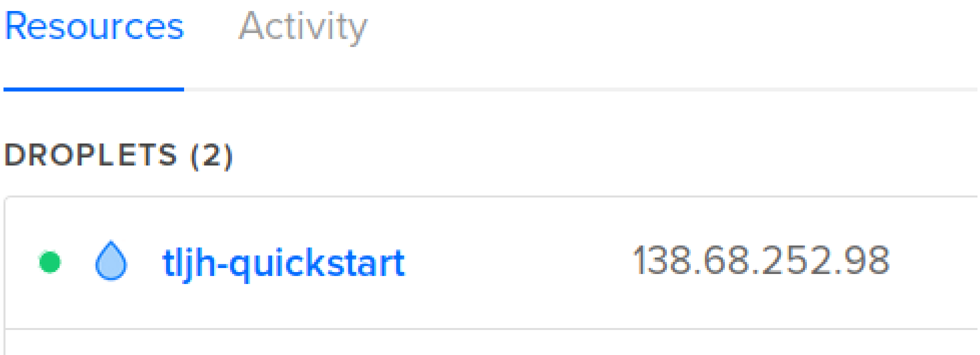
8. Click the **Create** button! You will be taken to a different screen, where you can see progress of your server being created.



DROPLETS (2)

tljh-quickstart

9. In a few seconds your server will be created, and you can see the **public IP** used to access it.



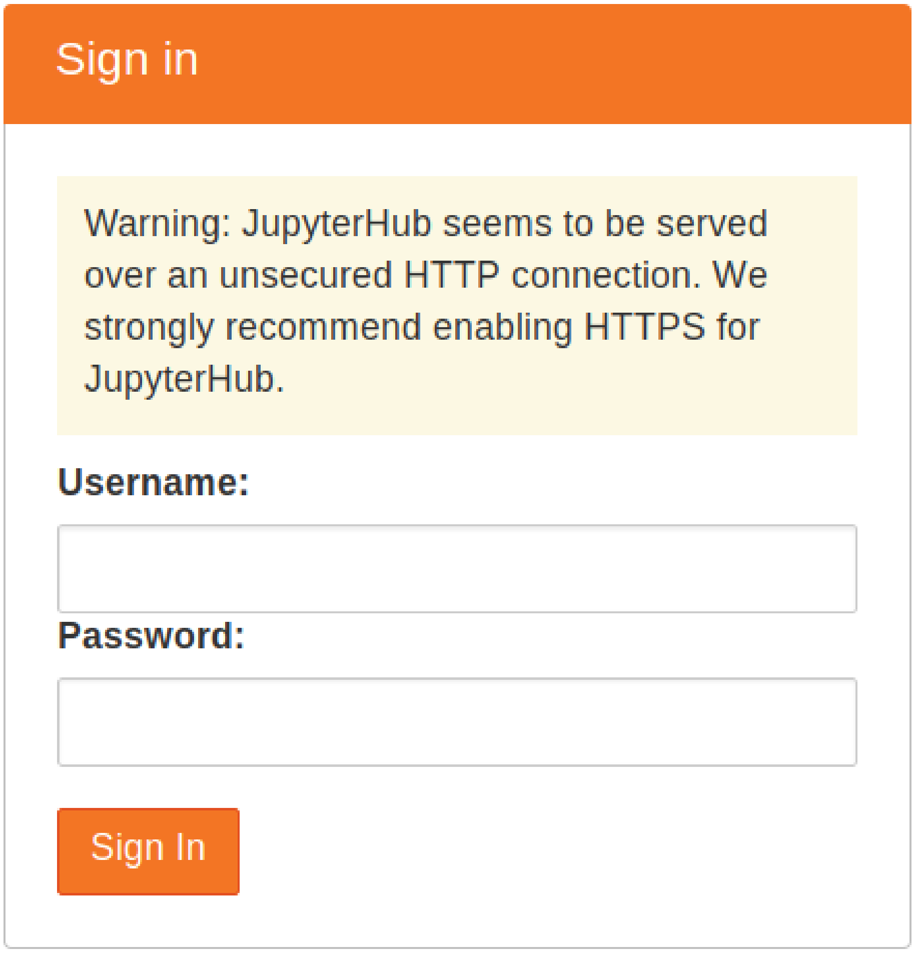
DROPLETS (2)

tljh-quickstart 138.68.252.98

10. The Littlest JupyterHub is now installing in the background on your new server. It takes around 5-10 minutes for this installation to complete.

11. Check if the installation is complete by copying the **public ip** of your server, and trying to access it with a browser. This will fail until the installation is complete, so be patient.

12. When the installation is complete, it should give you a JupyterHub login page.



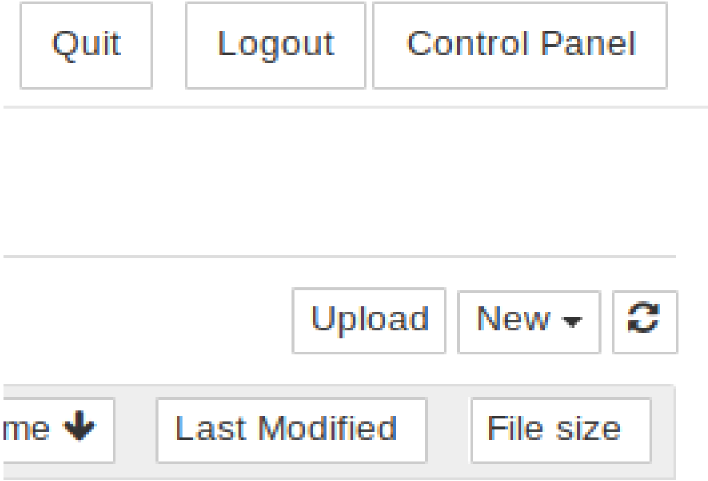
Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub.

13. Login using the **admin user name** you used in step 6, and a password. Use a strong password & note it down somewhere, since this will be the password for the admin user account from now on.
14. Congratulations, you have a running working JupyterHub!

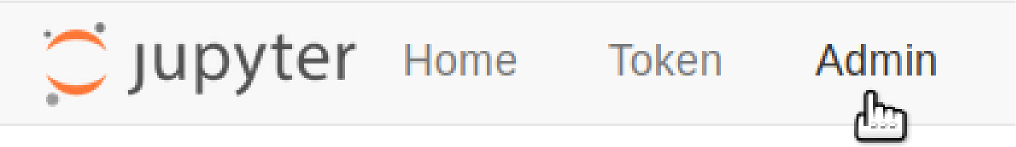
### Step 2: Adding more users

Most administration & configuration of the JupyterHub can be done from the web UI directly. Let's add a few users who can log in!

1. Open the **Control Panel** by clicking the control panel button on the top right of your JupyterHub.

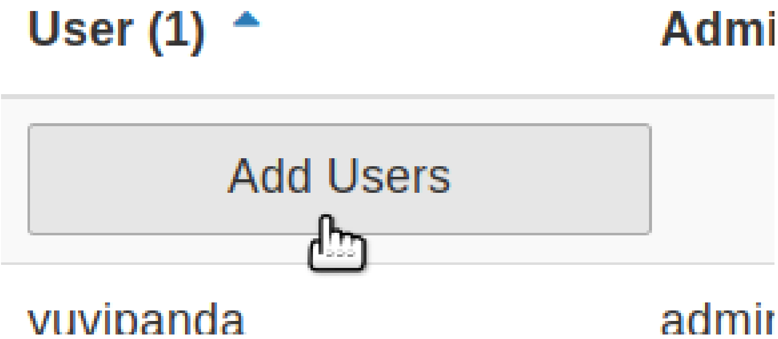


2. In the control panel, open the **Admin** link in the top left.



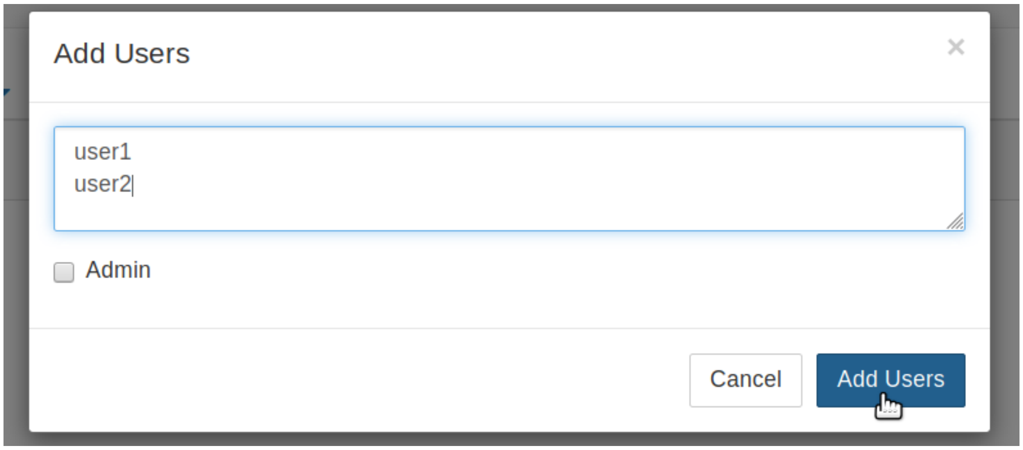
This opens up the JupyterHub admin page, where you can add / delete users, start / stop peoples' servers and see who is online.

3. Click the **Add Users** button.



A **Add Users** dialog box opens up.

4. Type the names of users you want to add to this JupyterHub in the dialog box, one per line.



user1
user2

You can tick the **Admin** checkbox if you want to give admin rights to all these users too.

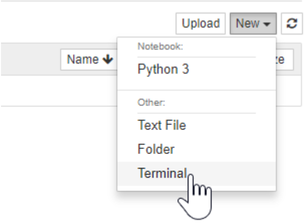
5. Click the **Add Users** button in the dialog box. Your users are now added to the JupyterHub! When they log in for the first time, they can set their password - and use it to log in again in the future.

Congratulations, you now have a multi user JupyterHub that you can add arbitrary users to!

### Step 3: Install conda / pip packages for all users

The **User Environment** is a conda environment that is shared by all users in the JupyterHub. Libraries installed in this environment are immediately available to all users. Admin users can install packages in this environment with `sudo -E`.

1. Log in as an admin user and open a Terminal in your Jupyter Notebook.



2. Install gdal from conda-forge.

```
sudo -E conda install -c conda-forge gdal
```

The `sudo -E` is very important!

3. Install `there` with `pip`

```
sudo -E pip install there
```

The packages `gdal` and `there` are now available to all users in JupyterHub. If a user already had a python notebook running, they have to restart their notebook's kernel to make the new libraries available.

See *Install conda, pip or apt packages* for more information.

## 2.1.2 Installing on OVH

### Goal

By the end of this tutorial, you should have a JupyterHub with some admin users and a user environment with packages you want installed running on OVH.

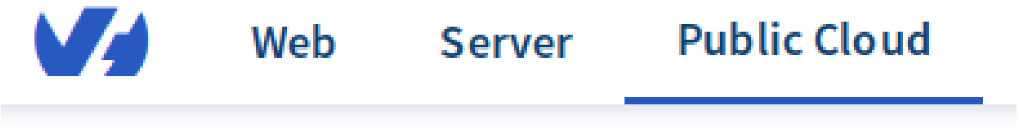
### Pre-requisites

1. An OVH account.

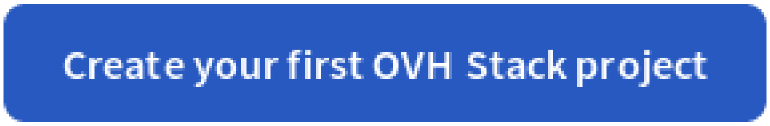
### Step 1: Installing The Littlest JupyterHub

Let's create the server on which we can run JupyterHub.

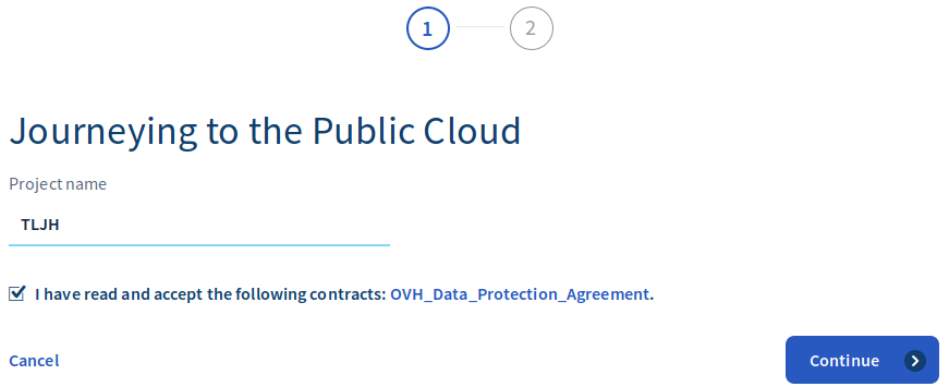
1. Log in to the OVH Control Panel.
2. Click the **Public Cloud** button in the navigation bar.



3. If you don't have an OVH Stack, you can create one by clicking on the following button:



4. Select a name for the project:



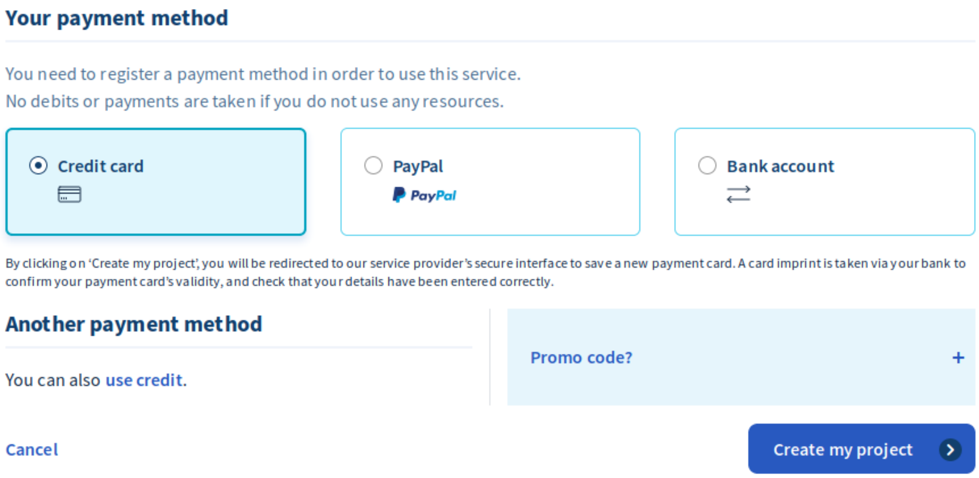
Journeying to the Public Cloud

Project name

TLJH

☑ I have read and accept the following contracts: OVH_Data_Protection_Agreement.

5. If you don't have a payment method yet, select one and click on "Create my project":



**Your payment method**

You need to register a payment method in order to use this service.
No debits or payments are taken if you do not use any resources.

● Credit card

○ PayPal

○ Bank account

By clicking on 'Create my project', you will be redirected to our service provider's secure interface to save a new payment card. A card imprint is taken via your bank to confirm your payment card's validity, and check that your details have been entered correctly.

**Another payment method**

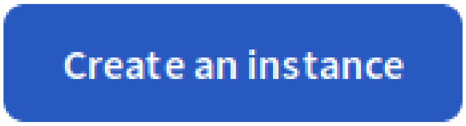
You can also use credit.

Promo code? +

Cancel

Create my project

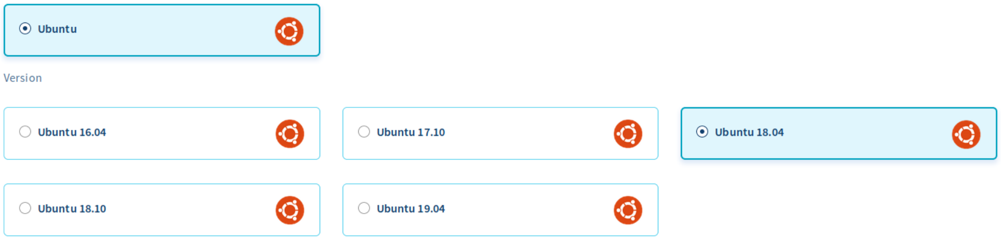
6. Using the **Public Cloud interface**, click on **Create an instance**:



7. **Select a model** for the instance. A good start is the **S1-4** model under **Shared resources** which comes with 4GB RAM, 1 vCores and 20GB SSD.

8. **Select a region**.

9. Select **Ubuntu 18.04** as the image:



● Ubuntu

Version

○ Ubuntu 16.04

○ Ubuntu 17.10

● Ubuntu 18.04

○ Ubuntu 18.10

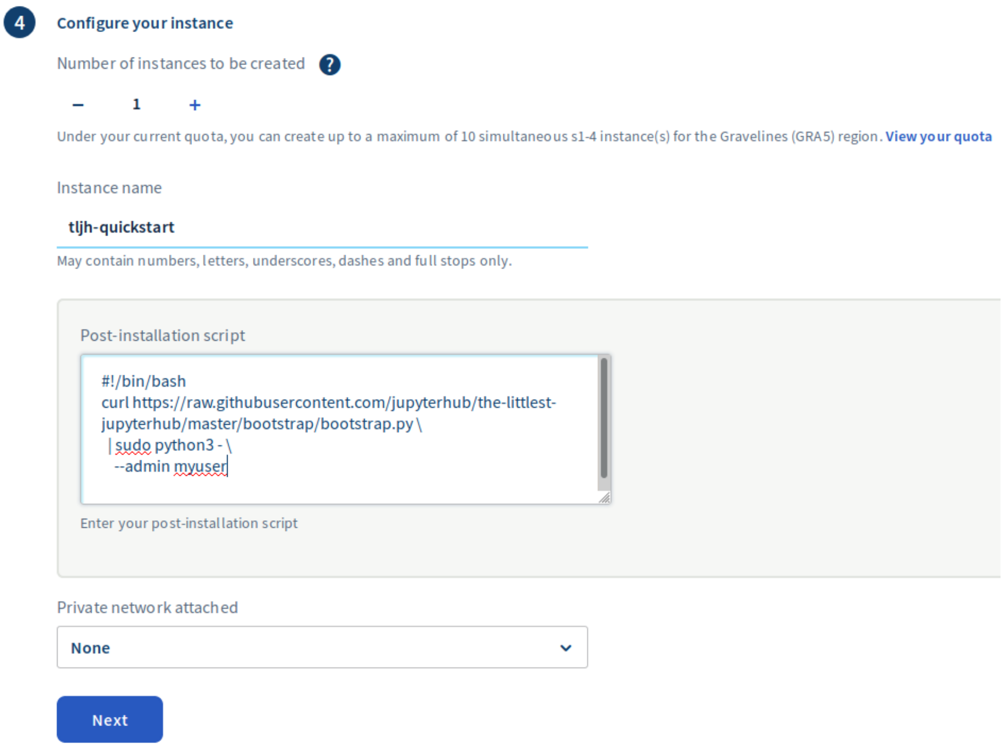
○ Ubuntu 19.04

10. OVH requires setting an SSH key to be able to connect to the instance. You can create a new SSH by following these instructions. Be sure to copy the content of the `~/.ssh/id_rsa.pub` file, which corresponds to the **public part** of the SSH key.

11. Select **Configure your instance**, and select a name for the instance. Under **Post-installation script**, copy the text below and paste it in the text box. Replace `<admin-user-name>` with the name of the first **admin user** for this JupyterHub. This admin user can log in after the JupyterHub is set up, and can configure it to their needs. **Remember to add your username**!

```
#!/bin/bash
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-
↪jupyterhub/master/bootstrap/bootstrap.py \
  | sudo python3 - \
    --admin <admin-user-name>
```

**Note:** See *What does the installer do?* if you want to understand exactly what the installer is doing. *Customizing the Installer* documents other options that can be passed to the installer.



Configure your instance

Number of instances to be created ?

1

Under your current quota, you can create up to a maximum of 10 simultaneous s1-4 instance(s) for the Gravelines (GRA5) region. View your quota

Instance name

tljh-quickstart

May contain numbers, letters, underscores, dashes and full stops only.

Post-installation script

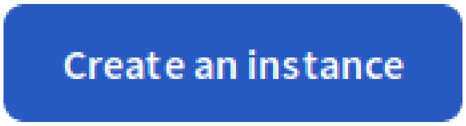
```
#!/bin/bash
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-
jupyterhub/master/bootstrap/bootstrap.py \
 | sudo python3 - \
  --admin myuser
```

Enter your post-installation script

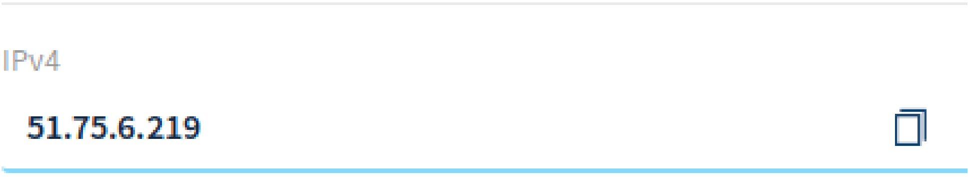
Private network attached

None

12. Select a billing period: monthly or hourly.
13. Click the **Create an instance** button! You will be taken to a different screen, where you can see progress of your server being created.



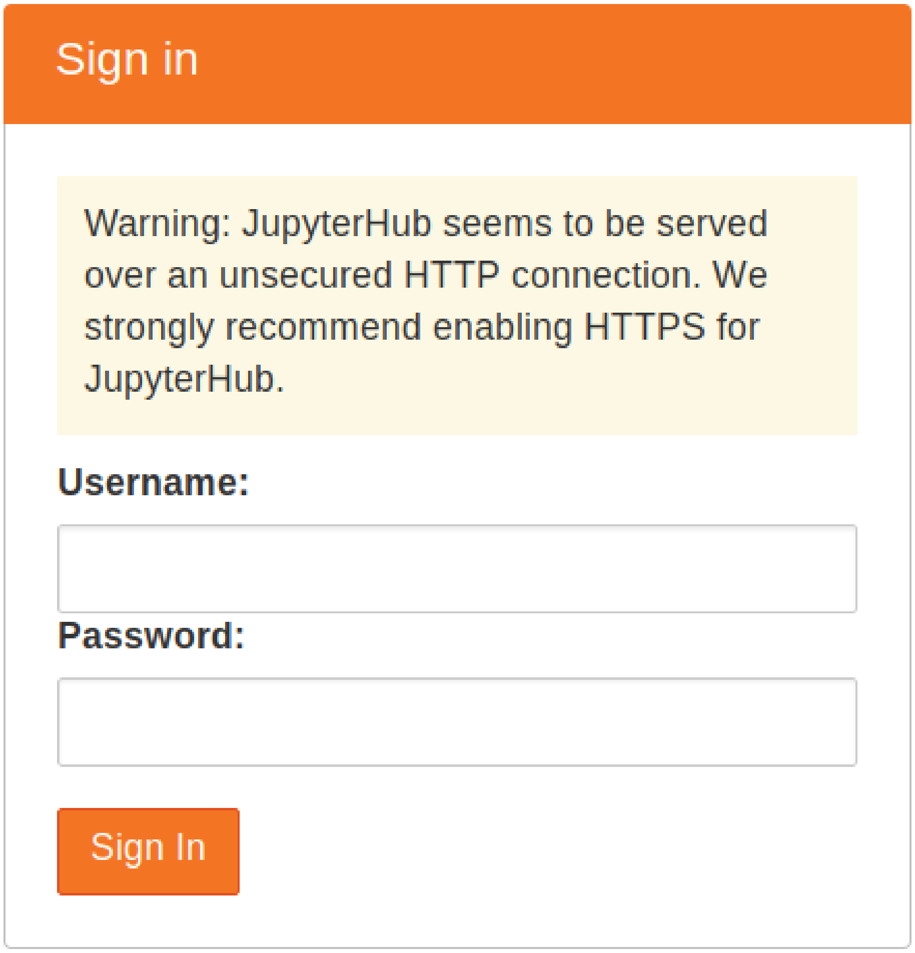
14. In a few seconds your server will be created, and you can see the **public IP** used to access it.



IPv4

51.75.6.219

15. The Littlest JupyterHub is now installing in the background on your new server. It takes around 5-10 minutes for this installation to complete.
16. Check if the installation is complete by copying the **public ip** of your server, and trying to access it with a browser. This will fail until the installation is complete, so be patient.
17. When the installation is complete, it should give you a JupyterHub login page.



Sign in

Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub.

Username:

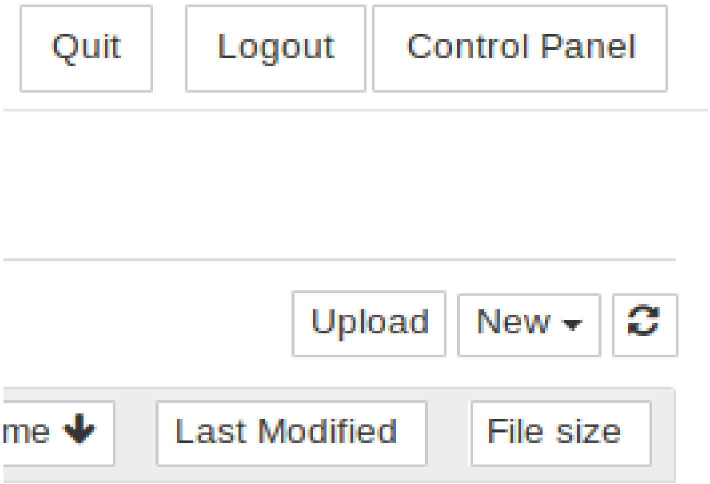
Password:

18. Login using the **admin user name** you used in step 6, and a password. Use a strong password & note it down somewhere, since this will be the password for the admin user account from now on.
19. Congratulations, you have a running working JupyterHub!

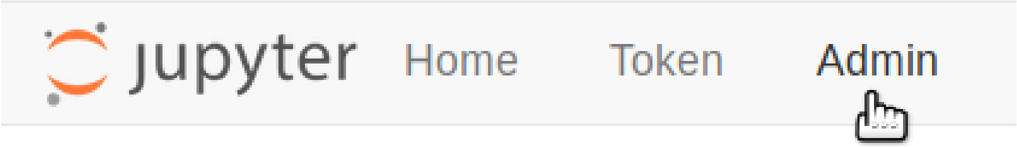
### Step 2: Adding more users

Most administration & configuration of the JupyterHub can be done from the web UI directly. Let's add a few users who can log in!

1. Open the **Control Panel** by clicking the control panel button on the top right of your JupyterHub.

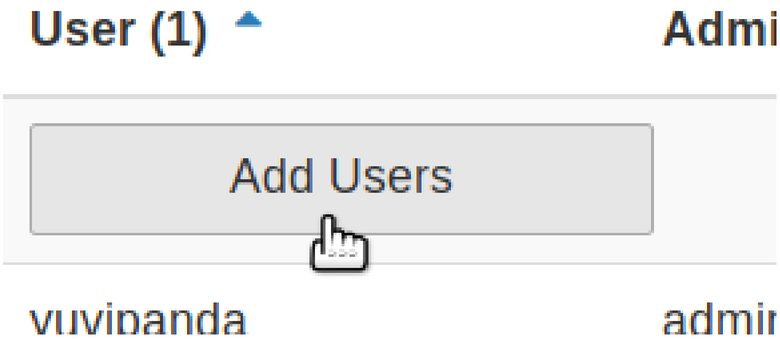


2. In the control panel, open the **Admin** link in the top left.



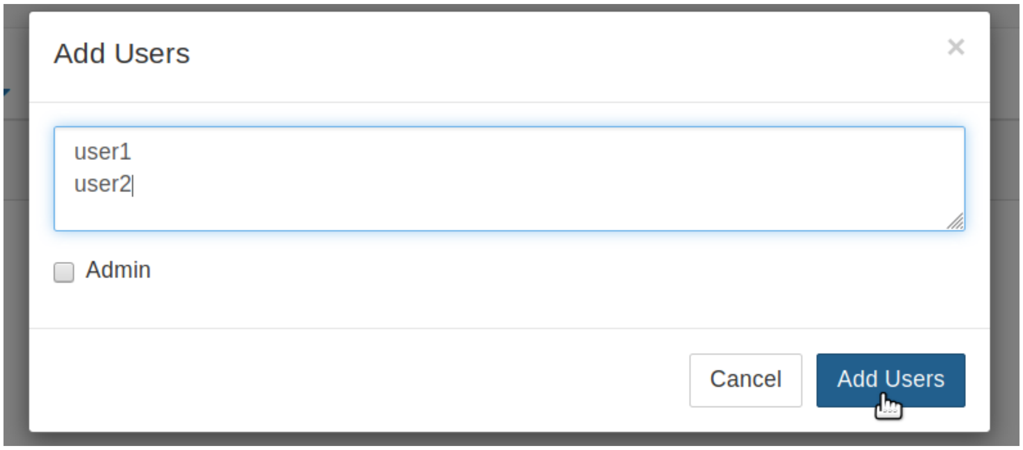
This opens up the JupyterHub admin page, where you can add / delete users, start / stop peoples' servers and see who is online.

3. Click the **Add Users** button.



A **Add Users** dialog box opens up.

4. Type the names of users you want to add to this JupyterHub in the dialog box, one per line.



user1
user2

You can tick the **Admin** checkbox if you want to give admin rights to all these users too.

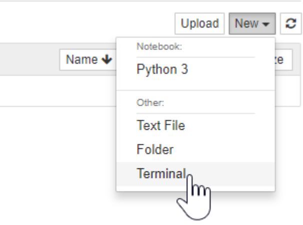
5. Click the **Add Users** button in the dialog box. Your users are now added to the JupyterHub! When they log in for the first time, they can set their password - and use it to log in again in the future.

Congratulations, you now have a multi user JupyterHub that you can add arbitrary users to!

### Step 3: Install conda / pip packages for all users

The **User Environment** is a conda environment that is shared by all users in the JupyterHub. Libraries installed in this environment are immediately available to all users. Admin users can install packages in this environment with `sudo -E`.

1. Log in as an admin user and open a Terminal in your Jupyter Notebook.



2. Install gdal from conda-forge.

```
sudo -E conda install -c conda-forge gdal
```

   The `sudo -E` is very important!

3. Install `there` with `pip`

```
sudo -E pip install there
```

The packages `gdal` and `there` are now available to all users in JupyterHub. If a user already had a python notebook running, they have to restart their notebook's kernel to make the new libraries available.

See *Install conda, pip or apt packages* for more information.

### 2.1.3 Installing on Jetstream

#### Goal

By the end of this tutorial, you should have a JupyterHub with some admin users and a user environment with packages you want installed running on Jetstream.

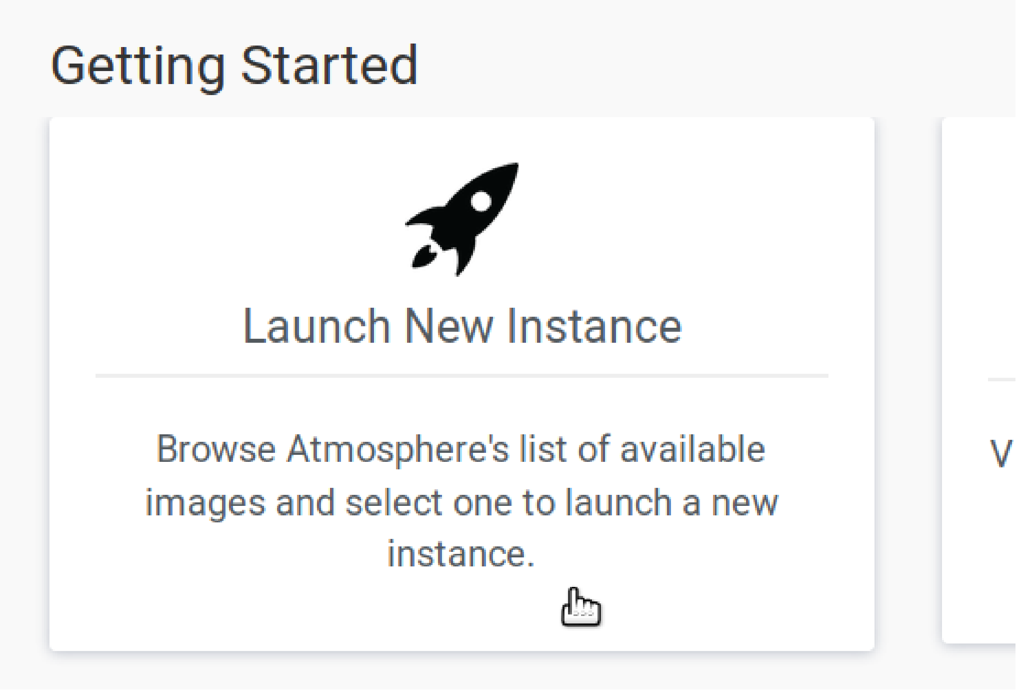
#### Prerequisites

1. A Jetstream account with an XSEDE allocation; for more information, see the Jetstream Allocations help page.

#### Step 1: Installing The Littlest JupyterHub

Let's create the server on which we can run JupyterHub.

1. Log in to the Jetstream portal. You need an allocation to launch instances.
2. Select the **Launch New Instance** option to get going.



Browse Atmosphere's list of available images and select one to launch a new instance.

   This takes you to a page with a list of base images you can choose for your server.

3. Under **Image Search**, search for **Ubuntu 18.04**, and select the **Ubuntu 18.04 Devel and Docker** image.

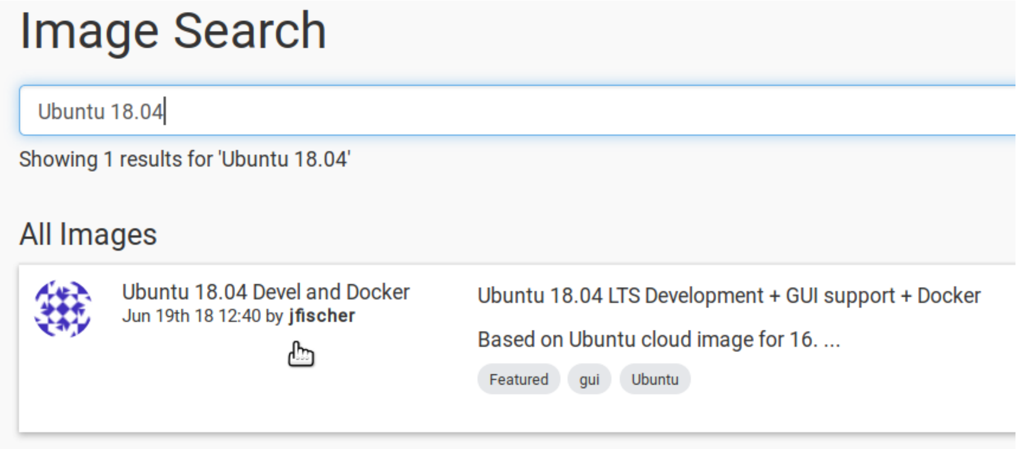


Image Search

Ubuntu 18.04

Showing 1 results for 'Ubuntu 18.04'

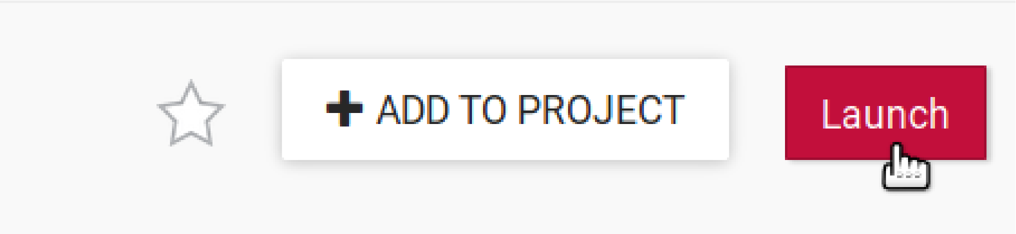
All Images

Ubuntu 18.04 Devel and Docker
Jun 19th 18 12:40 by jfischer

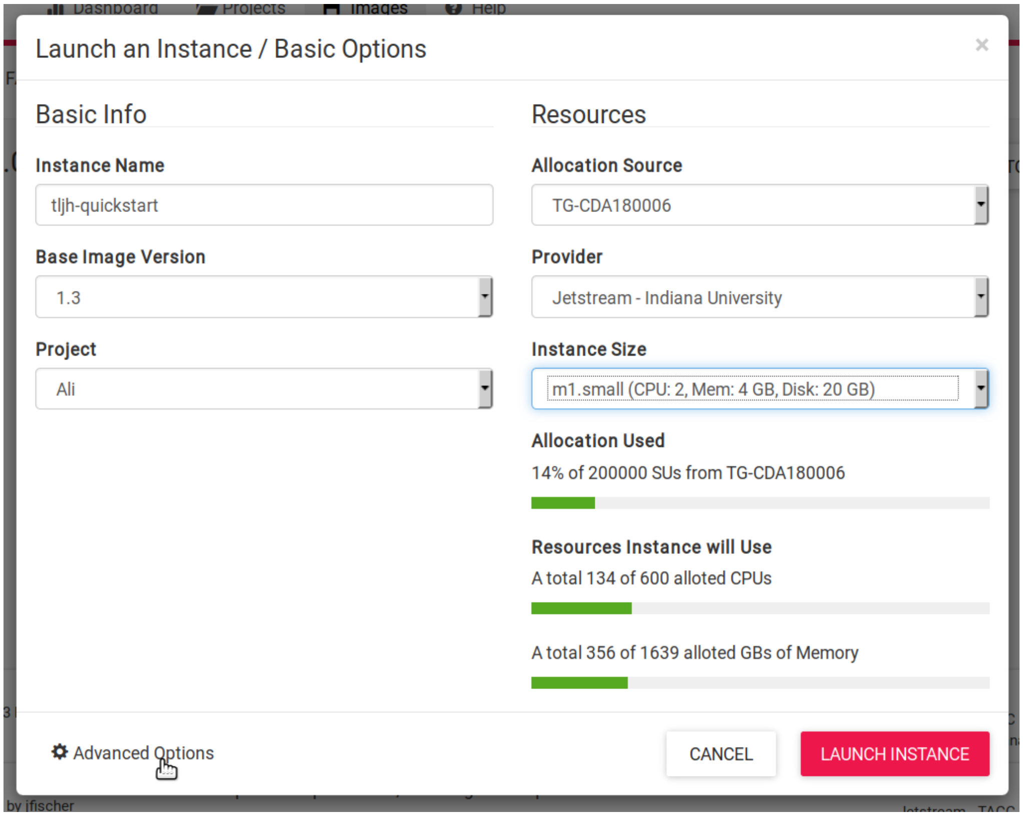
Ubuntu 18.04 LTS Development + GUI support + Docker

Based on Ubuntu cloud image for 16. ...

4. Once selected, you will see more information about this image. Click the **Launch** button on the top right.



5. A dialog titled **Launch an Instance / Basic Options** pops up, with various options for configuring your instance.



Launch an Instance / Basic Options

Basic Info

Instance Name: tljh-quickstart

Base Image Version: 1.3

Project: Ali

Resources

Allocation Source: TG-CDA180006

Provider: Jetstream - Indiana University

Instance Size: m1.small (CPU: 2, Mem: 4 GB, Disk: 20 GB)

Allocation Used

14% of 200000 SUs from TG-CDA180006

Resources Instance will Use

A total 134 of 600 alloted CPUs

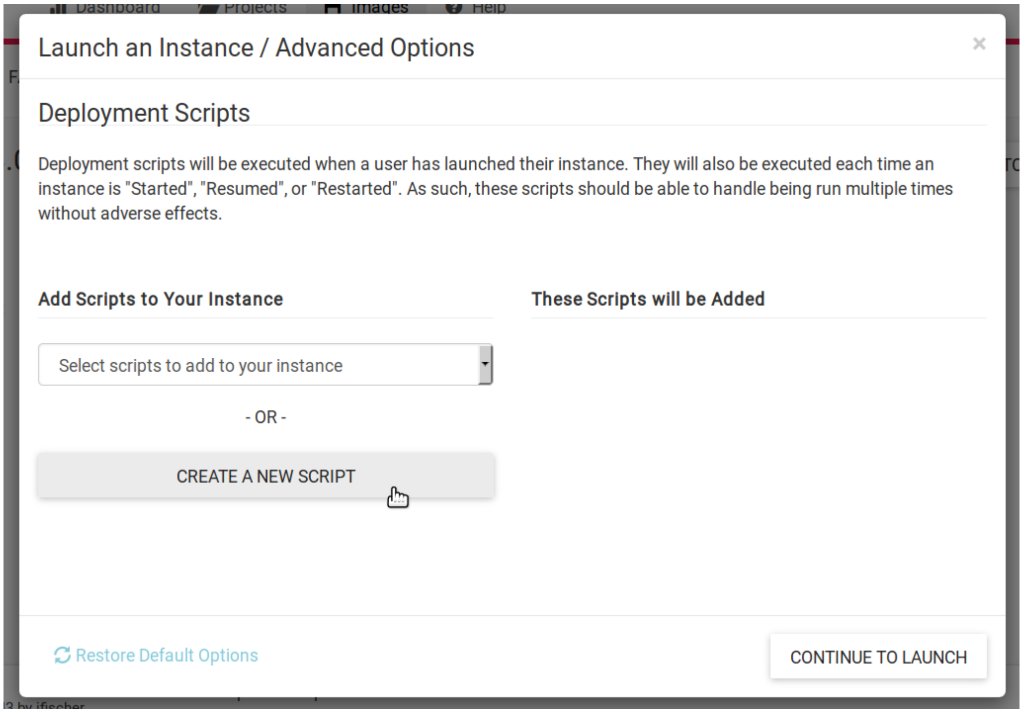
A total 356 of 1639 alloted GBs of Memory

   1. Give your server a descriptive **Instance Name**.
   2. Select an appropriate **Instance Size**. We suggest m1.medium or larger. Make sure your instance has at least **1.15GB** of RAM.

      Check out our guide on How To *Estimate Memory / CPU / Disk needed* to help pick how much Memory, CPU & disk space your server needs.
   3. If you have multiple allocations, make sure you are 'charging' this server to the correct allocation.

6. Click the **Advanced Options** link in the bottom left of the popup. This lets us configure what the server should do when it starts up. We will use this to install The Littlest JupyterHub.

   A dialog titled **Launch an Instance / Advanced Options** should pop up.



Launch an Instance / Advanced Options

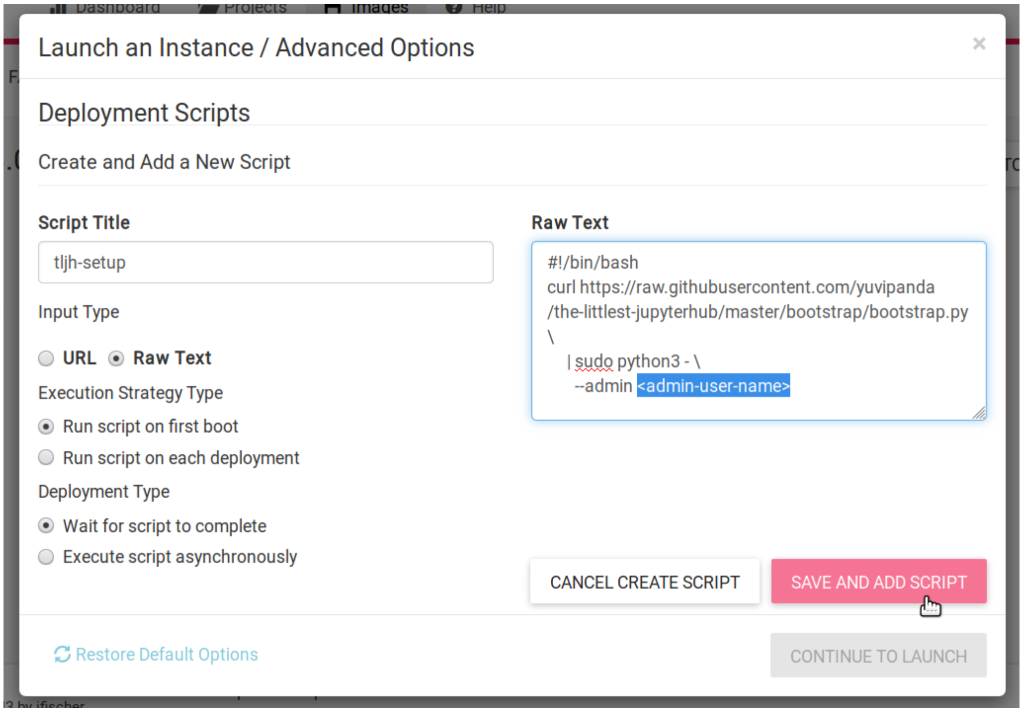
Deployment Scripts

Deployment scripts will be executed when a user has launched their instance. They will also be executed each time an instance is "Started", "Resumed", or "Restarted". As such, these scripts should be able to handle being run multiple times without adverse effects.

Add Scripts to Your Instance

These Scripts will be Added

7. Click the **Create New Script** button. This will open up another dialog box!



Launch an Instance / Advanced Options

Deployment Scripts

Create and Add a New Script

Script Title: tljh-setup

Input Type

○ URL ● Raw Text

Execution Strategy Type

● Run script on first boot

○ Run script on each deployment

Deployment Type

● Wait for script to complete

○ Execute script asynchronously

Raw Text:

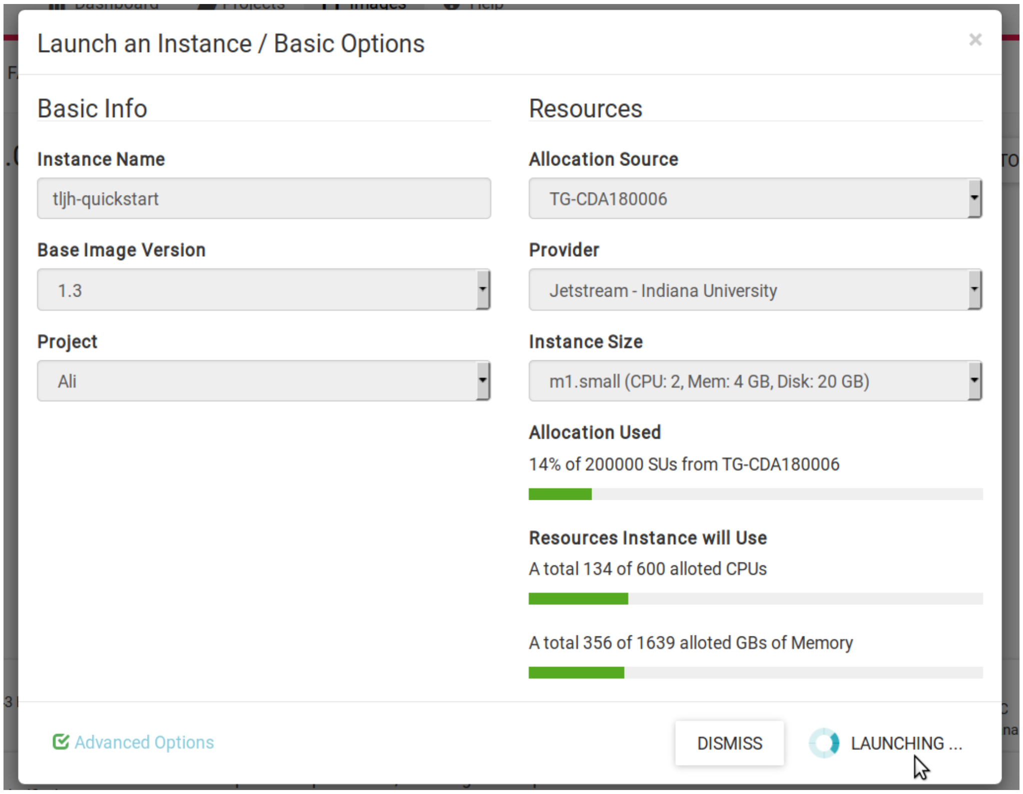
```
#!/bin/bash
curl https://raw.githubusercontent.com/yuvipanda
/the-littlest-jupyterhub/master/bootstrap/bootstrap.py
\
   | sudo python3 - \
    --admin <admin-user-name>
```

8. Under **Input Type**, select **Raw Text**. This should make a text box titled **Raw Text** visible on the right side of the dialog box. Copy the text below, and paste it into the **Raw Text** text box. Replace `<admin-user-name>` with the name of the first **admin user** for this JupyterHub. This admin user can log in after the JupyterHub is set up, and can configure it to their needs. **Remember to add your username**!

```
#!/bin/bash
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-
↪jupyterhub/master/bootstrap/bootstrap.py \
  | sudo python3 - \
    --admin <admin-user-name>
```

**Note:** See *What does the installer do?* if you want to understand exactly what the installer is doing. *Customizing the Installer* documents other options that can be passed to the installer.

9. Under **Execution Strategy Type**, select **Run script on first boot**.
10. Under **Deployment Type**, select **Wait for script to complete**.
11. Click the **Save and Add Script** button on the bottom right. This should hide the dialog box.
12. Click the **Continue to Launch** button on the bottom right. This should put you back in the **Launch an Instance / Basic Options** dialog box again.
13. Click the **Launch Instance** button on the bottom right. This should turn it into a spinner, and your server is getting created!



Launch an Instance / Basic Options

Basic Info

Instance Name: tljh-quickstart

Base Image Version: 1.3

Project: Ali

Resources

Allocation Source: TG-CDA180006

Provider: Jetstream - Indiana University

Instance Size: m1.small (CPU: 2, Mem: 4 GB, Disk: 20 GB)

Allocation Used

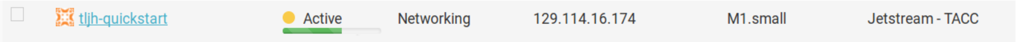
14% of 200000 SUs from TG-CDA180006

Resources Instance will Use

A total 134 of 600 alloted CPUs

A total 356 of 1639 alloted GBs of Memory

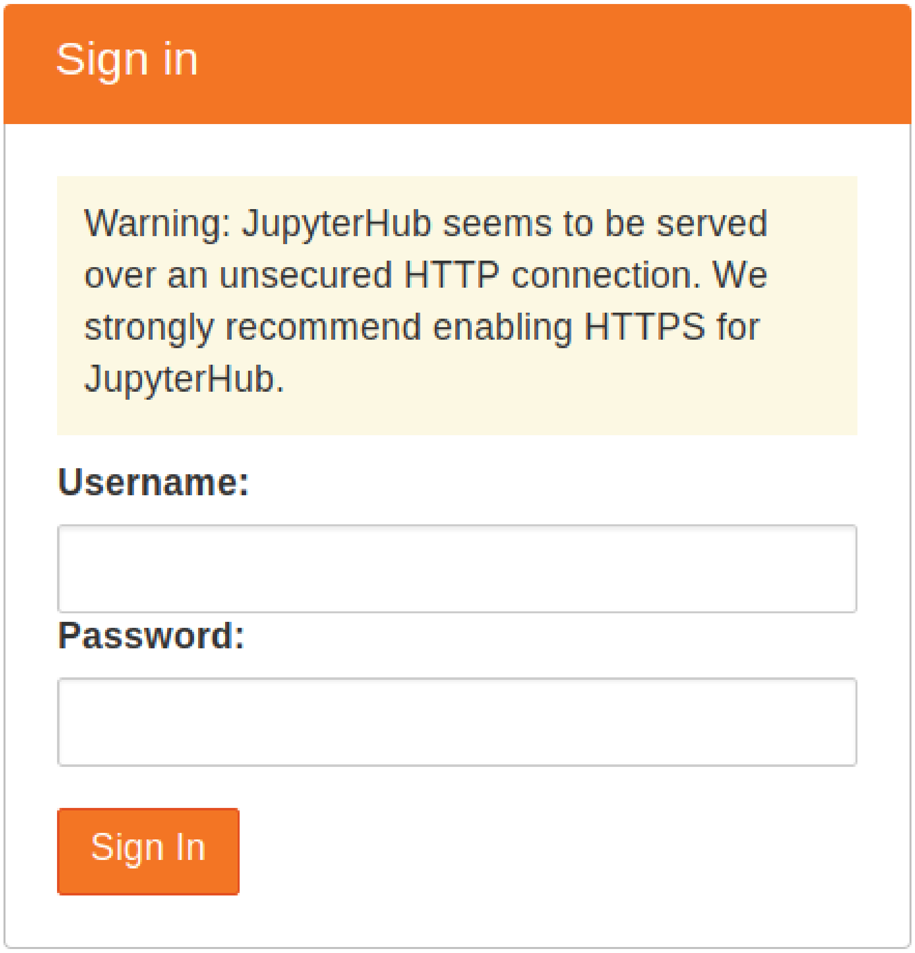
14. You'll now be shown a dashboard with all your servers and their states. The server you just launched will progress through various stages of set up, and you can see the progress here.



| | | | | | | |
|---|---|---|---|---|---|---|
| □ | tljh-quickstart | ● Active | Networking | 129.114.16.174 | M1.small | Jetstream - TACC |

15. It will take about ten minutes for your server to come up. The status will say **Active** and the progress bar will be a solid green. At this point, your JupyterHub is ready for use!

16. Copy the **IP Address** of your server, and try accessing it from a web browser. It should give you a JupyterHub login page.



Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub.

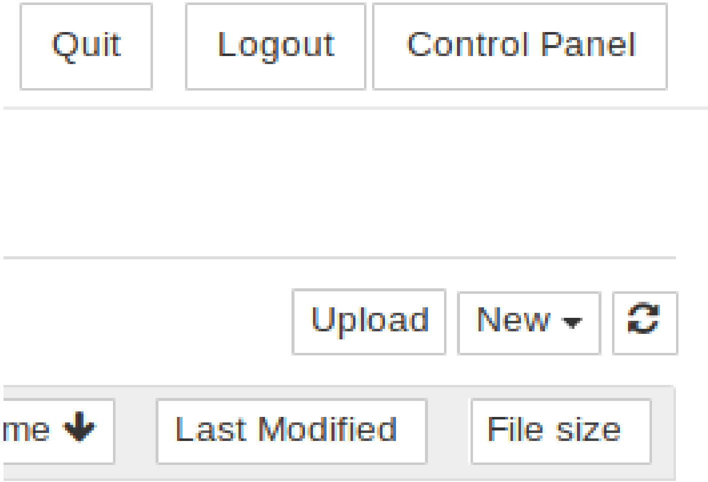
17. Login using the **admin user name** you used in step 8, and a password. Use a strong password & note it down somewhere, since this will be the password for the admin user account from now on.

18. Congratulations, you have a running working JupyterHub!

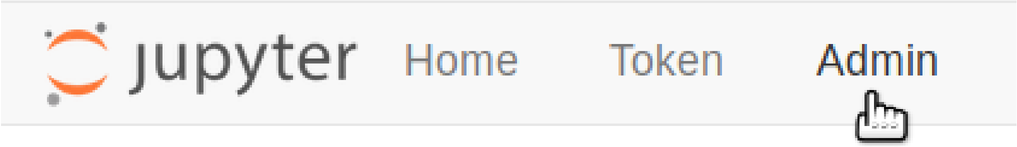
### Step 2: Adding more users

Most administration & configuration of the JupyterHub can be done from the web UI directly. Let's add a few users who can log in!

1. Open the **Control Panel** by clicking the control panel button on the top right of your JupyterHub.

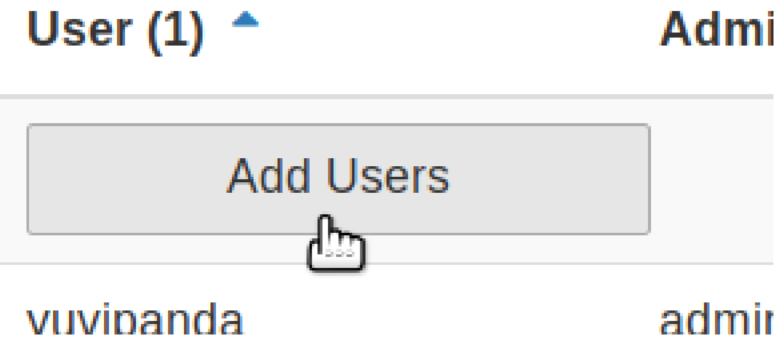


2. In the control panel, open the **Admin** link in the top left.



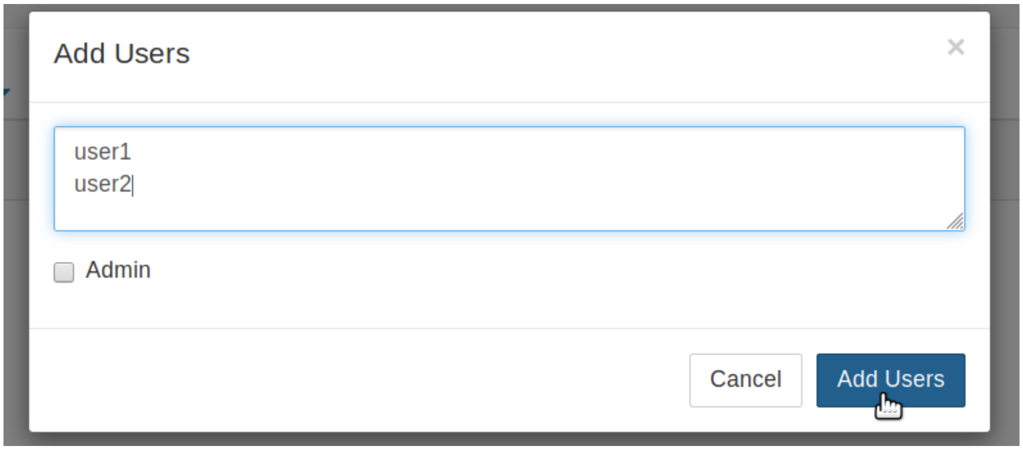
This opens up the JupyterHub admin page, where you can add / delete users, start / stop peoples' servers and see who is online.

3. Click the **Add Users** button.



A **Add Users** dialog box opens up.

4. Type the names of users you want to add to this JupyterHub in the dialog box, one per line.



Add Users

user1
user2

You can tick the **Admin** checkbox if you want to give admin rights to all these users too.

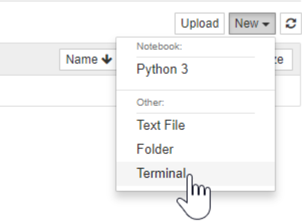
5. Click the **Add Users** button in the dialog box. Your users are now added to the JupyterHub! When they log in for the first time, they can set their password - and use it to log in again in the future.

Congratulations, you now have a multi user JupyterHub that you can add arbitrary users to!

### Step 3: Install conda / pip packages for all users

The **User Environment** is a conda environment that is shared by all users in the JupyterHub. Libraries installed in this environment are immediately available to all users. Admin users can install packages in this environment with `sudo -E`.

1. Log in as an admin user and open a Terminal in your Jupyter Notebook.



2. Install gdal from conda-forge.

```
sudo -E conda install -c conda-forge gdal
```

The `sudo -E` is very important!

3. Install `there` with `pip`

```
sudo -E pip install there
```

The packages `gdal` and `there` are now available to all users in JupyterHub. If a user already had a python notebook running, they have to restart their notebook's kernel to make the new libraries available.

See *Install conda, pip or apt packages* for more information.

## 2.1.4 Installing on Google Cloud

### Goal

By the end of this tutorial, you should have a JupyterHub with some admin users and a user environment with packages you want installed running on Google Cloud.

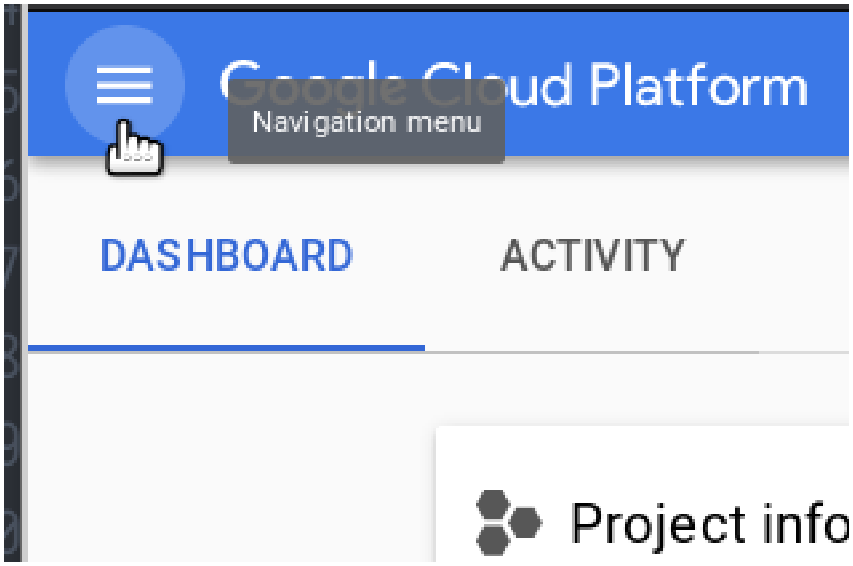
### Prerequisites

1. A Google Cloud account. You might use the free credits for trying it out!

### Step 1: Installing The Littlest JupyterHub

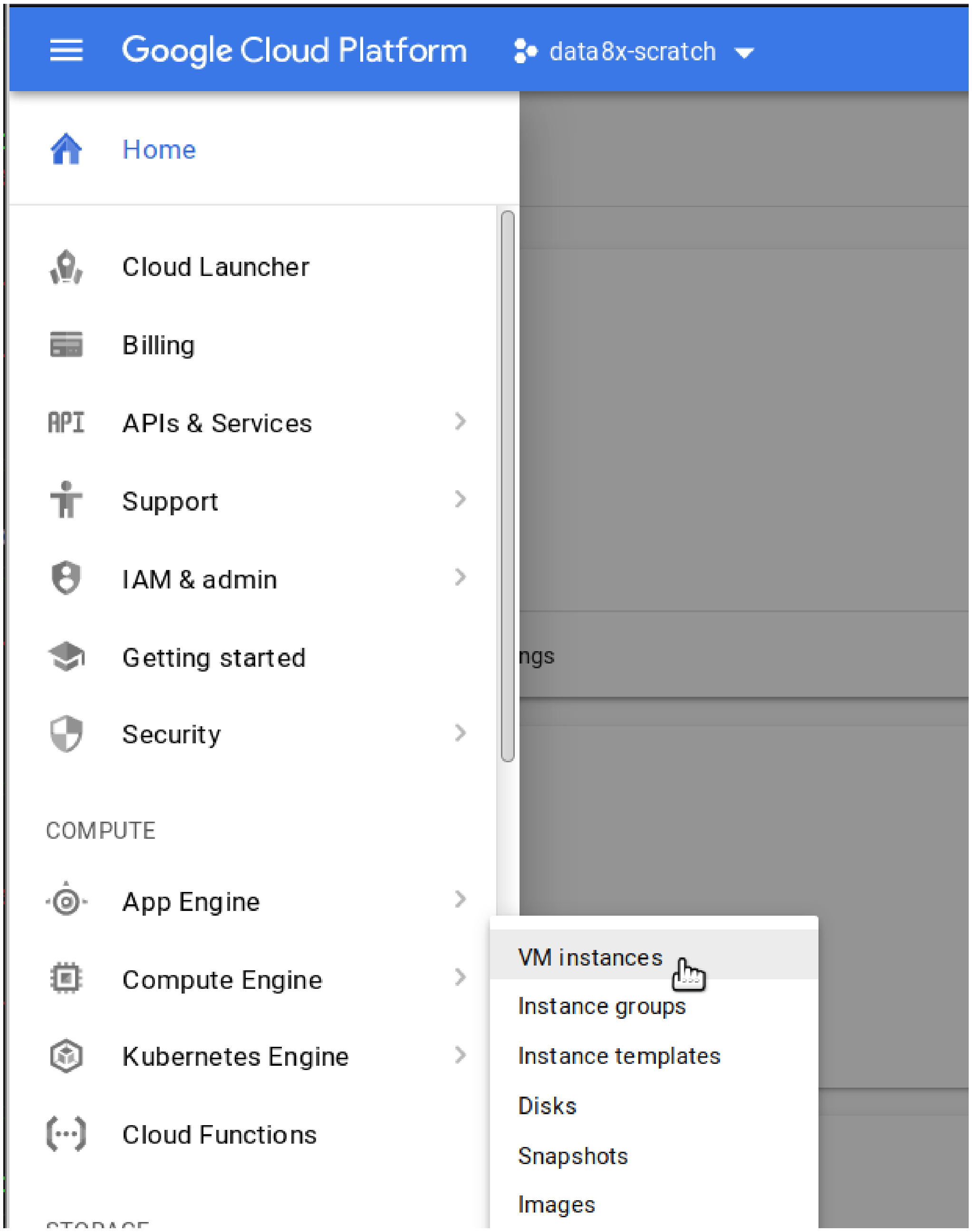
Let's create the server on which we can run JupyterHub.

1. Log in to Google Cloud Console with your Google Account.
2. Open the navigation menu by clicking the button with three lines on the top left corner of the page.

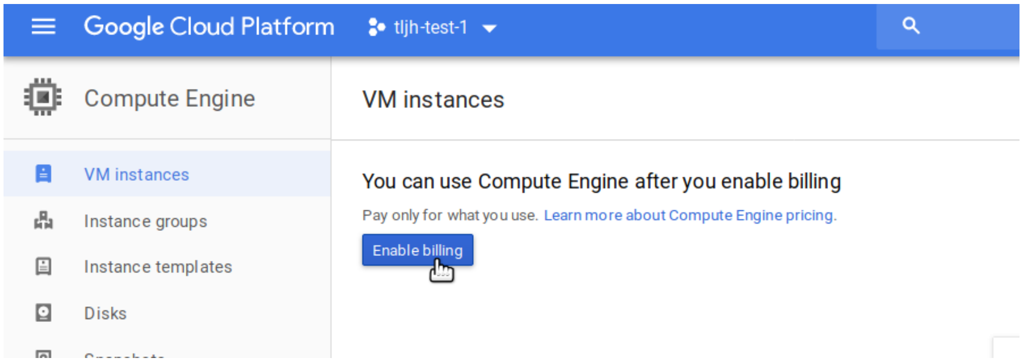


This opens a menu with all the cloud products Google Cloud offers.

3. Under **Compute Engine**, select **VM Instances**.



4. If you are using Google Cloud for the first time, you might have to enable billing. Google will present a screen asking you to enable billing to proceed. Click the **Enable Billing** button and follow any prompts that appear.

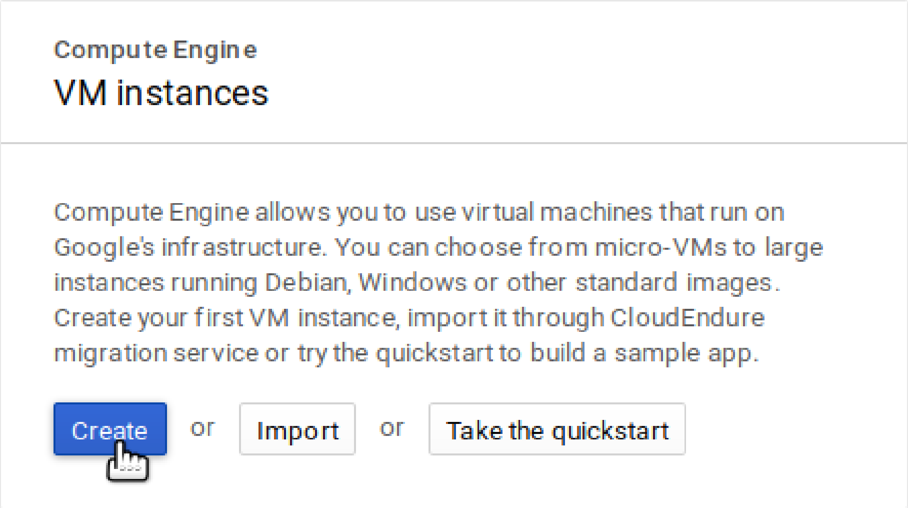


You can use Compute Engine after you enable billing

Pay only for what you use. Learn more about Compute Engine pricing.

It might take a few minutes for your account to be set up.

5. Once Compute Engine is ready, click the **Create** button to start creating the server that'll run TLJH.

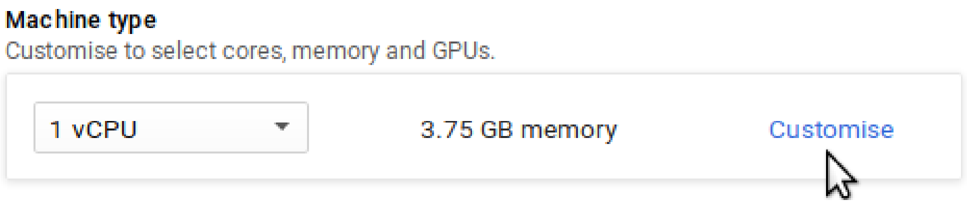


Compute Engine allows you to use virtual machines that run on Google's infrastructure. You can choose from micro-VMs to large instances running Debian, Windows or other standard images. Create your first VM instance, import it through CloudEndure migration service or try the quickstart to build a sample app.

   If you already have VMs running in your project, the screen will look different. But you can find the **Create** button still!

6. This shows you a page titled **Create an instance**. This lets you customize the kind of server you want, the resources it will have & what it'll be called.
7. Under **Name**, give it a memorable name that identifies what purpose this JupyterHub will be used for.
8. **Region** specifies the physical location where this server will be hosted. Generally, pick something close to where most of your users are. Note that it might increase the cost of your server in some cases!
9. For **Zone**, pick any of the options. Leaving the default as is is fine.
10. Under **Machine** type, select the amount of CPU / RAM / GPU you want for your server. You need at least **1.15GB** of RAM.

    You can select a preset combination in the default **basic view**.



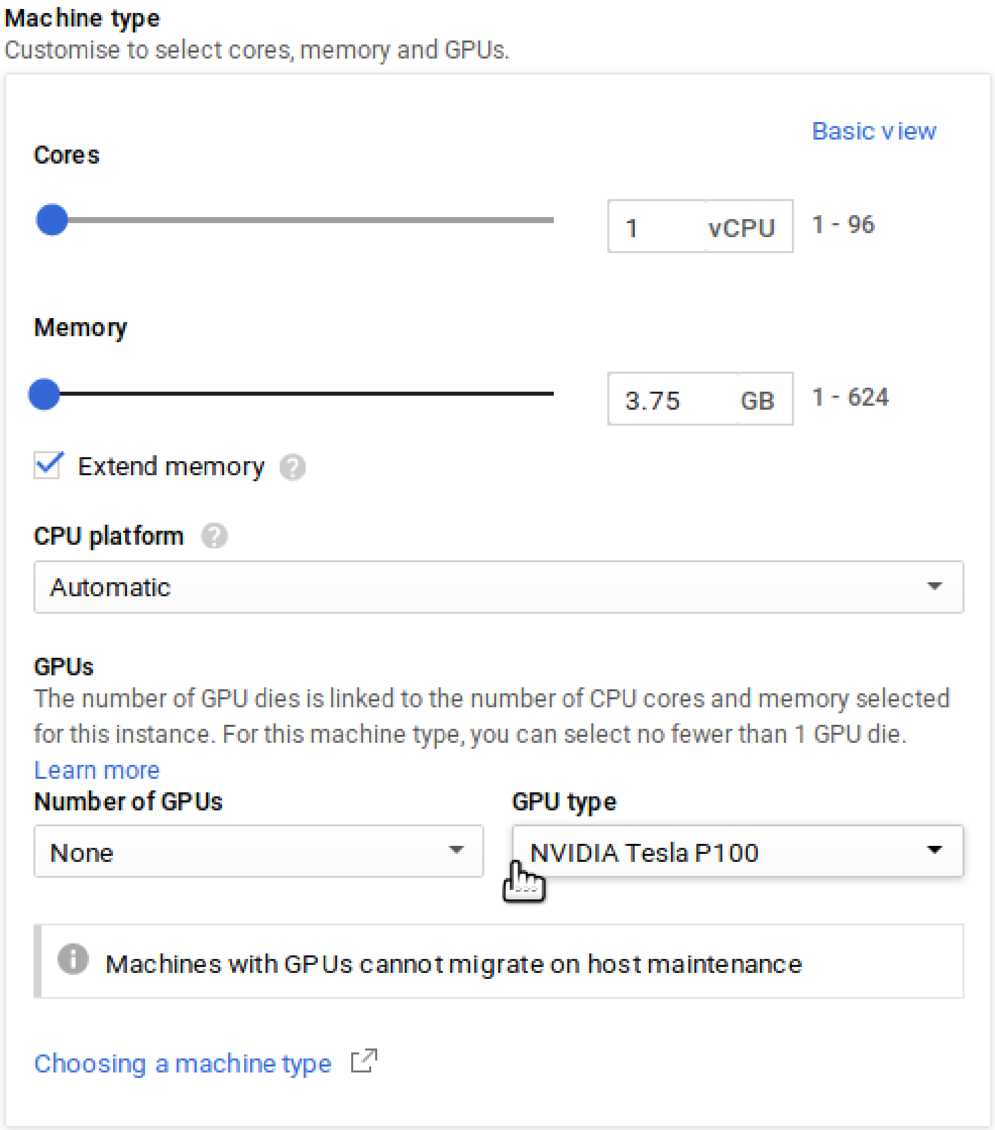
Machine type

Customise to select cores, memory and GPUs.

1 vCPU

3.75 GB memory

If you want to add **GPUs**, you should click the **Customize** button & use the **Advanced View**. You need to request a quota increase before you can use GPUs.



Machine type

Customise to select cores, memory and GPUs.

Cores

1 vCPU 1 - 96

Memory

3.75 GB 1 - 624

☑ Extend memory

CPU platform

Automatic

GPUs

The number of GPU dies is linked to the number of CPU cores and memory selected for this instance. For this machine type, you can select no fewer than 1 GPU die.

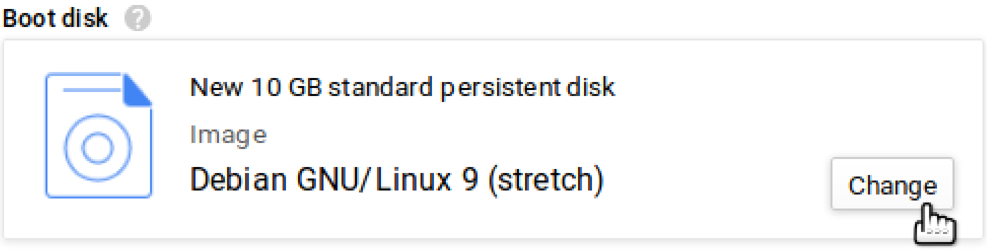
Number of GPUs: None

GPU type: NVIDIA Tesla P100

Machines with GPUs cannot migrate on host maintenance

Check out our guide on How To *Estimate Memory / CPU / Disk needed* to help pick how much Memory / CPU your server needs.

11. Under **Boot Disk**, click the **Change** button. This lets us change the operating system and the size of your disk.



Boot disk

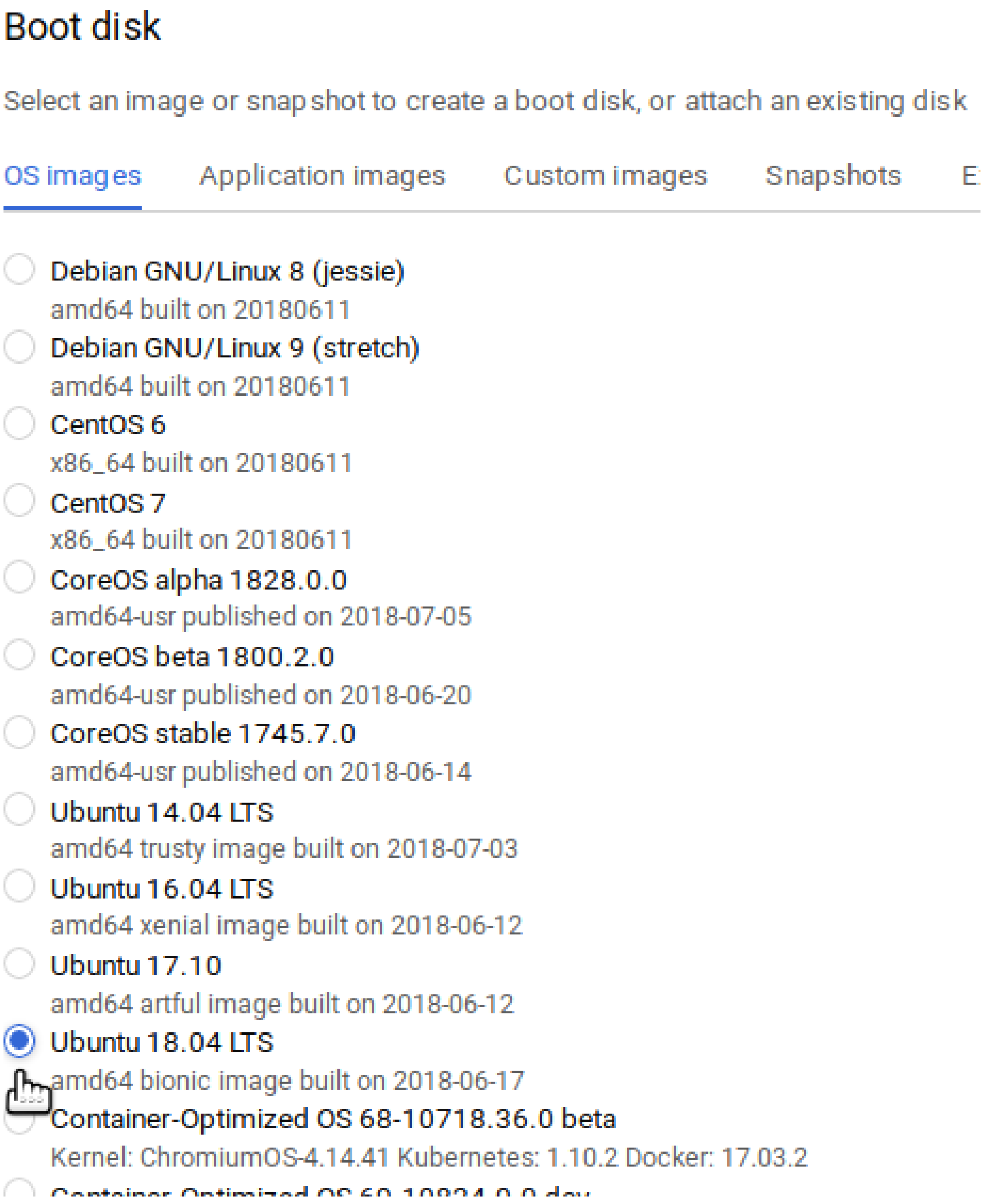
New 10 GB standard persistent disk

Image

Debian GNU/Linux 9 (stretch)

This should open a **Boot disk** popup.

12. Select **Ubuntu 18.04 LTS** from the list of operating system images.



Boot disk

Select an image or snapshot to create a boot disk, or attach an existing disk

○ Debian GNU/Linux 8 (jessie)
amd64 built on 20180611

○ Debian GNU/Linux 9 (stretch)
amd64 built on 20180611

○ CentOS 6
x86_64 built on 20180611

○ CentOS 7
x86_64 built on 20180611

○ CoreOS alpha 1828.0.0
amd64-usr published on 2018-07-05

○ CoreOS beta 1800.2.0
amd64-usr published on 2018-06-20

○ CoreOS stable 1745.7.0
amd64-usr published on 2018-06-14

○ Ubuntu 14.04 LTS
amd64 trusty image built on 2018-07-03

○ Ubuntu 16.04 LTS
amd64 xenial image built on 2018-06-12

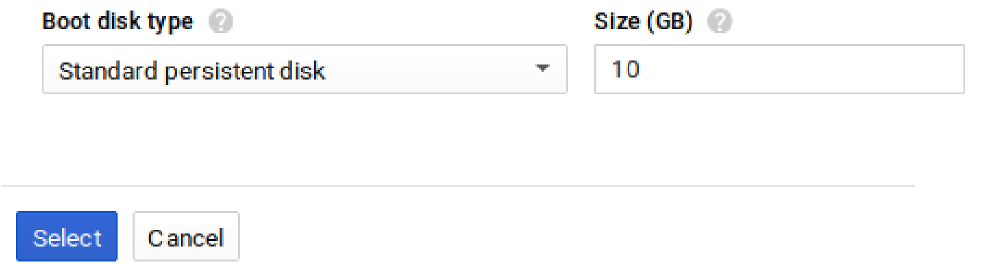
○ Ubuntu 17.10
amd64 artful image built on 2018-06-12

● Ubuntu 18.04 LTS
amd64 bionic image built on 2018-06-17

○ Container-Optimized OS 68-10718.36.0 beta
Kernel: ChromiumOS-4.14.41 Kubernetes: 1.10.2 Docker: 17.03.2

○ Container-Optimized OS 69-10824.0.0 dev

13. You can also change the **type** and **size** of your disk at the bottom of this popup.



Boot disk type: Standard persistent disk

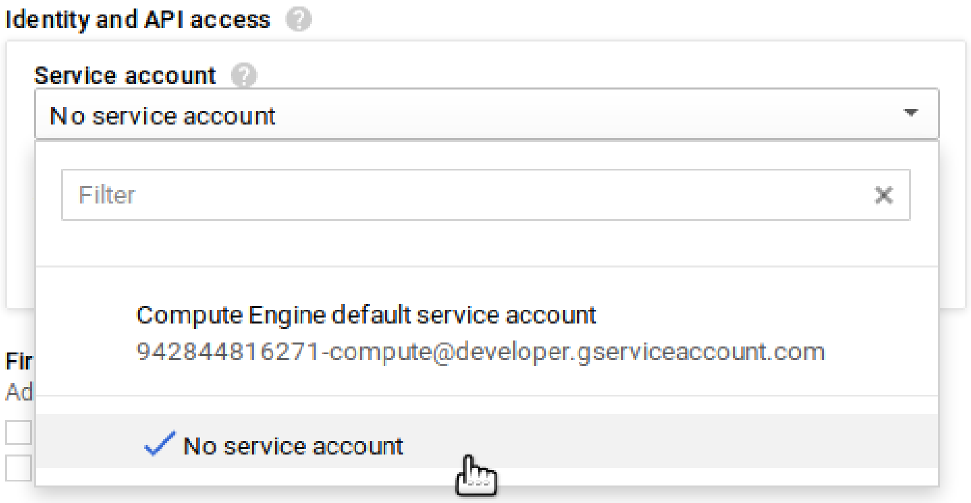
Size (GB): 10

**Standard persistent disk** type gives you a slower but cheaper disk, similar to a hard drive. **SSD persistent disk** gives you a faster but more expensive disk, similar to an SSD.

Check out our guide on How To *Estimate Memory / CPU / Disk needed* to help pick how much Disk space your server needs.

14. Click the **Select** button to dismiss the Boot disk popup and go back to the Create an instance screen.

15. Under **Identity and API access**, select **No service account** for the **Service account** field. This prevents your JupyterHub users from automatically accessing other cloud services, increasing security.

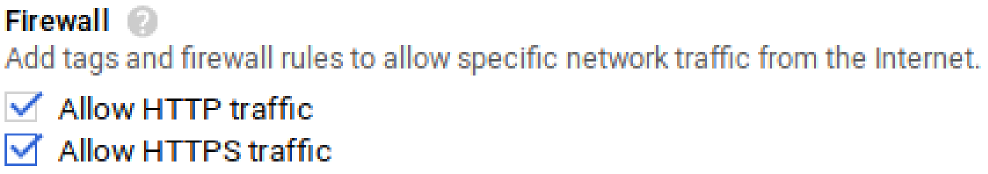


Identity and API access

Service account: No service account

Filter
Compute Engine default service account
942844816271-compute@developer.gserviceaccount.com
✓ No service account
Fir
Ad

16. Under **Firewall**, check both **Allow HTTP Traffic** and **Allow HTTPS Traffic** checkboxes.



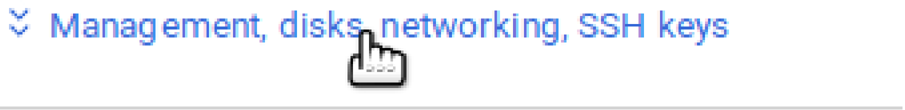
Firewall

Add tags and firewall rules to allow specific network traffic from the Internet.

☑ Allow HTTP traffic

☑ Allow HTTPS traffic

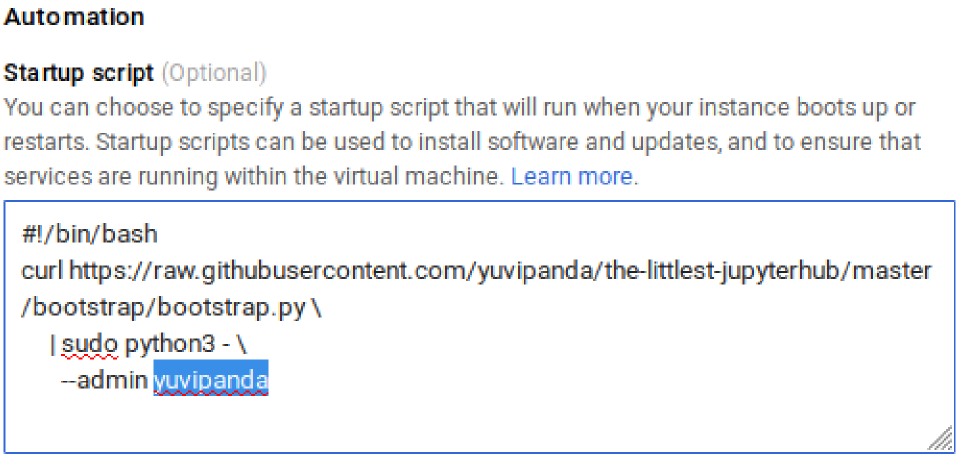
17. Click the **Management, disks, networking, SSH keys** link to expand more options.



This displays a lot of advanced options, but we'll be only using one of them.

18. Copy the text below, and paste it into the **Startup script** text box. Replace `<admin-user-name>` with the name of the first **admin user** for this JupyterHub. This admin user can log in after the JupyterHub is set up, and can configure it to their needs. **Remember to add your username**!

```
#!/bin/bash
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-
↪jupyterhub/master/bootstrap/bootstrap.py \
  | sudo python3 - \
    --admin <admin-user-name>
```



**Automation**

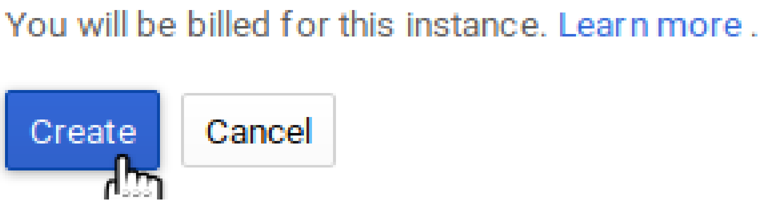
**Startup script** (Optional)

You can choose to specify a startup script that will run when your instance boots up or restarts. Startup scripts can be used to install software and updates, and to ensure that services are running within the virtual machine. Learn more.

```
#!/bin/bash
curl https://raw.githubusercontent.com/yuvipanda/the-littlest-jupyterhub/master
/bootstrap/bootstrap.py \
  | sudo python3 - \
   --admin yuvipanda
```

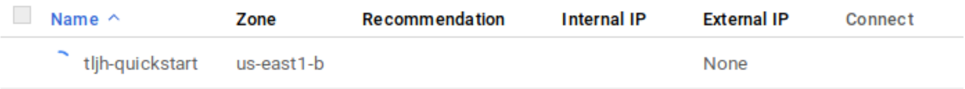
**Note:** See *What does the installer do?* if you want to understand exactly what the installer is doing. *Customizing the Installer* documents other options that can be passed to the installer.

19. Click the **Create** button at the bottom to start your server!



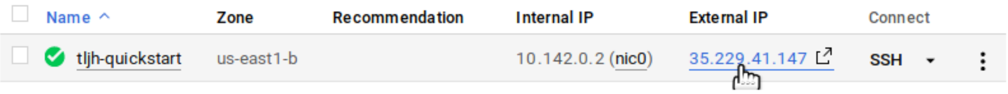
You will be billed for this instance. Learn more .

20. We'll be sent to the **VM instances** page, where we can see that our server is being created.



| □ | Name ^ | Zone | Recommendation | Internal IP | External IP | Connect |
|---|---|---|---|---|---|---|
| | tljh-quickstart | us-east1-b | | | None | |

21. In a few seconds your server will be created, and you can see the **External IP** used to access it.



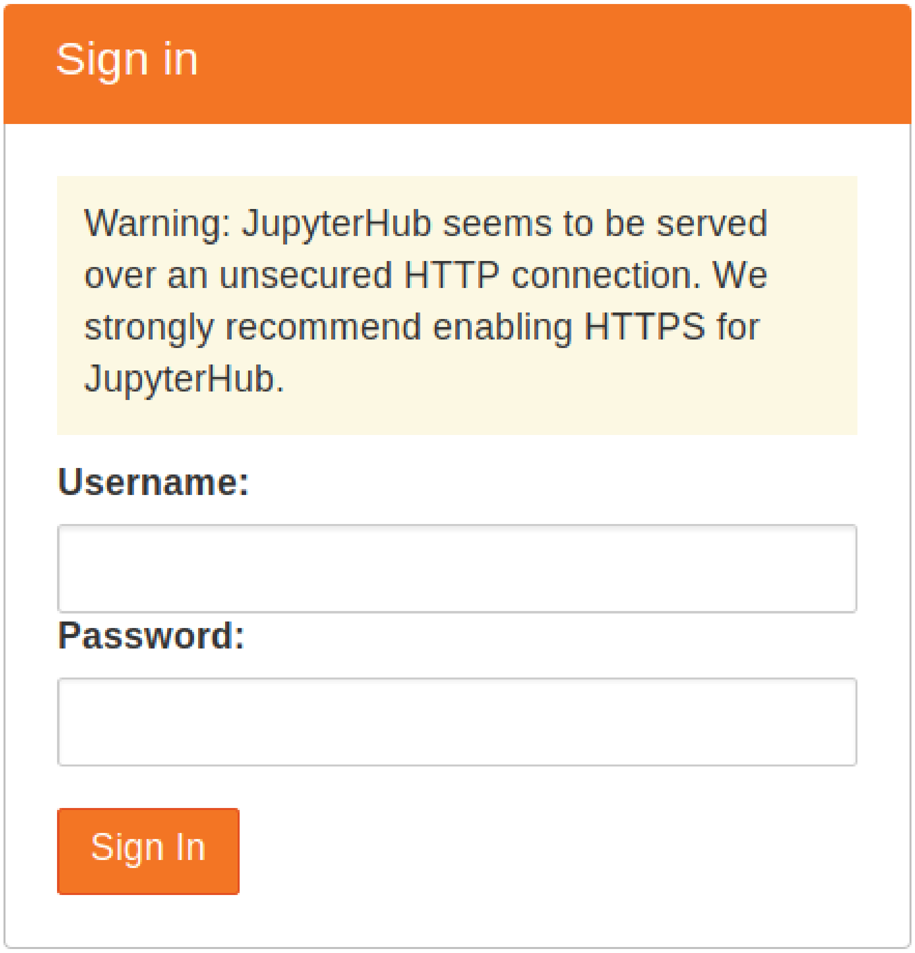
| □ | Name ^ | Zone | Recommendation | Internal IP | External IP | Connect |
|---|---|---|---|---|---|---|
| □ | ✓ tljh-quickstart | us-east1-b | | 10.142.0.2 (nic0) | 35.229.41.147 | SSH ▾ |

22. The Littlest JupyterHub is now installing in the background on your new server. It takes around 5-10 minutes for this installation to complete.

23. Check if the installation is complete by **copying** the **External IP** of your server, and trying to access it with a browser. Do **not click** on the IP - this will open the link with HTTPS, and will not work.

    Accessing the JupyterHub will also fail until the installation is complete, so be patient.

24. When the installation is complete, it should give you a JupyterHub login page.



Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub.

Username:

Password:

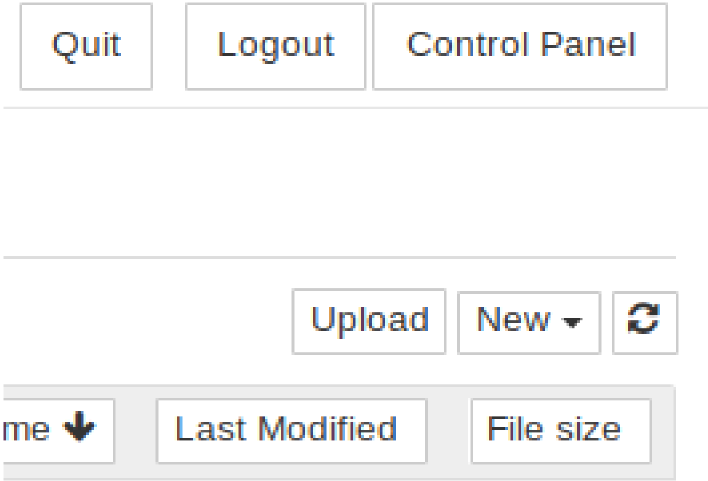
25. Login using the **admin user name** you used in step 6, and a password. Use a strong password & note it down somewhere, since this will be the password for the admin user account from now on.

26. Congratulations, you have a running working JupyterHub!

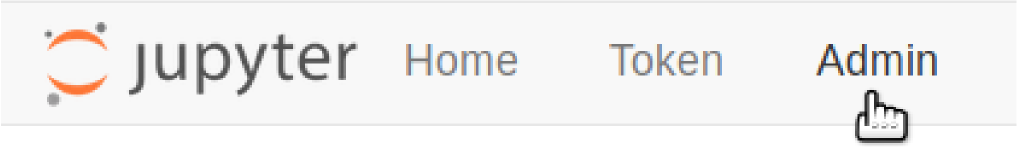
### Step 2: Adding more users

Most administration & configuration of the JupyterHub can be done from the web UI directly. Let's add a few users who can log in!

1. Open the **Control Panel** by clicking the control panel button on the top right of your JupyterHub.

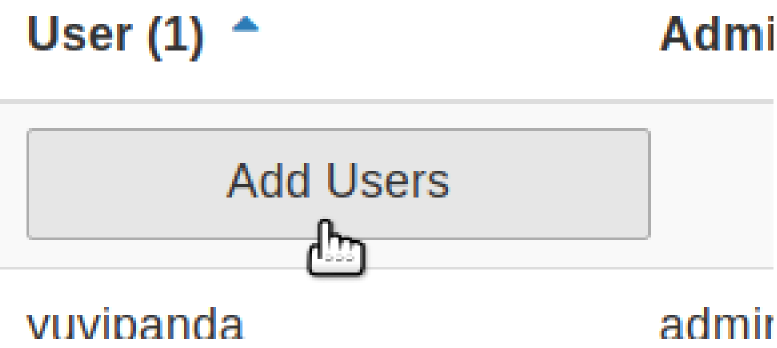


2. In the control panel, open the **Admin** link in the top left.



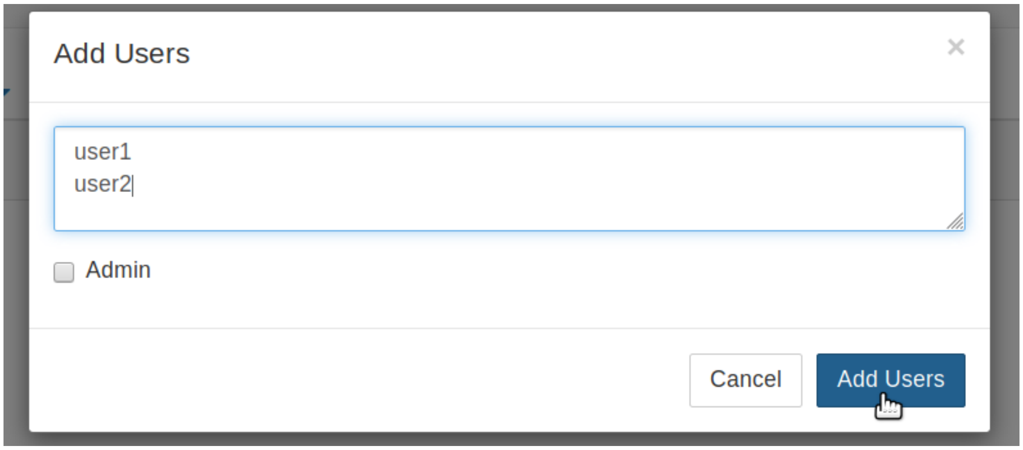
This opens up the JupyterHub admin page, where you can add / delete users, start / stop peoples' servers and see who is online.

3. Click the **Add Users** button.



A **Add Users** dialog box opens up.

4. Type the names of users you want to add to this JupyterHub in the dialog box, one per line.



You can tick the **Admin** checkbox if you want to give admin rights to all these users too.

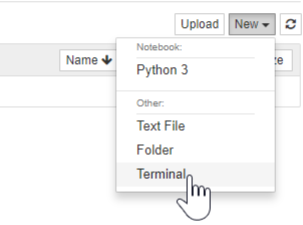
5. Click the **Add Users** button in the dialog box. Your users are now added to the JupyterHub! When they log in for the first time, they can set their password - and use it to log in again in the future.

Congratulations, you now have a multi user JupyterHub that you can add arbitrary users to!

### Step 3: Install conda / pip packages for all users

The **User Environment** is a conda environment that is shared by all users in the JupyterHub. Libraries installed in this environment are immediately available to all users. Admin users can install packages in this environment with `sudo -E`.

1. Log in as an admin user and open a Terminal in your Jupyter Notebook.



2. Install gdal from conda-forge.

```
sudo -E conda install -c conda-forge gdal
```

The `sudo -E` is very important!

3. Install `there` with `pip`

```
sudo -E pip install there
```

The packages `gdal` and `there` are now available to all users in JupyterHub. If a user already had a python notebook running, they have to restart their notebook's kernel to make the new libraries available.

See *Install conda, pip or apt packages* for more information.

### 2.1.5 Installing on Amazon Web Services

#### Goal

To have a JupyterHub with admin users and a user environment with conda / pip packages.

#### Prerequisites

1. An Amazon Web Services account.

   The AWS free tier is fully capable of running a minimal littlest Jupyterhub for testing purposes.

   If asked to choose a default region, choose the one closest to the majority of your users.

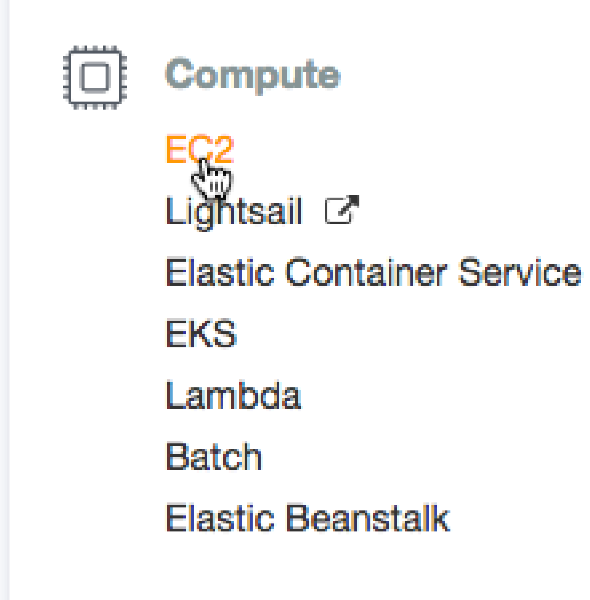
#### Step 1: Installing The Littlest JupyterHub

Let's create the server on which we can run JupyterHub.

1. Go to Amazon Web Services and click the gold button 'Sign In to the Console' in the upper right. Log in with your Amazon Web Services account.

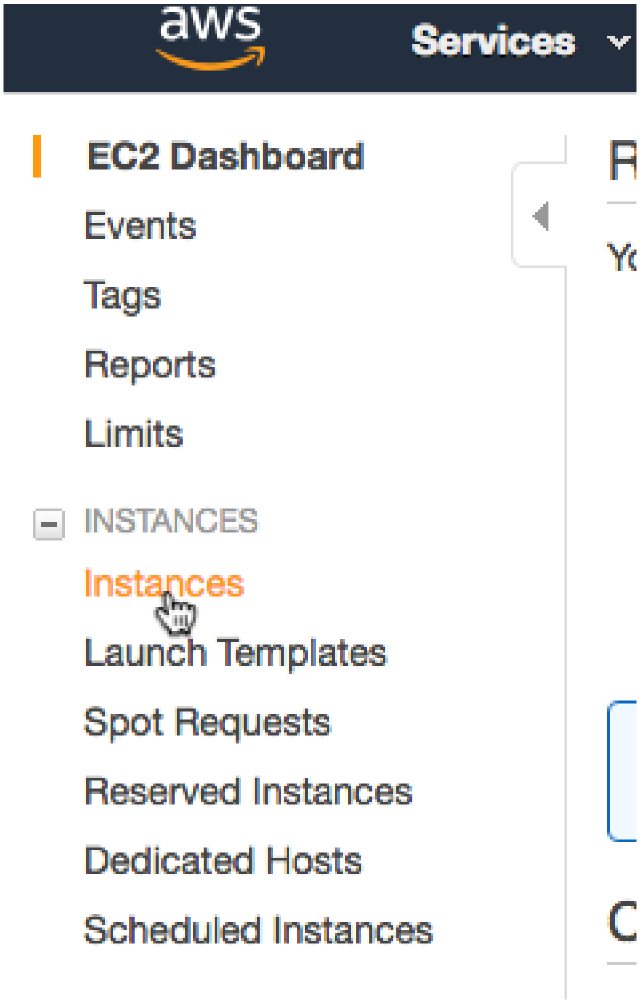
   If you need to adjust your region from your default, there is a drop-down menu between your name and the **Support** menu on the far right of the dark navigation bar across the top of the window. Adjust the region to match the closest one to the majority of your users.

2. On the screen listing all the available services, pick **EC2** under **Compute** on the left side at the top of the first column.

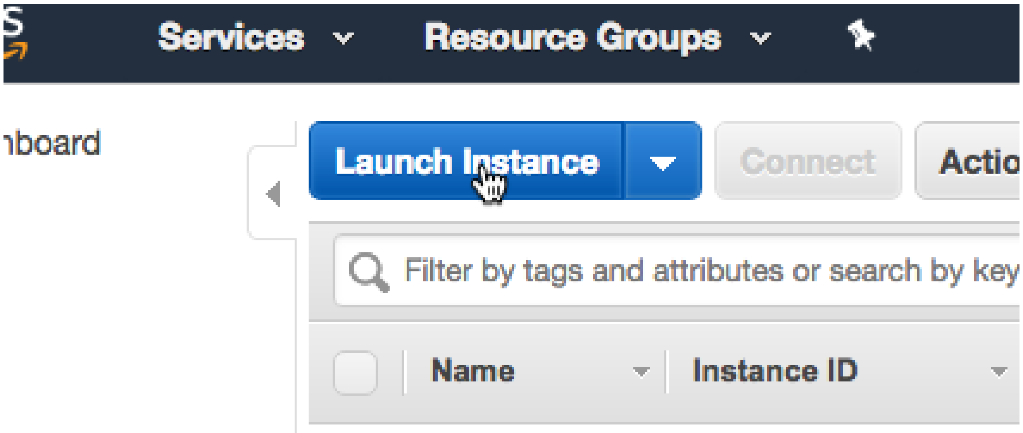
   

   This will take you to the **EC2 Management Console**.

3. From the navigation menu listing on the far left side of the **EC2 Management Console**, choose **Instances** under the light gray **INSTANCES** sub-heading.



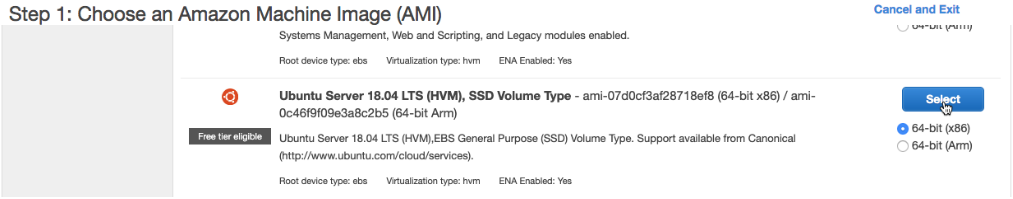
4. In the main window of the **EC2 Management Console**, towards the top left, click on the bright blue **Launch Instance** button.



This will start the ‘launch instance wizard’ process. This lets you customize the kind of server you want, the resources it will have and its name.

5. On the page **Step 1: Choose an Amazon Machine Image (AMI)** you are going to pick the base image your remote server will have. The view will default to the ‘Quick-start’ tab selected and just a few down the page, select **Ubuntu Server 18.04 LTS (HVM), SSD Volume Type -**

**ami-XXXXXXXXXXXXXXXXX**, leaving *64-bit (x86)* toggled.



Step 1: Choose an Amazon Machine Image (AMI)

○ 64-bit (Arm)

Systems Management, Web and Scripting, and Legacy modules enabled.

Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

**Ubuntu Server 18.04 LTS (HVM), SSD Volume Type** - ami-07d0cf3af28718ef8 (64-bit x86) / ami-0c46f9f09e3a8c2b5 (64-bit Arm)

● 64-bit (x86)
○ 64-bit (Arm)

Ubuntu Server 18.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).

Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

The *ami* alpha-numeric at the end references the specific Amazon machine image, ignore this as Amazon updates them routinely. The **Ubuntu Server 18.04 LTS (HVM)** is the important part.

6. After selecting the AMI, you'll be at **Step 2: Choose an Instance Type**.

   There will be a long listing of the types and numbers of CPUs that Amazon offers. Select the one you want and then select the button *Next: Configure Instance Details* in the lower right corner.

   Check out our guide on How To *Estimate Memory / CPU / Disk needed* to help pick how much Memory / CPU your server needs. You need to have at least **1.15GB** of RAM.

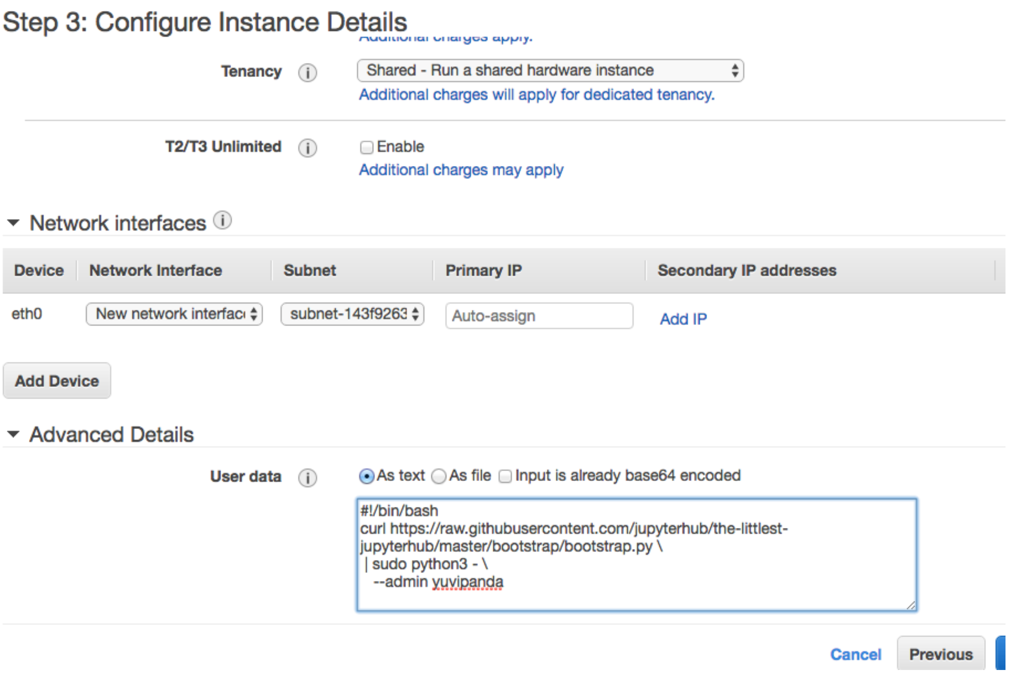
   You may wish to consult the listing here because it shows cost per hour. The **On Demand** price is the pertinent cost.

   (For reference, a minimal hub that worked for developing this tutorial used a **t2.micro** tier, which is free for Amazon users the first year they sign up. Two users were able to concurrently utilize this development hub without issue.)

   `GPU graphics` and `GPU compute` products are also available around half way down the page

7. Under **Step 3: Configure Instance Details**, scroll to the bottom of the page and toggle the arrow next to **Advanced Details**. Scroll down to 'User data'. Copy the text below, and paste it into the **User data** text box. Replace `<admin-user-name>` with the name of the first **admin user** for this JupyterHub. This admin user can log in after the JupyterHub is set up, and configure it. **Remember to add your username**!

```
#!/bin/bash
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-
↪jupyterhub/master/bootstrap/bootstrap.py \
  | sudo python3 - \
    --admin <admin-user-name>
```



Step 3: Configure Instance Details

Additional charges apply.

Tenancy: Shared - Run a shared hardware instance

Additional charges will apply for dedicated tenancy.

T2/T3 Unlimited: □ Enable

Additional charges may apply

Network interfaces

| Device | Network Interface | Subnet | Primary IP | Secondary IP addresses |
|---|---|---|---|---|
| eth0 | New network interfac | subnet-143f9263 | Auto-assign | Add IP |

Add Device

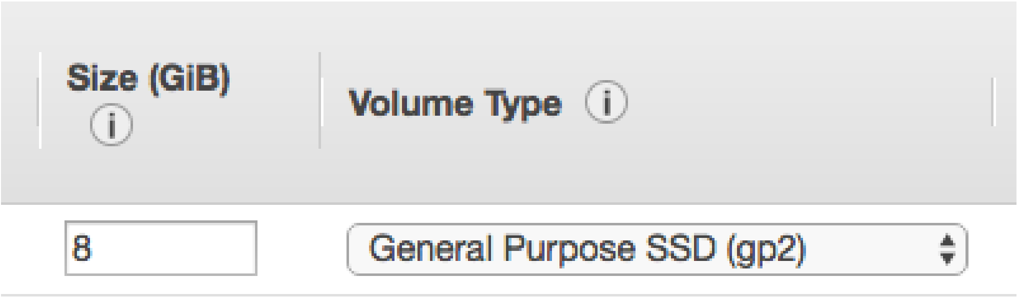
Advanced Details

User data: ● As text ○ As file □ Input is already base64 encoded

```
#!/bin/bash
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-
jupyterhub/master/bootstrap/bootstrap.py \
 | sudo python3 - \
   --admin yuvipanda
```

**Note:** See *What does the installer do?* for a detailed description and *Customizing the Installer* for other options that can be used.

8. Under **Step 4: Add Storage**, you can change the **size** and **type of your disk by adjusting the value in **Size (GiB)** and selecting **Volume Type**.



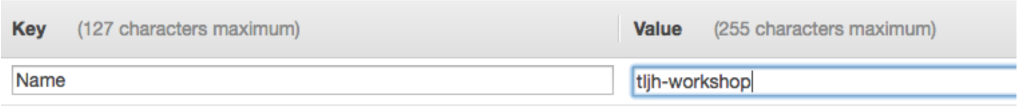
| Size (GiB) | Volume Type |
|---|---|
| 8 | General Purpose SSD (gp2) |

Check out *Estimate Memory / CPU / Disk needed* to help pick how much Disk space your server needs.

Hover over the encircled *i* next to **Volume Type** for an explanation of each. Leaving the default as is is fine. *General Purpose SSD (gp2)* is recommended for most workloads. With *Provisioned IOPS SSD (io1)* being the highest-performance SSD volume. Magnetic (standard) is a previous generation volume and not suited for a hub for multi-users.

When finished, click **Next: Add Tags** in the bottom right corner.

9. Under **Step 5: Add Tags**, click **Add Tag** and enter **Name** under the **Key** field. In the **Value** field in the **Name** row, give your new server a memorable name that identifies what purpose this JupyterHub will be used for.



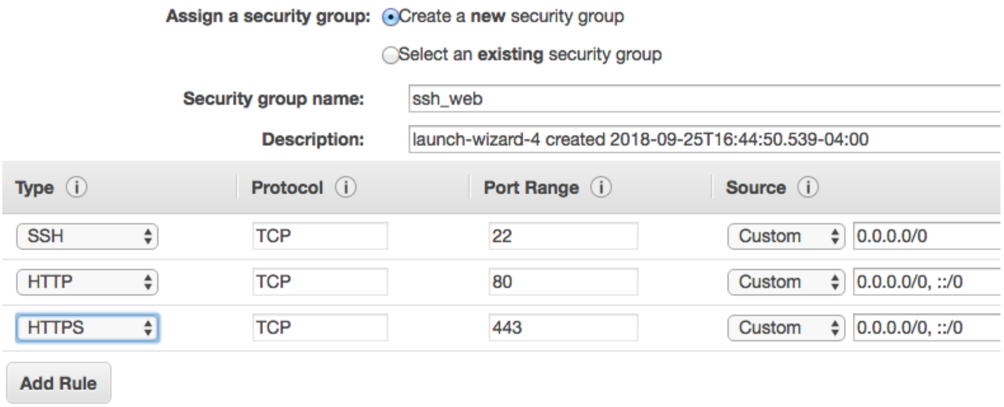
| Key (127 characters maximum) | Value (255 characters maximum) |
|---|---|
| Name | tljh-workshop |

10. Under **Step 6: Configure Security Group**, you'll set the firewall rules that control the traffic for your instance. Specifically you'll want to add rules to allow both **HTTP Traffic** and **HTTPS Traffic**. For advanced troubleshooting, it will be helpful to set rules so you can use SSH to connect (port 22).

    If you have never used your Amazon account before, you'll have to select **Create a new security group**. You should give it a disitnguishing name under **Security group name** such as *ssh_web* for future reference. If you have, one from before you can select it and adjust it to have the rules you need, if you prefer.

    The rules will default to include *SSH*. Leave that there, and then click on the **Add Rule** button. Under **Type** for the new rule, change the field to **HTTP**. The other boxes will get filled in appropritely. Again, click on the **Add Rule** button. This time under **Type** for the new rule, change the field to **HTTPS**.

    The warning is there to remind you this opens things up to some degree but this is necessary in order to let your users connect. However, this warning is a good reminder that you should monitor your server to insure it is available for users who may need it.



Assign a security group: ● Create a **new** security group
○ Select an **existing** security group

Security group name: ssh_web

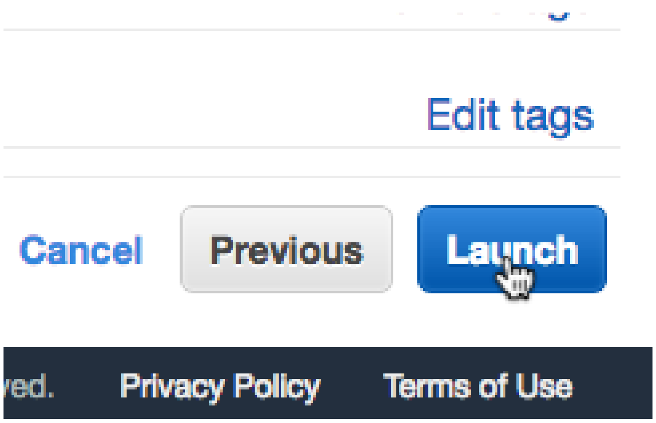
Description: launch-wizard-4 created 2018-09-25T16:44:50.539-04:00

| Type | Protocol | Port Range | Source | |
|---|---|---|---|---|
| SSH | TCP | 22 | Custom | 0.0.0.0/0 |
| HTTP | TCP | 80 | Custom | 0.0.0.0/0, ::/0 |
| HTTPS | TCP | 443 | Custom | 0.0.0.0/0, ::/0 |

11. When the security rules are set, click on the blue button in the bottom right **Review and Launch**. This will give you a chance to review things because very soon you'll be launching and start paying for any resources you use.

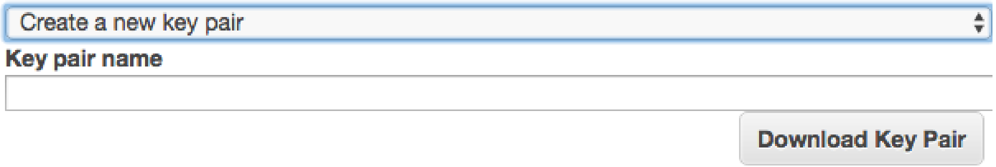
    Note that you'll see two HTTP listings and two HTTPS listings under **Security Groups** even though you only made one for each. This is normal & necessary to match both IPv4 & IPv6 types of IP addresses.

    When you are happy, press the blue **Launch** button in the bottom right corner to nearly conclude your journey through the instance launch wizard.



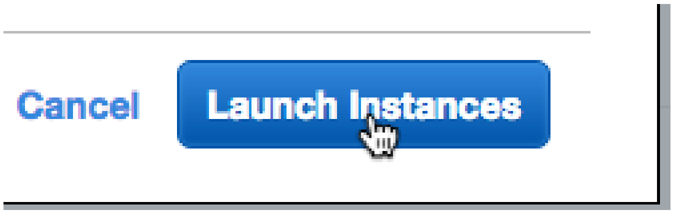
12. In the dialog box that pops up as the last step before launching is triggered, you need to choose what to do about an identifying key pair and acknowledge your choice in order to proceed. If you already have a key pair you can select to associate it with this instance, otherwise you need to **Create a new key pair**. Choosing to *Proceed without a key pair* is not recommended as you'll have no way to access your server via SSH if anything goes wrong with the Jupyterhub and have no way to recover files via download.

    Download and keep the key pair file unless you are associating one you already have.

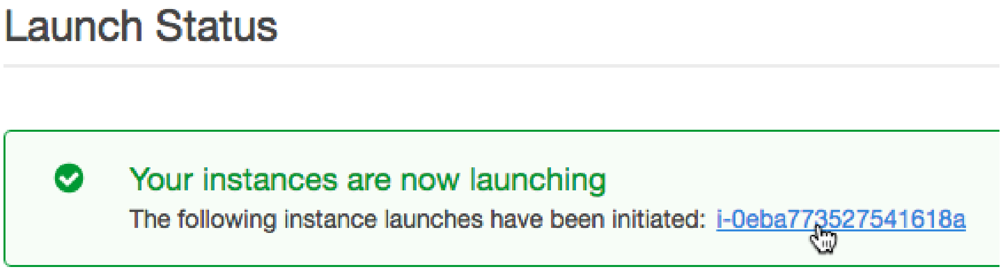
    

    Create a new key pair

13. With the key pair associated, click the **Launch instances** button to start creating the server that'll run TLJH.

14. Following the launch initiation, you'll be taken to a **Launch Status** notification screen. You can see more information about the details if you click on the alphanumeric link to the launching instance following the text, "*The following instance launches have been initiated:*".

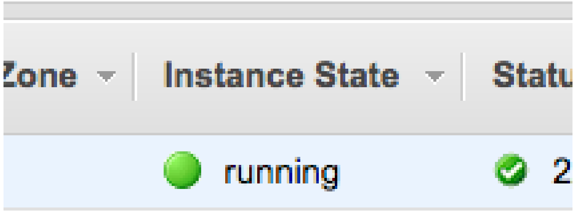
    Launch Status

    Your instances are now launching

    The following instance launches have been initiated: i-0eba773527541618a

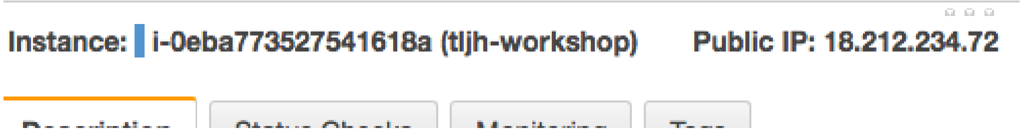
15. That link will take you back to the **EC2 Management Console** with settings that will limit the view in the console to just that instance. (Delete the filter in the search bar if you want to see

any other instances you may have.) At first the server will be starting up, and then when the **Instance state** is green the server is running.



If you already have instances running in your account, the screen will look different if you disable that filter. But you want to pay attention to the row with the name of the server you made.

16. In a few seconds your server will be created, and you can see the **Public IP** used to access it in the panel at the bottom of the console. If it isn't displayed, click on the row for that instance in the console. It will look like a pattern similar to **12.30.230.127**.

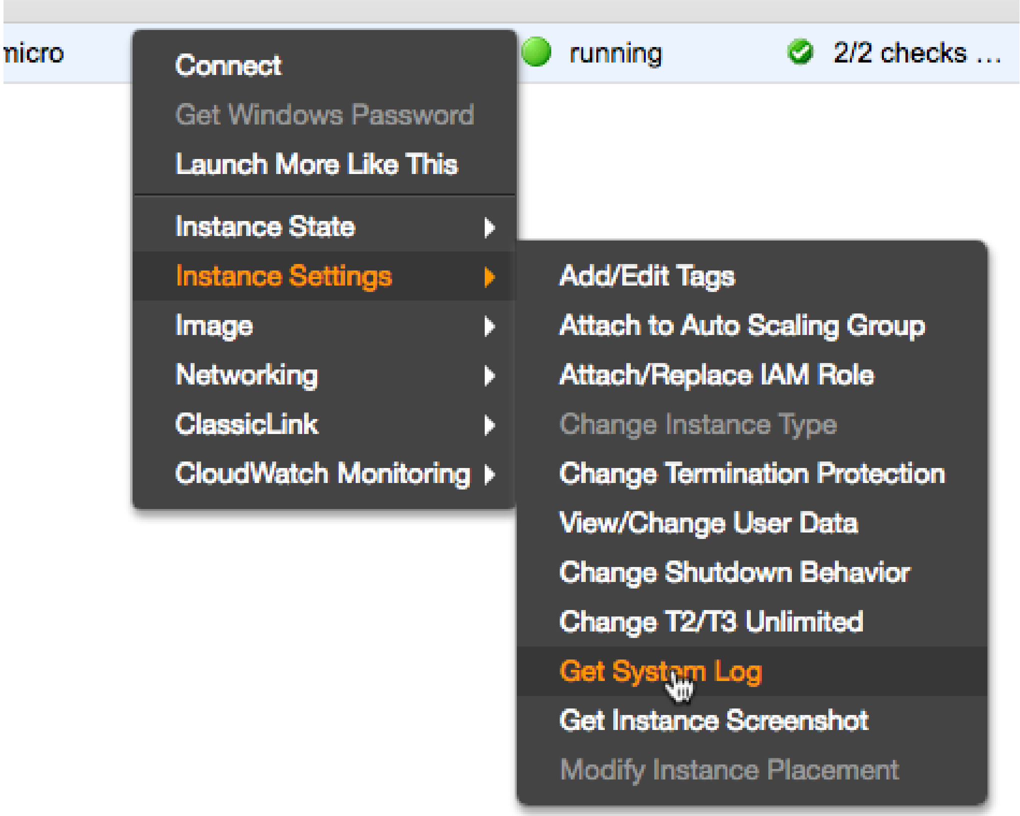


Instance: i-0eba773527541618a (tljh-workshop)    Public IP: 18.212.234.72

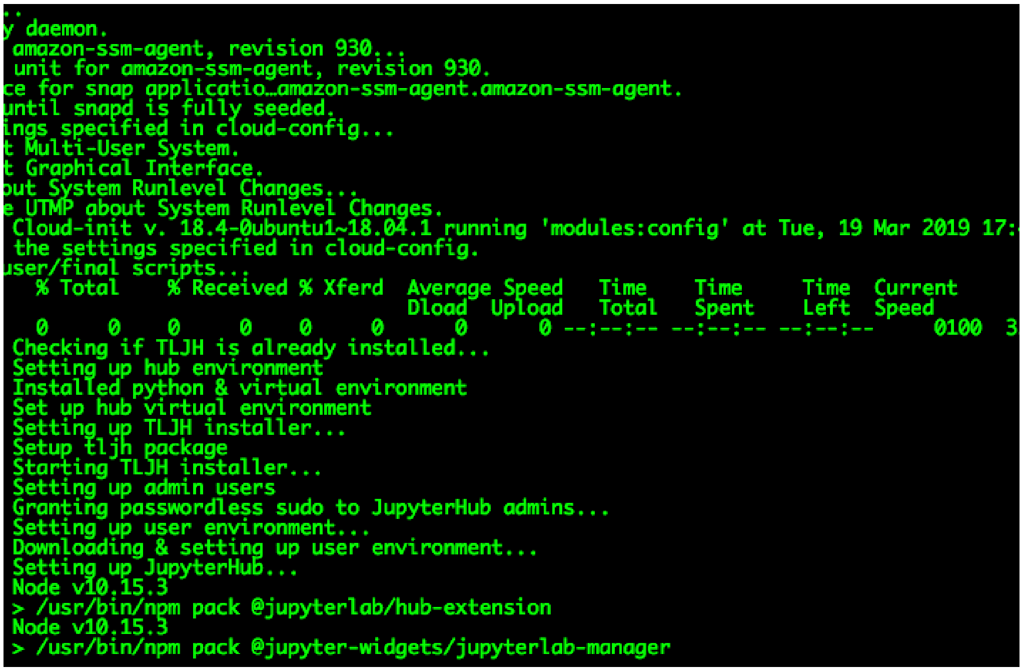
17. The Littlest JupyterHub is now installing in the background on your new server. It takes around 10 minutes for this installation to complete.

18. Check if the installation is complete by copying the **Public IP** of your server, and trying to access it from within a browser. If it has been 10 minutes, paste the public IP into the URL bar of your browser and hit return to try to connect.

    Accessing the JupyterHub will fail until the installation is complete, so be patient. The next step below this one shows the login window you are expecting to see when trying the URL and things work. While waiting until the appropriate time to try, another way to check if things are churning away, is to open the **System Log**. To do this, go to the **EC2 Management Console** & highlight the instance by clicking on that row and then right-click **Instance Settings > Get System Log**.

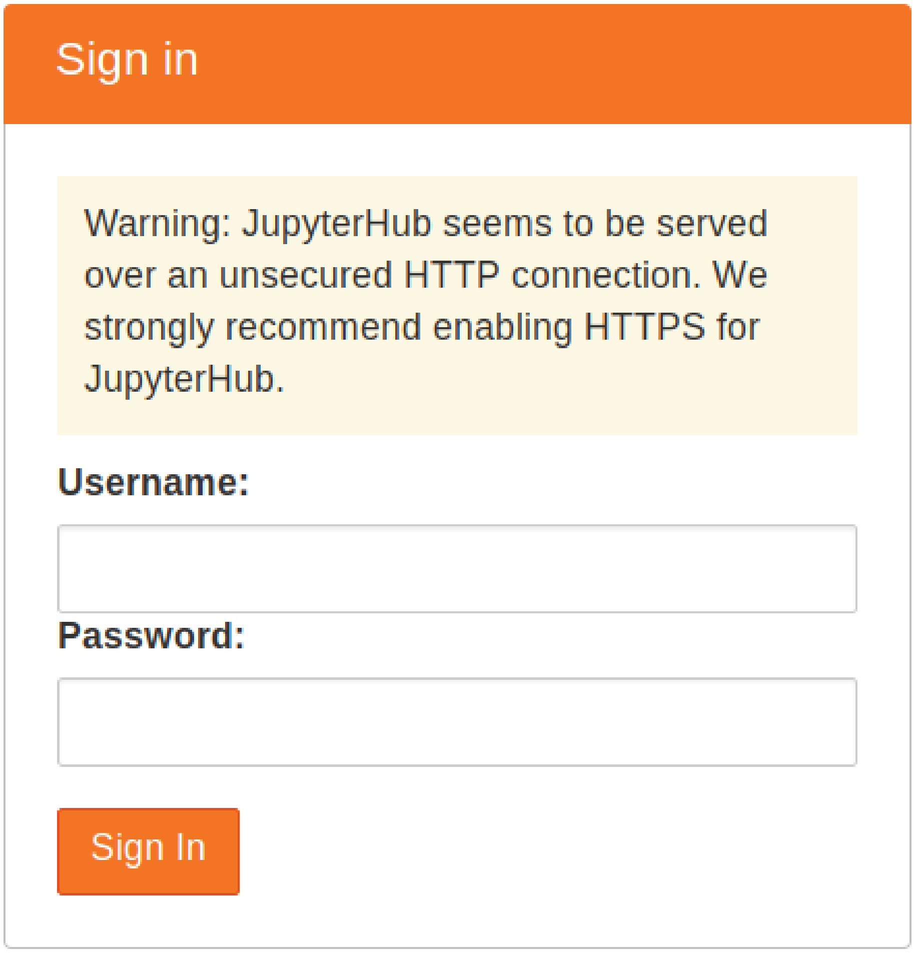


19. When the Jupyterhub creation process finishes and the hub is ready to show the login, the **System Log** should look similar to the image below. Scroll to the bottom of your output from the previous step. Note the line **Starting TLJH installer**, you may also see **Started jupyterhub.service**



```
..
y daemon.
 amazon-ssm-agent, revision 930...
 unit for amazon-ssm-agent, revision 930.
ce for snap applicatio…amazon-ssm-agent.amazon-ssm-agent.
until snapd is fully seeded.
ings specified in cloud-config...
t Multi-User System.
t Graphical Interface.
out System Runlevel Changes...
e UTMP about System Runlevel Changes.
 Cloud-init v. 18.4-0ubuntu1~18.04.1 running 'modules:config' at Tue, 19 Mar 2019 17:
 the settings specified in cloud-config.
user/final scripts...
   % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
   0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0100  3
 Checking if TLJH is already installed...
 Setting up hub environment
 Installed python & virtual environment
 Set up hub virtual environment
 Setting up TLJH installer...
 Setup tljh package
 Starting TLJH installer...
 Setting up admin users
 Granting passwordless sudo to JupyterHub admins...
 Setting up user environment...
 Downloading & setting up user environment...
 Setting up JupyterHub...
 Node v10.15.3
 > /usr/bin/npm pack @jupyterlab/hub-extension
 Node v10.15.3
 > /usr/bin/npm pack @jupyter-widgets/jupyterlab-manager
```

20. When the installation is complete, it should give you a JupyterHub login page.



Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub.

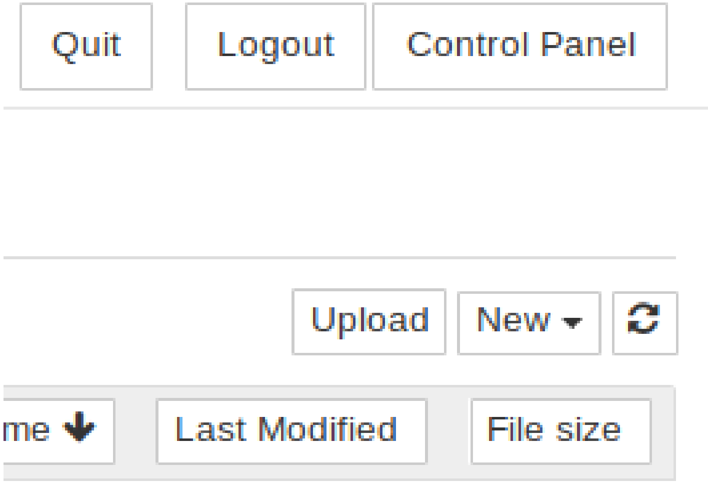
21. Login using the **admin user name** you used in step 7, and a password. Use a strong password & note it down somewhere, since this will be the password for the admin user account from now on.

22. Congratulations, you have a running working JupyterHub!

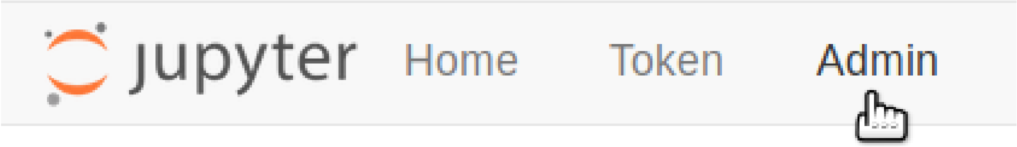
### Step 2: Adding more users

Most administration & configuration of the JupyterHub can be done from the web UI directly. Let's add a few users who can log in!

1. Open the **Control Panel** by clicking the control panel button on the top right of your JupyterHub.

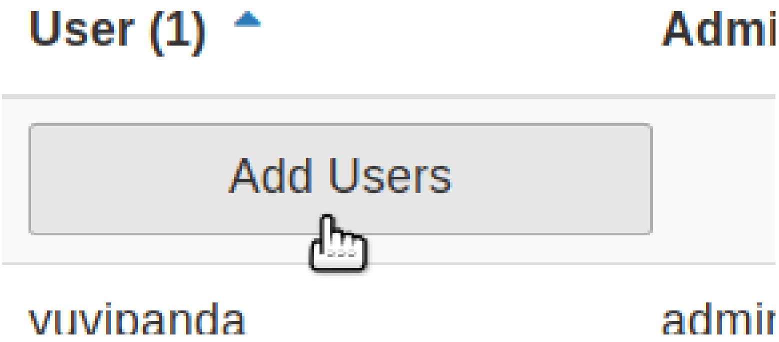


2. In the control panel, open the **Admin** link in the top left.



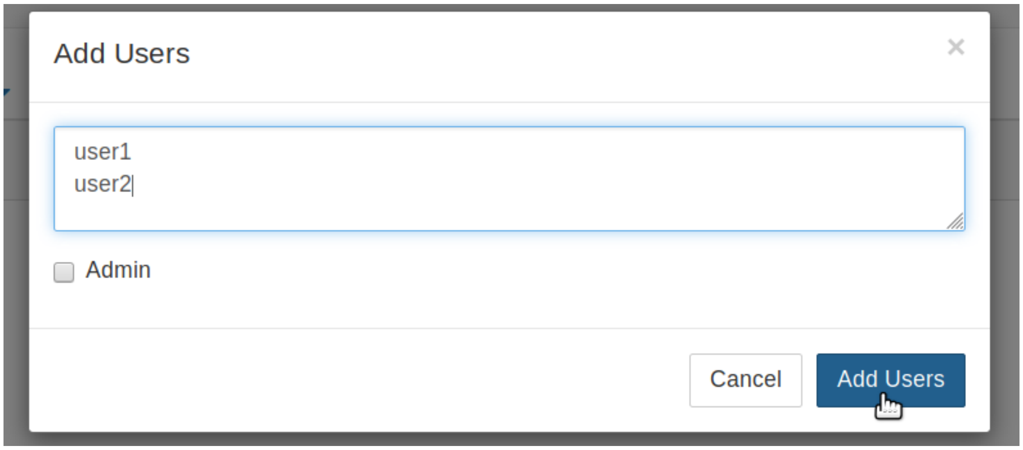
This opens up the JupyterHub admin page, where you can add / delete users, start / stop peoples' servers and see who is online.

3. Click the **Add Users** button.



A **Add Users** dialog box opens up.

4. Type the names of users you want to add to this JupyterHub in the dialog box, one per line.


Add Users

user1
user2

You can tick the **Admin** checkbox if you want to give admin rights to all these users too.

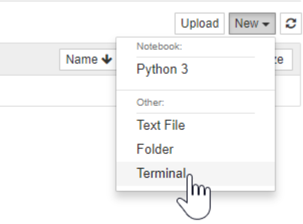
5. Click the **Add Users** button in the dialog box. Your users are now added to the JupyterHub! When they log in for the first time, they can set their password - and use it to log in again in the future.

Congratulations, you now have a multi user JupyterHub that you can add arbitrary users to!

### Step 3: Install conda / pip packages for all users

The **User Environment** is a conda environment that is shared by all users in the JupyterHub. Libraries installed in this environment are immediately available to all users. Admin users can install packages in this environment with `sudo -E`.

1. Log in as an admin user and open a Terminal in your Jupyter Notebook.



2. Install gdal from conda-forge.

```
sudo -E conda install -c conda-forge gdal
```

The `sudo -E` is very important!

3. Install `there` with `pip`

```
sudo -E pip install there
```

The packages `gdal` and `there` are now available to all users in JupyterHub. If a user already had a python notebook running, they have to restart their notebook's kernel to make the new libraries available.

See *Install conda, pip or apt packages* for more information.

## 2.1.6 Installing on Azure

### Goal

By the end of this tutorial, you should have a JupyterHub with some admin users and a user environment with packages you want to be installed running on Microsoft Azure.

This tutorial leads you step-by-step for you to manually deploy your own JupyterHub on Azure cloud.

**Note:** The `Deploy to Azure button` project allows you to deploy your own JupyterHub with minimal manual configuration steps. The deploy to Azure button allows you to have a vanilla configuration in just one-click and by assigning some variables.

Check it out at https://github.com/trallard/TLJH-azure-button.

### Prerequisites

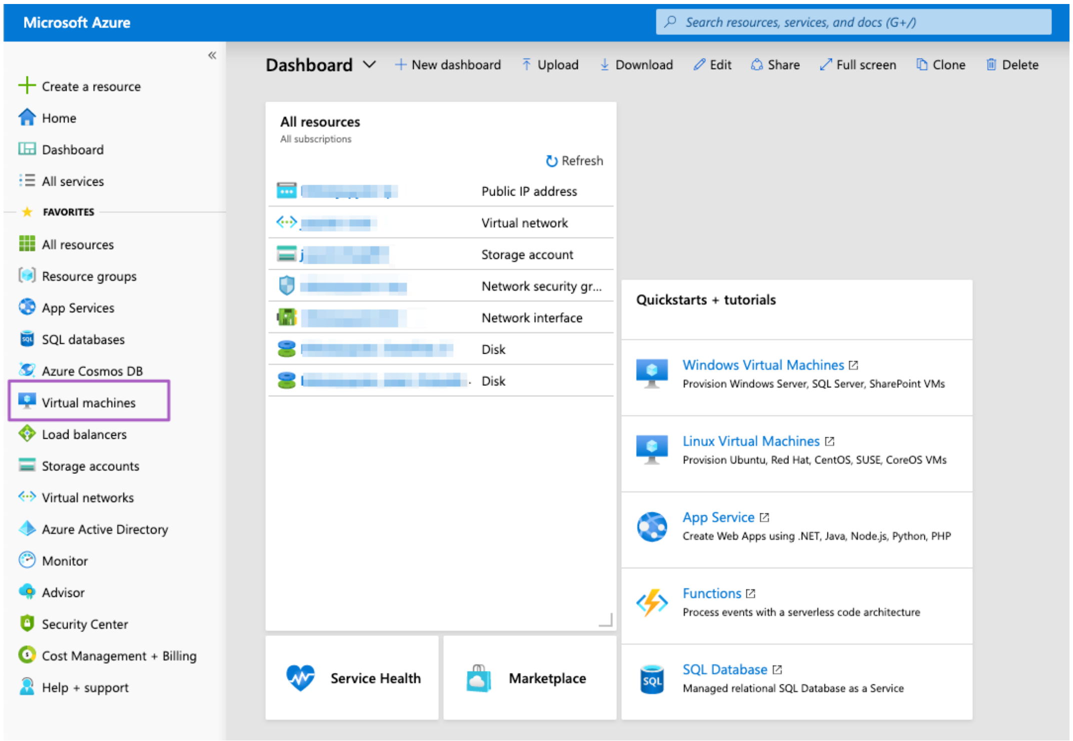
- A Microsoft Azure account.
- To get started you can get a free account which includes 150 dollars worth of Azure credits (get a free account here)

These instructions cover how to set up a Virtual Machine on Microsoft Azure. For subsequent information about creating your JupyterHub and configuring it, see The Littlest JupyterHub guide.

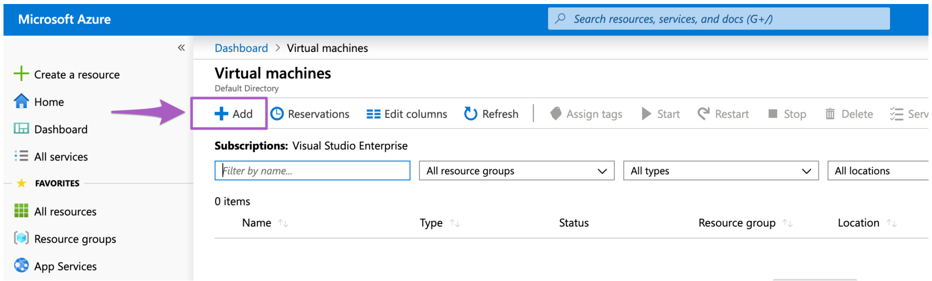
### Step 1: Installing The Littlest JupyterHub

We start by creating the Virtual Machine in which we can run TLJH (The Littlest JupyterHub).

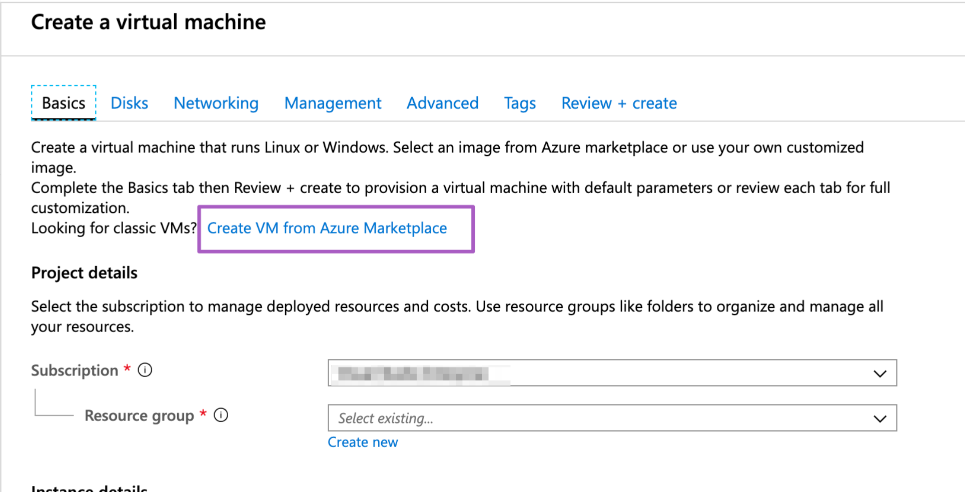
1. Go to Azure portal and login with your Azure account.
2. Expand the left-hand panel by clicking on the ">>" button on the top left corner of your dashboard. Find the Virtual Machines tab and click on it.



1. Click **+ add** to create a new Virtual Machine



#. Select **Create VM from Marketplace** in the next screen. A new screen with all the options for Virtual Machines in Azure will displayed.



Create a virtual machine

Basics Disks Networking Management Advanced Tags Review + create

Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image.
Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization.
Looking for classic VMs? Create VM from Azure Marketplace

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

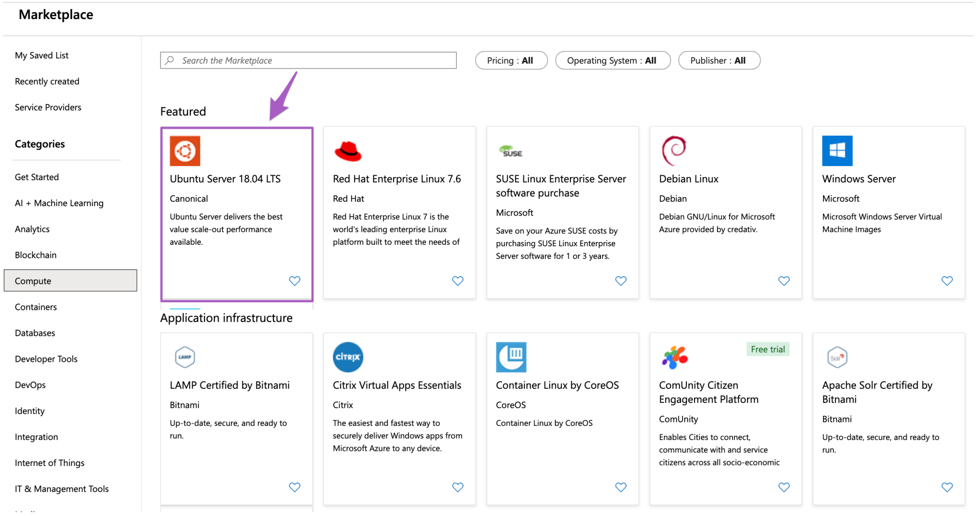
Subscription * ⓘ

Resource group * ⓘ Select existing...

Create new

1. **Choose an Ubuntu server for your VM:**

- Click *Ubuntu Server 18.04 LTS*.
- Make sure *Resource Manager* is selected in the next screen and click **Create**



Marketplace

My Saved List
Recently created
Service Providers

Categories

Get Started
AI + Machine Learning
Analytics
Blockchain
Compute
Containers
Databases
Developer Tools
DevOps
Identity
Integration
Internet of Things
IT & Management Tools

Featured

Ubuntu Server 18.04 LTS
Canonical
Ubuntu Server delivers the best value scale-out performance available.

Red Hat Enterprise Linux 7.6
Red Hat
Red Hat Enterprise Linux 7 is the world's leading enterprise Linux platform built to meet the needs of

SUSE Linux Enterprise Server software purchase
Microsoft
Save on your Azure SUSE costs by purchasing SUSE Linux Enterprise Server software for 1 or 3 years.

Debian Linux
Debian
Debian GNU/Linux for Microsoft Azure provided by credativ.

Windows Server
Microsoft
Microsoft Windows Server Virtual Machine Images

Application infrastructure

LAMP Certified by Bitnami
Bitnami
Up-to-date, secure, and ready to run.

Citrix Virtual Apps Essentials
Citrix
The easiest and fastest way to securely deliver Windows apps from Microsoft Azure to any device.

Container Linux by CoreOS
CoreOS
Container Linux by CoreOS

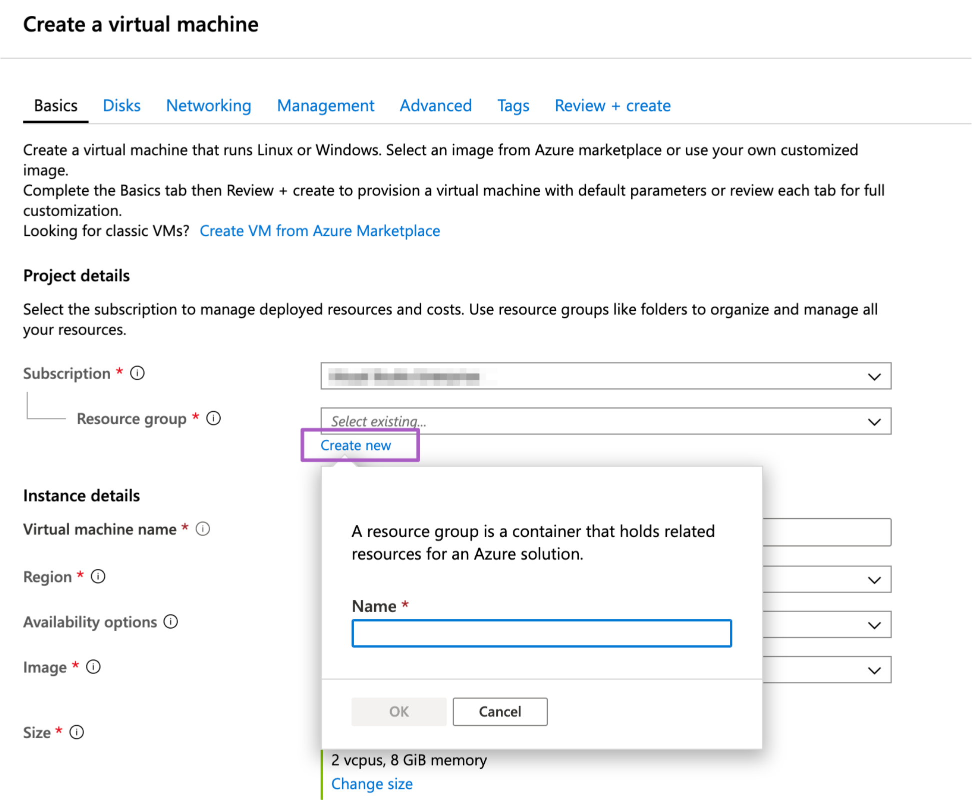
Free trial
ComUnity Citizen Engagement Platform
ComUnity
Enables Cities to connect, communicate with and service citizens across all socio-economic

Apache Solr Certified by Bitnami
Bitnami
Up-to-date, secure, and ready to run.

2. **Customise the Virtual Machine basics:**

   - **Subscription**. Choose the "Free Trial" if this is what you're using. Otherwise, choose a different plan. This is the billing account that will be charged.
   - **Resource group**. Resource groups let you keep your Azure tools/resources together in an availability region (e.g. WestEurope). If you already have one you'd like to use it select that resource.

**Note:** If you have never created a Resource Group, click on **Create new**



Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image.
Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization.
Looking for classic VMs? Create VM from Azure Marketplace

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * [redacted]

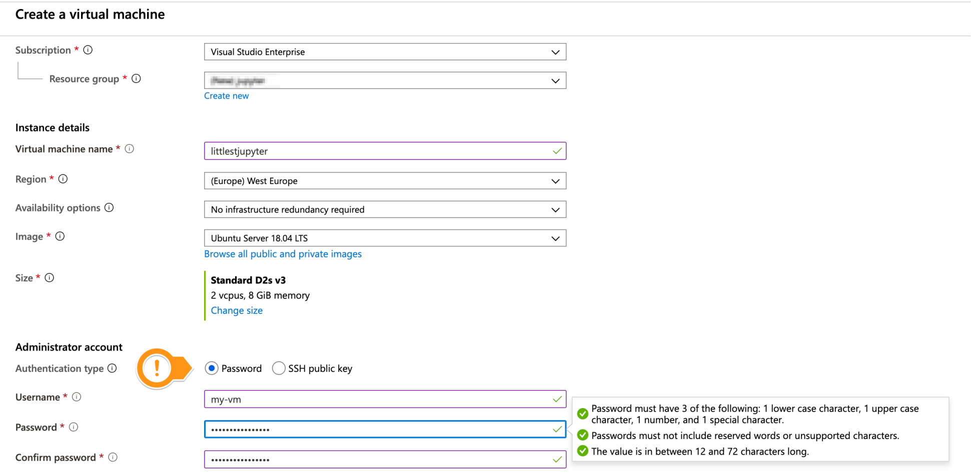
Resource group * *Select existing...*

A resource group is a container that holds related resources for an Azure solution.

Name *

Size * 2 vcpus, 8 GiB memory

- **Name**. Use a descriptive name for your virtual machine (note that you cannot use spaces or special characters).
- **Region**. Choose a location near where you expect your users to be located.
- **Availability options**. Choose "No infrastructure redundancy required".
- **Image**. Make sure "Ubuntu Server 18.04 LTS" is selected (from the previous step).
- **Authentication type**. Change authentication type to "password".
- **Username**. Choose a memorable username, this will be your "root" user, and you'll need it later on.
- **Password**. Type in a password, this will be used later for admin access so make sure it is something memorable.



Create a virtual machine

Subscription * ⓘ: Visual Studio Enterprise

Resource group * ⓘ: [redacted]

Create new

Instance details

Virtual machine name * ⓘ: littlestjupyter ✓

Region * ⓘ: (Europe) West Europe

Availability options ⓘ: No infrastructure redundancy required

Image * ⓘ: Ubuntu Server 18.04 LTS

Browse all public and private images

Size * ⓘ: **Standard D2s v3**
2 vcpus, 8 GiB memory
Change size

Administrator account

Authentication type ⓘ: ● Password ○ SSH public key

Username * ⓘ: my-vm ✓

Password * ⓘ: ••••••••••••••• ✓

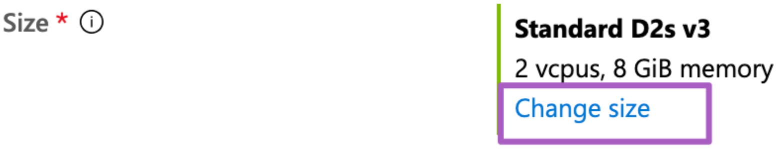
Confirm password * ⓘ: ••••••••••••••• ✓

✓ Password must have 3 of the following: 1 lower case character, 1 upper case character, 1 number, and 1 special character.
✓ Passwords must not include reserved words or unsupported characters.
✓ The value is in between 12 and 72 characters long.

- **Login with Azure Active Directory**. Choose “Off” (usually the default)
- **Inbound port rules**. Leave the defaults for now, and we will update these later on in the Network configuration step.

3. **Before clicking on “Next” we need to select the RAM size for the image.**

   - For this we need to make sure we have enough RAM to accommodate your users. For example, if each user needs 2GB of RAM, and you have 10 total users, you need at least 20GB of RAM on the machine. It’s also good to have a few GB of “buffer” RAM beyond what you think you’ll need.
   - Click on **Change size** (see image below)



Size * ⓘ

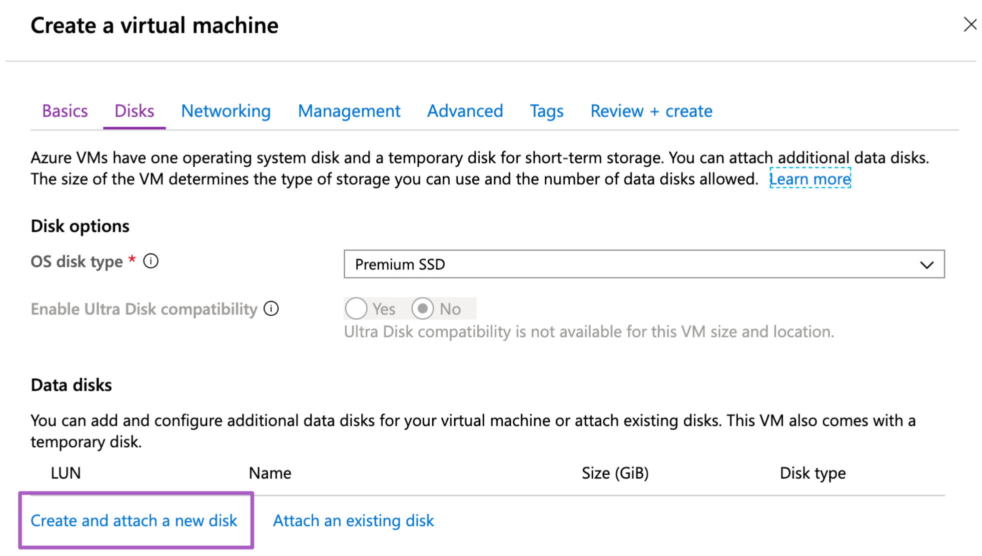
**Standard D2s v3**
2 vcpus, 8 GiB memory
Change size

**Note:** For more information about estimating memory, CPU and disk needs check The memory section in the TLJH documentation

   - Select a suitable image (to check available images and prices in your region click on this link).

4. **Disks (Storage):**

   - **Disk options**: select the OS disk type there are options for SDD and HDD. **SSD persistent disk** gives you a faster but more expensive disk than HDD.
   - **Data disk**. Click on create and attach a new disk. Select an appropriate type and size and click ok.
   - Click “Next”.



Create a virtual machine

Azure VMs have one operating system disk and a temporary disk for short-term storage. You can attach additional data disks. The size of the VM determines the type of storage you can use and the number of data disks allowed. Learn more

Disk options

OS disk type * ⓘ: Premium SSD

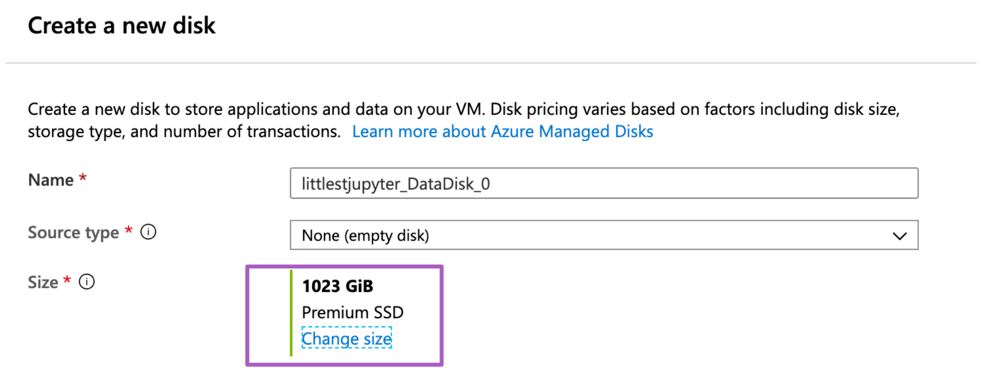
Enable Ultra Disk compatibility ⓘ: ○ Yes ● No

Ultra Disk compatibility is not available for this VM size and location.

Data disks

You can add and configure additional data disks for your virtual machine or attach existing disks. This VM also comes with a temporary disk.

| LUN | Name | Size (GiB) | Disk type |
|---|---|---|---|



Create a new disk

Create a new disk to store applications and data on your VM. Disk pricing varies based on factors including disk size, storage type, and number of transactions. Learn more about Azure Managed Disks

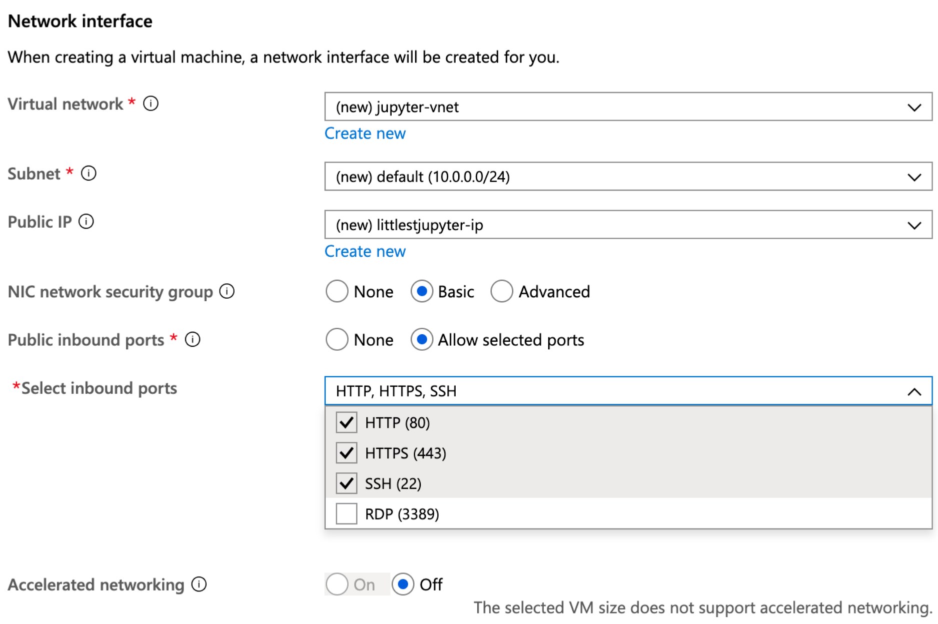
Name *: littlestjupyter_DataDisk_0

Source type * ⓘ: None (empty disk)

Size * ⓘ: 1023 GiB Premium SSD

5. **Networking**

   - **Virtual network**. Leave the default values selected.
   - **Subnet**. Leave the default values selected.
   - **Public IP address**.Leave the default values selected. This will make your server accessible from a browser.
   - **Network Security Group**. Choose “Basic”
   - **Public inbound ports**. Check **HTTP**, **HTTPS**, and **SSH**.



Network interface

When creating a virtual machine, a network interface will be created for you.

Virtual network * : (new) jupyter-vnet

Create new

Subnet * : (new) default (10.0.0.0/24)

Public IP : (new) littlestjupyter-ip

Create new

NIC network security group: ○ None ● Basic ○ Advanced

Public inbound ports * : ○ None ● Allow selected ports

*Select inbound ports: HTTP, HTTPS, SSH

☑ HTTP (80)

☑ HTTPS (443)

☑ SSH (22)

□ RDP (3389)

Accelerated networking: ○ On ● Off

The selected VM size does not support accelerated networking.

6. **Management**

   - **Monitoring**

     - **Boot diagnostics**. Choose "On".
     - **OS guest diagnostics**. Choose "Off".
     - **Diagnostics storage account**. Leave as the default.

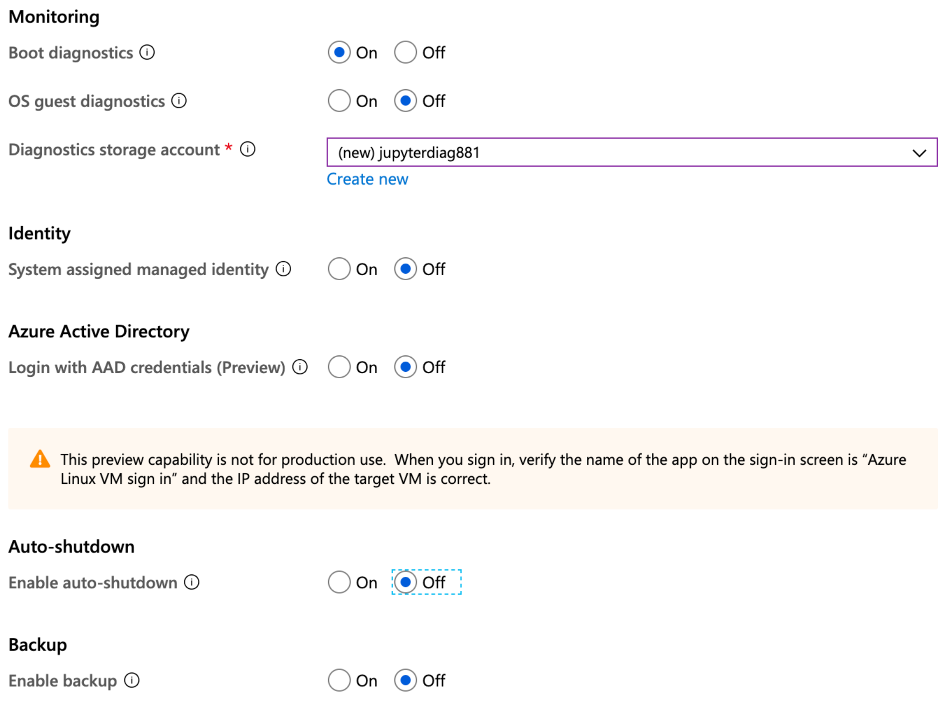
   - **Auto-Shutdown**

     - **Enable auto-shutdown**. Choose "Off".

   - **Backup**

     - **Backup**. Choose "Off".

   - System assigned managed identity Select "Off"



**Monitoring**

Boot diagnostics ⓘ ● On ○ Off

OS guest diagnostics ⓘ ○ On ● Off

Diagnostics storage account * ⓘ (new) jupyterdiag881

Create new

**Identity**

System assigned managed identity ⓘ ○ On ● Off

**Azure Active Directory**

Login with AAD credentials (Preview) ⓘ ○ On ● Off

⚠ This preview capability is not for production use. When you sign in, verify the name of the app on the sign-in screen is "Azure Linux VM sign in" and the IP address of the target VM is correct.

**Auto-shutdown**

Enable auto-shutdown ⓘ ○ On ● Off

**Backup**

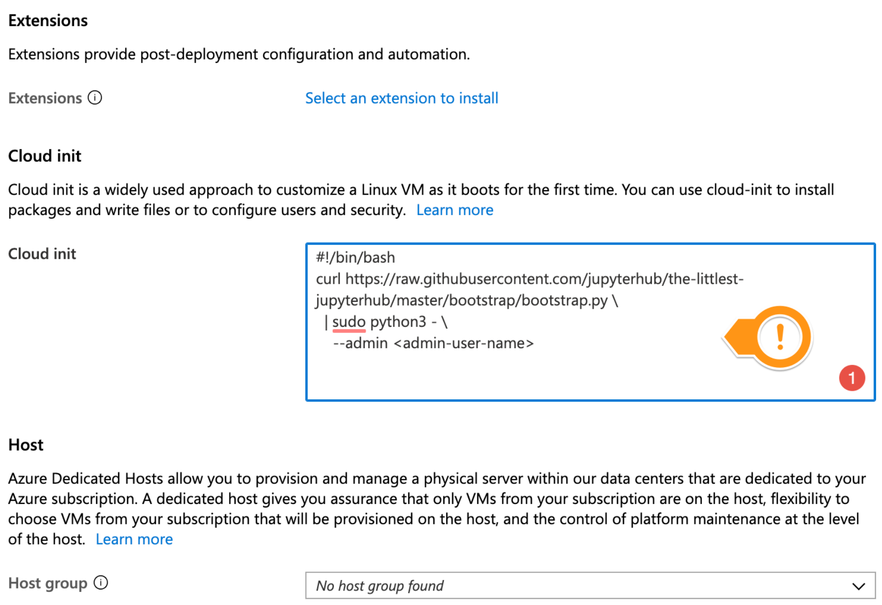
Enable backup ⓘ ○ On ● Off

7. **Advanced settings**

   - **Extensions**. Make sure there are no extensions listed
   - **Cloud init**. We are going to use this section to install TLJH directly into our Virtual Machine. Copy the code snippet below:

```
#!/bin/bash
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-
↪jupyterhub/master/bootstrap/bootstrap.py \
  | sudo python3 - \
    --admin <admin-user-name>
```

   where the `username` is the root username you chose for your Virtual Machine.



**Extensions**

Extensions provide post-deployment configuration and automation.

Extensions ⓘ Select an extension to install

**Cloud init**

Cloud init is a widely used approach to customize a Linux VM as it boots for the first time. You can use cloud-init to install packages and write files or to configure users and security. Learn more

Cloud init

```
#!/bin/bash
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-
jupyterhub/master/bootstrap/bootstrap.py \
  | sudo python3 - \
    --admin <admin-user-name>
```

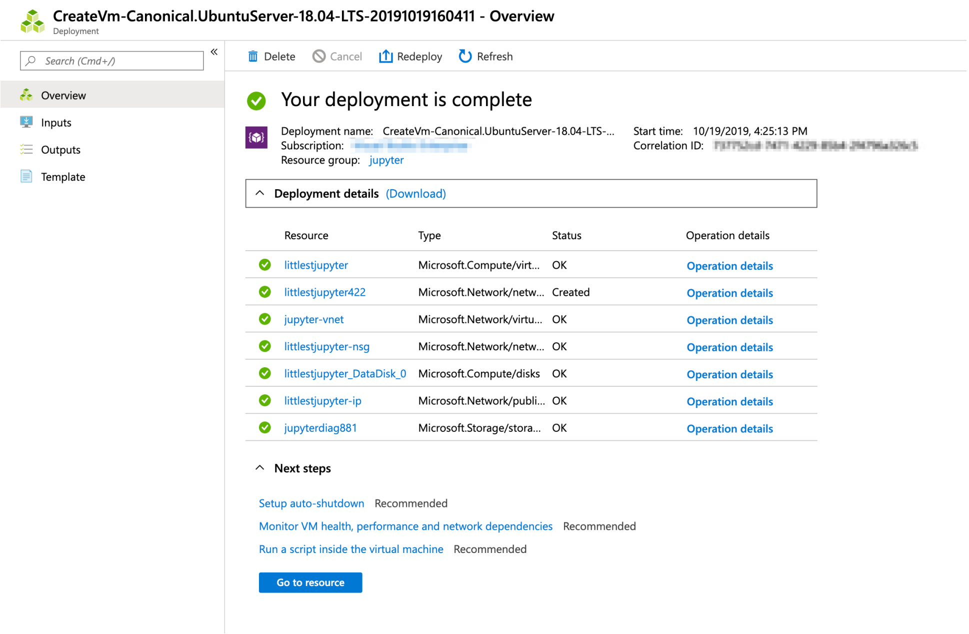
**Host**

Azure Dedicated Hosts allow you to provision and manage a physical server within our data centers that are dedicated to your Azure subscription. A dedicated host gives you assurance that only VMs from your subscription are on the host, flexibility to choose VMs from your subscription that will be provisioned on the host, and the control of platform maintenance at the level of the host. Learn more

Host group ⓘ *No host group found*

**Note:** See *What does the installer do?* if you want to understand exactly what the installer is doing. *Customizing the Installer* documents other options that can be passed to the installer.

8. Check the summary and confirm the creation of your Virtual Machine.
9. **Check that the creation of your Virtual Machine worked.**
   - Wait for the virtual machine to be created. This might take about 5-10 minutes.
   - After completion, you should see a similar screen to the one below:



**CreateVm-Canonical.UbuntuServer-18.04-LTS-20191019160411 - Overview**
Deployment

Your deployment is complete

Deployment name: CreateVm-Canonical.UbuntuServer-18.04-LTS-...
Subscription: [redacted]
Resource group: jupyter
Start time: 10/19/2019, 4:25:13 PM
Correlation ID: [redacted]

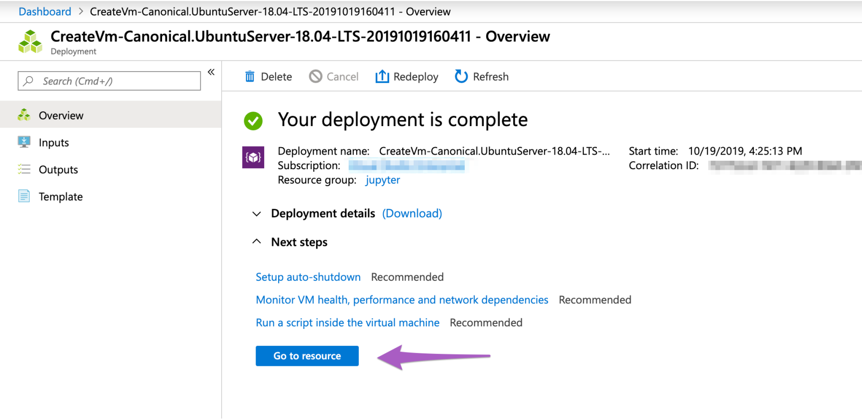
Deployment details (Download)

| Resource | Type | Status | Operation details |
|---|---|---|---|
| littlestjupyter | Microsoft.Compute/virt... | OK | Operation details |
| littlestjupyter422 | Microsoft.Network/netw... | Created | Operation details |
| jupyter-vnet | Microsoft.Network/virtu... | OK | Operation details |
| littlestjupyter-nsg | Microsoft.Network/netw... | OK | Operation details |
| littlestjupyter_DataDisk_0 | Microsoft.Compute/disks | OK | Operation details |
| littlestjupyter-ip | Microsoft.Network/publi... | OK | Operation details |
| jupyterdiag881 | Microsoft.Storage/stora... | OK | Operation details |

Next steps

Setup auto-shutdown Recommended
Monitor VM health, performance and network dependencies Recommended
Run a script inside the virtual machine Recommended

10. **Note that the Littlest JupyterHub should be installing in the background on your new server.** It takes around 5-10 minutes for this installation to complete.


Dashboard > CreateVm-Canonical.UbuntuServer-18.04-LTS-20191019160411 - Overview
Search (Cmd+/)
Overview
Inputs
Outputs
Template
Delete
Cancel
Redeploy
Refresh
Deployment details (Download)
Next steps
Go to resource

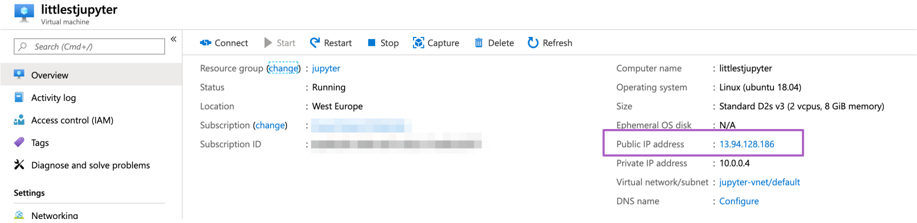
CreateVm-Canonical.UbuntuServer-18.04-LTS-20191019160411 - Overview
Deployment

Your deployment is complete

Deployment name: CreateVm-Canonical.UbuntuServer-18.04-LTS-...
Subscription: [redacted]
Resource group: jupyter
Start time: 10/19/2019, 4:25:13 PM
Correlation ID: [redacted]

Setup auto-shutdown Recommended
Monitor VM health, performance and network dependencies Recommended
Run a script inside the virtual machine Recommended

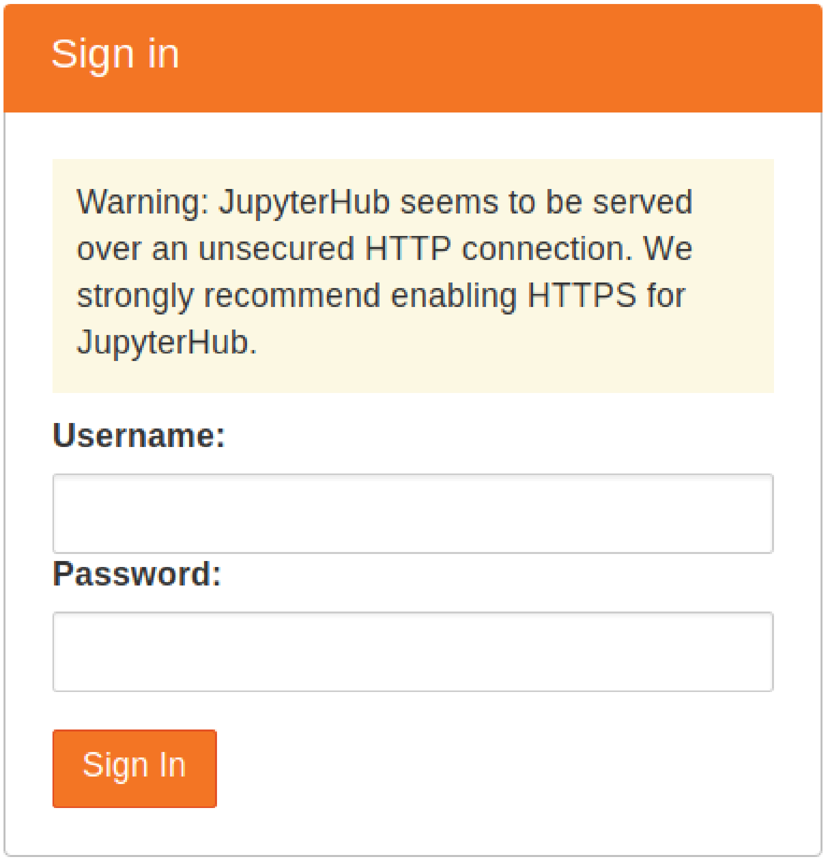
11. **Click on the Go to resource button**
12. Check if the installation is completed by **copying** the **Public IP address** of your virtual machine, and trying to access it with a browser.



Resource group (change) : jupyter
Status : Running
Location : West Europe
Subscription (change) : [redacted]
Subscription ID : [redacted]
Computer name : littlestjupyter
Operating system : Linux (ubuntu 18.04)
Size : Standard D2s v3 (2 vcpus, 8 GiB memory)
Ephemeral OS disk : N/A
Public IP address : 13.94.128.186
Private IP address : 10.0.0.4
Virtual network/subnet : jupyter-vnet/default
DNS name : Configure

Note that accessing the JupyterHub will fail until the installation is complete, so be patient.

13. When the installation is complete, it should give you a JupyterHub login page.



Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub.

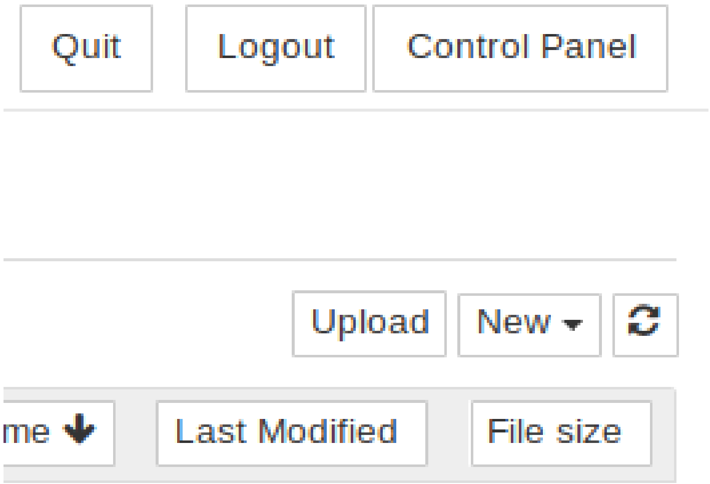
14. Login using the **admin user name** you used in step 6, and a password. Use a strong password & note it down somewhere, since this will be the password for the admin user account from now on.

15. Congratulations, you have a running working JupyterHub!

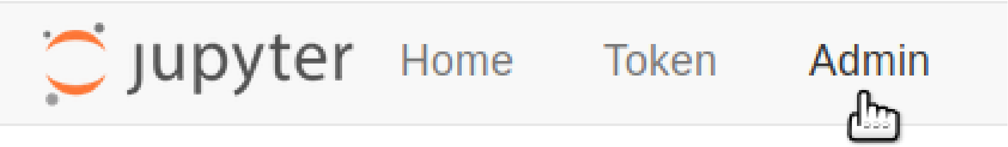
### Step 2: Adding more users

Most administration & configuration of the JupyterHub can be done from the web UI directly. Let's add a few users who can log in!

1. Open the **Control Panel** by clicking the control panel button on the top right of your JupyterHub.

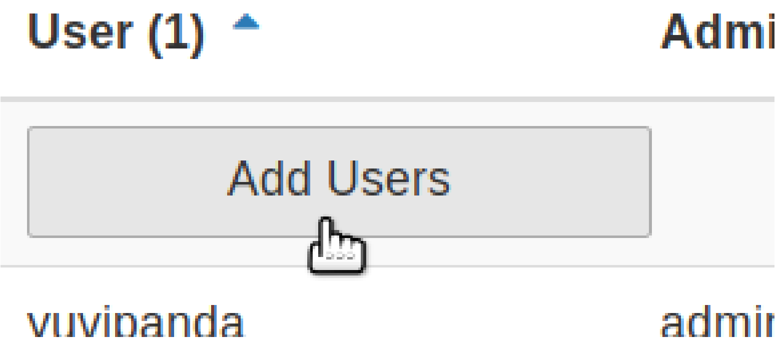


2. In the control panel, open the **Admin** link in the top left.



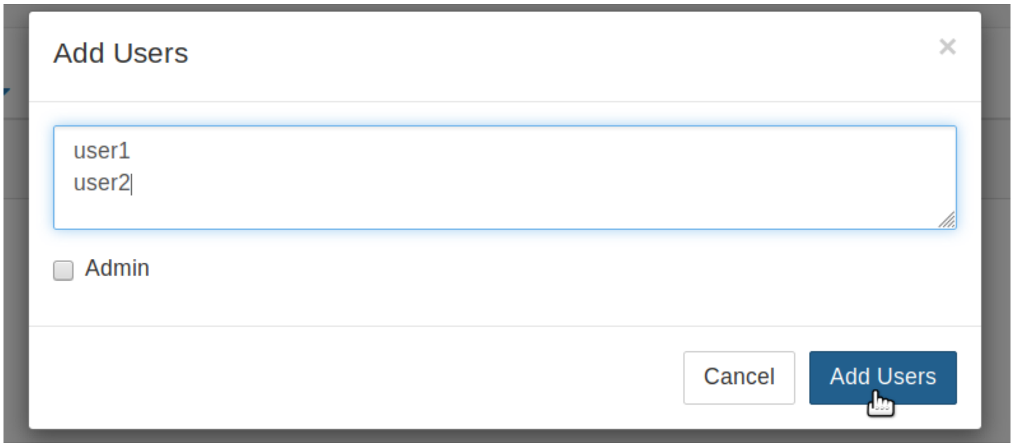
   This opens up the JupyterHub admin page, where you can add / delete users, start / stop peoples' servers and see who is online.

3. Click the **Add Users** button.



   A **Add Users** dialog box opens up.

4. Type the names of users you want to add to this JupyterHub in the dialog box, one per line.



user1
user2

□ Admin

You can tick the **Admin** checkbox if you want to give admin rights to all these users too.

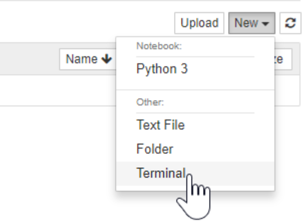
5. Click the **Add Users** button in the dialog box. Your users are now added to the JupyterHub! When they log in for the first time, they can set their password - and use it to log in again in the future.

Congratulations, you now have a multi user JupyterHub that you can add arbitrary users to!

### Step 3: Install conda / pip packages for all users

The **User Environment** is a conda environment that is shared by all users in the JupyterHub. Libraries installed in this environment are immediately available to all users. Admin users can install packages in this environment with `sudo -E`.

1. Log in as an admin user and open a Terminal in your Jupyter Notebook.



2. Install gdal from conda-forge.

```
sudo -E conda install -c conda-forge gdal
```

The `sudo -E` is very important!

3. Install `there` with `pip`

```
sudo -E pip install there
```

The packages `gdal` and `there` are now available to all users in JupyterHub. If a user already had a python notebook running, they have to restart their notebook's kernel to make the new libraries available.

See *Install conda, pip or apt packages* for more information.

### 2.1.7 Installing on your own server

Follow this guide if your cloud provider doesn't have a direct tutorial, or you are setting this up on a bare metal server.

**Warning:** Do **not** install TLJH directly on your laptop or personal computer! It will most likely open up exploitable security holes when run directly on your personal computer.

**Note:** Running TLJH *inside* a docker container is not supported, since we depend on systemd. If you want to run TLJH locally for development, see *Setting up Development Environment*.

#### Goal

By the end of this tutorial, you should have a JupyterHub with some admin users and a user environment with packages you want installed running on a server you have access to.

#### Pre-requisites

1. Some familiarity with the command line.
2. A server running Ubuntu 18.04 where you have root access.
3. At least **1.15GB** of RAM on your server.
4. Ability to `ssh` into the server & run commands from the prompt.
5. A **IP address** where the server can be reached from the browsers of your target audience.

If you run into issues, look at the specific *troubleshooting guide* for custom server installations.

#### Step 1: Installing The Littlest JupyterHub

1. Using a terminal program, SSH into your server. This should give you a prompt where you can type commands.
2. Make sure you have `python3`, `python3-dev`, `curl` and `git` installed.

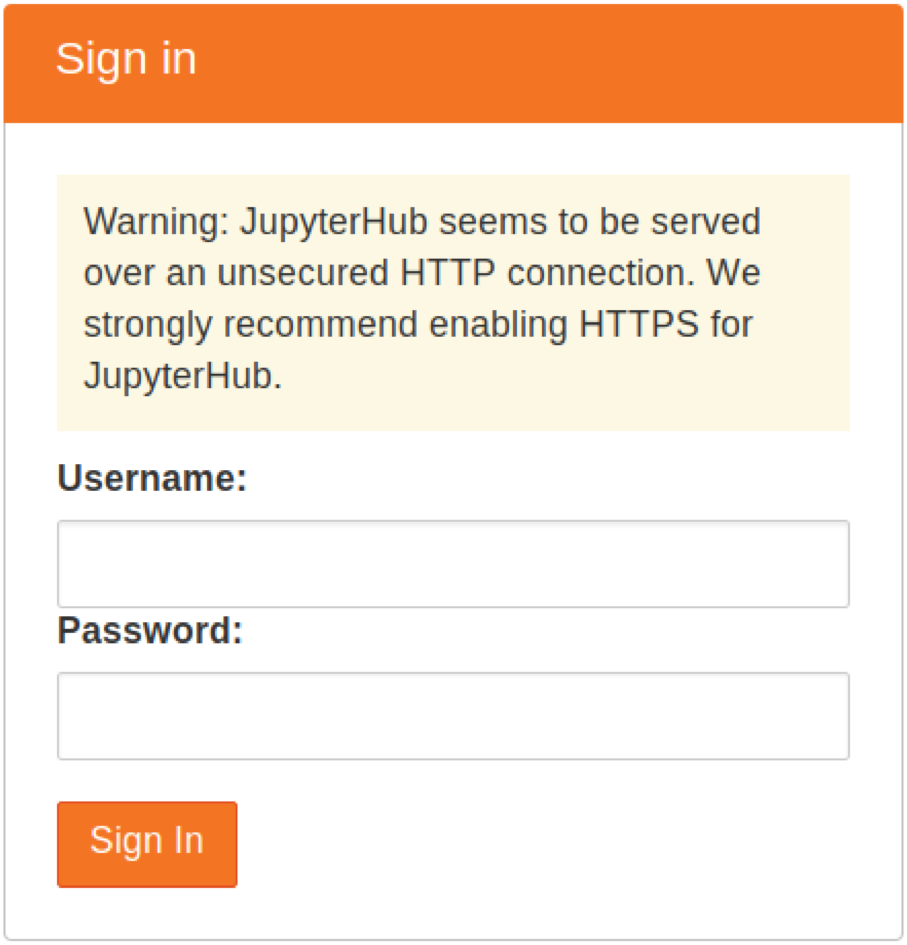
```
sudo apt install python3 python3-dev git curl
```

3. Copy the text below, and paste it into the terminal. Replace `<admin-user-name>` with the name of the first **admin user** for this JupyterHub. Choose any name you like (don't forget to remove the brackets!). This admin user can log in after the JupyterHub is set up, and can configure it to their needs. **Remember to add your username**!

```
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-
↪jupyterhub/master/bootstrap/bootstrap.py | sudo -E python3 - --
↪admin <admin-user-name>
```

**Note:** See *What does the installer do?* if you want to understand exactly what the installer is doing. *Customizing the Installer* documents other options that can be passed to the installer.

4. Press `Enter` to start the installation process. This will take 5-10 minutes, and will say 'Done!' when the installation process is complete.
5. Copy the **Public IP** of your server, and try accessing `http://<public-ip>` from your browser. If everything went well, this should give you a JupyterHub login page.



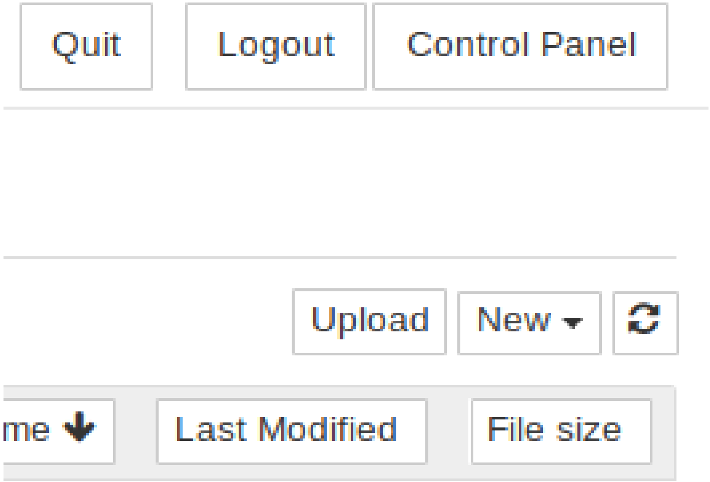
Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub.

6. Login using the **admin user name** you used in step 2. You can choose any password that you wish. Use a strong password & note it down somewhere, since this will be the password for the admin user account from now on.
7. Congratulations, you have a running working JupyterHub!

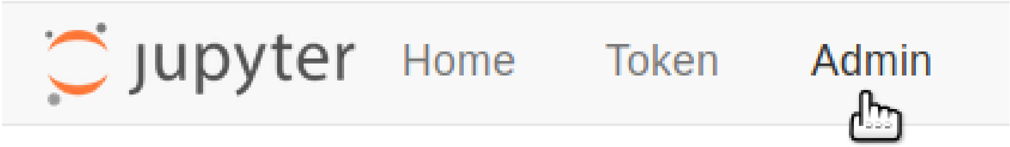
### Step 2: Adding more users

Most administration & configuration of the JupyterHub can be done from the web UI directly. Let's add a few users who can log in!

1. Open the **Control Panel** by clicking the control panel button on the top right of your JupyterHub.

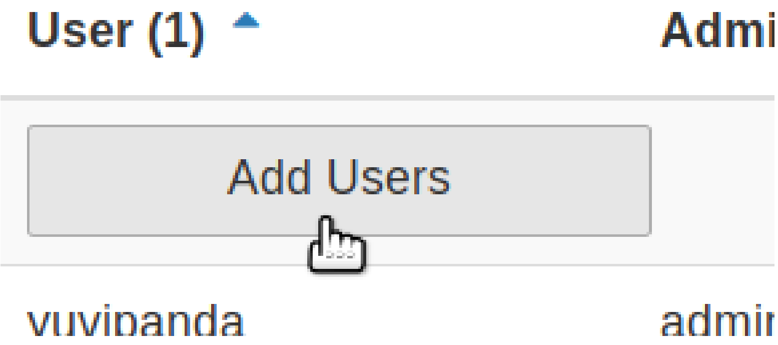


2. In the control panel, open the **Admin** link in the top left.



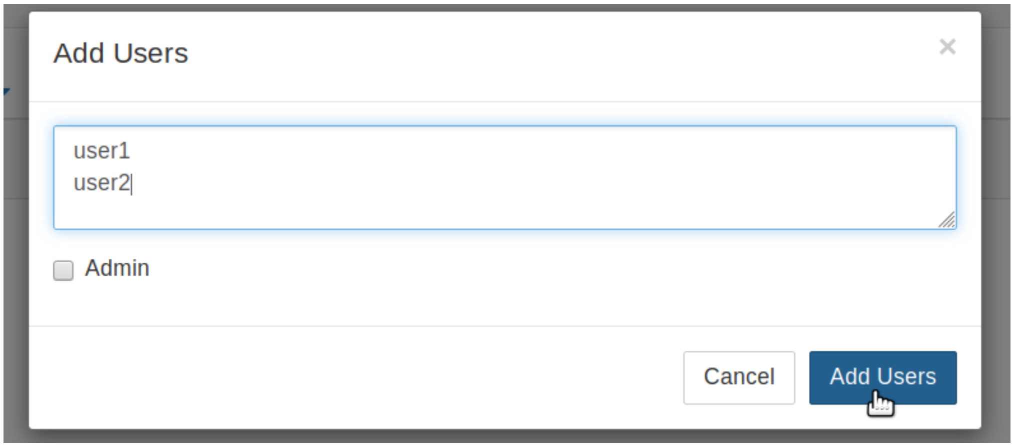
This opens up the JupyterHub admin page, where you can add / delete users, start / stop peoples' servers and see who is online.

3. Click the **Add Users** button.



A **Add Users** dialog box opens up.

4. Type the names of users you want to add to this JupyterHub in the dialog box, one per line.



user1
user2

You can tick the **Admin** checkbox if you want to give admin rights to all these users too.

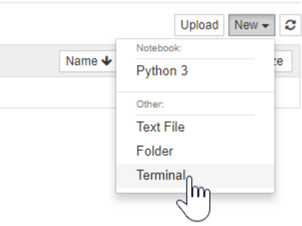
5. Click the **Add Users** button in the dialog box. Your users are now added to the JupyterHub! When they log in for the first time, they can set their password - and use it to log in again in the future.

Congratulations, you now have a multi user JupyterHub that you can add arbitrary users to!

### Step 3: Install conda / pip packages for all users

The **User Environment** is a conda environment that is shared by all users in the JupyterHub. Libraries installed in this environment are immediately available to all users. Admin users can install packages in this environment with `sudo -E`.

1. Log in as an admin user and open a Terminal in your Jupyter Notebook.



2. Install gdal from conda-forge.

```
sudo -E conda install -c conda-forge gdal
```

The `sudo -E` is very important!

3. Install `there` with `pip`

```
sudo -E pip install there
```

The packages `gdal` and `there` are now available to all users in JupyterHub. If a user already had a python notebook running, they have to restart their notebook's kernel to make the new libraries available.

See *Install conda, pip or apt packages* for more information.

#### Step 4: Setup HTTPS

Once you are ready to run your server for real, and have a domain, it's a good idea to proceed directly to *Enable HTTPS*.

Once you are ready to run your server for real, it's a good idea to proceed directly to *Enable HTTPS*.

# CHAPTER 3

# How-To Guides

How-To guides answer the question 'How do I. . . ?' for a lot of topics.

## 3.1 How-To Guides

How-To guides answer the question 'How do I. . . ?' for a lot of topics.

### 3.1.1 Content and Data

#### Distributing materials to users with nbgitpuller

#### Goal

A very common need when using JupyterHub is to easily distribute study materials / lab notebooks to students.

Students should be able to:

1. Easily get the latest version of materials, including any updates the instructor has made to materials the student already has a copy of.
2. Be confident they won't lose any of their work. If an instructor has modified something the student has also modified, the student's modification should never be overwritten.
3. Not have to deal with manual merge conflicts or other complex operations.

Instructors should be able to:

1. Use modern collaborative version control tools to author & store their materials. This currently means using Git.

**nbgitpuller** is a Jupyter Notebook extension that helps achieve these goals. This tutorial will walk you through the process of creating a magic nbgitpuller link that users of your JupyterHub can click to fetch the latest version of materials from a git repo.

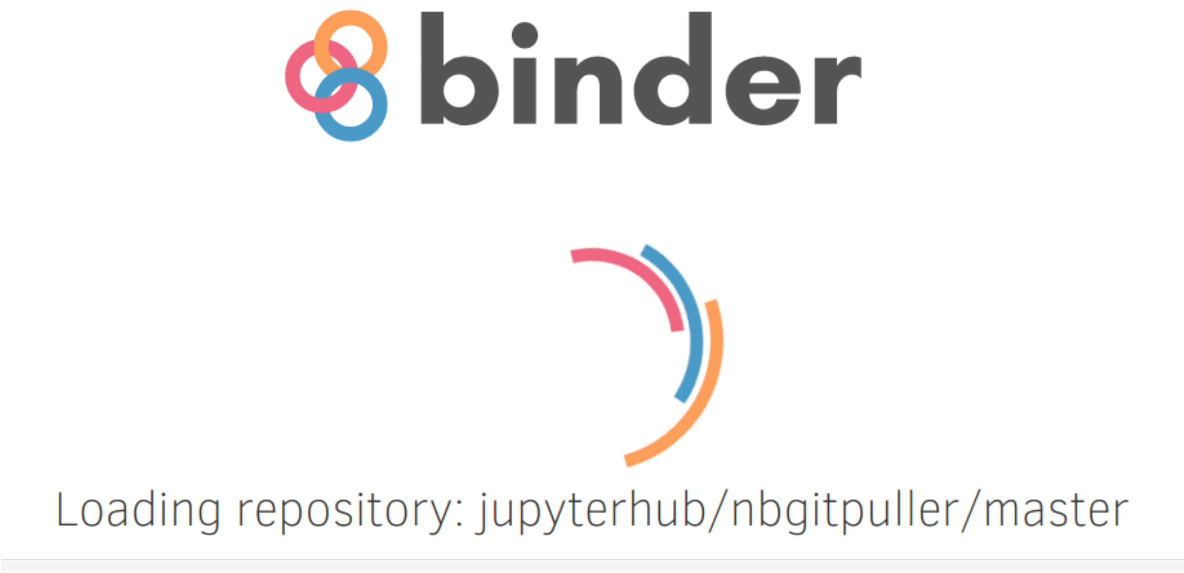
### Pre-requisites

1. A JupyterHub set up with The Littlest JupyterHub
2. A git repository containing materials to distribute

### Step 1: Generate nbgitpuller link

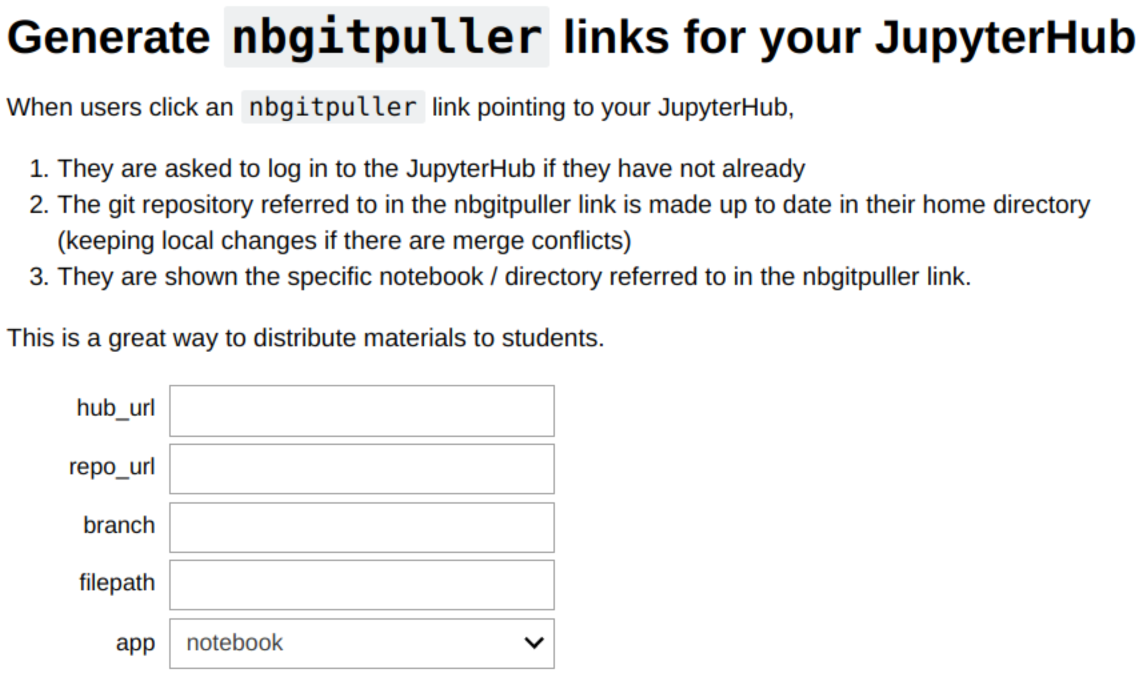
**Generate the link with a Binder app**.

1. The easiest way to generate an nbgitpuller link is to use the mybinder.org based application. Open it, and wait for it to load.



Loading repository: jupyterhub/nbgitpuller/master

2. A blank form with some help text will open up.



# Generate nbgitpuller links for your JupyterHub

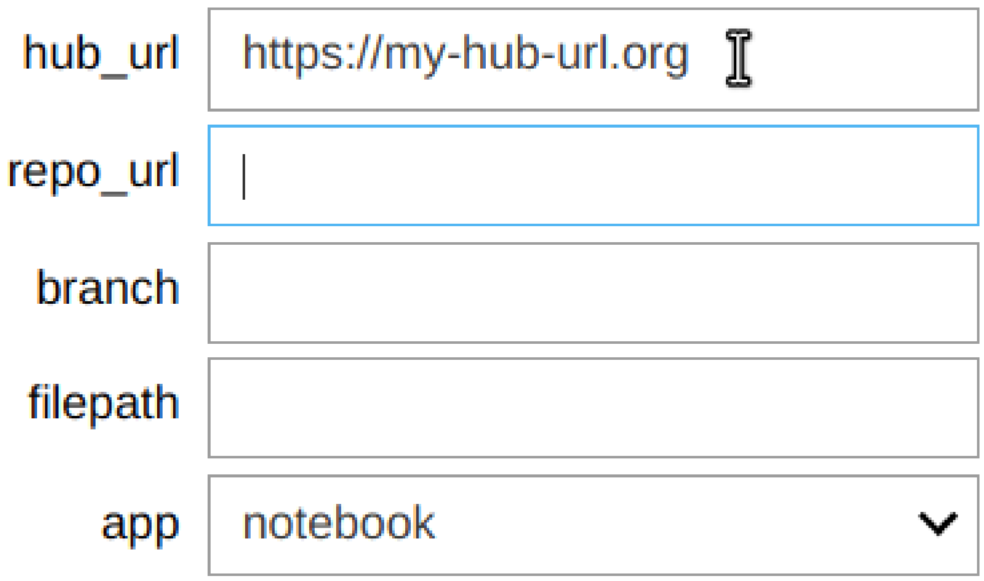
When users click an `nbgitpuller` link pointing to your JupyterHub,

1. They are asked to log in to the JupyterHub if they have not already
2. The git repository referred to in the nbgitpuller link is made up to date in their home directory (keeping local changes if there are merge conflicts)
3. They are shown the specific notebook / directory referred to in the nbgitpuller link.

This is a great way to distribute materials to students.

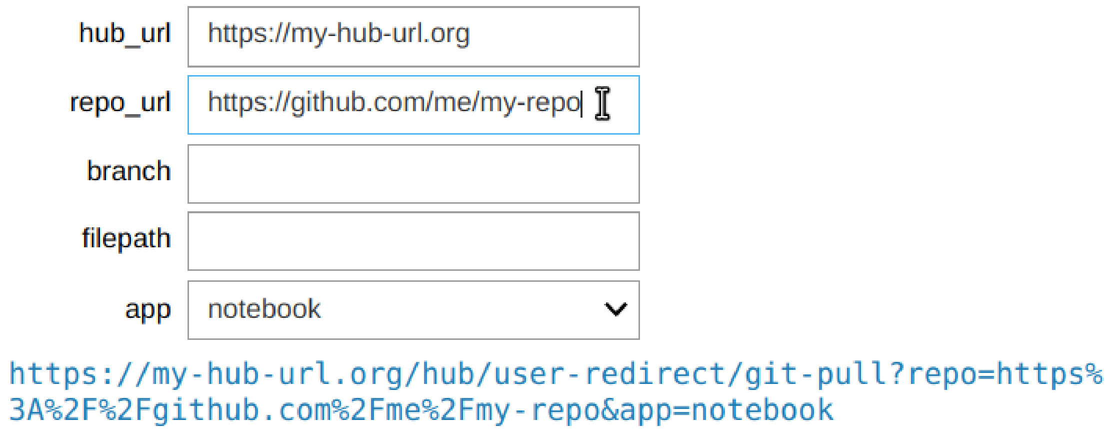
| | |
|---|---|
| hub_url | |
| repo_url | |
| branch | |
| filepath | |
| app | notebook |

3. Enter the IP address or URL to your JupyterHub under `hub_url`. Include `http://` or `https://` as appropriate.



| | |
|---|---|
| hub_url | https://my-hub-url.org |
| repo_url | |
| branch | |
| filepath | |
| app | notebook |

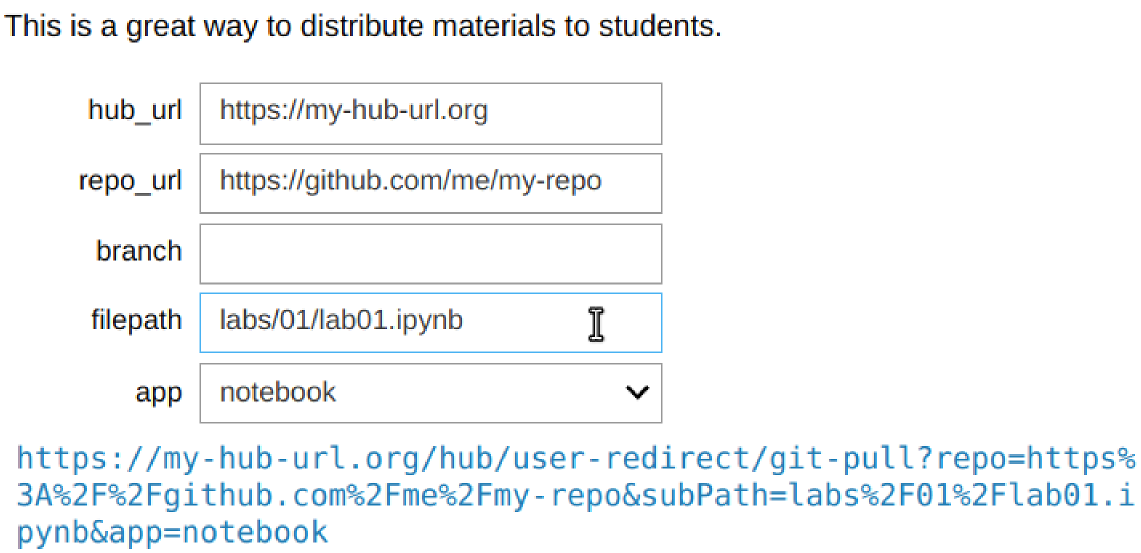
4. Enter the URL to your Git repository. This could be from GitHub, GitLab or any other git provider - including the disk of the server The Littlest JupyterHub is installed on. As you start typing the URL here, you'll notice that the link is already being printed below!



| | |
|---|---|
| hub_url | https://my-hub-url.org |
| repo_url | https://github.com/me/my-repo |
| branch | |
| filepath | |
| app | notebook |

https://my-hub-url.org/hub/user-redirect/git-pull?repo=https%3A%2F%2Fgithub.com%2Fme%2Fmy-repo&app=notebook

5. If your git repository is using a non-default branch name, you can specify that under `branch`. Most people do not need to customize this.
6. If you want to open a specific notebook when the user clicks on the link, specify the path to the notebook under `filepath`. Make sure this file exists, otherwise users will get a 'File not found' error.



This is a great way to distribute materials to students.

| | |
|---|---|
| hub_url | https://my-hub-url.org |
| repo_url | https://github.com/me/my-repo |
| branch | |
| filepath | labs/01/lab01.ipynb |
| app | notebook |

https://my-hub-url.org/hub/user-redirect/git-pull?repo=https%3A%2F%2Fgithub.com%2Fme%2Fmy-repo&subPath=labs%2F01%2Flab01.ipynb&app=notebook

If you do not specify a file path, the user will be shown the directory listing for the repository.

7. By default, notebooks will be opened in the classic Jupyter Notebook interface. You can select `lab` under `application` to open it in the JupyterLab instead.

The link printed at the bottom of the form can be distributed to students now! You can also click it to test that it is working as intended, and adjust the form values until you get something you are happy with.

### Hand-craft your nbgitpuller link

If you'd prefer to hand-craft your `nbgitpuller` link (e.g. if the Binder link above doesn't work), you can use the following pattern:

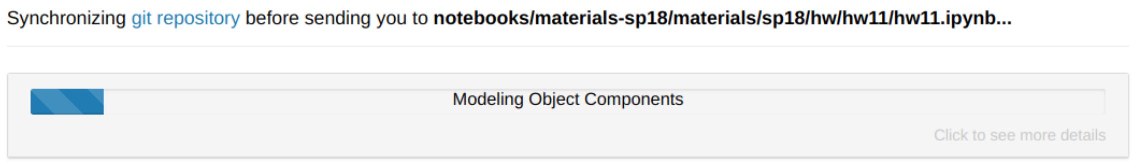
```
http://<my-jhub-address>/hub/user-redirect/git-pull?repo=<your-repo-url>&branch=<your-
↪branch-name>&subPath=<subPath>&app=<notebook | lab>
```

- **repo** is the URL of the git repository you want to clone. This parameter is required.

- **branch** is the branch name to use when cloning from the repository. This parameter is optional and defaults to `master`.
- **subPath** is the path of the directory / notebook inside the repo to launch after cloning. This parameter is optional, and defaults to opening the base directory of the linked Git repository.
- **app** This parameter is optional and defaults to either the environment variable *NBGITPULLER_APP*'s value or *notebook* if it is undefined. The allowed values are *lab* and *notebook*, the value will determine in what application view you end up in.
- **urlPath** will, if specified, override *app* and *subPath* and redirect blindly to the specified path.

### Step 2: Users click on the nbgitpuller link

1. Send the link to your users in some way - email, slack, post a shortened version (with bit.ly maybe) on the wall, or put it on your syllabus page (like UC Berkeley's data8 does). Whatever works for you :)
2. When users click the link, they will be asked to log in to the hub if they have not already.
3. Users will see a progress bar as the git repository is fetched & any automatic merging required is performed.



Synchronizing git repository before sending you to **notebooks/materials-sp18/materials/sp18/hw/hw11/hw11.ipynb...**

Modeling Object Components

4. Users will now be redirected to the notebook specified in the URL!

This workflow lets users land directly in the notebook you specified without having to understand much about git or the JupyterHub interface.

### Advanced: hand-crafting an nbgitpuller link

For information on hand-crafting an `nbgitpuller` link, see the nbgitpuller README.

### Adding data to the JupyterHub

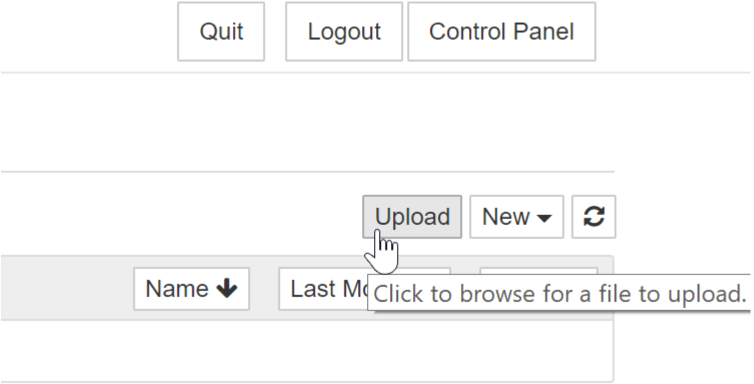
This section covers how to add data to your JupyterHub either from the internet or from your own machine. To learn how to **share data** that is already on your JupyterHub, see *Share data with your users*.

**Note:** When you add data using the methods on this page, you will **only add it to your user directory**. This is not a place that is accessible to others. For information on sharing this data with users on the JupyterHub, see *Share data with your users*.

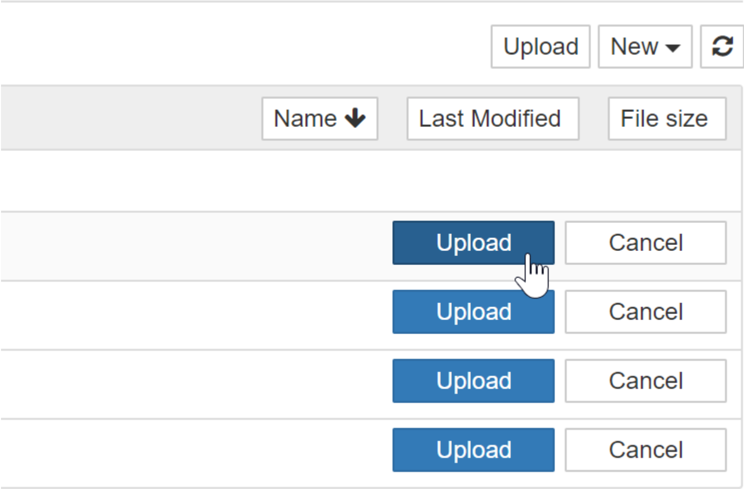
### Adding data from your local machine

The easiest way to add data to your JupyterHub is to use the "Upload" user interface. To do so, follow these steps:

1. First, navigate to the Jupyter Notebook interface home page. You can do this by going to the URL `<my-hub-url>/user/<my-username>/tree`.
2. Click the "Upload" button to open the file chooser window.



3. Choose the file you wish to upload. You may select multiple files if you wish.
4. Click "Upload" for each file that you wish to upload.



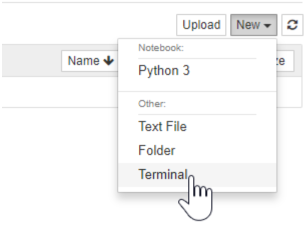
5. Wait for the progress bar to finish for each file. These files will now be on your JupyterHub, your home user's home directory.

To learn how to **share** this data with new users on the JupyterHub, see *Share data with your users*.

### Downloading data from the command line

If the data of interest is on the internet, you may also use code in order to download it to your JupyterHub. There are several ways of doing this, so we'll cover the simplest approach using the unix tool `wget`.

1. Log in to your JupyterHub and open a terminal window.



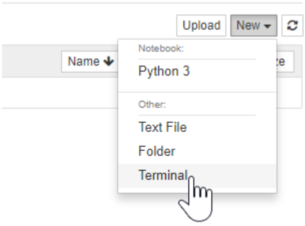
2. Use `wget` to download the file to your current directory in the terminal.

```
wget <MY-FILE-URL>
```

### Example: Downloading the gapminder dataset.

In this example we'll download the gapminder dataset, which contains information about country GDP and live expectancy over time. You can download it from your browser at this link.

1. Log in to your JupyterHub and open a terminal window.



2. Use `wget` to download the gapminder dataset to your current directory in the terminal.

```
wget https://swcarpentry.github.io/python-novice-gapminder/files/python-novice-
↪gapminder-data.zip
```

3. This is a **zip** file, so we'll need to download a unix tool called "unzip" in order to unzip it.

```
sudo apt install unzip
```

4. Finally, unzip the file:

```
unzip python-novice-gapminder-data.zip
```

5. Confirm that your data was unzipped. It could be in a folder called `data/`.

To learn how to **share** this data with new users on the JupyterHub, see *Share data with your users*.

### Share data with your users

There are a few options for sharing data with your users, this page covers a few useful patterns.

### Option 1: Distributing data with *nbgitpuller*

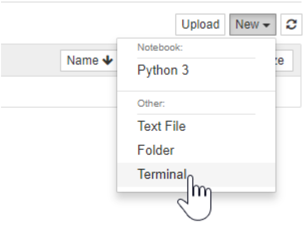
For small datasets, the simplest way to share data with your users is via `nbgitpuller` links. In this case, users click on your link and the dataset contained in the link's target repository is downloaded to the user's home directory. Note that a copy of the dataset will be made for each user.

For information on creating and sharing `nbgitpuller` links, see *Distributing materials to users with nbgitpuller*.

### Option 2: Create a read-only shared folder for data

If your data is large or you don't want copies of it to exist, you can create a read-only shared folder that users have access to. To do this, follow these steps:

1. **Log** in to your JupyterHub as an **administrator user**.
2. **Create a terminal session** with your JupyterHub interface.



3. **Create a folder** where your data will live. We recommend placing shared data in `/srv`. The following command creates two folders (`/srv/data` and `/srv/data/my_shared_data_folder`).

```
sudo mkdir -p /srv/data/my_shared_data_folder
```

4. **Download the data** into this folder. See *Adding data to the JupyterHub* for details on how to do this.
5. All users now have read access to the data in this folder.

### Add a link to the shared folder in the user home directory

Optionally, you may also **create a symbolic link to the shared data folder** that you created above in each **new user's** home directory.

To do this, you can use the server's **user skeleton directory** (`/etc/skel`). Anything that is placed in this directory will also show up in a new user's home directory.

To create a link to the shared folder in the user skeleton directory, follow these steps:

1. `cd` into the skeleton directory:

```
cd /etc/skel
```

2. **Create a symbolic link** to the data folder

```
sudo ln -s /srv/data/my_shared_data_folder my_shared_data_folder
```

3. **Confirm that this worked** by logging in as a new user. You can do this by opening a new "incognito" browser window and accessing your JupyterHub. After you log in as a **new user**, the folder should appear in your new user home directory.

From now on, when a new user account is created, their home directory will have this symbolic link (and any other files in `/etc/skel`) in their home directory. This will have **no effect on the directories of existing users**.

## 3.1.2 The user environment

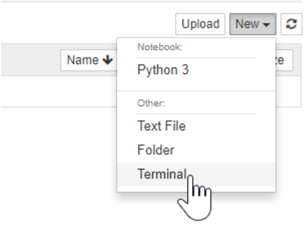
### Install conda, pip or apt packages

TLJH (The Littlest JupyterHub) starts all users in the same conda environment. Packages / libraries installed in this environment are available to all users on the JupyterHub. Users with *admin rights* can install packages easily.

### Installing pip packages

pip is the recommended tool for installing packages in Python from the Python Packaging Index (PyPI). PyPI has almost 145,000 packages in it right now, so a lot of what you need is going to be there!

1. Log in as an admin user and open a Terminal in your Jupyter Notebook.



If you already have a terminal open as an admin user, that should work too!

2. Install a package!

```
sudo -E pip install numpy
```

This installs the `numpy` library from PyPI and makes it available to all users.

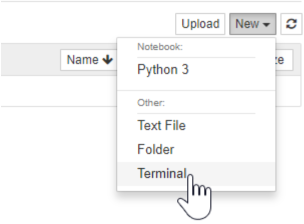
**Note:** If you get an error message like `sudo: pip: command not found`, make sure you are not missing the `-E` parameter after `sudo`.

### Installing conda packages

Conda lets you install new languages (such as new versions of python, node, R, etc) as well as packages in those languages. For lots of scientific software, installing with conda is often simpler & easier than installing with pip - especially if it links to C / Fortran code.

We recommend installing packages from conda-forge, a community maintained repository of conda packages.

1. Log in as an admin user and open a Terminal in your Jupyter Notebook.



If you already have a terminal open as an admin user, that should work too!

2. Install a package!

```
sudo -E conda install -c conda-forge gdal
```

This installs the `gdal` library from `conda-forge` and makes it available to all users. `gdal` is much harder to install with pip.

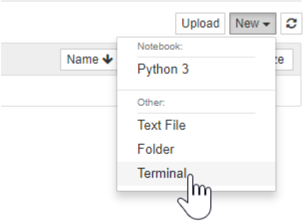
**Note:** If you get an error message like `sudo: conda: command not found`, make sure you are not missing the `-E` parameter after `sudo`.

### Installing apt packages

apt is the official package manager for the Ubuntu Linux distribution. You can install utilities (such as `vim`, `sl`, `htop`, etc), servers (`postgres`, `mysql`, `nginx`, etc) and a lot more languages than present in `conda` (`haskell`, `prolog`, `INTERCAL`). Some third party software (such as RStudio) is distributed as `.deb` files, which are the files `apt` uses to install software.

You can search for packages with Ubuntu Package search - make sure to look in the version of Ubuntu you are using!

1. Log in as an admin user and open a Terminal in your Jupyter Notebook.



If you already have a terminal open as an admin user, that should work too!

2. Update list of packages available. This makes sure you get the latest version of the packages possible from the repositories.

```
sudo apt update
```

3. Install the packages you want.

```
sudo apt install mysql-server git
```

This installs (and starts) a `MySQL <https://www.mysql.com/>` database server and `git`.

### User environment location

The user environment is a conda environment set up in `/opt/tljh/user`, with a Python3 kernel as the default. It is readable by all users, but writeable only by users who have root access. This makes it possible for JupyterHub admins (who have root access with `sudo`) to install software in the user environment easily.

### Accessing user environment outside JupyterHub

We add `/opt/tljh/user/bin` to the `$PATH` environment variable for all JupyterHub users, so everything installed in the user environment is available to them automatically. If you are using `ssh` to access your server instead, you can get access to the same environment with:

```
export PATH=/opt/tljh/user/bin:${PATH}
```

Whenever you run any command now, the user environment will be searched first before your system environment is. So if you run `python3 <somefile>`, it'll use the `python3` installed in the user environment (`/opt/tljh/user/bin/python3`) rather than the `python3` installed in your system environment (`/usr/bin/python3`). This is usually what you want!

To make this change 'stick', you can add the line to the end of the `.bashrc` file in your home directory.

When using `sudo`, the `PATH` environment variable is usually reset, for security reasons. This leads to error messages like:

```
$ sudo conda install -c conda-forge gdal
sudo: conda: command not found
```

The most common & portable way to fix this when using `ssh` is:

```
sudo PATH=${PATH} conda install -c conda-forge gdal
```

### Upgrade to a newer Python version

All new TLJH installs use miniconda 4.7.10, which comes with a Python 3.7 environment for the users. The previously TLJH installs came with miniconda 4.5.4, which meant a Python 3.6 environment.

To upgrade the Python version of the user environment, one can:

- **Start fresh on a machine that doesn't have TLJH already installed.**

  See the *installation guide* section about how to install TLJH.

- **Upgrade Python manually.**

  Because upgrading Python for existing installs can break packages alaredy installed under the old Python, upgrading your current TLJH installation, will NOT upgrade the Python version of the user environment, but you may do so manually.

  **Steps:**

  1. Activate the user environment, if using ssh. If the terminal was started with JupyterHub, this step can be skipped:

```
source /opt/tljh/user/bin/activate
```

2. Get the list of currently installed pip packages (so you can later install them under the new Python):

```
pip freeze > pip_pkgs.txt
```

3. Update all conda installed packages in the environment:

```
sudo PATH=${PATH} conda update --all
```

4. Update Python version:

```
sudo PATH=${PATH} conda install python=3.7
```

5. Install the pip packages previously saved:

```
pip install -r pip_pkgs.txt
```

### Change default User Interface for users

By default, logging into TLJH puts you in the classic Jupyter Notebook interface we all know and love. However, there are at least two other popular notebook interfaces you can use:

1. JupyterLab
2. nteract

Both these interfaces are also shipped with tljh by default. You can try them temporarily, or set them to be the default interface whenever you login.

#### Trying an alternate interface temporarily

When you log in & start your server, by default the URL in your browser will be something like `/user/<username>/tree`. The `/tree` is what tells the notebook server to give you the classic notebook interface.

- **For the JupyterLab interface**: change `/tree` to `/lab`.
- **For the nteract interface**: change `/tree` to `/nteract`

You can play around with them and see what fits your use cases best.

#### Changing the default user interface

You can change the default interface users get when they log in by modifying `config.yaml` as an admin user.

1. To launch **JupyterLab** when users log in, run the following in an admin console:

```
sudo tljh-config set user_environment.default_app jupyterlab
```

2. Alternatively, to launch **nteract** when users log in, run the following in the admin console:

```
sudo tljh-config set user_environment.default_app nteract
```

3. Apply the changes by restarting JupyterHub. This should not disrupt current users.

```
sudo tljh-config reload hub
```

If this causes problems, check the *JupyterHub Logs* for clues on what went wrong.

Users might have to restart their servers from control panel to get the new interface.

### Configure resources available to users

To configure the resources that are available to your users (such as RAM, CPU and Disk Space), see the section *User Server Limits*. For information on **resizing** the environment available to users *after* you've created your JupyterHub, see *Resize the resources available to your JupyterHub*.

## 3.1.3 Authentication

We have a special set of How-To Guides on using various forms of authentication with your JupyterHub. For more information on Authentication, see *Configuring JupyterHub authenticators*

### Authenticate *any* user with a single shared password

The **Dummy Authenticator** lets *any* user log in with the given password. This authenticator is **extremely insecure**, so do not use it if you can avoid it.

### Enabling the authenticator

1. Always use DummyAuthenticator with a password. You can communicate this password to all your users via an out of band mechanism (like writing on a whiteboard). Once you have selected a password, configure TLJH to use the password by executing the following from an admin console.

   ```
   sudo tljh-config set auth.DummyAuthenticator.password <password>
   ```

   Remember to replace `<password>` with the password you choose.

2. Enable the authenticator and reload config to apply configuration:

   ```
   sudo tljh-config set auth.type dummyauthenticator.DummyAuthenticator
   ```

   ```
   sudo tljh-config reload
   ```

Users who are currently logged in will continue to be logged in. When they log out and try to log back in, they will be asked to provide a username and password.

### Changing the password

The password used by DummyAuthenticator can be changed with the following commands:

```
tljh-config set auth.DummyAuthenticator.password <new-password>
```

```
tljh-config reload
```

### Authenticate using GitHub Usernames

The **GitHub Authenticator** lets users log into your JupyterHub using their GitHub user ID / password. To do so, you'll first need to register an application with GitHub, and then provide information about this application to your `tljh` configuration.

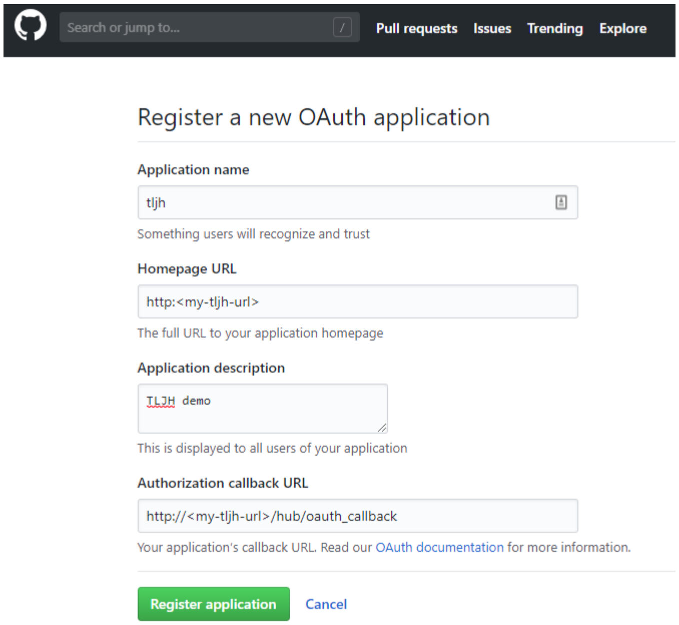
**Note:** You'll need a GitHub account in order to complete these steps.

### Step 1: Create a GitHub application

1. Go to the GitHub OAuth app creation page.
   - **Application name**: Choose a descriptive application name (e.g. `tljh`)
   - **Homepage URL**: Use the IP address or URL of your JupyterHub. e.g. `http(s)://<my-tljh-url>`.
   - **Application description**: Use any description that you like.
   - **Authorization callback URL**: Insert text with the following form:

   ```
   http(s)://<my-tljh-ip-address>/hub/oauth_callback
   ```

   - When you're done filling in the page, it should look something like this:



Register a new OAuth application

Application name

tljh

Something users will recognize and trust

Homepage URL

http:<my-tljh-url>

The full URL to your application homepage

Application description

TLJH demo

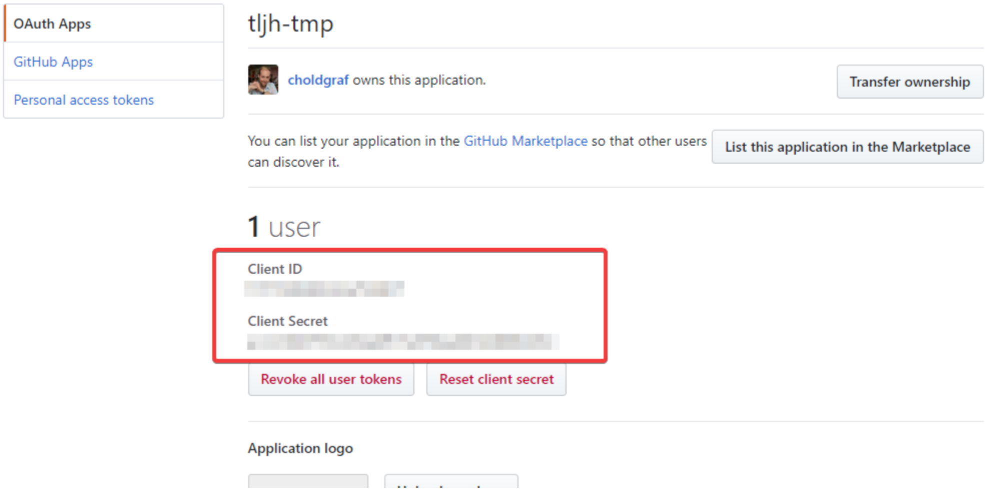
This is displayed to all users of your application

Authorization callback URL

http://<my-tljh-url>/hub/oauth_callback

Your application's callback URL. Read our OAuth documentation for more information.

2. Click "Register application". You'll be taken to a page with the registered application details.
3. Copy the **Client ID** and **Client Secret** from the application details page. You will use these later to configure your JupyterHub authenticator.



tljh-tmp

choldgraf owns this application.

You can list your application in the GitHub Marketplace so that other users can discover it.

1 user

Client ID
[redacted]

Client Secret
[redacted]

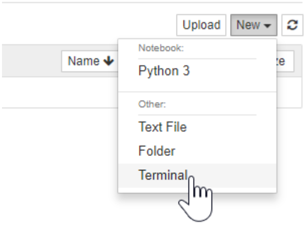
Application logo

**Important:** If you are using a virtual machine from a cloud provider and **stop the VM**, then when you re-start the VM, the provider will likely assign a **new public IP address** to it. In this case, **you must update your GitHub application information** with the new IP address.

### Configure your JupyterHub to use the GitHub Oauthenticator

We'll use the `tljh-config` tool to configure your JupyterHub's authentication. For more information on `tljh-config`, see *Configuring TLJH with tljh-config*.

1. Log in as an administrator account to your JupyterHub.
2. Open a terminal window.



3. Configure the GitHub OAuthenticator to use your client ID, client secret and callback URL with the following commands:

```
sudo tljh-config set auth.GitHubOAuthenticator.client_id '<my-tljh-client-id>'
```

```
sudo tljh-config set auth.GitHubOAuthenticator.client_secret '<my-tljh-client-secret>'
```

```
sudo tljh-config set auth.GitHubOAuthenticator.oauth_callback_url 'http(s)://<my-tljh-ip-address>/hub/oauth_callback'
```

4. Tell your JupyterHub to *use* the GitHub OAuthenticator for authentication:

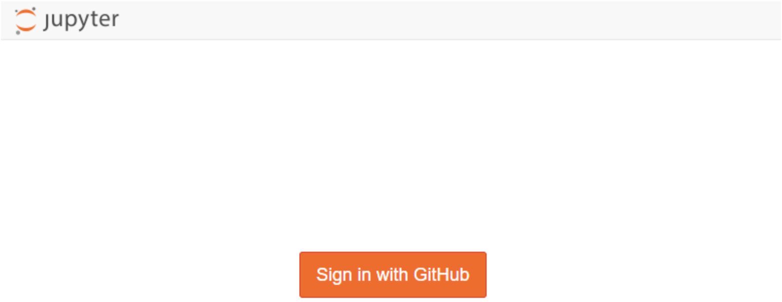
```
sudo tljh-config set auth.type oauthenticator.github.GitHubOAuthenticator
```

5. Restart your JupyterHub so that new users see these changes:

```
sudo tljh-config reload
```

### Confirm that the new authenticator works

1. **Open an incognito window** in your browser (do not log out until you confirm that the new authentication method works!)
2. Go to your JupyterHub URL.
3. You should see a GitHub login button like below:



4. After you log in with your GitHub credentials, you should be directed to the Jupyter interface used in this JupyterHub.
5. **If this does not work** you can revert back to the default JupyterHub authenticator by following the steps in *Let users choose a password when they first log in*.

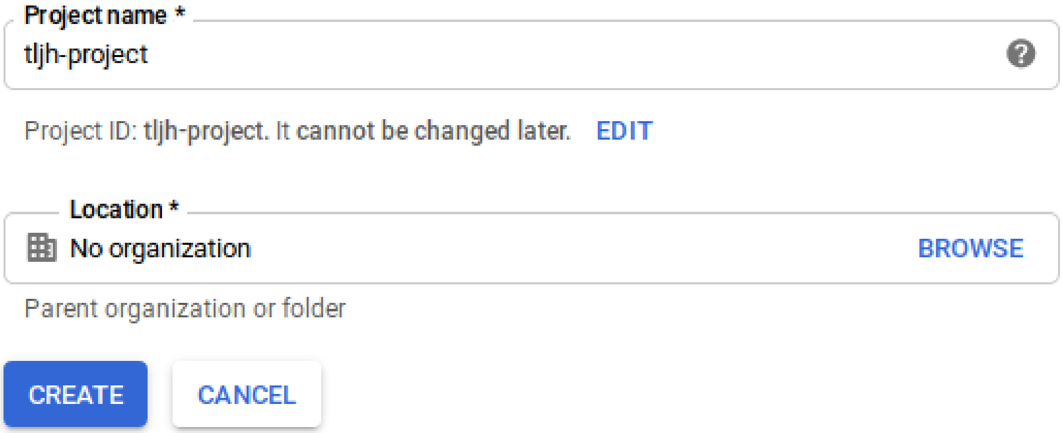
### Authenticate using Google

The **Google Authenticator** lets users log into your JupyterHub using their Google user ID / password. To do so, you'll first need to register an application with Google, and then provide information about this application to your `tljh` configuration. See Google's documentation on how to create OAUth 2.0 client credentials.

**Note:** You'll need a Google account in order to complete these steps.

### Step 1: Create a Google project

Go to Google Developers Console and create a new project:



Project name *
tljh-project

Project ID: tljh-project. It cannot be changed later.

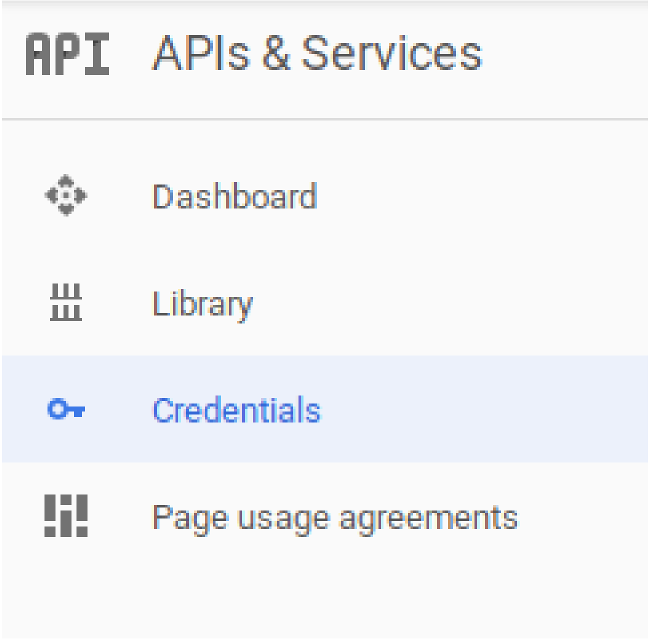
Location *
No organization

Parent organization or folder

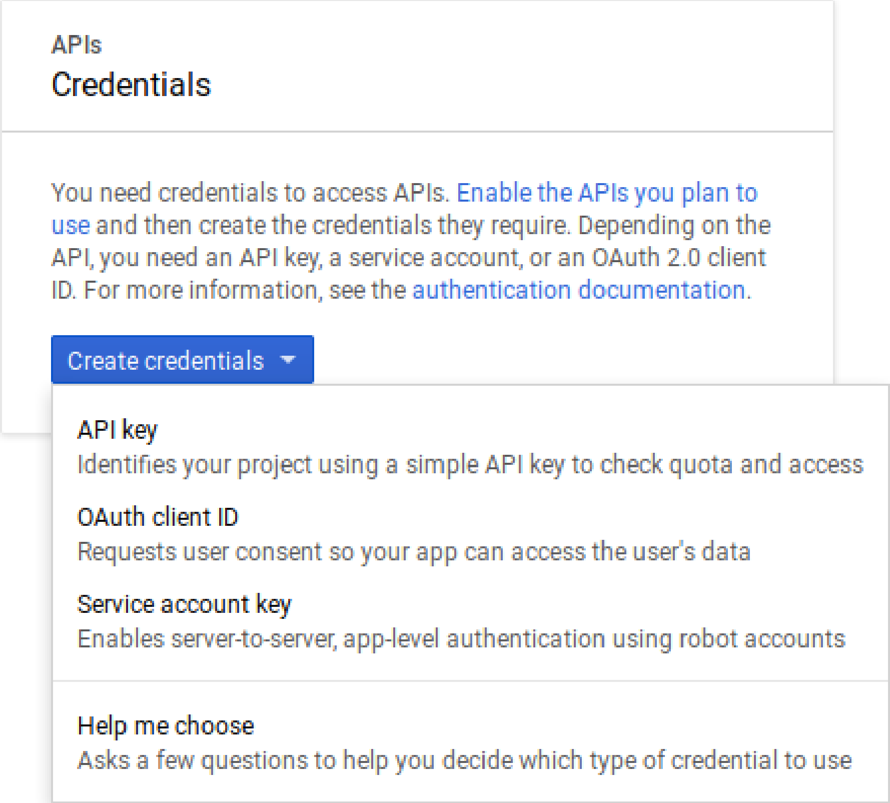
### Step 2: Set up a Google OAuth client ID and secret

1. After creating and selecting the project:

    - Go to the credentials menu:



    - Click "Create credentials" and from the dropdown menu select **"OAuth client ID"**:



APIs

Credentials

You need credentials to access APIs. Enable the APIs you plan to use and then create the credentials they require. Depending on the API, you need an API key, a service account, or an OAuth 2.0 client ID. For more information, see the authentication documentation.

Create credentials ▾

API key
Identifies your project using a simple API key to check quota and access

OAuth client ID
Requests user consent so your app can access the user's data

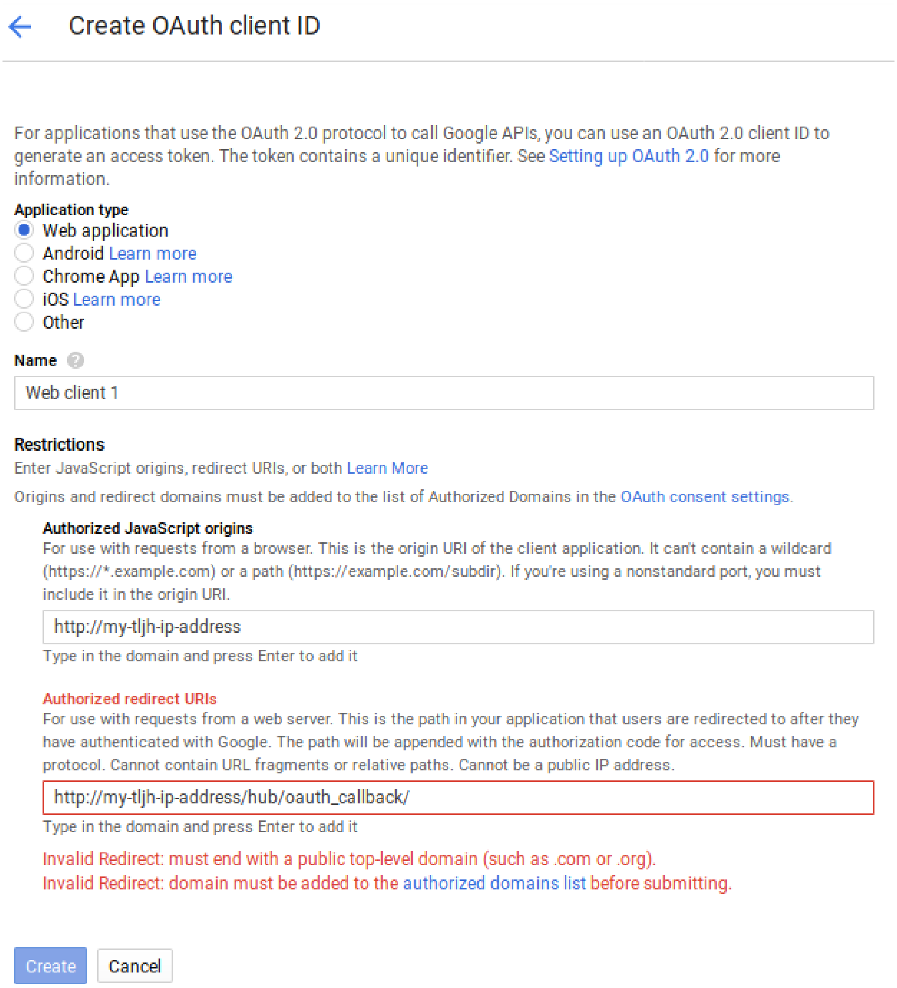
Service account key
Enables server-to-server, app-level authentication using robot accounts

Help me choose
Asks a few questions to help you decide which type of credential to use

- **You will have to fill a form with:**
  - **Application type**: Choose *Web application*
  - **Name**: A descriptive name for your OAuth client ID (e.g. `tljh-client`)
  - **Authorized JavaScript origins**: Use the IP address or URL of your JupyterHub. e.g. `http(s)://<my-tljh-url>`.
  - **Authorized redirect URIs**: Insert text with the following form:

    ```
    http(s)://<my-tljh-ip-address>/hub/oauth_callback
    ```

  - When you're done filling in the page, it should look something like this (ideally without the red warnings):



Create OAuth client ID

For applications that use the OAuth 2.0 protocol to call Google APIs, you can use an OAuth 2.0 client ID to generate an access token. The token contains a unique identifier. See Setting up OAuth 2.0 for more information.

**Application type**

- ● Web application
- ○ Android Learn more
- ○ Chrome App Learn more
- ○ iOS Learn more
- ○ Other

**Name**

Web client 1

**Restrictions**

Enter JavaScript origins, redirect URIs, or both Learn More

Origins and redirect domains must be added to the list of Authorized Domains in the OAuth consent settings.

**Authorized JavaScript origins**

For use with requests from a browser. This is the origin URI of the client application. It can't contain a wildcard (https://*.example.com) or a path (https://example.com/subdir). If you're using a nonstandard port, you must include it in the origin URI.

http://my-tljh-ip-address

Type in the domain and press Enter to add it

**Authorized redirect URIs**

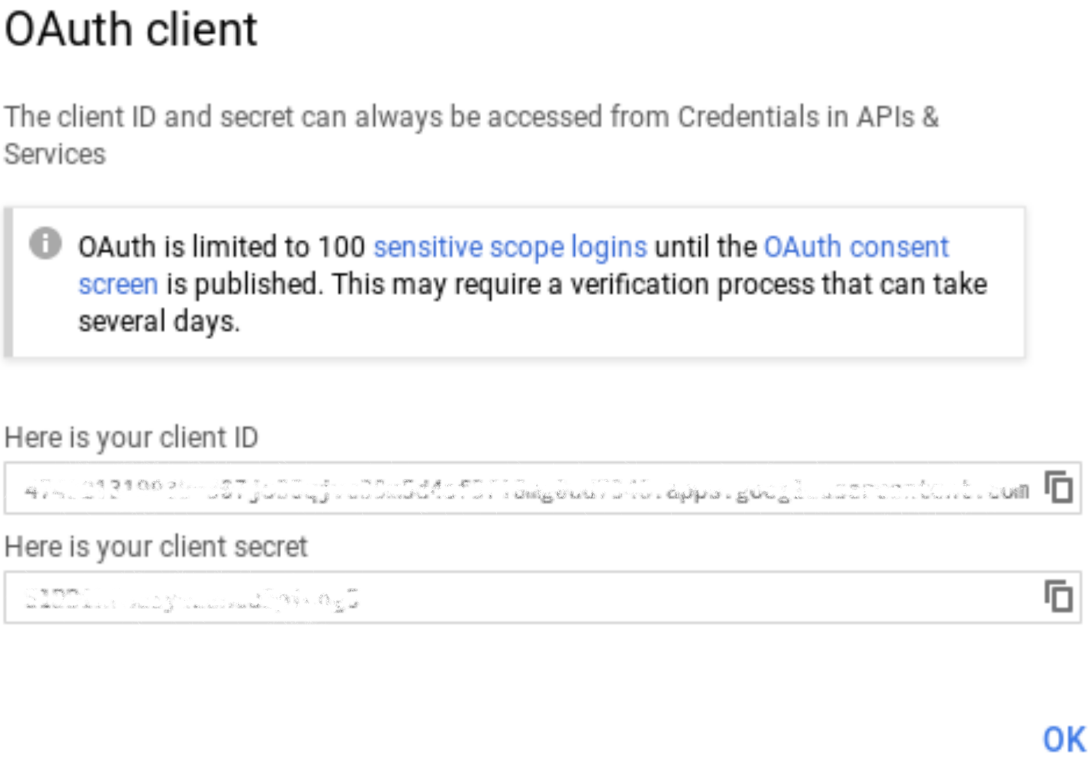
For use with requests from a web server. This is the path in your application that users are redirected to after they have authenticated with Google. The path will be appended with the authorization code for access. Must have a protocol. Cannot contain URL fragments or relative paths. Cannot be a public IP address.

http://my-tljh-ip-address/hub/oauth_callback/

Type in the domain and press Enter to add it

Invalid Redirect: must end with a public top-level domain (such as .com or .org).
Invalid Redirect: domain must be added to the authorized domains list before submitting.

2. Click "Create". You'll be taken to a page with the registered application details.
3. Copy the **Client ID** and **Client Secret** from the application details page. You will use these later to configure your JupyterHub authenticator.



OAuth client

The client ID and secret can always be accessed from Credentials in APIs & Services

OAuth is limited to 100 sensitive scope logins until the OAuth consent screen is published. This may require a verification process that can take several days.

Here is your client ID

[redacted]

Here is your client secret

[redacted]

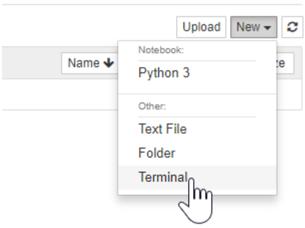
OK

**Important:** If you are using a virtual machine from a cloud provider and **stop the VM**, then when you re-start the VM, the provider will likely assign a **new public IP address** to it. In this case, **you must update your Google application information** with the new IP address.

### Configure your JupyterHub to use the Google Oauthenticator

We'll use the `tljh-config` tool to configure your JupyterHub's authentication. For more information on `tljh-config`, see *Configuring TLJH with tljh-config*.

1. Log in as an administrator account to your JupyterHub.
2. Open a terminal window.



3. Configure the Google OAuthenticator to use your client ID, client secret and callback URL with the following commands:

```
sudo tljh-config set auth.GoogleOAuthenticator.client_id '<my-tljh-client-id>'
```

```
sudo tljh-config set auth.GoogleOAuthenticator.client_secret '<my-tljh-client-
↪secret>'
```

```
sudo tljh-config set auth.GoogleOAuthenticator.oauth_callback_url 'http(s)://<my-
↪tljh-ip-address>/hub/oauth_callback'
```

4. Tell your JupyterHub to *use* the Google OAuthenticator for authentication:

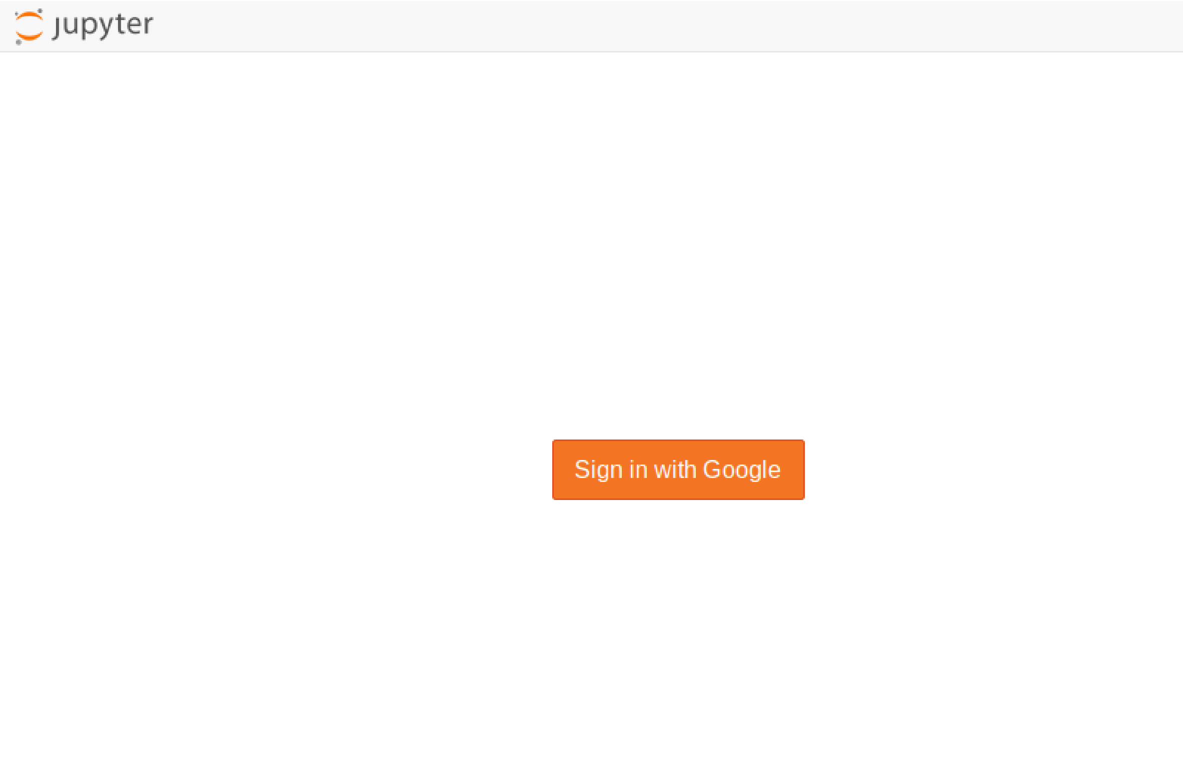
```
sudo tljh-config set auth.type oauthenticator.google.GoogleOAuthenticator
```

5. Restart your JupyterHub so that new users see these changes:

```
sudo tljh-config reload
```

### Confirm that the new authenticator works

1. **Open an incognito window** in your browser (do not log out until you confirm that the new authentication method works!)
2. Go to your JupyterHub URL.
3. You should see a Google login button like below:



4. After you log in with your Google credentials, you should be directed to the Jupyter interface used in this JupyterHub.
5. **If this does not work** you can revert back to the default JupyterHub authenticator by following the steps in *Let users choose a password when they first log in*.

### Authenticate using AWS Cognito

The **AWS Cognito Authenticator** lets users log into your JupyterHub using cognito user pools. To do so, you'll first need to register and configure a cognito user pool and app, and then provide information about this application to your `tljh` configuration.

### Create an AWS Cognito application

1. Create a user pool Getting Started with User Pool.

   When you have completed creating a user pool, app, and domain you should have the following settings available to you:

   - **App client id**: From the App client page
   - **App client secret** From the App client page
   - **Callback URL** This should be the domain you are hosting you server on:

     ```
     http(s)://<my-tljh-ip-address>/hub/oauth_callback
     ```

   - **Signout URL**: This is the landing page for a user when they are not logged on:

     ```
     http(s)://<my-tljh-ip-address>
     ```

   - **Auth Domain** Create an auth domain e.g. <my_jupyter_hub>:

     ```
     https://<<my_jupyter_hub>.auth.eu-west-1.amazoncognito.com
     ```

### Install and configure an AWS EC2 Instance with userdata

By adding following script to the ec2 instance user data you should be able to configure the instance automatically, replace relevant config variables:

```
#!/bin/bash
##############################################
# Setup systemd environment variable overrides
##############################################
mkdir /etc/systemd/system/jupyterhub.service.d

echo "[Service]
Environment=AWSCOGNITO_DOMAIN=${awscognito_domain}" >> /etc/systemd/system/jupyterhub.
↪service.d/jupyterhub.conf

##############################################
# Need to ensure oauthenticator is bumped to 0.10.0
##############################################
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-jupyterhub/master/
↪bootstrap/bootstrap.py \
  | sudo python3 - \
    --admin insightadmin

##############################################
# Setup aws Cognito Authenticator
##############################################
```

(continues on next page)

(continued from previous page)

```
echo "c.AWSCognitoAuthenticator.client_id='${client_id}'
c.AWSCognitoAuthenticator.client_secret='${client_secret}'
c.AWSCognitoAuthenticator.oauth_callback_url='${callback_url}'
c.AWSCognitoAuthenticator.username_key='username'
c.AWSCognitoAuthenticator.oauth_logout_redirect_url='${logout_url}'" >> /opt/tljh/
↪config/jupyterhub_config.d/awscognito.py


tljh-config set auth.type oauthenticator.awscognito.AWSCognitoAuthenticator

tljh-config reload
```

### Manual configuration to use the AWS Cognito Oauthenticator

Assuming tljh has already been installed, we need to make sure the oautheneticator module is at 0.10.0 and if not do a pip install oauthenticator>=0.10.0

Because the AWS Congito authenticator uses environment variables and the systemd script we need to pass the the AWS Cognito domain in via systemd we can do this by creating a systemd service overide file:

```
/etc/systemd/system/jupyterhub.service.d/jupyterhub.conf
```

and add the following:

```
[Service]
Environment=AWSCOGNITO_DOMAIN=https://<<my_jupyter_hub>.auth.eu-west-1.amazoncognito.
↪com
```

Using your prefered editor create the config file:

```
/opt/tljh/config/jupyterhub_config.d/awscognito.py
```

subsituting the relevant variables:

```
c.AWSCognitoAuthenticator.client_id='${client_id}'
c.AWSCognitoAuthenticator.client_secret='${client_secret}'
c.AWSCognitoAuthenticator.oauth_callback_url='${callback_url}'
c.AWSCognitoAuthenticator.username_key='username'
c.AWSCognitoAuthenticator.oauth_logout_redirect_url='${logout_url}'
```

We'll use the `tljh-config` tool to configure your JupyterHub's authentication. For more information on `tljh-config`, see *Configuring TLJH with tljh-config*.

1. Tell your JupyterHub to *use* the AWS Cognito OAuthenticator for authentication:

   ```
   tljh-config set auth.type oauthenticator.awscognito.AWSCognitoAuthenticator
   ```

2. Restart your JupyterHub so that new users see these changes:

   ```
   sudo tljh-config reload
   ```

### Confirm that the new authenticator works

1. **Open an incognito window** in your browser (do not log out until you confirm that the new authentication method works!)
2. Go to your JupyterHub URL.
3. You should see an AWS Cognito login button:
4. You will likely have to create a new user (sign up) and then you should be directed to the Jupyter interface used in this JupyterHub.
5. **If this does not work** you can revert back to the default JupyterHub authenticator by following the steps in *Let users choose a password when they first log in*.

### Let users choose a password when they first log in

The **First Use Authenticator** lets users choose their own password. Upon their first log-in attempt, whatever password they use will be stored as their password for subsequent log in attempts. This is the default authenticator that ships with TLJH.

### Enabling the authenticator

**Note:** the FirstUseAuthenticator is enabled by default in TLJH.

1. Enable the authenticator and reload config to apply the configuration:

   sudo tljh-config set auth.type firstuseauthenticator.FirstUseAuthenticator sudo tljh-config reload

Users who are currently logged in will continue to be logged in. When they log out and try to log back in, they will be asked to provide a username and password.

### Allowing anyone to log in to your JupyterHub

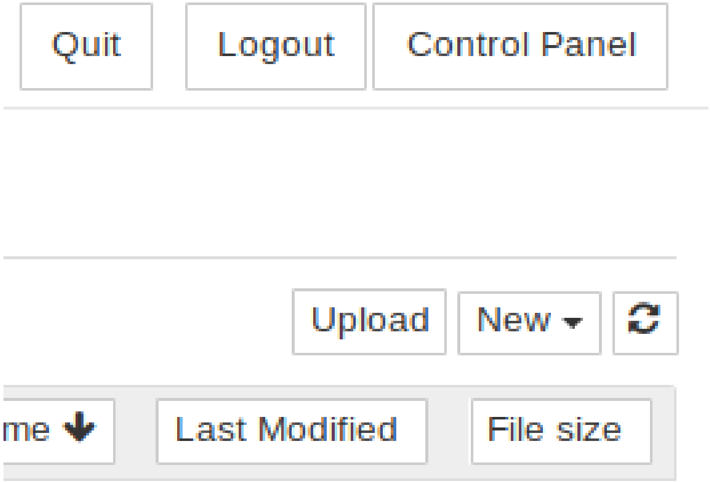
By default, you need to manually create user accounts before they will be able to log in to your JupyterHub. If you wish to allow **any** user to access the JupyterHub, run the following command.

```
tljh-config set auth.FirstUseAuthenticator.create_users true
tljh-config reload
```

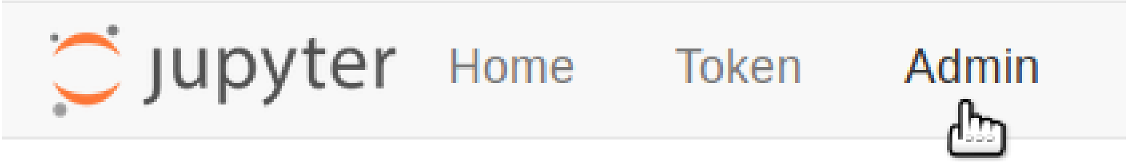
### Resetting user password

You can reset user passwords by *deleting* the user from the JupyterHub admin page. This logs the user out, but does **not** remove any of their data or home directories. The user can then set a new password by logging in again with their new password.

1. As an admin user, open the **Control Panel** by clicking the control panel button on the top right of your JupyterHub.

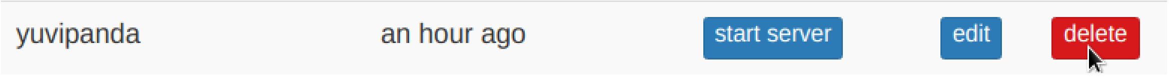


2. In the control panel, open the **Admin** link in the top left.

   This opens up the JupyterHub admin page, where you can add / delete users, start / stop peoples' servers and see who is online.

3. **Delete** the user whose password needs resetting. Remember this **does not** delete their data or home directory.

   If there is a confirmation dialog, confirm the deletion. This will also log the user out if they were currently running.

4. Ask the user to log in again with their new password as usual. This will be their new password going forward.

### Let users sign up with a username and password

The **Native Authenticator** lets users signup for creating a new username and password. When they signup, they won't be able to login until they are authorized by an admin. Users that are characterized as admin have to signup as well, but they will be authorized automatically.

### Enabling the authenticator

Enable the authenticator and reload config to apply the configuration:

```
sudo tljh-config set auth.type nativeauthenticator.NativeAuthenticator
sudo tljh-config reload
```

### Allowing all users to be authorized after signup

By default, all users created on signup don't have authorization to login. If you wish to allow **any** user to access the JupyterHub just after the signup, run the following command:

```
tljh-config set auth.NativeAuthenticator.open_signup true
tljh-config reload
```

### Optional features

More optional features are available on the *authenticator documentation <https://native-authenticator.readthedocs.io/en/latest/>*

## 3.1.4 Administration and security

### Add / Remove admin users

Admin users in TLJH have the following powers:

1. Full root access to the server with passwordless `sudo`. This lets them do literally whatever they want in the server
2. Access servers / home directories of all other users
3. Install new packages for everyone with `conda`, `pip` or `apt`
4. Change configuration of TLJH

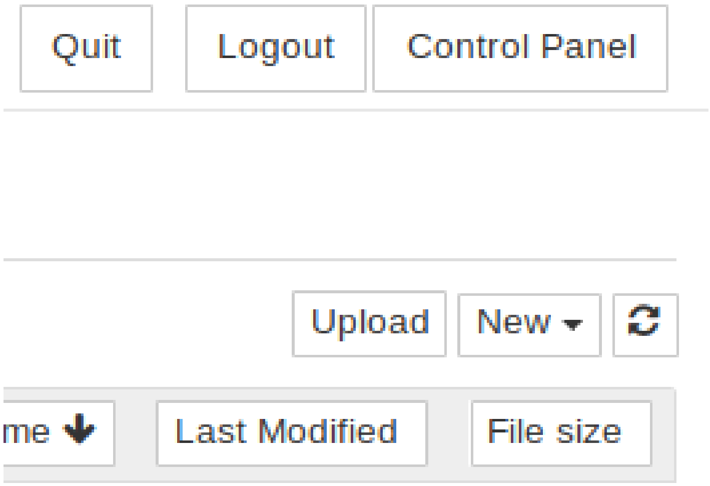
This is a lot of power, so make sure you know who you are giving it to. Admin users should have decent passwords / secure login mechanisms, so attackers can not easily gain control of the system.

**Important:** You should make sure an admin user is present when you **install** TLJH the very first time. It is recommended that you also set a password for the admin at this step. The *–admin* flag passed to the installer does this. If you had forgotten to do so, the easiest way to fix this is to run the installer again.

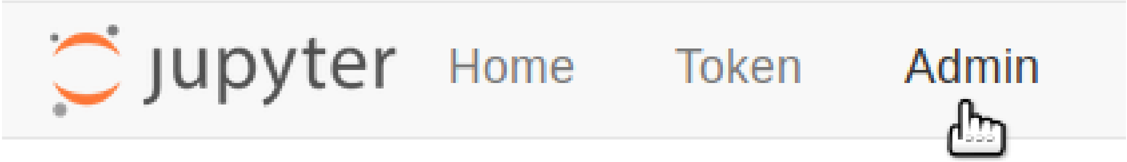
### Adding admin users from the JupyterHub interface

There are two primary user interfaces for doing work on your JupyterHub. By default, this is the Notebook Interface, and will be used in this section. If you are using JupyterLab, you can access the Notebook Interface by replacing `/lab` with `/tree` in your URL.

1. First, navigate to the Jupyter Notebook interface home page. You can do this by going to the URL `<my-hub-url>/user/<my-username>/tree`.
2. Open the **Control Panel** by clicking the control panel button on the top right of your JupyterHub.

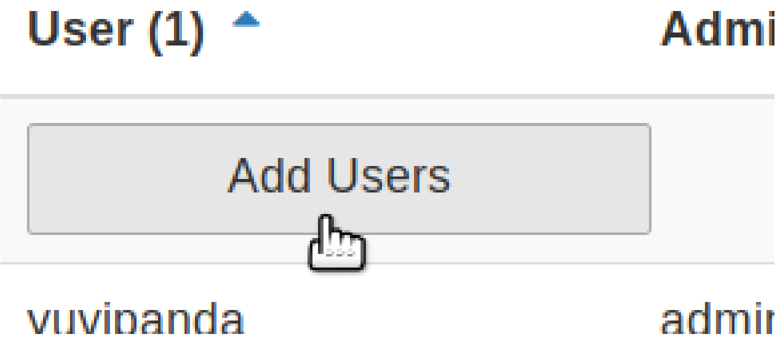


3. In the control panel, open the **Admin** link in the top left.



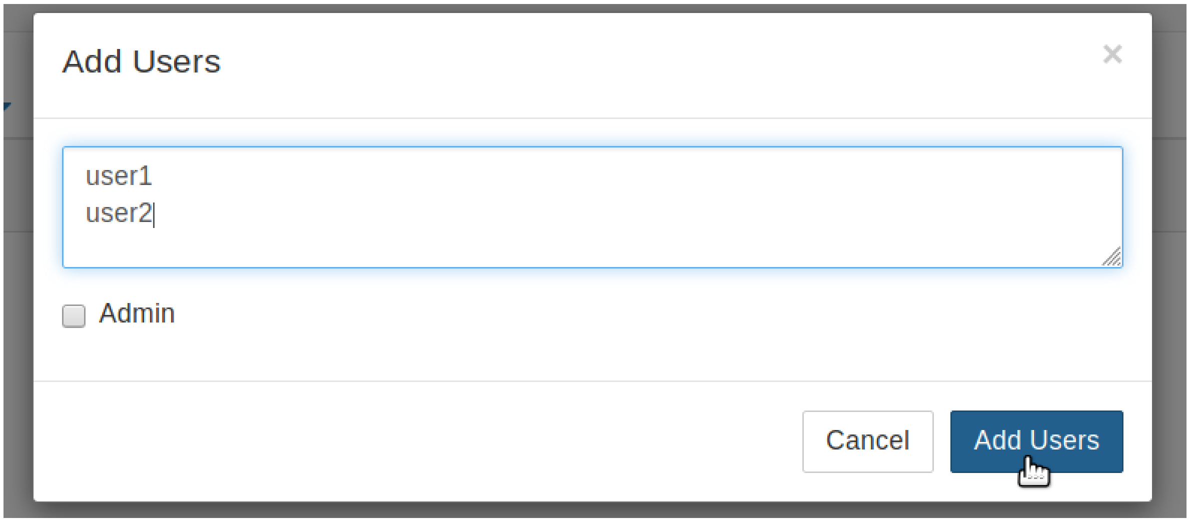
This opens up the JupyterHub admin page, where you can add / delete users, start / stop peoples' servers and see who is online.

4. Click the **Add Users** button.



A **Add Users** dialog box opens up.

5. Type the names of users you want to add to this JupyterHub in the dialog box, one per line. **Make sure to tick the Admin checkbox**.



6. Click the **Add Users** button in the dialog box. Your users are now added to the JupyterHub with administrator privileges!

### Adding admin users from the command line

Sometimes it is easier to add or remove admin users from the command line (for example, if you forgot to add an admin user when first setting up your JupyterHub).

#### Adding new admin users

New admin users can be added by executing the following commands on an admin terminal:

```
sudo tljh-config add-item users.admin <username>
sudo tljh-config reload
```

If the user is already using the JupyterHub, they might have to stop and start their server from the control panel to gain new powers.

#### Removing admin users

You can remove an existing admin user by executing the following commands in an admin terminal:

```
sudo tljh-config remove-item users.admin <username>
sudo tljh-config reload
```

If the user is already using the JupyterHub, they will continue to have some of their admin powers until their server is stopped. Another admin can force their server to stop by clicking 'Stop Server' in the admin panel.

### Estimate Memory / CPU / Disk needed

This page helps you estimate how much Memory / CPU / Disk the server you install The Littlest JupyterHub on should have. These are just guidelines to help with estimation - your actual needs will vary.

### Memory

Memory is usually the biggest determinant of server size in most JupyterHub installations. At minimum, your server must have at least **1.15GB** of RAM for TLJH to install.

$$Recommended\,Memory = (Max\,concurrent\,users \times Max\,mem\,per\,user) + 128MB$$

The `128MB` is overhead for TLJH and related services. **Server Memory Recommended** is the amount of Memory (RAM) the server you acquire should have - we recommend erring on the side of 'more Memory'. The other terms are explained below.

### Maximum concurrent users

Even if your class has 100 students, most of them will not be using the JupyterHub actively at a single given moment. At 2am on a normal night, maybe you'll have 10 students using it. At 2am before a final, maybe you'll have 60 students using it. Maybe you'll have a lab session with all 100 of your students using it at the same time.

The *maximum* number of users actively using the JupyterHub at any given time determines how much memory your server will need. You'll get better at estimating this number over time. We generally recommend between 40-60% of your total class size to start with.

### Maximum memory allowed per user

Depending on what kind of work your users are doing, they will use different amounts of memory. The easiest way to determine this is to run through a typical user workflow yourself, and measure how much memory is used. You can use *Check your memory usage* to determine how much memory your user is using.

A good rule of thumb is to take the maximum amount of memory you used during your session, and add 20-40% headroom for users to 'play around'. This is the maximum amount of memory that should be given to each user.

If users use *more* than this alloted amount of memory, their notebook kernel will *restart*.

### CPU

CPU estimation is more forgiving than Memory estimation. If there isn't enough CPU for your users, their computation becomes very slow - but does not stop, unlike with RAM.

$$Recommended\,CPU = (Max\,concurrent\,users \times Max\,CPU\,usage\,per\,user) + 20\%$$

The `20%` is overhead for TLJH and related services. This is around 20% of a single modern CPU. This, of course, is just an estimate. We recommend using the same process used to estimate Memory required for estimating CPU required. You cannot use nbresuse for this, but you should carry out normal workflow and investigate the CPU usage on the machine.

### Disk space

Unlike Memory & CPU, disk space is predicated on **total** number of users, rather than **maximum concurrent** users.

$$Recommended\,Disk\,Size = (Total\,users \times Max\,disk\,usage\,per\,user) + 2GB$$

### Resizing your server

Lots of cloud providers let your dynamically resize your server if you need it to be larger or smaller. Usually this requires a restart of the whole server - active users will be logged out, but otherwise usually nothing bad happens.

### Resize the resources available to your JupyterHub

As you are using your JupyterHub, you may need to increase or decrease the amount of resources allocated to your TLJH install. The kinds of resources that can be allocated, as well as the process to do so, will depend on the provider / interface for your VM. We recommend consulting the installation page for your provider for more information. This page covers the steps your should take on your JupyterHub *after* you've reallocated resources on the cloud provider of your choice.

Currently there are instructions to resize your resources on the following providers:

- *Digital Ocean*.

Once resources have been reallocated, you must tell TLJH to make use of these resources, and verify that the resources have become available.

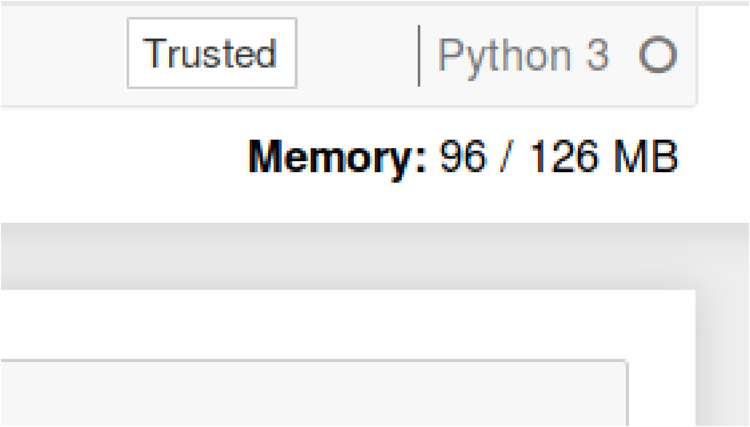
### Verifying a Resize

1. Once you have resized your server, tell the JupyterHub to make use of these new resources. To accomplish this, follow the instructions in *Configuring TLJH with tljh-config* to set new memory or CPU limits and reload the hub. This can be completed using the terminal in the JupyterHub (or via SSH-ing into your VM and using this terminal).
2. TLJH configuration options can be verified by viewing the tljh-config output.

```
sudo tljh-config show
```

   Double-check that your changes are reflected in the output.

3. **To verify changes to memory**, confirm that it worked by starting a new server (if you had one previously running, click "Control Panel -> Stop My Server" to shut down your active server first), opening a notebook, and checking the value of the nbresuse extension in the upper-right.



Memory: 96 / 126 MB

4. **To verify changes to CPU**, use the `nproc` from a terminal. This command displays the number of available cores, and should be equal to the number of cores you selected in your provider's interface.

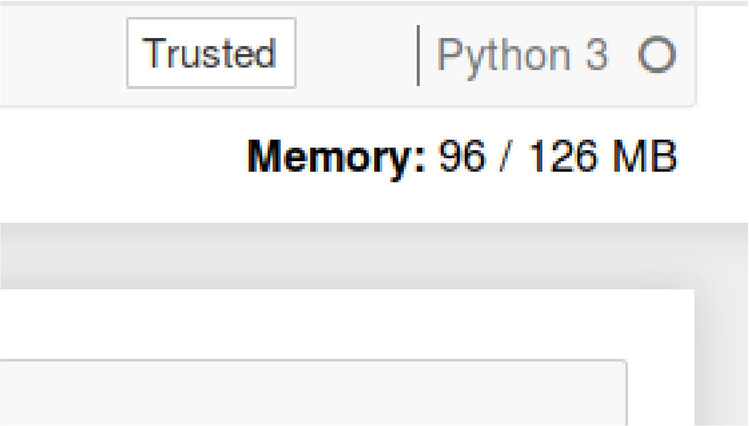
```
nproc --all
```

5. **To verify currently-available disk space**, use the `df` command in a terminal. This shows how much disk space is available. The `-hT` argument allows us to have this printed in a human readable format, and condenses the output to show one storage volume. Note that currently you cannot change the disk space on a per-user basis.

```
df -hT /home
```

### Check your memory usage

The nbresuse extension is part of the default installation, and tells you how much memory your user is using right now, and what the memory limit for your user is. It is shown in the top right corner of the notebook interface. Note that this is memory usage for everything your user is running through the Jupyter notebook interface, not just the specific notebook it is shown on.



Memory: 96 / 126 MB

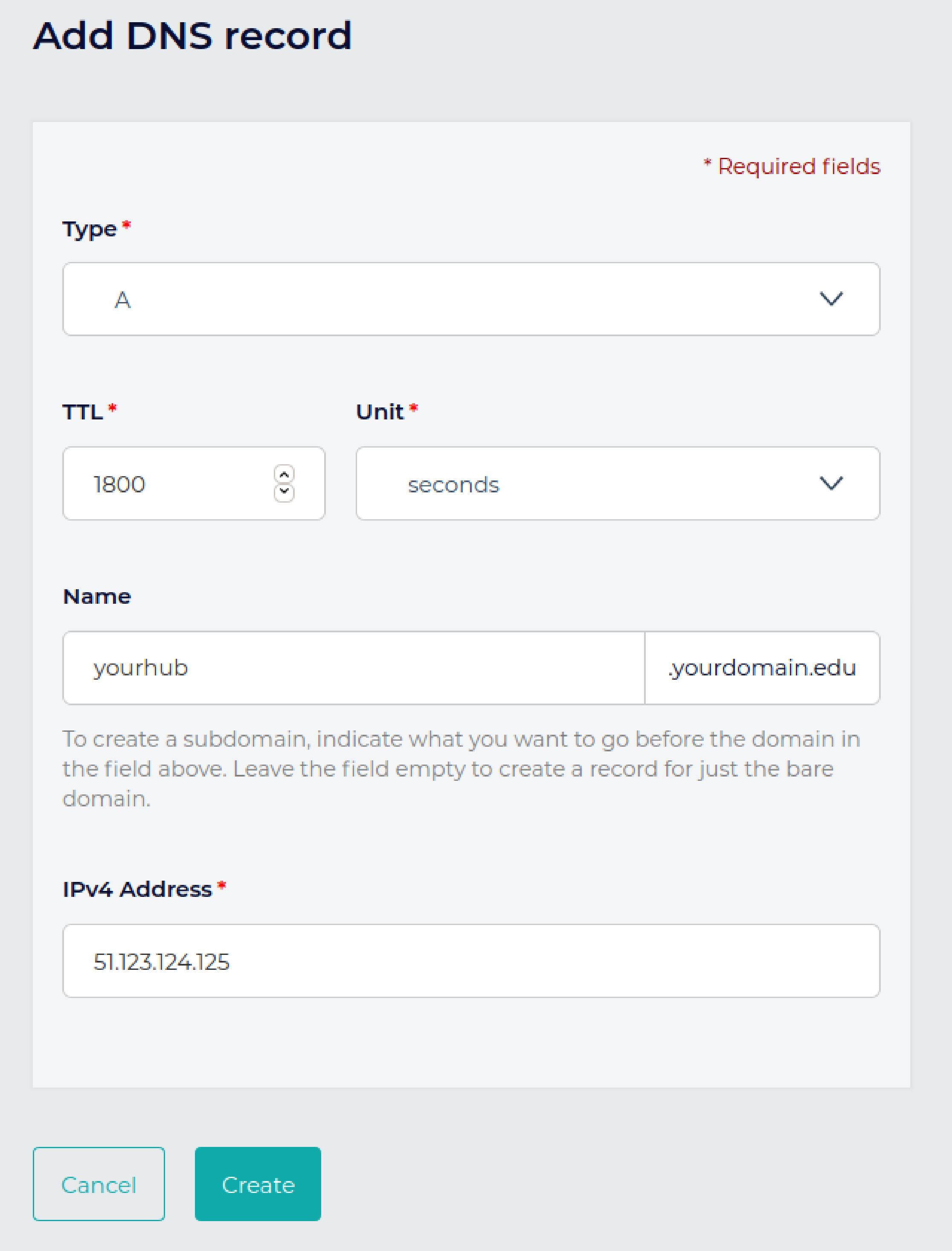
### Enable HTTPS

Every JupyterHub deployment should enable HTTPS!

HTTPS encrypts traffic so that usernames, passwords and your data are communicated securely. sensitive bits of information are communicated securely. The Littlest JupyterHub supports automatically configuring HTTPS via Let's Encrypt, or setting it up *manually* with your own TLS key and certificate. Unless you have a strong reason to use the manual method, you should use the *Let's Encrypt* method.

**Note:** You *must* have a domain name set up to point to the IP address on which TLJH is accessible before you can set up HTTPS.

To do that, you would have to log in to the website of your registrar and go to the DNS records section. The interface will look like something similar to this:



Add DNS record

* Required fields

Type *

A

TTL *

1800

Unit *

seconds

Name

yourhub .yourdomain.edu

To create a subdomain, indicate what you want to go before the domain in the field above. Leave the field empty to create a record for just the bare domain.

IPv4 Address *

51.123.124.125

### Automatic HTTPS with Let's Encrypt

**Note:** If the machine you are running on is not reachable from the internet - for example, if it is a machine internal to your organization that is cut off from the internet - you can not use this method. Please set up a DNS entry and HTTPS *manually*.

To enable HTTPS via letsencrypt:

```
sudo tljh-config set https.enabled true
sudo tljh-config set https.letsencrypt.email you@example.com
sudo tljh-config add-item https.letsencrypt.domains yourhub.yourdomain.edu
```

where `you@example.com` is your email address and `yourhub.yourdomain.edu` is the domain where your hub will be running.

Once you have loaded this, your config should look like:

```
sudo tljh-config show
```

```
https:
  enabled: true
  letsencrypt:
    email: you@example.com
    domains:
    - yourhub.yourdomain.edu
```

Finally, you can reload the proxy to load the new configuration:

```
sudo tljh-config reload proxy
```

At this point, the proxy should negotiate with Let's Encrypt to set up a trusted HTTPS certificate for you. It may take a moment for the proxy to negotiate with Let's Encrypt to get your certificates, after which you can access your Hub securely at https://yourhub.yourdomain.edu.

These certificates are valid for 3 months. The proxy will automatically renew them for you before they expire.

### Manual HTTPS with existing key and certificate

You may already have an SSL key and certificate. If so, you can tell your deployment to use these files:

```
sudo tljh-config set https.enabled true
sudo tljh-config set https.tls.key /etc/mycerts/mydomain.key
sudo tljh-config set https.tls.cert /etc/mycerts/mydomain.cert
```

Once you have loaded this, your config should look like:

```
sudo tljh-config show
```

```
https:
  enabled: true
  tls:
    key: /etc/mycerts/mydomain.key
    cert: /etc/mycerts/mydomain.cert
```

Finally, you can reload the proxy to load the new configuration:

```
sudo tljh-config reload proxy
```

and now access your Hub securely at https://yourhub.yourdomain.edu.

### Troubleshooting

If you're having trouble with HTTPS, looking at the *traefik proxy logs* might help.

### Enabling Jupyter Notebook extensions

Jupyter contributed notebook extensions are community-contributed and maintained plug-ins to the Jupyter notebook. These extensions serve many purposes, from pedagogical tools to tools for converting and editing notebooks.

Extensions are often added and enabled through the graphical user interface of the notebook. However, this interface only makes the extension available to the user, not all users on a hub. Instead, to make contributed extensions available to your users, you will use the command line. This can be completed using the terminal in the JupyterHub (or via SSH-ing into your VM and using this terminal).

### Enabling extensions via the command line

1. There are multiple ways to install contributed extensions. For this example, we will use `pip`.

```
sudo -E pip install jupyter_contrib_nbextensions
```

2. Next, add the notebook extension style files to the Jupyter configuration files.

```
sudo -E jupyter contrib nbextension install --sys-prefix
```

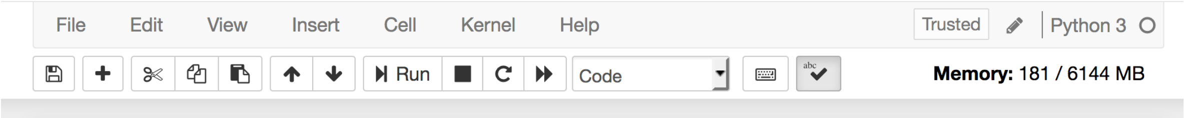
3. Then, you will enable the extensions you would like to use. The syntax for this is `jupyter nbextension enable` followed by the path to the desired extension's main file. For example, to enable scratchpad, you would type the following:

```
sudo -E jupyter nbextension enable scratchpad/main --sys-prefix
```

4. When this is completed, the enabled extension should be visible in the extension list:

```
jupyter nbextension list
```

5. You can also verify the availability of the extension via its user interface in the notebook. For example, spellchecker adds an ABC checkmark icon to the interface.



## 3.1.5 Cloud provider configuration

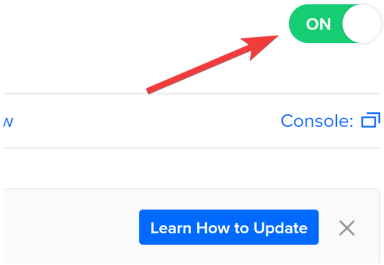
### Perform common Digital Ocean configuration tasks

This page lists various common tasks you can perform on your Digital Ocean virtual machine.

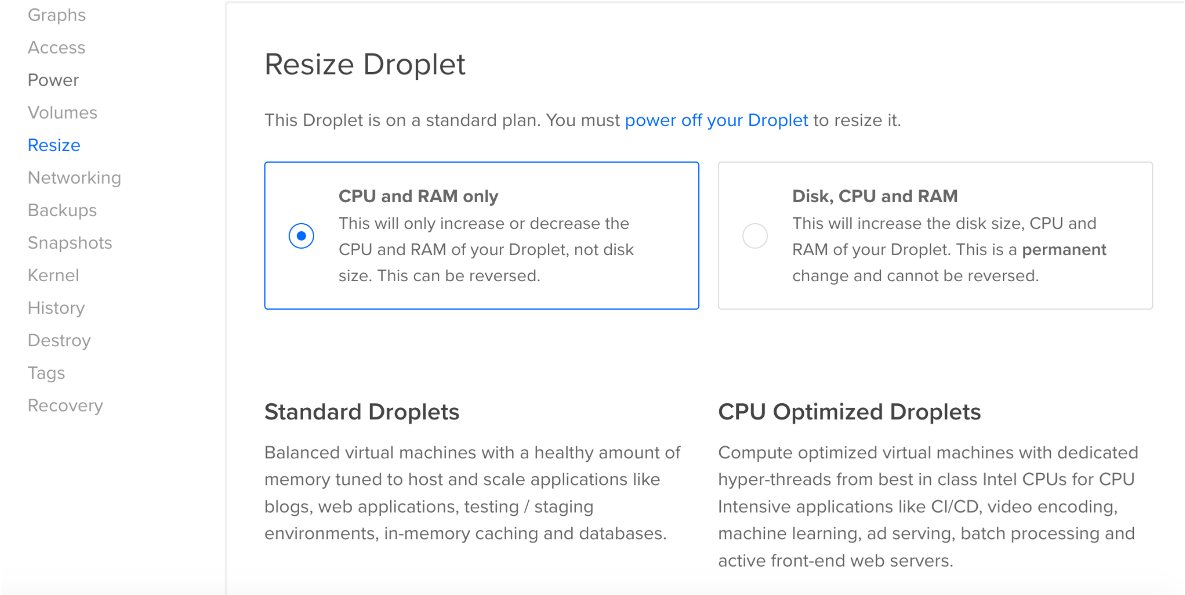
### Resizing your droplet

As you use your JupyterHub, you may find that you need more memory, disk space, or CPUs. Digital Ocean servers can be resized in the "Resize Droplet" panel. These instructions take you through the process.

1. First, click on the name of your newly-created Droplet to enter its configuration page.
2. Next, **turn off your Droplet**. This allows DigitalOcean to make modifications to your VM. This will shut down your JupyterHub (temporarily).



3. Once your Droplet has been turned off, click "Resize", which will take you to a menu with options to resize your VM.



Resize Droplet

This Droplet is on a standard plan. You must power off your Droplet to resize it.

● **CPU and RAM only**
This will only increase or decrease the CPU and RAM of your Droplet, not disk size. This can be reversed.

○ **Disk, CPU and RAM**
This will increase the disk size, CPU and RAM of your Droplet. This is a **permanent** change and cannot be reversed.

Standard Droplets

Balanced virtual machines with a healthy amount of memory tuned to host and scale applications like blogs, web applications, testing / staging environments, in-memory caching and databases.

CPU Optimized Droplets

Compute optimized virtual machines with dedicated hyper-threads from best in class Intel CPUs for CPU Intensive applications like CI/CD, video encoding, machine learning, ad serving, batch processing and active front-end web servers.

4. Decide what kinds of resources you'd like to resize, then click on a new VM type in the list below. Finally, click "Resize". This may take a few moments!
5. Once your Droplet is resized, **turn your Droplet back on**. This makes your JupyterHub available to the world once again. This will take a few moments to complete.

Now that you've resized your Droplet, you may want to change the resources available to your users. Further information on making more resources available to users and verifying resource availability can be found in *Resize the resources available to your JupyterHub*.

### Perform common Microsoft Azure configuration tasks

This page lists various common tasks you can perform on your Microsoft Azure virtual machine.

### Deleting or stopping your virtual machine

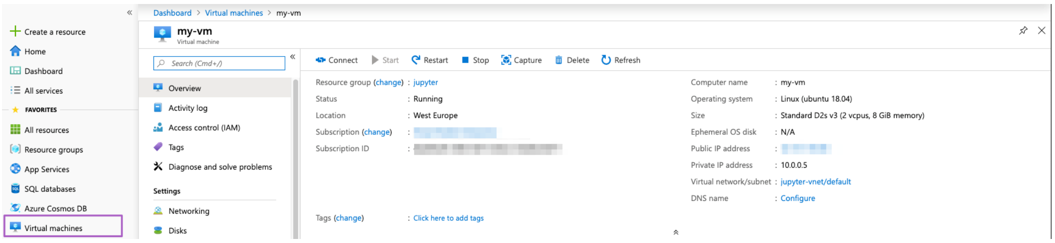
After you have finished using your TLJH you might wanto to either Stop or completely delete the Virtual Machine to avoid incurring in subsequent costs.

The difference between these two approaches is that **Stop** will keep the VM resources (e.g. storage and network) but will effectively stop any compute / runtime activities.

If you choose to delete the VM then all the resources associated with it will be wiped out.

To do either of this:

- Go to "Virtual Machines" on the left hand panel
- Click on your machine name
- Click on "Stop" to stop the machine temporarily, or "Delete" to delete it permanently.



Resource group (change) : jupyter
Status : Running
Location : West Europe
Subscription (change) : [redacted]
Subscription ID : [redacted]
Tags (change) : Click here to add tags

Computer name : my-vm
Operating system : Linux (ubuntu 18.04)
Size : Standard D2s v3 (2 vcpus, 8 GiB memory)
Ephemeral OS disk : N/A
Public IP address : [redacted]
Private IP address : 10.0.0.5
Virtual network/subnet : jupyter-vnet/default
DNS name : Configure

---

**Note:** It is important to mention that even if you stop the machine you will still be charged for the use of the data disk.

---

If you no longer need any of your resources you can delete the entire resource group.

- Go to "Reosurce groups" on the left hand panel
- Click on your resource group
- Click on "Delete resource group" you will then be asked to confirm the operation. This operation will take between 5 and 10 minutes.

# CHAPTER 4

# Topic Guides

Topic guides provide in-depth explanations of specific topics.

## 4.1 Topic Guides

Topic guides provide in-depth explanations of specific topics.

### 4.1.1 When to use The Littlest JupyterHub

This page is a brief guide to determining whether to use The Littlest JupyterHub (TLJH) or Zero to JupyterHub for Kubernetes (Z2JH). Many of these ideas were first laid out in a blog post announcing TLJH.

**The Littlest JupyterHub (TLJH)** is an opinionated and pre-configured distribution to deploy a JupyterHub on a **single machine** (in the cloud or on your own hardware). It is designed to be a more lightweight and maintainable solution for use-cases where size, scalability, and cost-savings are not a huge concern.

**Zero to JupyterHub on Kubernetes** allows you to deploy JupyterHub on **Kubernetes**. This allows JupyterHub to scale to many thousands of users, to flexibly grow/shrink the size of resources it needs, and to use container technology in administering user sessions.

#### When to use TLJH vs. Z2JH

The choice between TLJH and Z2JH ultimately comes down to only a few questions:

1. Do you want your hub and all users to live on a **single, larger machine** vs. spreading users on a **cluster of smaller machines** that are scaled up or down?
    - If you can use a single machine, we recommend **The Littlest JupyterHub**.
    - If you wish to use multiple machines, we recommend **Zero to JupyterHub for Kubernetes**.
2. Do you **need to use container technology**?

- If no, we recommend **The Littlest JupyterHub**.
- If yes, we recommend **Zero to JupyterHub for Kubernetes**.

## 4.1.2 Server Requirements

### Operating System

We require using Ubuntu 18.04 as the base operating system for your server.

### Root access

Full `root` access to this server is required. This might be via `sudo` (recommended) or by direct access to `root` (not recommended!)

### External IP

An external IP allows users on the internet to reach your JupyterHub. Most VPS / Cloud providers give you a public IP address along with your server. If you are hosting on a physical machine somewhere, talk to your system administrators about how to get HTTP traffic from the world into your server.

### CPU / Memory / Disk Space

See how to ref:*howto/admin/resource-estimation*

## 4.1.3 Security Considerations

The Littlest JupyterHub is in beta state & should not be used in security critical situations. We will try to keep things as secure as possible, but sometimes trade security for massive gains in convenience. This page contains information about the security model of The Littlest JupyterHub.

### System user accounts

Each JupyterHub user gets their own Unix user account created when they first start their server. This protects users from each other, gives them a home directory at a well known location, and allows sharing based on file system permissions.

1. The unix user account created for a JupyterHub user named `<username>` is `jupyter-<username>`. This prefix helps prevent clashes with users that already exist - otherwise a user named `root` can trivially gain full root access to your server. If the username (including the `jupyter-` prefix) is longer than 26 characters, it is truncated at 26 characters & a 5 charcter hash is appeneded to it. This keeps usernames under the linux username limit of 32 characters while also reducing chances of collision.
2. A home directory is created for the user under `/home/jupyter-<username>`.
3. The default permission of the home directory is change with `o-rwx` (remove non-group members the ability to read, write or list files and folders in the Home directory).
4. No password is set for this unix system user by default. The password used to log in to JupyterHub (if using an authenticator that requires a password) is not related to the unix user's password in any form.
5. All users created by The Littlest JupyterHub are added to the user group `jupyterhub-users`.

### `sudo` access for admins

JupyterHub admin users are added to the user group `jupyterhub-admins`, which is granted complete root access to the whole server with the `sudo` command on the terminal. No password required.

This is a **lot** of power, and they can do pretty much anything they want to the server - look at other people's work, modify it, break the server in cool & funky ways, etc. This also means **if an admin's credentials are compromised (easy to guess password, password re-use, etc) the entire JupyterHub is compromised.**

### Off-boarding users securely

When you delete users from the JupyterHub admin console, their unix user accounts are **not** removed. This means they might continue to have access to the server even after you remove them from JupyterHub. Admins should manually remove the user from the server & archive their home directories as needed. For example, the following command deletes the unix user associated with the JupyterHub user `yuvipanda`.

```
sudo userdel jupyter-yuvipanda
```

If the user removed from the server is an admin, extra care must be taken since they could have modified the system earlier to continue giving them access.

### Per-user `/tmp`

`/tmp` is shared by all users in most computing systems, and this has been a consistent source of security issues. The Littlest JupyterHub gives each user their own ephemeral `/tmp` using the PrivateTmp feature of systemd.

### HTTPS

Any internet-facing JupyterHub should use HTTPS to secure its traffic. For information on how to use HTTPS with your JupyterHub, see *Enable HTTPS*.

## 4.1.4 Customizing the Installer

The installer can be customized with commandline parameters. The default installer is executed as:

```
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-jupyterhub/master/
↪bootstrap/bootstrap.py \
 | sudo python3 - \
   <parameters>
```

This page documents the various options you can pass as commandline parameters to the installer.

### Adding admin users

`--admin <username>:<password>` adds user `<username>` to JupyterHub as an admin user and sets its password to be `<password>`. Although it is not recommended, it is possible to only set the admin username at this point and set the admin password after the installation.

Also, the `--admin` flag can be repeated multiple times. For example, to add `admin-user1` and `admin-user2` as admins when installing, depending if you would like to set their passwords during install you would:

- set `admin-user1` with password `password-user1` and `admin-user2` with `password-user2` using:

```
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-jupyterhub/master/
↪bootstrap/bootstrap.py \
 | sudo python3 - \
   --admin admin-user1:password-user1 --admin admin-user2:password-user2
```

- set `admin-user1` and `admin-user2` to be admins, without any passwords at this stage, using:

```
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-jupyterhub/master/
↪bootstrap/bootstrap.py \
 | sudo python3 - \
   --admin admin-user1 --admin admin-user2
```

- set `admin-user1` with password `password-user1` and `admin-user2` with no password at this stage using:

```
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-jupyterhub/master/
↪bootstrap/bootstrap.py \
 | sudo python3 - \
   --admin admin-user1:password-user1 --admin admin-user2
```

### Installing python packages in the user environment

`--user-requirements-txt-url <url-to-requirements.txt>` installs packages specified in the `requirements.txt` located at the given URL into the user environment at install time. This is very useful when you want to set up a hub with a particular user environment in one go.

For example, to install the latest requirements to run UC Berkeley's data8 course in your new hub, you would run:

```
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-jupyterhub/master/
↪bootstrap/bootstrap.py \
 | sudo python3 - \
   --user-requirements-txt-url https://raw.githubusercontent.com/data-8/materials-
↪sp18/master/requirements.txt
```

The URL **must** point to a working requirements.txt. If there are any errors, the installation will fail.

**Note:** When pointing to a file on GitHub, make sure to use the 'Raw' version. It should point to `raw.githubusercontent.com`, not `github.com`.

### Installing TLJH plugins

The Littlest JupyterHub can install additional *plugins* that provide additional features. They are most commonly used to install a particular *stack* - such as the PANGEO Stack for earth sciences research, a stack for a particular class, etc.

`--plugin <plugin-to-install>` installs and activates a plugin. You can pass it however many times you want. Since plugins are distributed as python packages, `<plugin-to-install>` can be anything that can be passed to `pip install` - `plugin-name-on-pypi==<version>` and `git+https://github.com/user/repo@tag` are the most popular ones. Specifying a version or tag is highly recommended.

For example, to install the PANGEO Plugin version 0.1 in your new TLJH install, you would use:

```
curl https://raw.githubusercontent.com/jupyterhub/the-littlest-jupyterhub/master/
↪bootstrap/bootstrap.py \
 | sudo python3 - \
   --plugin git+https://github.com/yuvipanda/tljh-pangeo@v0.1
```

Multiple plugins can be installed at once with: `--plugin <first-plugin-to-install> <second-plugin-to-install>`.

See *TLJH Plugins* for more information about plugins.

**Note:** Plugins are extremely powerful and can do a large number of arbitrary things. Only install plugins you trust.

### 4.1.5 What does the installer do?

This document details what exactly the installer does to the machine it is run on.

#### `apt` Packages installed

The packages `python3` and `python3-venv` are installed from the apt repositories. Since we need an recent & supported version of `nodejs`, we install it from nodesource.

#### Hub environment

JupyterHub is run from a python3 virtual environment located in `/opt/tljh/hub`. It uses the system's installed python and is owned by root. It also contains a binary install of traefik. This virtual environment is completely managed by TLJH.

**Note:** If you try to remove TLJH, revert this action using:

```
sudo rm -rf /opt/tljh/hub
```

#### User environment

By default, a `miniconda` environment is installed in `/opt/tljh/user`. This contains the notebook interface used to launch all users, and the various packages available to all users. The environment is owned by the `root` user. JupyterHub admins may use to `sudo -E conda install` or `sudo -E pip install` packages into this environment.

This conda environment is added to `$PATH` for all users started with JupyterHub. If you are using `ssh` instead, you can activate this environment by running the following:

```
source /opt/tljh/user/bin/activate
```

This should let you run various `conda` and `pip` commands. If you run into errors like `Command 'conda' not found`, try prefixing your command with:

```
sudo PATH=${PATH} <command>
```

By default, sudo does not respect any custom environments you have activated. The PATH=${PATH} 'fixes' that.

**Note:** If you try to remove TLJH, revert this action using:

```
sudo rm -rf /opt/tljh/user
```

### tljh-config symlink

We create a symlink from /usr/bin/tljh-config to /opt/tljh/hub/bin/tljh-config, so users can run sudo tljh-config <something> from their terminal. While the user environment is added to users' $PATH when they launch through JupyterHub, the hub environment is not. This makes it hard to access the tljh-config command used to change most config parameters. Hence we symlink the tljh-config command to /usr/bin, so it is directly accessible with sudo tljh-config <command>.

**Note:** If you try to remove TLJH, revert this action using:

```
sudo unlink /usr/bin/tljh-config
```

### jupyterhub_config.d directory for custom configuration snippets

Any files in /opt/tljh/config/jupyterhub_config.d that end in .py and are a valid JupyterHub configuration will be loaded after any of the config options specified with tljh-config are loaded.

**Note:** If you try to remove TLJH, revert this action using:

```
sudo rm -rf /opt/tljh/config
```

### Systemd Units

TLJH places 2 systemd units on your computer. They all start on system startup.

1. jupyterhub.service - starts the JupyterHub service.
2. traefik.service - starts traefik proxy that manages HTTPS

In addition, each running Jupyter user gets their own systemd unit of the name jupyter-<username>.

**Note:** If you try to remove TLJH, revert this action using:

```
# stop the services
systemctl stop jupyterhub.service
systemctl stop traefik.service
systemctl stop jupyter-<username>

# disable the services
systemctl disable jupyterhub.service
systemctl disable traefik.service
# run this command for all the Jupyter users
```

(continues on next page)

(continued from previous page)

```
systemctl disable jupyter-<username>

# remove the systemd unit
rm /etc/systemd/system/jupyterhub.service
rm /etc/systemd/system/traefik.service

# reset the state of all units
systemctl daemon-reload
systemctl reset-failed
```

### State files

TLJH places 3 *jupyterhub.service* and 4 *traefik.service* state files in */opt/tljh/state*. These files save the state of JupyterHub and Traefik services and are meant to be used and modified solely by these services.

**Note:** If you try to remove TLJH, revert this action using:

```
sudo rm -rf /opt/tljh/state
```

### User groups

TLJH creates two user groups when installed:

1. `jupyterhub-users` contains all users managed by this JupyterHub
2. `jupyterhub-admins` contains all users with admin rights managed by this JupyterHub.

When a new JupyterHub user logs in, a unix user is created for them. The unix user is always added to the `jupyterhub-users` group. If the user is an admin, they are added to the `jupyterhub-admins` group whenever they start / stop their notebook server.

If you uninstall TLJH, you should probably remove all user accounts associated with both these user groups, and then remove the groups themselves. You might have to archive or delete the home directories of these users under `/home/`.

**Note:** If you try to remove TLJH, in order to remove a user and its home directory, use:

```
sudo userdel -r <user>
```

Keep in mind that the files located in other parts of the file system will have to be searched for and deleted manually.

**Note:** To remove the user groups units:

```
sudo delgroup jupyterhub-users
sudo delgroup jupyterhub-admins
# remove jupyterhub-admins from the sudoers group
sudo rm /etc/sudoers.d/jupyterhub-admins
```

### Passwordless sudo for JupyterHub admins

/etc/sudoers.d/jupyterhub-admins is created to provide passwordless sudo for all JupyterHub admins. We also set it up to inherit $PATH with sudo -E, to more easily call conda, pip, etc.

### Removing TLJH

If trying to wipe out a fresh TLJH installation, follow the instructions on how to revert each specific modification the TLJH installer does to the system.

**Note:** If using a VM, the recommended way to remove TLJH is destroying the VM and start fresh.

**Warning:** Completely uninstalling TLJH after it has been used is a difficult task because it's highly coupled to how the system changed after it has been used and modified by the users. Thus, we cannot provide instructions on how to proceed in this case.

## 4.1.6 Configuring TLJH with tljh-config

tljh-config is the commandline program used to make configuration changes to TLJH.

### Running tljh-config

You can run tljh-config in two ways:

1. From inside a terminal in JupyterHub while logged in as an admin user. This method is recommended.
2. By directly calling /opt/tljh/hub/bin/tljh-config as root when logged in to the server via other means (such as SSH). This is an advanced use case, and not covered much in this guide.

### Set / Unset a configuration property

TLJH's configuration is organized in a nested tree structure. You can set a particular property with the following command:

```
sudo tljh-config set <property-path> <value>
```

where:

1. <property-path> is a dot-separated path to the property you want to set.
2. <value> is the value you want to set the property to.

For example, to set the password for the DummyAuthenticator, you need to set the auth.DummyAuthenticator.password property. You would do so with the following:

```
sudo tljh-config set auth.DummyAuthenticator.password mypassword
```

This can only set string and numerical properties, not lists.

To unset a configuration property you can use the following command:

```
sudo tljh-config unset <property-path>
```

Unsetting a configuration property removes the property from the configuration file. If what you want is only to change the property's value, you should use `set` and overwrite it with the desired value.

Some of the existing `<property-path>` are listed below by categories:

### Authentication

Use `auth.type` to determine authenticator to use. All parameters in the config under `auth.{auth.type}` will be passed straight to the authenticators themselves.

### Ports

Use `http.port` and `https.port` to set the ports that TLJH will listen on, which are 80 and 443 by default. However, if you change these, note that TLJH does a lot of other things to the system (with user accounts and sudo rules primarily) that might break security assumptions your other applications have, so use with extreme caution.

```
sudo tljh-config set http.port 8080
sudo tljh-config set https.port 8443
sudo tljh-config reload proxy
```

### User Lists

- `users.allowed` takes in usernames to whitelist
- `users.banned` takes in usernames to blacklist
- `users.admin` takes in usernames to designate as admins

### User Server Limits

- `limits.memory` Specifies the maximum memory that can be used by each individual user. By default there is no memory limit. The limit can be specified as an absolute byte value. You can use the suffixes K, M, G or T to mean Kilobyte, Megabyte, Gigabyte or Terabyte respectively. Setting it to `None` disables memory limits.

  ```
  sudo tljh-config set limits.memory 4G
  ```

  Even if you want individual users to use as much memory as possible, it is still good practice to set a memory limit of 80-90% of total physical memory. This prevents one user from being able to single handedly take down the machine accidentally by OOMing it.

- `limits.cpu` A float representing the total CPU-cores each user can use. By default there is no CPU limit. 1 represents one full CPU, 4 represents 4 full CPUs, 0.5 represents half of one CPU, etc. This value is ultimately converted to a percentage and rounded down to the nearest integer percentage, i.e. 1.5 is converted to 150%, 0.125 is converted to 12%, etc. Setting it to `None` disables CPU limits.

  ```
  sudo tljh-config set limits.cpu 2
  ```

### User Environment

user_environment.default_app Set default application users are launched into. Currently can be set to the following values jupyterlab or nteract

```
sudo tljh-config set user_environment.default_app jupyterlab
```

### Extra User Groups

users.extra_user_groups is a configuration option that can be used to automatically add a user to a specific group. By default, there are no extra groups defined.

Users can be "paired" with the desired, **existing** groups using:

- tljh-config set, if only one user is to be added to the desired group:

```
tljh-config set users.extra_user_groups.group1 user1
```

- tljh-config add-item, if multiple users are to be added to the group:

```
tljh-config add-item users.extra_user_groups.group1 user1
tljh-config add-item users.extra_user_groups.group1 user2
```

### View current configuration

To see the current configuration, you can run the following command:

```
sudo tljh-config show
```

This will print the current configuration of your TLJH. This is very useful when asking for support!

### Reloading JupyterHub to apply configuration

After modifying the configuration, you need to reload JupyterHub for it to take effect. You can do so with:

```
sudo tljh-config reload
```

This should not affect any running users. The JupyterHub will be restarted and loaded with the new configuration.

### Advanced: config.yaml

tljh-config is a simple program that modifies the contents of the config.yaml file located at /opt/tljh/config/config.yaml. tljh-config is the recommended method of editing / viewing configuration since editing YAML by hand in a terminal text editor is a large source of errors.

## 4.1.7 Configuring JupyterHub authenticators

Any JupyterHub authenticator can be used with TLJH. A number of them ship by default with TLJH:

1. OAuthenticator - Google, GitHub, CILogon, GitLab, Globus, Mediawiki, auth0, generic OpenID connect (for KeyCloak, etc) and other OAuth based authentication methods.

2. LDAPAuthenticator - LDAP & Active Directory.
3. DummyAuthenticator - Any username, one shared password. A *how-to guide on using DummyAuthenticator* is also available.
4. FirstUseAuthenticator - Users set their password when they log in for the first time. Default authenticator used in TLJH.
5. TmpAuthenticator - Opens the JupyterHub to the world, makes a new user every time someone logs in.
6. NativeAuthenticator - Allow users to signup, add password security verification and block users after failed attempts oflogin.

We try to have specific how-to guides & tutorials for common authenticators. Since we can not cover everything, this guide shows you how to use any authenticator you want with JupyterHub by following the authenticator's documentation.

### Setting authenticator properties

JupyterHub authenticators are customized by setting *traitlet properties*. In the authenticator's documentation, you will find these are usually represented as:

```
c.<AuthenticatorName>.<property-name> = <some-value>
```

You can set these with `tljh-config` with:

```
sudo tljh-config set auth.<AuthenticatorName>.<property-name> <some-value>
```

### Example

LDAPAuthenticator's documentation lists the various configuration options you can set for LDAPAuthenticator. When the documentation asks you to set `LDAPAuthenticator.server_address` to some value, you can do that with the following command:

```
sudo tljh-config set auth.LDAPAuthenticator.server_address 'my-ldap-server'
```

Most authenticators require you set multiple configuration options before you can enable them. Read the authenticator's documentation carefully for more information.

### Enabling the authenticator

Once you have configured the authenticator as you want, you should then enable it. Usually, the documentation for the authenticator would ask you to add something like the following to your `jupyterhub_config.py` to enable it:

```
c.JupyterHub.authenticator_class = 'fully-qualified-authenticator-name'
```

You can accomplish the same with `tljh-config`:

```
sudo tljh-config set auth.type <fully-qualified-authenticator-name>
```

Once enabled, you need to reload JupyterHub for the config to take effect.

```
sudo tljh-config reload
```

Try logging in a separate incognito window to check if your configuration works. This lets you preserve your terminal in case there were errors. If there are errors, *Looking at Logs* should help you debug them.

### Example

From the documentation for LDAPAuthenticator, we see that the fully qualified name is `ldapauthenticator.LDAPAuthenticator`. Assuming you have already configured it, the following commands enable LDAPAuthenticator.

```
sudo tljh-config set auth.type ldapauthenticator.LDAPAuthenticator
sudo tljh-config reload
```

## 4.1.8 Custom configuration snippets

The two main TLJH components are **JupyterHub** and **Traefik**.

- JupyterHub takes its configuration from the `jupyterhub_config.py` file.
- Traefik takes its configuration from the `traefik.toml` file.

These files are created by TLJH during installation and can be edited by the user only through `tljh-config`. Any direct modification to these files is unsupported, and will cause hard to debug issues.

But because sometimes TLJH needs to be customized in ways that are not officially supported, an escape hatch has been introduced to allow easily extending the configuration. Please follow the sections below for how to extend JupyterHub's and Traefik's configuration outside of `tljh-config` scope.

### Extending `jupyterhub_config.py`

The `jupyterhub_config.d` directory lets you load multiple `jupyterhub_config.py` snippets for your configuration.

- Any files in `/opt/tljh/config/jupyterhub_config.d` that end in `.py` will be loaded in alphabetical order as python files to provide configuration for JupyterHub.
- The configuration files can have any name, but they need to have the *.py* extension and to respect this format.
- Any config that can go in a regular `jupyterhub_config.py` file is valid in these files.
- They will be loaded *after* any of the config options specified with `tljh-config` are loaded.

Once you have created and defined your custom JupyterHub config file/s, just reload the hub for the new configuration to take effect:

```
sudo tljh-config reload hub
```

### Extending `traefik.toml`

The `traefik_config.d` directory lets you load multiple `traefik.toml` snippets for your configuration.

- Any files in `/opt/tljh/config/traefik_config.d` that end in `.toml` will be loaded in alphabetical order to provide configuration for Traefik.
- The configuration files can have any name, but they need to have the *.toml* extension and to respect this format.
- Any config that can go in a regular `traefik.toml` file is valid in these files.

- They will be loaded *after* any of the config options specified with `tljh-config` are loaded.

Once you have created and defined your custom Traefik config file/s, just reload the proxy for the new configuration to take effect:

```
sudo tljh-config reload proxy
```

**Warning:** This instructions might change when TLJH will switch to Traefik > 2.0

### 4.1.9 Culling idle notebook servers

The idle culler automatically shuts down user notebook servers when they have not been used for a certain time period, in order to reduce the total resource usage on your JupyterHub.

JupyterHub pings the user's notebook server at certain time intervals. If no response is received from the server during this checks and the timeout expires, the server is considered to be *inactive (idle)* and will be culled.

#### Default settings

By default, JupyterHub will ping the user notebook servers every 60s to check their status. Every server found to be idle for more than 10 minutes will be culled.

```
services.cull.every = 60
services.cull.timeout = 600
```

Because the servers don't have a maximum age set, an active server will not be shut down regardless of how long it has been up and running.

```
services.cull.max_age = 0
```

If after the culling process, there are users with no active notebook servers, by default, the users will not be culled alongside their notebooks and will continue to exist.

```
services.cull.users = False
```

#### Configuring the idle culler

The available configuration options are:

#### Idle timeout

The idle timeout is the maximum time (in seconds) a server can be inactive before it will be culled. The timeout can be configured using:

```
sudo tljh-config set services.cull.timeout <max-idle-sec-before-server-is-culled>
sudo tljh-config reload
```

### Idle check interval

The idle check interval represents how frequent (in seconds) the Hub will check if there are any idle servers to cull. It can be configured using:

```
sudo tljh-config set services.cull.every <number-of-sec-this-check-is-done>
sudo tljh-config reload
```

### Maximum age

The maximum age sets the time (in seconds) a server should be running. The servers that exceed the maximum age, will be culled even if they are active. A maximum age of 0, will deactivate this option. The maximum age can be configured using:

```
sudo tljh-config set services.cull.max_age <server-max-age>
sudo tljh-config reload
```

### User culling

In addition to servers, it is also possible to cull the users. This is usually suited for temporary-user cases such as *tmpnb*. User culling can be activated using the following command:

```
sudo tljh-config set services.cull.users True
sudo tljh-config reload
```

### Concurrency

Deleting a lot of users at the same time can slow down the Hub. The number of concurrent requests made to the Hub can be configured using:

```
sudo tljh-config set services.cull.concurrency <number-of-concurrent-hub-requests>
sudo tljh-config reload
```

Because TLJH it's used for a small number of users, the cases that may require to modify the concurrency limit should be rare.

### Disabling the idle culler

The idle culling service is enabled by default. To disable it, use the following command:

```
sudo tljh-config set services.cull.enabled False
sudo tljh-config reload
```

# CHAPTER 5

# Troubleshooting

In time, all systems have issues that need to be debugged. Troubleshooting guides help you find what is broken & hopefully fix it.

## 5.1 Troubleshooting

In time, all systems have issues that need to be debugged. Troubleshooting guides help you find what is broken & hopefully fix it.

### 5.1.1 Looking at Logs

**Logs** are extremely useful in piecing together what went wrong when things go wrong. They contain a forensic record of what individual pieces of software were doing before things went bad, and can help us understand the problem so we can fix it.

TLJH collects logs from JupyterHub, Traefik Proxy, & from each individual user's notebook server. All the logs are accessible via journalctl. The installer also writes logs to disk, to help with cases where the installer did not succeed.

**Warning:** If you are providing a snippet from the logs to someone else to help debug a problem you might have, be careful to redact any private information (such as usernames) from the snippet first!

#### Installer Logs

The JupyterHub installer writes log messages to `/opt/tljh/installer.log`. This is very useful if the installation fails for any reason.

### JupyterHub Logs

JupyterHub is responsible for user authentication, & starting / stopping user notebook servers. When there is a general systemic issue with JupyterHub (rather than a specific issue with a particular user's notebook), looking at the JupyterHub logs is a great first step.

```
sudo journalctl -u jupyterhub
```

This command displays logs from JupyterHub itself. See *journalctl tips* for tips on navigating the logs.

### Traefik Proxy Logs

traefik redirects traffic to JupyterHub / user notebook servers as necessary & handles HTTPS. Look at this if all you can see in your browser is one line cryptic error messages, or if you are having trouble with HTTPS.

```
sudo journalctl -u traefik
```

This command displays logs from Traefik. See *journalctl tips* for tips on navigating the logs.

### User Server Logs

Each user gets their own notebook server, and this server also produces logs. Looking at these can be useful when a user can launch their server but run into problems after that.

```
sudo journalctl -u jupyter-<name-of-user>
```

This command displays logs from the given user's notebook server. You can get a list of all users from the "users" button at the top-right of the Admin page. See *journalctl tips* for tips on navigating the logs.

### journalctl tips

`journalctl` has a lot of options to make your life as an administrator easier. Here are some very basic tips on effective `journalctl` usage.

1. When looking at full logs (via `sudo journalctl -u <some-name>`), the output usually does not fit into one screen. Hence, it is *paginated* with less. This allows you to scroll up / down, search for specific words, etc. Some common keyboard shortcuts are:

    - Arrow keys to move up / down / left / right
    - `G` to navigate to the end of the logs
    - `g` to navigate to the start of the logs
    - `/` followed by a string to search for & `enter` key to search the logs from current position on screen to the end of the logs. If there are multiple results, you can use `n` key to jump to the next search result. Use `?` instead of `/` to search backwards from current position
    - `q` or `Ctrl + C` to exit

    There are plenty of other commands & options to explore if you wish.

2. Add `-f` to any `journalctl` command to view live logging output that updates as new log lines are written. This is extremely useful when actively debugging an issue.

    For example, to watch live logs of JupyterHub, you can run:

```
sudo journalctl -u jupyterhub -f
```

Often, your issues are not related to TLJH itself but to the cloud provider your server is running on. We have some documentation on common issues you might run into with various providers and how to fix them. We welcome contributions here to better support your favorite provider!

## 5.1.2 Troubleshooting issues on Google Cloud

This is an incomplete list of issues people have run into when running TLJH on Google Cloud, and how they have fixed them!

### 'Connection Refused' error after restarting server

If you restarted your server from the Google Cloud console & then try to access your JupyterHub from a browser, you might get a **Connection Refused** error. This is most likely because the **External IP** of your server has changed.

Check the **External IP** in the Google Cloud Console -> Compute Engine -> VM instances screen matches the IP you are trying to access. If you have a domain name pointing to the IP address, you might have to change it to point to the new correct IP.

You can prevent External IP changes by reserving the static IP your server is using.

## 5.1.3 Troubleshooting issues on Amazon Web Services

This is an incomplete list of issues people have run into when running TLJH on Amazon Web Services (AWS), and how they have fixed them!

### 'Connection Refused' error after restarting server

If you restarted your server from the EC2 Management Console & then try to access your JupyterHub from a browser, you might get a **Connection Refused** error. This is most likely because the **External IP** of your server has changed.

Check the **IPv4 Public IP** dislayed in the bottom of the *EC2 Management Console* screen for that instance matches the IP you are trying to access. If you have a domain name pointing to the IP address, you might have to change it to point to the new correct IP.

You can prevent public IP changes by associating a static IP with your server. In the Amazon Web Services ecosystem, the public static IP addresses are handled under *Elastic IP addresses* category of AWS; these addresses are tied to the overall AWS account. This guide might be helpful. Notice there can be a cost to this. Although the guide is outdated (generally half that price now), Amazon describes here how the Elastic IP address feature is free when associated with a running instance, but that you'll be charged by the hour for maintaining that specific IP address when it isn't associated with a running instance.

## 5.1.4 Troubleshooting issues on your own server

This is an incomplete list of issues people have run into when installing TLJH on their own servers, and ways they have fixed them.

### Outgoing HTTP proxy required

If your server is behind a firewall that requires a HTTP proxy to reach the internet, run these commands before running the installer

```
export http_proxy=<your_proxy-server>
```

### HTTPS certificate interception

If your server is behind a firewall that intercepts HTTPS requests and re-signs them, you might have to explicitly tell TLJH which certificates to use.

```
export REQUESTS_CA_BUNDLE=</directory/with/your/ssl/certificates>
sudo npm config set cafile=</directory/with/your/ssl/certificates>
```

CHAPTER 6

# Contributing

We want you to contribute to TLJH in the ways that are most useful and exciting to you. This section contains documentation helpful to people contributing in various ways.

## 6.1 Contributing

Thank you for thinking about contributing to the littlest JupyterHub!

This is an open source project that is developed and maintained by volunteers. Your contribution is integral to the future of the project. Thank you!

This section contains documentation for people who want to contribute.

You can find the source code on GitHub

### 6.1.1 Writing documentation

**Note:** Heavily inspired by the django project's guidelines

We place a high importance on consistency, readability and completeness of documentation. If a feature is not documented, it does not exist. If a behavior is not documented, it is a bug! We try to treat our documentation like we treat our code: we aim to improve it as often as possible.

Documentation changes generally come in two forms:

- General improvements: typo corrections, error fixes and better explanations through clearer writing and more examples.
- New features: documentation of features that have been added to the framework since the last release.

This section explains how writers can craft their documentation changes in the most useful and least error-prone ways.

### Getting the raw documentation

Though TLJH's documentation is intended to be read as HTML at https://the-littlest-jupyterhub.readthedocs.io/, we edit it as a collection of text files for maximum flexibility. These files live in the top-level `docs/` directory of TLJH's repository.

If you'd like to start contributing to our docs, get the development version of TLJH from the source code repository. The development version has the latest-and-greatest documentation, just as it has latest-and-greatest code.

### Getting started with Sphinx

TLJH's documentation uses the Sphinx documentation system, which in turn is based on docutils. The basic idea is that lightly-formatted plain-text documentation is transformed into HTML, PDF, and any other output format.

To build the documentation locally, install the Sphinx dependencies:

```
$ cd docs/
$ pip install -r requirements.txt
```

Then from the `docs` directory, build the HTML:

```
$ make html
```

If you encounter this error, it's likely that you are running inside a virtual environment.

```
Error in "currentmodule" directive:
```

To get started contributing, you'll want to read the reStructuredText reference

Your locally-built documentation will be themed differently than the documentation at the-littlest-jupyterhub.readthedocs.io. This is OK! If your changes look good on your local machine, they'll look good on the website.

### How the documentation is organized

The documentation is organized into several categories:

- **Tutorials** take the reader by the hand through a series of steps to create something.

  The important thing in a tutorial is to help the reader achieve something useful, preferably as early as possible, in order to give them confidence.

  Explain the nature of the problem we're solving, so that the reader understands what we're trying to achieve. Don't feel that you need to begin with explanations of how things work - what matters is what the reader does, not what you explain. It can be helpful to refer back to what you've done and explain afterwards.

  **Installation Tutorials** are a special subcategory of tutorials that teach the user how to install TLJH in various cloud providers / bare metal systems. These should cross-link a lot to other parts of the documentation, avoid forcing the user to learn to SSH if possible & have lots of screenshots.

- **Topic guides** aim to explain a concept or subject at a fairly high level.

  Link to reference material rather than repeat it. Use examples and don't be reluctant to explain things that seem very basic to you - it might be the explanation someone else needs.

  Providing background context helps a newcomer connect the topic to things that they already know.

- **Reference guides** contain technical reference for APIs. They describe the functioning of TLJH's internal machinery and instruct in its use.

  Keep reference material tightly focused on the subject. Assume that the reader already understands the basic concepts involved but needs to know or be reminded of how TLJH does it.

  Reference guides aren't the place for general explanation. If you find yourself explaining basic concepts, you may want to move that material to a topic guide.

- **How-to guides** are recipes that take the reader through steps in key subjects.

  What matters most in a how-to guide is what a user wants to achieve. A how-to should always be result-oriented rather than focused on internal details of how TLJH implements whatever is being discussed.

  These guides are more advanced than tutorials and assume some knowledge about how TLJH works. Assume that the reader has followed the tutorials and don't hesitate to refer the reader back to the appropriate tutorial rather than repeat the same material.

- **Troubleshooting guides** help reader answer the question "Why is my JupyterHub not working?".

  These guides help readers try find causes for their symptoms, and hopefully fix the issues. Some of these need to be specific to cloud providers, and that is acceptable.

### Writing style

Typically, documentation is written in second person, referring to the reader as "you". When using pronouns in reference to a hypothetical person, such as "a user with a running notebook", gender neutral pronouns (they/their/them) should be used. Instead of:

- he or she… use they.
- him or her… use them.
- his or her… use their.
- his or hers… use theirs.
- himself or herself… use themselves.

### Commonly used terms

Here are some style guidelines on commonly used terms throughout the documentation:

- **TLJH** – common abbreviation of The Littlest JupyterHub. Fully capitalized except when used in code / the commandline.
- **Python** – when referring to the language, capitalize Python.
- **Notebook Interface** – generic term for referring to JupyterLab, nteract, classic notebook & other user interfaces for accessing

### Guidelines for reStructuredText files

These guidelines regulate the format of our reST (reStructuredText) documentation:

- In section titles, capitalize only initial words and proper nouns.
- Wrap the documentation at 120 characters wide, unless a code example is significantly less readable when split over two lines, or for another good reason.
- Use these heading styles:

```
===
One
===

Two
===

Three
-----

Four
~~~~

Five
^^^^
```

### Documenting new features

Our policy for new features is:

> All new features must have appropriate documentation before they can be merged.

### Choosing image size

When adding images to the documentation, try to keep them as small as possible. Larger images make the site load more slowly on browsers, and may make the site inaccessible for people with a slow internet connection.

If you're adding screenshots, make the size of your shot as small as possible. If you're uploading large images, consider using an image optimizer in order to reduce its size.

For example, for PNG files, use OptiPNG and AdvanceCOMP's `advpng`:

```
$ cd docs
$ optipng -o7 -zm1-9 -i0 -strip all `find . -type f -not -path "./_build/*" -name "*.
↪png"`
$ advpng -z4 `find . -type f -not -path "./_build/*" -name "*.png"`
```

This is based on OptiPNG version 0.7.5. Older versions may complain about the `--strip all` option being lossy.

### Spelling check

Before you commit your docs, it's a good idea to run the spelling checker. You'll need to install a couple packages first:

- pyenchant (which requires enchant)
- sphinxcontrib-spelling

Then from the `docs` directory, run `make spelling`. Wrong words (if any) along with the file and line number where they occur will be saved to `_build/spelling/output.txt`.

If you encounter false-positives (error output that actually is correct), do one of the following:

- Surround inline code or brand/technology names with grave accents (').
- Find synonyms that the spell checker recognizes.

- If, and only if, you are sure the word you are using is correct - add it to `docs/spelling_wordlist` (please keep the list in alphabetical order).

### 6.1.2 Setting up Development Environment

The easiest & safest way to develop & test TLJH is with Docker.

1. Install Docker Community Edition by following the instructions on their website.
2. Clone the [git repo](https://github.com/jupyterhub/the-littlest-jupyterhub) (or your fork of it).
3. Build a docker image that has a functional systemd in it.

   ```
   docker build -t tljh-systemd . -f integration-tests/Dockerfile
   ```

4. Run a docker container with the image in the background, while bind mounting your TLJH repository under `/srv/src`.

   ```
   docker run \
     --privileged \
     --detach \
     --name=tljh-dev \
     --publish 12000:80 \
     --mount type=bind,source=$(pwd),target=/srv/src \
     tljh-systemd
   ```

5. Get a shell inside the running docker container.

   ```
   docker exec -it tljh-dev /bin/bash
   ```

6. Run the bootstrapper from inside the container (see step above): The container image is already set up to default to a `dev` install, so it'll install from your local repo rather than from github.

   ```
   python3 /srv/src/bootstrap/bootstrap.py --admin admin
   ```

   Or, if you would like to setup the admin's password during install, you can use this command (replace "admin" with the desired admin username and "password" with the desired admin password):

   ```
   python3 /srv/src/bootstrap/bootstrap.py --admin admin:password
   ```

   The primary hub environment will also be in your PATH already for convenience.

1. You should be able to access the JupyterHub from your browser now at http://localhost:12000. Congratulations, you are set up to develop TLJH!
2. Make some changes to the repository. You can test easily depending on what you changed.
   - If you changed the `bootstrap/bootstrap.py` script or any of its dependencies, you can test it by running `python3 /srv/src/bootstrap/bootstrap.py`.
   - If you changed the `tljh/installer.py` code (or any of its dependencies), you can test it by running `python3 -m tljh.installer`.
   - If you changed `tljh/jupyterhub_config.py`, `tljh/configurer.py`, `/opt/tljh/config/` or any of their dependencies, you only need to restart jupyterhub for them to take effect. `tljh-config reload hub` should do that.

*Looking at Logs* has information on looking at various logs in the container to debug issues you might have.

### 6.1.3 Testing TLJH

Unit and integration tests are a core part of TLJH, as important as the code & documentation. They help validate that the code works as we think it does, and continues to do so when changes occur. They also help communicate in precise terms what we expect our code to do.

#### Integration tests

TLJH is a *distribution* where the primary value is the many opinionated choices we have made on components to use and how they fit together. Integration tests are perfect for testing that the various components fit together and work as they should. So we write a lot of integration tests, and put in more effort towards them than unit tests.

All integration tests are run on CircleCI for each PR and merge, making sure we don't have broken tests for too long.

The integration tests are in the `integration-tests` directory in the git repository. `py.test` is used to write the integration tests. Each file should contain tests that can be run in any order against the same installation of TLJH.

#### Running integration tests locally

You need `docker` installed and callable by the user running the integration tests without needing sudo.

You can then run the tests with:

```
.circleci/integration-test.py run-test <name-of-run> <test-file-names>
```

- `<name-of-run>` is an identifier for the tests - you can choose anything you want
- `<test-file-names>>` is list of test files (under `integration-tests`) that should be run in one go.

For example, to run all the basic tests, you would write:

```
.circleci/integration-test.py run-test basic-tests \
                                test_hub.py \
                                test_install.py \
                                test_extensions.py
```

This will run the tests in the three files against the same installation of TLJH and report errors.

If you would like to run the tests with a custom pip spec for the bootstrap script, you can use the `--bootstrap-pip-spec` parameter:

```
.circleci/integration-test.py run-test <name-of-run> <test-file-names> \
   --bootstrap-pip-spec="git+https://github.com/your-username/the-littlest-jupyterhub.
↪git@branch-name"
```

### 6.1.4 TLJH Plugins

TLJH plugins are the official way to make customized 'spins' or 'stacks' with TLJH as the base. For example, the earth sciences community can make a plugin that installs commonly used packages, set up authentication and pre-download useful datasets. The mybinder.org community can make a plugin that gives you a single-node, single-repository mybinder.org. Plugins are very powerful, so the possibilities are endless.

### Design

pluggy is used to implement plugin functionality. TLJH exposes specific **hooks** that your plugin can provide implementations for. This allows us to have specific hook points in the application that can be explicitly extended by plugins, balancing the need to change TLJH internals in the future with the stability required for a good plugin ecosystem.

### Writing a simple plugins

We shall try to write a simple plugin that installs a few libraries, and use it to explain how the plugin mechanism works. We shall call this plugin `tljh-simple`.

### Plugin directory layout

We recommend creating a new git repo for your plugin. Plugins are normal python packages - however, since they are usually simpler, we recommend they live in one file.

For `tljh-simple`, the repository's structure should look like:

```
tljh_simple:
 - tljh_simple.py
 - setup.py
 - README.md
 - LICENSE
```

The `README.md` (or `README.rst` file) contains human readable information about what your plugin does for your users. `LICENSE` specifies the license used by your plugin - we recommend the 3-Clause BSD License, since that is what is used by TLJH itself.

### `setup.py` - metadata & registration

`setup.py` marks this as a python package, and contains metadata about the package itself. It should look something like:

```
from setuptools import setup

setup(
    name="tljh-simple",
    author="YuviPanda",
    version="0.1",
    license="3-clause BSD",
    url='https://github.com/yuvipanda/tljh-simple',
    entry_points={"tljh": ["simple = tljh_simple"]},
    py_modules=["tljh_simple"],
)
```

This is a mostly standard `setup.py` file. `entry_points={"tljh": ["simple = tljh_simple]}` 'registers' the module `tljh_simple` (in file `tljh_simple.py`) with TLJH as a plugin.

### `tljh_simple.py` - implementation

In `tljh_simple.py`, you provide implementations for whichever hooks you want to extend.

A hook implementation is a function that has the following characteristics:

1. Has same name as the hook
2. Accepts some or all of the parameters defined for the hook
3. Is decorated with the `hookimpl` decorator function, imported from `tljh.hooks`.

The current list of available hooks and when they are called can be seen in tljh/hooks.py in the source repository.

This example provides an implementation for the `tljh_extra_user_conda_packages` hook, which can return a list of conda packages that'll be installed in users' environment from conda-forge.

```
from tljh.hooks import hookimpl

@hookimpl
def tljh_extra_user_conda_packages():
    return [
        'xarray',
        'iris',
        'dask',
    ]
```

### Publishing plugins

Plugins are python packages and should be published on PyPI. Users can also install them directly from GitHub - although this is not good long term practice.

The python package should be named `tljh-<pluginname>`.

### List of known plugins

If you are looking for a way to extend or customize your TLJH deployment, you might want to look for existing plugins.

Here is a non-exhaustive list of known TLJH plugins:

- tljh-pangeo: TLJH Plugin for setting up the Pangeo Stack
- tljh-voila-gallery: TLJH plugin that installs a gallery of Voilà dashboards
- tljh-repo2docker: TLJH plugin to build multiple user environments with repo2docker.

If you have authored a plugin, please open a PR to add it to this list!

We also recommend adding the `tljh-plugin` topic to the GitHub repository to make it more discoverable: https://github.com/topics/tljh-plugin

## 6.1.5 Code Review guidelines

This document outlines general guidelines to follow when you are making or reviewing a Pull Request.

### Have empathy

We recommend reading On Empathy & Pull Requests and How about code reviews to learn more about being empathetic in code reviews.

### Write documentation

If your pull request touches any code, you must write or update documentation for it. For this project, documentation is a lot more important than the code. If a feature is not documented, it does not exist. If a behavior is not documented, it is a bug.

Do not worry about having perfect documentation! Documentation improves over time. The requirement is to have documentation before merging a pull request, not to have *perfect* documentation before merging a pull request. If you are new and not sure how to add documentation, other contributors will be happy to guide you.

See *Writing documentation* for guidelines on writing documentation.

### Write tests

If your pull request touches any code, you must write unit or integration tests to exercise it. This helps validate & communicate that your pull request works the way you think it does. It also makes sure you do not accidentally break other code, and makes it harder for future pull requests to break the code added in your pull request.

Since TLJH is a distribution that integrates many JupyterHub components, integration tests provide more value for effort than unit tests do. Unit tests are easier to write & faster to run, so if the code being changed feels exhaustively unit-testable, write unit tests too. When in doubt, add more tests.

If you are unsure what kind of tests to add for your pull request, other contributors to the repo will be happy to help guide you!

See *Testing TLJH* for guidelines on writing tests.

## 6.1.6 Environments & Packages

TLJH installs packages from different sources during installation. This document describes the various sources and how to upgrade versions of packages installed.

### Python Environments

TLJH sets up two python environments during installation.

1. **Hub Environment**. JupyterHub, authenticators, spawners, TLJH plugins and the TLJH configuration management code is installed into this environment. A venv is used, primarily since conda does not support ARM CPUs and we'd like to support the RaspberryPI someday. Admins generally do not install custom packages in this environment.
2. **User Environment**. Jupyter Notebook, JupyterLab, nteract, kernels, and packages the users wanna use (such as numpy, scipy, etc) are installed here. A conda environment is used here, since a lot of scientific packages are available from Conda. `pip` is still used to install Jupyter specific packages, primarily because most notebook extensions are still available only on PyPI. Admins can install packages here for use by all users.

### Python package versions

In `installer.py`, most Python packages have a version specified. This can be upgraded freely whenever needed. Some of them have version checks in `integration-tests/test_extensions.py`, so those might need updating too.

### Apt packages

Base operating system packages, including Python itself, are installed via `apt` from the base Ubuntu repositories. The one exception to this is nodejs, which is installed from the nodesource apt repository. The Ubuntu provided version of nodejs is usually too old.

We generally do not pin versions of packages provided by apt, instead just using the latest versions provided by Ubuntu.