JupyterHub Server Setup Guide

Using JupyterHub with Docker Containers, Authentication Services and NbGrader

Requirements

1. Server with a fresh copy of Ubuntu 16.04 Server installed.
2. Root access to your server.
3. Internet access on server with all ports open.
4. Static ip on server.
5. Domain name that points to your server's static ip address, with url forwarding such that www. versions of your domain are redirected to the non www. versions, for example www.example.com is forwarded to example.com.
6. A computer that you can use to ssh into your server.

Useful Resources

JupyterHub Documentation	Docker Documentation	NbGrader Documentation

Python3 Documentation	NumPy & SciPy Documentation	Matplotlib Documentation

PyCav Github

Preliminary setup

In this section you will complete the basic set up required for any new server.

1. If your computer is running windows please download putty.exe and puttygen.exe from here if however you are running Linux, or Mac OS then you can ssh into your server from the terminal.
2. On Mac and Linux SSH into your server by running the command ssh root@**[ip]** in your terminal, replacing **[ip]** with your servers ip address. On Windows you can carry out the same process using the putty gui.
3. Your ssh client will then prompt you to input the server's root password, input the root users password to gain access to your server.
4. It is good practice to change the root users password to something more secure, you can do this by running the command

   ```
   passwd
   ```

   and following the on screen instructions.
5. Now update your software sources and software packages to their latest version by running

   ```
   apt-get -y update && apt-get -y upgrade
   ```

   in your ssh client.
6. You now need to create a new user on the server as remaining logged in as root all of the time is a bad idea. You can create a new user using the command below.

```
adduser [username]
```

into your client and replacing [**username**] with the name of the user you wish to create. You will then be prompted for various things relating to the new user, follow these on screen instructions.

7. You must now give this new user sudo access which stands for super user do, this allows the new user to execute commands as the root user. You can do this using the command

   ```
   usermod -aG sudo [username]
   ```

   where [**username**] is replaced by the name of the user that you just created.

8. To aid security you need to prevent unauthorized computers from logging in to your server. To do this you must create an ssh key on your local machine. To do this on Mac or Linux open a local terminal and run the command

   ```
   ssh-keygen
   ```

   and save in the default location. To view the generated key type in the same local terminal the command

   ```
   cat ~/.ssh/id_rsa.pub
   ```

   To do this on Windows open the file named puttygen.exe that was downloaded earlier and use that to generate and view your machine's ssh key. **Note this key down somewhere that is easily accessible.**

9. Switch back to your ssh client and input the following two commands to create and set permissions for the folder containing the authorised keys.

   ```
   runuser -l [username] -c 'mkdir ~/.ssh'

   runuser -l [username] -c 'chmod 700 ~/.ssh'
   ```

   Making sure to replace [**username**] with the name of the user that you created earlier.

10. You will now need to authorise your computer so that it can access the server using the generated key, run the command below in your ssh client making sure to replace [**public-key**] with the key that was generated and [**username**] with the name of the user you created earlier.

    ```
    runuser -l [username] -c 'echo [public-key] > ~/.ssh/authorized_keys'

    runuser -l [username] -c 'chmod 600 ~/.ssh/authorized_keys'
    ```

11. Make sure you can log in using your public key before running the following commands as they may prevent access to the server if you haven't set up your public-key correctly. If you can log in then run the commands below to remove the ability to log in using passwords (only machines that have authorised public-keys will be able to log in) and to also remove the ability for people to log in as root to your server remotely. These steps will help to prevent unauthorised users from accessing your server.

    ```
    sed -i -- 's/PasswordAuthentication yes/PasswordAuthentication no/g' /etc/ssh/sshd_config

    sed -i -- 's/PermitRootLogin yes/PermitRootLogin no/g' /etc/ssh/sshd_config

    systemctl reload sshd
    ```

12. This next step will enable a basic firewall for your server and allow ssh connections to pass through it. Run the following commands to set this up.

    ```
    ufw allow OpenSSH

    ufw enable

    ufw status
    ```

13. You now need to install the rest of the LAMP software stack which includes Linux, Apache, MySQL and PHP. You already have Linux installed on your server, so you now need to install the next part the http server

software Apache. You will also need to allow incoming connections to Apache to pass through your firewall. These steps are easily carried out by running the following two commands.

```
apt-get -y install apache2
```

```
ufw allow in "Apache Full"
```

You can print the status of Apache by running the command below (although it doesn't need to be run to proceed).

```
systemctl status apache2
```

You must now install the database software MySQL, on screen prompts will appear after running each of the commands below, make sure to follow the on screen instructions that appear.

```
apt-get -y install mysql-server
```

```
mysql_secure_installation
```

Now to install the final component, the scripting language PHP.

```
apt-get -y install php libapache2-mod-php php-mcrypt php-mysql
```

14. The next four commands will reorder the priority of files that apache uses as your server's homepage and then restart the apache server so that this change takes effect, by default index.html in the folder /var/www/html/ is your server's default homepage, these commands will change it so that index.php (same directory) is the default homepage.

```
sed -i -- 's/index.php/temp.html/g' /etc/apache2/mods-enabled/dir.conf
```

```
sed -i -- 's/index.html/index.php/g' /etc/apache2/mods-enabled/dir.conf
```

```
sed -i -- 's/temp.html/index.html/g' /etc/apache2/mods-enabled/dir.conf
```

```
systemctl restart apache2
```

15. The command below will now create a sample homepage written in php that prints the words Hello World!

```
echo $'<?php\necho \'Hello World!\';?>' > /var/www/html/index.php
```

Test this out by directing your browser to your domain name. You should see a white page with the words Hello World! on it.

16. Now you must configure secure access (https) to your server by generating your own SSL certificates using the letsencrypt software package. First you should install letsencrypt by running

```
apt-get -y install python-letsencrypt-apache
```

, then you must run the following command making sure to replace [**domain**] with the domain name (in the format example.com) that points to your server's ip address.

```
letsencrypt --apache -d [domain]
```

After inputting this command a prompt will appear, follow the on screen instructions and select secure (to permit https access only) when prompted.

The certificates this software generates are only valid for 90 days before you must renew them, luckily you can set up a cron job that will renew them automatically. To set up a cron job that will try to renew your certificate at 4am every everyday you can run the following commands.

```
crontab -l > mycron
```

```
echo "00 04 * * * letsencrypt renew" >> mycron
```

```
crontab mycron
```

```
rm mycron
```

The commands above create a cron job that runs letsencrypt renew at 00 minutes, 04 hours, every day, every month and every day of the week.

17. The following commands will prevent users from accessing your server via its IP address and force them to access your site through https at your domain. First you need to download a python script that will alter the apache config file so that redirects are allowed, this script is provided by the PyCav project. Before this though you should back up a working copy of the config file. To carry out both of these steps just run the commands below.

```
cp /etc/apache2/apache2.conf /etc/apache2/apache2.conf.working

curl https://raw.githubusercontent.com/PyCav/Server/master/configure_apache.py \
     >> configure_apache.py

python3 configure_apache.py

rm configure_apache.py
```

Now you need to format your server's public IP address so that it is laid out in the format that Apache expects, you can use the website http://icanhazip.com to determine your server's IP address. Run the following commands to do this making sure you replace [**domain**] with your server's domain name (in the format example.com) in the final command.

```
ipformatted=$(echo "$(curl http://icanhazip.com)" | sed -s 's/[.]/''\\.''/g')

echo "RewriteCond %{HTTP_HOST} ^""$ipformatted" >> /var/www/html/.htaccess

rm ipformatted

echo "RewriteRule (.*) https://[domain]/$1 [R=301,L]" >> /var/www/html/.htaccess
```

You must now restart Apache for these changes to take effect.

```
a2enmod rewrite

systemctl restart apache2
```

18. Now on to the final part of preliminary setup, installing useful libraries. You should first install git so that you can clone git repos.

```
apt-get -y install git
```

It will also be useful to install the program htop which is a visual tool for managing running processes on your server.

```
apt-get install htop
```

Another useful piece of software is the python tool pip, which allows you to install python libraries and all of their dependencies very easily using commands such as "pip3 install numpy". To set up pip for both python2 and python3 you need to run the three commands below.

```
apt-get -y install python3-pip python-pip

pip3 install --upgrade pip

pip install --upgrade pip
```

Finally you should install the javascript libraries npm, nodejs and mathjax using the command below.

```
apt-get -y install npm nodejs nodejs-legacy libjs-mathjax
```

JupyterHub and Docker

Warning It is highly recommended that you do not edit the folder structure of the server directory as this will prevent your JupyterHub server from functioning correctly. It is also recommended that you that you clone all repositories to the specified location as doing otherwise may cause issues, the expected layout of folders is shown in the tree diagram below.

```
parent/
      --users/

      --demos/

      --data/

      --investigations/

      --server/
```

In this next section you will be instructed on how to set up a JupyterHub Server that isolates users using Docker containers.

1. Firstly you should make a new directory where all of the server files and each JupyterHub user's home directories live called public.

   ```
   mkdir /home/public

   mkdir /home/public/users

   chmod a+rxw -R /home/public
   ```

2. Now you need to install JupyterHub, Docker and all of the required dependencies. To do this run the following commands.

   ```
   apt-get -y install docker.io

   npm install -g configurable-http-proxy

   pip3 install --upgrade jupyterhub

   pip3 install --upgrade notebook

   pip3 install --upgrade dockerspawner netifaces
   ```

3. You will now need to allow users access to JupyterHub and Docker through your server's firewall, the default ports used are 8000 and 8081 (these can be changed in the jupyterhub_config.py file if you wish), you can allow access to ports 8000 and 8081 using the following commands (if you decide to use different ports change the commands below to the ports you want to use).

   ```
   ufw allow 8000

   ufw allow 8081
   ```

4. This next step sets up the image that Docker containers will use. This step can be customised to meet your needs. Docker images are images that are used to create containers with specific software installed. To install the PyCav docker image (recommended) you should run the following command (note this is a large download several GBs in size and may take some time to complete).

   ```
   docker pull jordanosborn/pycav
   ```

   If you need to customise what libraries are installed you can edit the Dockerfile and build the image yourself by following the steps below making sure to name your image by replacing [**image**] with the name you wish

to use (note build will download several GBs of data and will then build the image this may take a long time). If you need help with editing this file please use the Dockerfile Docs.

```
git clone https://github.com/PyCav/jupyterhub.git /home/public/Dockerfile

nano jupyterhub/Dockerfile

docker -t build docker build -t [image]:latest /home/public/Dockerfile/.
```

5. Finally you should download the PyCav server scripts repo, this contains a jupyterhub_config.py file, and several shell scripts that will help you to manage your installation. To download and make all shell scripts executable run the commands below.

   ```
   git clone https://github.com/pycav/server.git /home/public/server

   chmod a+x /home/public/server/*.sh

   chmod a+x /home/public/server/cron/*.sh

   chmod a+x /home/public/server/global/*.sh
   ```

   Warning If you have decided to download these scripts into a different directory please run the script below (replacing /home/public/ wherever it appears in setcustomparent.sh with the path to the parent directory you cloned the server repo into) if however you have used the default directory you can ignore this step.

   ```
   /home/public/setcustomparent.sh
   ```

   To tell JupyterHub to use a custom image you need to customise the jupyterhub_config.py file. If you are using the default image you **do not** need to run this step, if however you are using a custom image run the following command making sure to replace **[image]** with the name you used for your custom image.

   ```
   sed -i -- 's/jordanosborn\/pycav/[image]/g' /home/public/server/jupyterhub_config.py
   ```

   If you have forgotten what you have named your image you can run the command below (must be root) to list all currently installed images.

   ```
   docker images
   ```

Authentication

This section will describe how to set up a variety of authentication methods (Raven, GitHub, Local User), which you can use to prevent unauthorised users from accessing your JupyterHub server.

Raven

1. Firstly you need to install the PyCav Raven Authenticator plugin, this can be done by running the command below.

   ```
   pip3 install --upgrade git+git://github.com/PyCav/jupyterhub-raven-auth.git
   ```

2. Before Raven authentication is available you must enable it in the jupyterhub_config.py file to do this you should run the following command.

   ```
   sed -i -- 's/raven = False/raven = True/g' /home/public/server/jupyterhub_config.py
   ```

GitHub

1. Firstly you need to install oauthenticator as GitHub uses oauth to authorise users.

   ```
   pip3 install --upgrade oauthenticator
   ```

2. Secondly you need to set up an oauth application on GitHub, to do this log in to GitHub and Create a GitHub Oauth Application. Use a sensible application name , set the homepage url as https://[**domain**]:[**jupyterhubport**] and set the callback url as https://[**domain**]:[**jupyterhubport**]/hub/oauth_callback , replacing [**domain**] with the domain name of your server (in the format example.com) and also [**jupyterhubport**] with the port you decided to run your JupyterHub server on (default 8000).
3. Finally, before GitHub authentication is available you must enable it in the jupyterhub_config.py file to do this you should run the following command

```
sed -i -- 's/github = False/github = True/g' /home/public/server/jupyterhub_config.py
```

Local User

1. If however you would like to authenticate using system users, then all you need to do is run the command below

```
sed -i -- 's/local = False/local = True/g' /home/public/server/jupyterhub_config.py
```

NbGrader

This section will discuss how to set up NbGrader up on your server, so that you can create assignments for users to complete and hand in. It will also show you how to set up NbGrader so that assignments are automatically marked.

1. Firstly you need to install nbgrader.

```
pip3 install --upgrade nbgrader
```

to do

Final Configuration

This section will show you how to customise your installation, how to update containers, how to set up periodic backups of user data and how to kill resource draining containers.

Updating Containers

The easiest way to update containers is to first delete them all, this is okay as data created by the user is seperated from the container. After this you need to delete the old docker image and download/build the newer version. You must schedule a maintenance period where the server will remain offline during the update, this involves informing users of this scheduled down time and giving them appropriate time to prepare. Three scripts have been provided the first one stops all containers and then deletes them, the second will take the JupyterHub server offline, call the first script, delete the old Docker Image and then download/build the new image. The third one will trigger a build of the default image on hub.docker.com You need to move these scripts so that they will be on your server's path, to do this run the commands below.

```
cp /home/public/server/global/removecontainers.sh /usr/local/bin/removecontainers

cp /home/public/server/global/updatecontainers.sh /usr/local/bin/updatecontainers

cp /home/public/server/global/triggerbuild.sh /usr/local/bin/triggerbuild
```

If you are using the default image it is suggested that you trigger a build using the command below before updating the image (note this build will take roughly 30 minutes) you can check the build status here.

```
triggerbuild
```

You can now update the containers by running **updatecontainers** (as root) in your server's terminal. The updatecontainers script has 2 flags that can be activated independently three examples of running the updatecontainers script are shown below.

```
updatecontainers
```

```
updatecontainers -b [image] [path_to_Dockerfile]
```

```
updatecontainers -p [image]
```

The first command updates using the default jordanosborn/pycav image.

The second command updates by building a new image from a custom Dockerfile (replace [**image**] with the name of the image you wish to create and [**path_to_Dockerfile**] with the path to your custom Dockerfile).

Note if you run updatecontainers using just the -b flag with no further arguments you will build the default Docker image locally.

The final command will pull the named image (replace [**image**] with the name of the image you wish to use) from DockerHub.

Warning If you use a new image with a different name you will need to update your **jupyterhub_config.py** file to reflect this change.

Backing Up Containers

All user data is by default stored in the directory /home/public/users which contains sub directories for each user named after that user's username. Backing up is therefore as simple as copying the users directory to a backup hard disk. A script has been created that will back up this folder to a mounted hard disk. It creates a tar.gz archive with a name modified according to the date the folder was backed up (default is every monday at 04:30). **Before carrying out the steps below you must mount the hard disk you want to back up to to some directory on your computer (Help).** Run the following commands replacing [**mountpath**] with the full path to the mount directory (and folder if you require) of your backup hard disk (in the format /media/backup/).

```
backup_path=$(echo "[mountpath]" | sed -s 's/[/]/''\\\/''/g')

sudo sed -i -- 's/\/media\/backup\//'$backup_path'/g' /home/public/cron/backup.sh

rm backup_path

crontab -l > mycron

echo "30 04 * * 1 /home/public/server/cron/backup.sh" >> mycron

rm mycron
```

JupyterHub Configuration

The final steps to configuring JupyterHub are to customise the jupyterhub_config.py file. These include replacing the word website with your server's domain, 8000 with the port your server is running JupyterHub on and generating a proxy authorisation token. These three tasks can be completed by running the 4 commands below, making sure to change [**domain**] to your server's domain name (in the form example.com) and [**port**] to the port number your JupyterHub instance is running on (default is 8000).

```
sudo sed -i -- 's/website/[domain]/g' /home/public/server/jupyterhub_config.py

sudo sed -i -- 's/8000/[port]/g' /home/public/server/jupyterhub_config.py

proxy_key=$(openssl rand -hex 32)

sudo sed -i -- "s/auth_key=''/auth_key='"$proxy_key"'/g" /home/public/server/jupyterhub_config.py
```

You can also customise further parts of your JupyterHub installation in this file. Including login logos, application logos and shared directories etc. This document will not explain the details of how to do this, please refer to the JupyterHub Docs if you need more information.

NbGrader Configuration

Run the commands below to configure the environment variables that NbGrader uses for configuration making sure to replace [**username**] with the username of an admin on your JupyterHub server.

[username] is actually admin in jupyterhub must change command?

```
echo "CONFIGPROXY_AUTH_TOKEN='""$proxy_key""'" >> /etc/environment

JPY_tmp=$(jupyterhub token --db=sqlite:///home/public/server/jupyterhub.sqlite \
     -f /home/public/server/jupyterhub_config.py [username])

echo "JPY_API_TOKEN='""$JPY_tmp""'" >> /etc/environment

source /etc/environment
```

Killing Resource Draining Containers

A script was written in python to check if containers are idle or are using too much of the server's computing power. The script will kill any containers that are exceeding maximum usage thresholds (USER CPU usage) for a certain period of time and will also kill any containers that fall below a minimum usage threshold (USER CPU usage) for a certain period of time. These are all user definable constants in the python script. Thresholds are effectively percentage usage of CPU for x86 based hardware. You should modify the python script located at **/home/public/server/python/killcontainers.py** (if using default directories) to fit these parameters to your needs (idle Timeout, maxing Timeout, time increment, idle cpu threshold, maxing cpu threshold).

(Optional) Setting up a Basic Webpage

The PyCav server repo that you cloned earlier contains a basic index page and stats page. These can be used as a basic landing page for users accessing your website. To make these pages accessible at your domain you need to move them to the directory Apache uses to serve content from (/var/www/html/), you can do this by running the commands below. Make sure you replace [**domain**] with the domain name of your server (in the format example.com) and [**port**] with the port your JupyterHub server is running on (default is 8000).

```
sed -i -- 's/8000/[port]/g' /home/public/server/webpages/index.php

sed -i -- 's/website/[domain]/g' /home/public/server/webpages/index.php

mv /home/public/server/webpages/* /var/www/html/

echo "ErrorDocument 404 /notfound.php" >> /var/www/html/.htaccess
```

(Optional) Install PyCav Notebooks

If you wish to install the PyCav Notebooks so that they are viewable by your users and set them up so that they are automatically updated (at 04:10 am everyday) you need to run the following commands.

```
git clone https://github.com/pycav/demos.git /home/public/demos

git clone https://github.com/pycav/data.git /home/public/data

git clone https://github.com/pycav/investigations.git /home/public/investigations

sudo sed -i -- 's/#demos_//g' /home/public/server/jupyterhub_config.py
```

```
/home/public/server/cron/indexgen.sh

crontab -l > mycron

echo "10 04 * * * rm -R /home/public/cron/updatenotebooks.sh" >> mycron

crontab mycron

rm mycron
```

Updating Server Scripts

The PyCav server repo provides a script that updates all of the server scripts including itself and those that have been added to /usr/local/bin. However it does **not** update your server's webpages using the default webpages in the server repo. It will automatically customise the newly updated files to use the previous values where it can, see notes below to see if you should use the update script.

Note this script will **DELETE** and redownload the entire server repo, which means you should not use it if you have manually customised any of the files in the server directory as you will lose any changes you have made. Unfortunately if you have customised the files in the server directory you will need to manually update scripts as they are uploaded on to GitHub.

Note the script also assumes that Raven is being used as an authentication method and that you want users on your JupyterHub to be able to access the PyCav provided notebooks. You can however use the script and then modify your **jupyterhub_config.py** file (in the server folder) manually to meet your needs (see JupyterHub Docs.

Note your JupyterHub server will be taken offline while updating the scripts, you can start it back up after they complete by running the command **startserver** (as root).

If you are sure you meet the use cases for this update script then you can set it up by running the following commands.

```
sed -i -- 's/domain/[site_name]/g' /home/public/server/global/updatescripts.sh

sed -i -- 's/PORT/[port]/g' /home/public/server/global/updatescripts.sh

cp /home/public/server/global/updatescripts_subscript.sh /usr/local/bin/updatescripts_subscript

cp /home/public/server/global/updatescripts.sh /usr/local/bin/updatescripts
```

To update your server scripts you just need to run the command below (as root).

```
updatescripts
```

Running The Server

First you should add the startserver.sh and killserver.sh scripts to your path, you can do this by running the following commands.

```
cp /home/public/server/global/startserver.sh /usr/local/bin/startserver

cp /home/public/server/global/killserver.sh /usr/local/bin/killserver
```

You should now switch to the non-root user you created earlier, you can do this by running the command below (making sure to replace [**username**] with the username of the user you created earlier).

```
su [username]
```

You now need to run the command below which ensures that all the set environment variables and shell scripts are accessible to the user you created earlier.

```
source /etc/environment
```

```
source ~/.bashrc
```

With the scripts added to your path you can start the server in any directory by running the command **(as root using sudo)** below.

```
sudo startserver
```

This command runs the scripts fixpermissions.sh which gives users write access to their home directories, killcontainers.sh which kills resource intensive or idle containers (can be turned off with the flag -nk), and also starts the JupyterHub server.

To close the JupyterHub server you need to send a SIGINT to the process, you can do this by pressing **CTRL-C** with your ssh-client's window focussed.

You can also start the server in the background by running (as root using sudo) the following command.

```
screen sudo startserver
```

To detach from the process's running screen press **CTRL-A** followed by **CTRL-D** you can now safely exit your ssh-client without interrupting the JupyterHub process. To reattach to the JupyterHub screen run the following command (as root using sudo).

```
screen -r
```

You can also kill your JupyterHub server by running the command below (as root using sudo).

```
sudo killserver
```

To access your JupyterHub server you should direct your web browser to https://[**domain**]:[**port**] (replacing [**domain**] with your server's domain name and [**port**] with the port your JupyterHub server is running on).

This concludes the PyCav JupyterHub setup guide, please visit pycav.org if you would like to learn more about the PyCav project.

v1.2.1 PyCav 2016 - Jordan Osborn