**Running a notebook server**

The [Jupyter notebook](http://jupyter-notebook.readthedocs.io/en/stable/notebook.html) web application is based on a server-client structure. The notebook server uses a [two-process kernel architecture](https://ipython.readthedocs.io/en/stable/overview.html" \l "ipythonzmq" \o "(in IPython v6.2.1)) based on [ZeroMQ](http://zeromq.org), as well as [Tornado](http://www.tornadoweb.org) for serving HTTP requests.

Note

By default, a notebook server runs locally at 127.0.0.1:8888 and is accessible only from *localhost*. You may access the notebook server from the browser using *http://127.0.0.1:8888*.

This document describes how you can [secure a notebook server](http://jupyter-notebook.readthedocs.io/en/stable/public_server.html" \l "notebook-server-security) and how to [run it on a public interface](http://jupyter-notebook.readthedocs.io/en/stable/public_server.html" \l "notebook-public-server).

Important

**This is not the multi-user server you are looking for**. This document describes how you can run a public server with a single user. This should only be done by someone who wants remote access to their personal machine. Even so, doing this requires a thorough understanding of the set-ups limitations and security implications. If you allow multiple users to access a notebook server as it is described in this document, their commands may collide, clobber and overwrite each other.

If you want a multi-user server, the official solution is [JupyterHub](https://jupyterhub.readthedocs.io/en/latest/). To use JupyterHub, you need a Unix server (typically Linux) running somewhere that is accessible to your users on a network. This may run over the public internet, but doing so introduces additional [security concerns](https://jupyterhub.readthedocs.io/en/latest/getting-started/security-basics.html).

**Securing a notebook server**

You can protect your notebook server with a simple single password by configuring the NotebookApp.password setting in jupyter\_notebook\_config.py.

**Prerequisite: A notebook configuration file**

Check to see if you have a notebook configuration file, jupyter\_notebook\_config.py. The default location for this file is your Jupyter folder in your home directory, ~/.jupyter.

If you don’t already have one, create a config file for the notebook using the following command:

$ jupyter notebook --generate-config

**Preparing a hashed password**

As of notebook version 5.0, you can enter and store a password for your notebook server with a single command. **jupyter notebook password** will prompt you for your password and record the hashed password in your jupyter\_notebook\_config.json.

$ jupyter notebook password

Enter password: \*\*\*\*

Verify password: \*\*\*\*

[NotebookPasswordApp] Wrote hashed password to /Users/you/.jupyter/jupyter\_notebook\_config.json

You can prepare a hashed password manually, using the function notebook.auth.security.passwd():

In [1]: from notebook.auth import passwd

In [2]: passwd()

Enter password:

Verify password:

Out[2]: 'sha1:67c9e60bb8b6:9ffede0825894254b2e042ea597d771089e11aed'

Caution

passwd() when called with no arguments will prompt you to enter and verify your password such as in the above code snippet. Although the function can also be passed a string as an argument such as passwd('mypassword'), please **do not** pass a string as an argument inside an IPython session, as it will be saved in your input history.

**Adding hashed password to your notebook configuration file**

You can then add the hashed password to your jupyter\_notebook\_config.py. The default location for this file jupyter\_notebook\_config.py is in your Jupyter folder in your home directory, ~/.jupyter, e.g.:

c.NotebookApp.password = u'sha1:67c9e60bb8b6:9ffede0825894254b2e042ea597d771089e11aed'

**Using SSL for encrypted communication**

When using a password, it is a good idea to also use SSL with a web certificate, so that your hashed password is not sent unencrypted by your browser.

Important

Web security is rapidly changing and evolving. We provide this document as a convenience to the user, and recommend that the user keep current on changes that may impact security, such as new releases of OpenSSL. The Open Web Application Security Project ([OWASP](https://www.owasp.org)) website is a good resource on general security issues and web practices.

You can start the notebook to communicate via a secure protocol mode by setting the certfile option to your self-signed certificate, i.e. mycert.pem, with the command:

$ jupyter notebook --certfile=mycert.pem --keyfile mykey.key

Tip

A self-signed certificate can be generated with openssl. For example, the following command will create a certificate valid for 365 days with both the key and certificate data written to the same file:

$ openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout mykey.key -out mycert.pem

When starting the notebook server, your browser may warn that your self-signed certificate is insecure or unrecognized. If you wish to have a fully compliant self-signed certificate that will not raise warnings, it is possible (but rather involved) to create one, as explained in detail in this [tutorial](https://arstechnica.com/information-technology/2009/12/how-to-get-set-with-a-secure-sertificate-for-free/). Alternatively, you may use [Let’s Encrypt](https://letsencrypt.org) to acquire a free SSL certificate and follow the steps in [Using Let’s Encrypt](http://jupyter-notebook.readthedocs.io/en/stable/public_server.html#using-lets-encrypt) to set up a public server.

**Running a public notebook server**

If you want to access your notebook server remotely via a web browser, you can do so by running a public notebook server. For optimal security when running a public notebook server, you should first secure the server with a password and SSL/HTTPS as described in [Securing a notebook server](http://jupyter-notebook.readthedocs.io/en/stable/public_server.html#notebook-server-security).

Start by creating a certificate file and a hashed password, as explained in [Securing a notebook server](http://jupyter-notebook.readthedocs.io/en/stable/public_server.html#notebook-server-security).

If you don’t already have one, create a config file for the notebook using the following command line:

$ jupyter notebook --generate-config

In the ~/.jupyter directory, edit the notebook config file, jupyter\_notebook\_config.py. By default, the notebook config file has all fields commented out. The minimum set of configuration options that you should uncomment and edit in jupyter\_notebook\_config.py is the following:

# Set options for certfile, ip, password, and toggle off

# browser auto-opening

c.NotebookApp.certfile = u'/absolute/path/to/your/certificate/mycert.pem'

c.NotebookApp.keyfile = u'/absolute/path/to/your/certificate/mykey.key'

# Set ip to '\*' to bind on all interfaces (ips) for the public server

c.NotebookApp.ip = '\*'

c.NotebookApp.password = u'sha1:bcd259ccf...<your hashed password here>'

c.NotebookApp.open\_browser = False

# It is a good idea to set a known, fixed port for server access

c.NotebookApp.port = 9999

You can then start the notebook using the jupyter notebook command.

**Using Let’s Encrypt**

[Let’s Encrypt](https://letsencrypt.org) provides free SSL/TLS certificates. You can also set up a public server using a [Let’s Encrypt](https://letsencrypt.org) certificate.

[Running a public notebook server](http://jupyter-notebook.readthedocs.io/en/stable/public_server.html#notebook-public-server) will be similar when using a Let’s Encrypt certificate with a few configuration changes. Here are the steps:

1. Create a [Let’s Encrypt certificate](https://letsencrypt.org/getting-started/).
2. Use [Preparing a hashed password](http://jupyter-notebook.readthedocs.io/en/stable/public_server.html#hashed-pw) to create one.
3. If you don’t already have config file for the notebook, create one using the following command:
4. $ jupyter notebook --generate-config

4. In the ~/.jupyter directory, edit the notebook config file, jupyter\_notebook\_config.py. By default, the notebook config file has all fields commented out. The minimum set of configuration options that you should to uncomment and edit in jupyter\_notebook\_config.py is the following:

# Set options for certfile, ip, password, and toggle off

# browser auto-opening

c.NotebookApp.certfile = u'/absolute/path/to/your/certificate/fullchain.pem'

c.NotebookApp.keyfile = u'/absolute/path/to/your/certificate/privkey.pem'

# Set ip to '\*' to bind on all interfaces (ips) for the public server

c.NotebookApp.ip = '\*'

c.NotebookApp.password = u'sha1:bcd259ccf...<your hashed password here>'

c.NotebookApp.open\_browser = False

# It is a good idea to set a known, fixed port for server access

c.NotebookApp.port = 9999

You can then start the notebook using the jupyter notebook command.

Important

**Use ‘https’.** Keep in mind that when you enable SSL support, you must access the notebook server over https://, not over plain http://. The startup message from the server prints a reminder in the console, but *it is easy to overlook this detail and think the server is for some reason non-responsive*.

**When using SSL, always access the notebook server with ‘https://’.**

You may now access the public server by pointing your browser to https://your.host.com:9999 where your.host.com is your public server’s domain.

**Firewall Setup**

To function correctly, the firewall on the computer running the jupyter notebook server must be configured to allow connections from client machines on the access port c.NotebookApp.port set in jupyter\_notebook\_config.py to allow connections to the web interface. The firewall must also allow connections from 127.0.0.1 (localhost) on ports from 49152 to 65535. These ports are used by the server to communicate with the notebook kernels. The kernel communication ports are chosen randomly by ZeroMQ, and may require multiple connections per kernel, so a large range of ports must be accessible.

**Running the notebook with a customized URL prefix**

The notebook dashboard, which is the landing page with an overview of the notebooks in your working directory, is typically found and accessed at the default URL http://localhost:8888/.

If you prefer to customize the URL prefix for the notebook dashboard, you can do so through modifying jupyter\_notebook\_config.py. For example, if you prefer that the notebook dashboard be located with a sub-directory that contains other ipython files, e.g. http://localhost:8888/ipython/, you can do so with configuration options like the following (see above for instructions about modifying jupyter\_notebook\_config.py):

c.NotebookApp.base\_url = '/ipython/'

**Embedding the notebook in another website**

Sometimes you may want to embed the notebook somewhere on your website, e.g. in an IFrame. To do this, you may need to override the Content-Security-Policy to allow embedding. Assuming your website is at *https://mywebsite.example.com*, you can embed the notebook on your website with the following configuration setting in jupyter\_notebook\_config.py:

c.NotebookApp.tornado\_settings = {

'headers': {

'Content-Security-Policy': "frame-ancestors https://mywebsite.example.com 'self' "

}

}

When embedding the notebook in a website using an iframe, consider putting the notebook in single-tab mode. Since the notebook opens some links in new tabs by default, single-tab mode keeps the notebook from opening additional tabs. Adding the following to ~/.jupyter/custom/custom.js will enable single-tab mode:

define(['base/js/namespace'], function(Jupyter){

Jupyter.\_target = '\_self';

});

**Known issues**

**Proxies**

When behind a proxy, especially if your system or browser is set to autodetect the proxy, the notebook web application might fail to connect to the server’s websockets, and present you with a warning at startup. In this case, you need to configure your system not to use the proxy for the server’s address.

For example, in Firefox, go to the Preferences panel, Advanced section, Network tab, click ‘Settings…’, and add the address of the notebook server to the ‘No proxy for’ field.

**Docker CMD**

Using jupyter notebook as a [Docker CMD](https://docs.docker.com/engine/reference/builder/#cmd) results in kernels repeatedly crashing, likely due to a lack of [PID reaping](https://blog.phusion.nl/2015/01/20/docker-and-the-pid-1-zombie-reaping-problem/). To avoid this, use the [tini](https://github.com/krallin/tini) init as your Dockerfile *ENTRYPOINT*:

# Add Tini. Tini operates as a process subreaper for jupyter. This prevents

# kernel crashes.

ENV TINI\_VERSION v0.6.0

ADD https://github.com/krallin/tini/releases/download/${TINI\_VERSION}/tini /usr/bin/tini

RUN chmod +x /usr/bin/tini

ENTRYPOINT ["/usr/bin/tini", "--"]

EXPOSE 8888

CMD ["jupyter", "notebook", "--port=8888", "--no-browser", "--ip=0.0.0.0"]