# Running Jupyter Notebooks on Remote Servers

Tobias Skovgaard Jepsen Mar 4, 2019 · 6 min read

Jupyter Notebook is a staple tool in many data scientists' toolkit. As a tool, Jupyter Notebook can enhance productivity by making it easier to perform data analysis, model prototyping, and experiments in an interactive way, thus shortening the feedback loop from coding and seeing results.

processing, or learning complex models you probably need something more powerful than a laptop extra power. Maybe you're running graph convolutional networks on large graphs or doing machine translation with recurrent neural networks on large text corpora and need some more CPU cores, RAM, or a couple of GPUs. Luckily, you may be in the situation where you have these resources available on a remote server!

If your remote server has a Graphical User Interface (GUI), you are in luck. You can use remote desktop software to access the remote server and otherwise use Jupyter Notebook as your normally would on your laptop.

However, many servers do not have a GUI. If you are in this situation, you can set up your experiment by writing a Python script on your laptop, run it on a small subset of your data to verify that it can run, copy it to a remote server, and execute it from the command line. You could even set up the experiment in a notebook and export the notebook to a script using `jupyter nbconvert --to script your_notebook.ipynb`. Although this workflow certainly allows you to run your code on the remote server, you can no longer use Jupyter Notebook to, e.g., experiment with your models and visualize your results interactively. What a shame!

In this post I will show you how to run a Jupyter notebook on a remote server and how to access it on your laptop. I will also show how to setup two `bash` commands to make the whole process easier.

## Starting the Remote Notebook Server

We will use the Secure Shell Protocol (SSH) to start the Jupyter Notebook server on the remote server. SSH allows us to send commands to the remote server. The basic syntax is as follows:

```
ssh username:password@remote_server_ip command
```

The exact command you should send depends a little on your context. In my case, I share a remote server with other people have therefore not installed Jupyter in the shared environment. The first step for me is therefore to go to my project folder,

```
cd project_folder
. virtual_environment/bin/activate
jupyter notebook --no-browser --port=8889
```

I execute the `jupyter notebook` command with the `--no-browser` flag to start the Jupyter notebook with launching a browser since the remote server cannot display a browser if it doesn't have a GUI. I also change the port from the default port 8888 to port 8889 using the `--port=8889` flag. This is a personal preference; having local and remote notebooks on different ports to make it easier to see where my code is running.

To execute commands on the remote server, we run the combined command

```
nohup ssh -f username:password@remote_server_ip "cd project_folder;
. virtual_environment/bin/activate; jupyter notebook --no-browser --
port=8889"
```

Note that I have one-lined the three commands and separated them using `;` instead of line breaks. Executing this command will start the Jupyter Notebook server on port 8889 and let it run in the background. Finally, I have added the `-f` flag to the `ssh` command to push the process to the background and prepended the `nohup` command to silence all output from the process so you can continue using the terminal window. You can read more about the `nohup` command here.

## Accessing Your Notebook

You can now access the notebook typing in the url

```
remote_server_ip:8889
```

This command requires you to memorise the IP address or to bookmark the web page. However, we can make it just as easy to access the remote notebook as if it were a local notebook by using port forwarding:

The `-N` flag tells `ssh` that no remote commands will be executed. At this point we do not need to execute any remote commands. The `-f` flag pushes the `ssh` process to the background as mentioned previously. Finally, the `-L` flag specifies the port forwarding configuration using the syntax `local_server:local_port:remote_server:remote_port`. The configuration specifies that all requests sent to port `8889` on the local machine, e.g., your laptop, to port `8889` on the remote machine at `username:password@remote_server_ip`. As before, the `nohup` command has been prepended to silence the output.

The effect the above command is that you can now access the remote Jupyter Notebook server in your browser at

```
localhost:8889
```

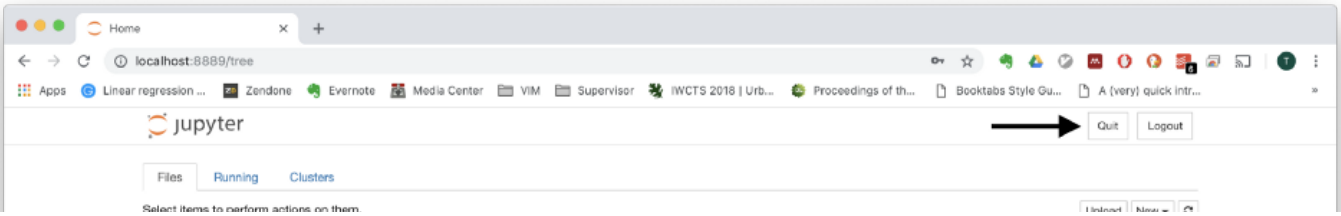
as if you ran the notebook locally.

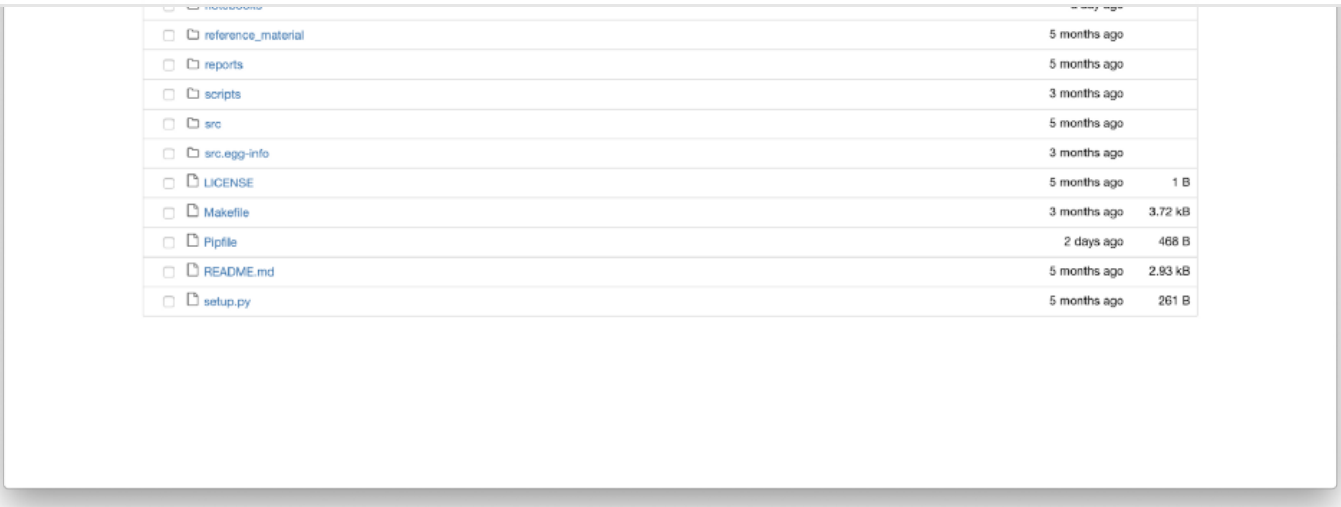
## Stopping the Remote Notebook Server

In principle, you can let the notebook server run indefinitely on the remote server (barring restarts or crashes), but you may need to stop the server, for instance to upgrade your version of `jupyter`. If you need to stop it there are two ways to do so: through the browser or through the command line.

### Through the Browser Window

In the recent versions of the Jupyter Notebook, you can find a Quit button at the top right of the browser window as indicated by the arrow in the image below. If you press it, you will have to relaunch the server again using the start-up command we saw previously.






The Quit Button

## Through the Command Line

If you are unable to upgrade to a newer version of Jupyter that has the Quit button or simply prefer working through a terminal, you can also stop the server from the command line. Jupyter has a shell command to stop notebooks:

```
jupyter notebook stop 8889
```

where `8889` is the port number. You can execute it on the remote server using the command

```
ssh username:password@remote_server_ip "jupyter notebook stop 8889"
```

Unfortunately, this command is currently bugged, but I have included it here in the hopes that it will work in the future. However, as a work-around you can instead kill the `jupyter` process instead using the command:

```
ssh username:password@remote_server_ip "pkill -u username jupyter"
```

notebook servers if you have more than one running at a time. Finally, you can of course manage the servers manually by logging on to the remote server, starting the notebook server, and keeping the terminal window open. This allows you to shutdown the notebook server using the usual `CTRL+C` keyboard command.

## Smoothening Your Workflow

Remembering all these commands can be quite cumbersome. Thankfully, we can make life easier by creating bash aliases for each of the commands. Add the following lines to your `~/.bashrc` file:

```
1  alias port_forward='nohup ssh -N -f -L localhost:8889:localhost:8889 username:password@r
2  alias remote_notebook_start='nohup ssh -f username:password@remote_server_ip "cd rne; .
3  alias remote_notebook_stop='ssh username:password@remote_server_ip "pkill -u username ju
```

**Running Jupyter Notebooks on Remote Servers** hosted with ❤ by **GitHub** **view raw**

Load the commands by typing `source .bashrc` in your terminal. You can now use the commands `remote_notebook_start` and `remote_notebook_stop` in your terminal to respectively start the remote notebook server (and setup port forwarding) and shut it down.

## Wrap-Up

In this post I have shown you how to start, access, and stop Jupyter notebooks on remote servers using bash commands and shown how to create bash aliases to make it easy to do so.

I hope that these commands can improve your data science productivity by nearly seamlessly allowing you to reap the benefits of both Jupyter notebook and any computing resources you have available on remote servers.

*Liked what you read? Consider following me on Twitter where I share papers, videos, and articles related to the practice, theory, and ethics of data science and machine learning that I find interesting, in addition to my own posts.*