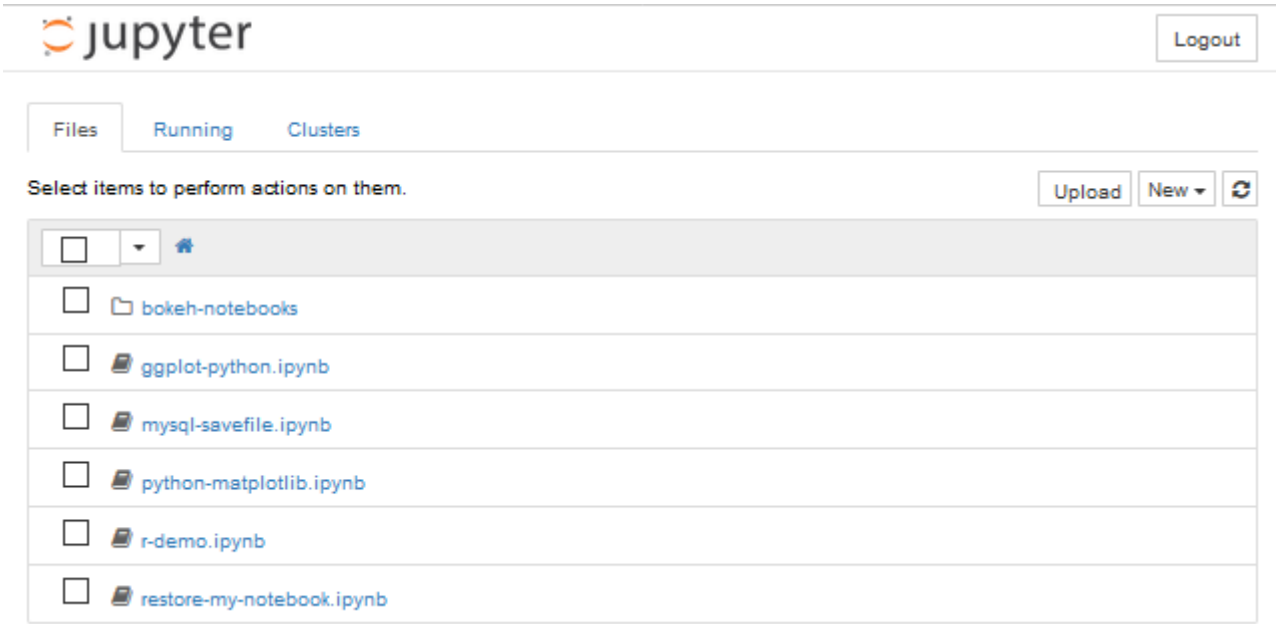
Getting to your Jupyter notebook environment

Duke provides a number of computing "containers", a sort of mini virtual computer built for a specific task, that you can create and keep for an entire semester. We'll be writing our Python code in a container built to run Python with a Jupyter Notebooks interface. There's a lot more to these containers and to the Jupyter Notebooks interface than we'll cover here, but now is not the time to go into that. Instead, we'll simply focus on using Jupyter to explore Python (and R too, if you wish...)

Below are instructions on how to create your Jupyter container. Once created, you'll be able to access it for the remainder of the semester, with an option to renew if you wish.

Creating your Jupyter notebook environment

- Navigate to https://vm-manage.oit.duke.edu/containers and log in.
- Find the entry for "**Jupyter - interactive data science and scientific computing notebooks**" and create your personal Jupyter environment.
- In just a few moments, your environment will be created and you will see the **Jupyter Console**.



The Jupyter console

The Jupyter console is where you control your Jupyter environment. It's a sort of file manager for creating, duplicating, removing, renaming, and uploading documents and folders. You'll see that your Duke Jupyter environment is created with a number of documents and one folder already.

Creating new documents

- Click the `New ▼` button and you'll see the different types of objects you can create.
  - "**Text File**" and "**Folder**" are self-evident.
  - The "**Terminal**" option opens up a Unix terminal for more advanced shell operations.
  - Under the "***Notebooks***" section you'll see options for each "kernel" installed in your environment. A "kernel" is an engine built to interpret a specific type of code. Thus our Duke Jupyter environment can interpret 5 types of code.
    - "**Bash**" executes Unix code (useful for writing scripts to run frequently used Git commands)
    - "**Julia**" is a high-performance dynamic programming language - *we won't be covering that here*.
    - "**Python 2**" executes Python v2 code, which is slightly different than v3.
    - "**Python 3**" is what we'll be using to create our Python v3 notebooks.

        - "**R**" runs R code. Yep!
- Create a new **Python 3** notebook. It will appear as a new tab in your browser.
- Under the File menu, rename your notebook as "`MyFirstNotebook`", then save your notebook.
- Go back and view your Jupyter Console page in your browser.
    - → Note that your new notebook appears as "MyFirstNotebook.ipynb". The `.ipynb` extension indicates it's an "interactive Python notebook". (Jupyter evolved from something called "interactive Python".)
    - → Also note that the icon next to it is green. This indicates that the document is associated with an active **kernel**, i.e. it is still "running". *No need to worry too much about whether a kernel is running or not, though it is good practice to "shutdown" a documents kernel if you are no longer using it. And a kernel will remain active even if its browser page is closed.*

The Jupyter notebook document

Jupyter notebook documents are much like R markdown documents in that the can contain both formatted text and executable code. The difference, as you'll soon see, is that the formatted text appears formatted (without knitting). There are other differences too, but we'll address them as needed.

Opening and editing a notebook

- Open your "MyFirstNotebook.ipynb" notebook, if not open already.
- Your new notebook should have one empty code cell in it. Type: `print("Hello World")` in it then hit the run button (`►|`), or hit `ctrl`-`Enter` with the cell active. This executes the Python code in the cell...
- Hit the `+` button to add a new cell (or hit `esc` then `b`).
- With this new cell active, change the dropdown in the menu bar that says `Code` to `Markdown`. This changes the cell from a code cell to a markdown cell.
    - Type `# This is really big font` in the cell and "run" the cell.
    - Double click the cell to edit the markdown and add two more `#` at the begging. Then "run" the cell...
- Add a new code cell to your notebook and run the code `! ls -la`
    - The `!` to start the command indicates the operation is a shell command. ("`ls -la`" *is a Unix command to list all contents of the current directory showing all attributes...*)

Saving and "shutting down" your notebook.

If you tried to close you notebook's browser tab right, it would ask confirmation to leave as you haven't saved your notebook. This is true; you need to save changes to your notebook as you would most documents you edit.

- Save your notebook by clicking the 💾 icon. Now your changes are logged.

The notebook is saved, but if you are finished, don't just close the browser tab as that leaves the notebook's kernel still running. Instead:

- From the `File` menu, select `Close and Halt`.

Now the notebook is properly closed. You can re-open the page from the Jupyter Console.

Cloning a GitHub repository

Git commands can be entered a number of ways. First you can open a new Terminal from your Jupyter console and type in commands at the Unix command prompt. Or, you can open a new Bash notebook and type Unix commands into code cells and run them there. And finally, you can run shell commands from a Python notebook by preceding the command with a `!` in a code cell.

As we may be running the same Git commands frequently, we'll enter them in a new Bash notebook:

- Create a new Bash document and name it "GitCommands"
- In the first code cell, enter the code below (replacing "`<gitName>`" with your GitHub account name:

```
git clone https://github.edu/<gitName>/Environmental_Data_Analytics
```

- Run the cell. You should be seeing returns that the repo is being cloned.
  - Check your Jupyter console and you should see a new folder called "Environmental_Data_Analytics"
- Back in the Bash notebook, add a new cell that sets your Git configuration (change values accordingly):

```
git config --global user.name "John Fay"
git config --global user.email "john.fay@duke.edu"
```

- Add a new cell that puts you into your new cloned directory:

```
cd ~/work/Environmental_Data_Analytics
pwd
```

- Add a new cell that adds the upstream repository

```
git remote add upstream https://github.com/KateriSalk/Environmental_Data_Analytics
git remove -v
```

- And finally, pull any upstream changes

```
git pull upstream master
```

Below is a figure from my completed session.

```
In [1]: git clone https://github.com/johnpfay/Environmental_Data_Analytics

Cloning into 'Environmental_Data_Analytics'...
remote: Enumerating objects: 40, done.
remote: Counting objects: 100% (40/40), done.
remote: Compressing objects: 100% (32/32), done.
remote: Total 107 (delta 8), reused 39 (delta 8), pack-reused 67
Receiving objects: 100% (107/107), 11.53 MiB | 0 bytes/s, done.
Resolving deltas: 100% (33/33), done.
Checking connectivity... done.

In [2]: git config --global user.name "John Fay"
        git config --global user.name "john.fay@duke.edu"

In [3]: cd ~/work/Environmental_Data_Analytics
        pwd

/home/jovyan/work/Environmental_Data_Analytics

In [4]: git remote add upstream https://github.com/KateriSalk/Environmental_Data_Analytics
        git remote -v

origin  https://github.com/johnpfay/Environmental_Data_Analytics (fetch)
origin  https://github.com/johnpfay/Environmental_Data_Analytics (push)
upstream        https://github.com/KateriSalk/Environmental_Data_Analytics (fetch)
upstream        https://github.com/KateriSalk/Environmental_Data_Analytics (push)

In [5]: git pull upstream master

From https://github.com/KateriSalk/Environmental_Data_Analytics
 * branch            master     -> FETCH_HEAD
 * [new branch]      master     -> upstream/master
Already up-to-date.
```

More info

For more info on Jupyter notebook, Interactive Python (or iPython), or on other places to run Python code, have a look at "A Whirlwind Tour of Python" - which is FREE and was written entirely in Jupyter notebooks!

https://jakevdp.github.io/WhirlwindTourOfPython/01-how-to-run-python-code.html