Managing Jupyter Kernels on ITaP Community Clusters

ITaP Research Computing Virtual Workshop Series

June 12, 2020

An intermediate level discussion of how the Jupyter system (JupyterHub, Jupyter Notebook, etc.) function at an application level on a distributed computing cluster. The workshop explains how Jupyter Kernels function and how to extend them.

Other topics related to Jupyter are open for discussion. A brief listing of possible topics is included at the end for reference.

Geoffrey Lentner
Senior Research Data Scientist
ITaP Research Computing
Prerequisites

- Basic Linux/Unix command-line knowledge.
- Basic understanding of Computing Clusters.
- Familiarity with Python
Content

• What is Jupyter

• Where does Jupyter Look for Kernels

• How do Kernels Work

• Anatomy of a Jupyter Kernel

• Extending and Customizing a Kernel

• Other Topics
**What is Jupyter**

**JupyterHub** only does **login** and **redirection**. It will start a Jupyter Notebook server on your behalf on one of the front-ends and redirect you.

The **notebook server** is fundamentally a web server that responds to requests from your browser and communicates with the **kernel** process its babysitting.

The browser is doing all of the **user interface**. When you execute a code cell it sends a request to the notebook server.

The **kernel** process is merely the interpreter running in a **headless** state (Python, R, etc...).

**JupyterHub** is a **Python application**. It behaves, responds, and can be debugged as such. To understand problems that arise in the Jupyter ecosystem you need to **identify at which layer** the issue is occurring.
Where Does Jupyter Look for Kernels

Jupyter looks for kernels in the system location it neighbors and in a fixed location in your home directory. A kernel is *installed* when it is *discoverable* by Jupyter in one of these locations. The *kernel.json* file specifies both *how to launch* the program and its *environment*. 
How Do Kernels Work

A kernel can be implemented in most languages (dozens are). Python is pretty easy to setup, only requiring a launch point. Others (e.g., R) can be more tricky and require detailed configuration.

Python can install its own kernel.
If you have your environment activated and IPython installed in that environment:

```
(my_env) $ ipython kernel install --user --name my_env --display-name "Python 3.7 (my_env)"
```

Jupyter is merely one of many clients that can manage an interactive Python session running in a headless state.

The `kernel.json` file specifies both **how to launch** the program and its **environment**.
Anatomy of a Jupyter Kernel

The “argv” section is literally the command-line arguments that will be invoked on your behalf.
Extending and Customizing a Kernel

The "argv" section is literally the command-line arguments that will be invoked on your behalf.

The "env" section is not present by default. You can add this section to define environment variables that you need for the libraries in the environment.

Many libraries require environment variables defined to function properly. When you install libraries with Anaconda these variables will be present after you activate the environment.

Jupyter doesn’t know anything about Anaconda.
Thank You
Other Topics

- Non-Python Kernels
- Integrating Modules with Jupyter Notebooks
- Distributing Computing from Jupyter Notebooks
- Notebook Extensions
- Customizing Notebooks with HTML/CSS/Javascript
- Debugging Jupyter, Notebooks, and Kernels

- OnDemand and Jupyter
- Running your own Jupyter from Backend Nodes
- Git and Jupyter
- JupyterLab
- Notebook Size and Visualization Libraries