Deploying a scalable JupyterHub environment for running Jupyter notebooks

Graham Dumpleton

Graham.Dumpleton@gmail.com
@GrahamDumpleton
The Jupyter Notebook is a web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, machine learning and much more.

Language of choice

The Notebook has support for over 40 programming languages, including those popular in Data Science such as Python, R, Julia and Scala.

Share notebooks

Notebooks can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer.

Interactive widgets

Code can produce rich output such as images, videos, LaTeX, and JavaScript. Interactive widgets can be used to manipulate and visualize data in realtime.

Big data integration

Leverage big data tools, such as Apache Spark, from Python, R and Scala. Explore that same data with pandas, scikit-learn, ggplot2, dplyr, etc.

http://jupyter.org/
Three-dimensional Points and Lines

The most basic three-dimensional plot is a line or collection of scatter plot created from sets of (x, y, z) triples. In analogy with the more common two-dimensional plots discussed earlier, these can be created using the `ax.plot3d` and `ax.scatter3d` functions. The call signature for these is nearly identical to that of their two-dimensional counterparts, so you can refer to `Simple Line Plots` and `Simple Scatter Plots` for more information on controlling the output. Here we'll plot a trigonometric spiral, along with some points drawn randomly near the line:

```python
In [4]: ax = plt.axes(projection='3d')

# Data for a three-dimensional line
zline = np.linspace(0, 15, 1000)
xline = np.sin(zline)
yline = np.cos(zline)
ax.plot3d(xline, yline, zline, 'gray')

# Data for three-dimensional scattered points
zdata = 15 * np.random.random(100)
xdata = np.sin(zdata) + 0.1 * np.random.random(100)
ydata = np.cos(zdata) + 0.1 * np.random.random(100)
ax.scatter3d(xdata, ydata, zdata, c=zdata, cmap='Greens');
```

Notice that by default, the scatter points have their transparency adjusted to give a sense of depth on the page. While the three-dimensional effect is sometimes difficult to see within a static image, an interactive view can lead to some nice intuition about the layout of the points.
In [11]:
# 1. Draw the map background
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='lcc', resolution='h',
lat_0=37.5, lon_0=-119,
width=180, height=120)
m.shadedrelief()
m.drawcoastlines(color='gray')
m.drawcountries(color='gray')
m.drawstates(color='gray')

# 2. scatter city data, with color reflecting population
# and size reflecting area
m.scatter(lon, lat, latlon=True,
c=np.log10(population), s=area,
cmap='reds', alpha=0.5)

# 3. create colorbar and legend
plt.colorbar(label=r'\$\log_{10}()$','\(\text{population}\)$',
shrink=0.5)
plt.clim(3, 7)

# make legend with dummy points
for a in [100, 300, 500]:
    plt.scatter([], [], c=a, alpha=0.5, s=a,
    label=str(a) + ' km$^2$'
)plt.legend(scatterpoints=1, frameon=False,
    labelspacing=1, loc='lower left');
How the Circle Line rogue train was caught with data

Text: Daniel Sim  |  Analysis: Lee Shangqian, Daniel Sim & Clarence Ng

Singapore’s MRT Circle Line was hit by a spate of mysterious disruptions in recent months, causing much confusion and distress to thousands of commuters.

Like most of my colleagues, I take a train on the Circle Line to my office at one-north every morning. So on November 5, when my team was given the chance to investigate the cause, I volunteered without hesitation.
2. The incidents happened at various locations on the Circle Line, with slightly more occurrences on the west side.

**Frequency by Station**

<table>
<thead>
<tr>
<th>Station</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harbourfront</td>
<td>10</td>
</tr>
<tr>
<td>Telok Blangah</td>
<td>8</td>
</tr>
<tr>
<td>Labrador Park</td>
<td>6</td>
</tr>
<tr>
<td>Peer Porong</td>
<td>5</td>
</tr>
<tr>
<td>Hwa Po Villa</td>
<td>4</td>
</tr>
<tr>
<td>Kent Ridge</td>
<td>3</td>
</tr>
<tr>
<td>one-north</td>
<td>2</td>
</tr>
<tr>
<td>Buona Vista</td>
<td>2</td>
</tr>
<tr>
<td>Holland Village</td>
<td>2</td>
</tr>
<tr>
<td>Furnace Road</td>
<td>2</td>
</tr>
<tr>
<td>Botanic Gardens</td>
<td>2</td>
</tr>
<tr>
<td>Cantonment</td>
<td>2</td>
</tr>
<tr>
<td>Marymount</td>
<td>1</td>
</tr>
<tr>
<td>Binan</td>
<td>1</td>
</tr>
<tr>
<td>Long Chuan</td>
<td>1</td>
</tr>
<tr>
<td>Senangan</td>
<td>1</td>
</tr>
<tr>
<td>Barry</td>
<td>1</td>
</tr>
<tr>
<td>Ta Sen</td>
<td>1</td>
</tr>
<tr>
<td>MacPherson</td>
<td>0</td>
</tr>
<tr>
<td>Pips Lorbar</td>
<td>0</td>
</tr>
<tr>
<td>Ekoosa</td>
<td>0</td>
</tr>
<tr>
<td>Mountainlea</td>
<td>0</td>
</tr>
<tr>
<td>Bellarm</td>
<td>0</td>
</tr>
<tr>
<td>North Highway</td>
<td>0</td>
</tr>
<tr>
<td>Promenade</td>
<td>0</td>
</tr>
<tr>
<td>Expo Line</td>
<td>0</td>
</tr>
<tr>
<td>Bish Bash</td>
<td>0</td>
</tr>
<tr>
<td>Dhuok Chau</td>
<td>0</td>
</tr>
<tr>
<td>Marina Bay</td>
<td>0</td>
</tr>
</tbody>
</table>

*Figure 2: The cause of the interference did not seem to be location-based.*

**How many rogue trains are there?**

As we showed in Figure 5, each end-to-end trip on the Circle Line takes about 1 hour. We drew best-fit lines through the incidents plots and the lines closely matched that of Figure 5. This strongly implied that there was only one “rogue train”.

*Figure 7: Time of clustered incidents strongly implies that the interference could be linked a single train*
Who’s Using It?

Individuals

Collaborators

Teachers
Getting Started

pip3 install jupyter

jupyter notebook
Empty Workspace
Upload Notebooks
Local File System

$ ls notebooks/01*.ipynb
notebooks/01.00-IPython-Beyond-Normal-Python.ipynb
notebooks/01.01-Help-And-Documentation.ipynb
notebooks/01.02-Shell-Keyboard-Shortcuts.ipynb
notebooks/01.03-Magic-Commands.ipynb
notebooks/01.04-Input-Output-History.ipynb
notebooks/01.05-IPython-And-Shell-Commands.ipynb
notebooks/01.06-Errors-and-Debugging.ipynb
notebooks/01.07-Timing-and-Profiling.ipynb
notebooks/01.08-More-IPython-Resources.ipynb
Browsing Files

![Jupyter Notebook Interface](image)
Interacting with a Notebook

IPython: Beyond Normal Python

There are many options for development environments for Python, and I’m often asked which one I use in my own work. My answer sometimes surprises people: my preferred environment is IPython plus a text editor (in my case, Emacs or Atom depending on my mood). IPython (short for Interactive Python) was started in 2001 by Fernando Perez as an enhanced Python interpreter, and has since grown into a project aiming to provide, in Perez’s words, “Tools for the entire life cycle of research computing.” If Python is the engine of our data science task, you might think of IPython as the interactive control panel.

As well as being a useful interactive interface to Python, IPython also provides a number of useful syntactic additions to the language; we’ll cover the most useful of these additions here. In addition, IPython is closely tied with the Jupyter project, which provides a browser-based notebook that is useful for development, collaboration, sharing, and even publication of data science results. The IPython notebook is actually a special case of the broader Jupyter notebook structure, which encompasses notebooks for Julia, R, and other programming languages. As an example of the usefulness of the notebook format, look no further than the page you are reading: the entire manuscript for this book was composed as a set of IPython notebooks.

IPython is about using Python effectively for interactive scientific and data-intensive computing. This chapter will start by stepping through some of the IPython features that are useful to the practice of data science, focusing especially on the syntax it offers beyond the standard features of Python. Next, we will go into a bit more depth on some of the more useful “magic commands” that can speed-up common tasks in creating and using data science code. Finally, we will touch on some of the features of the notebook that make it useful in understanding data and sharing results.

https://github.com/jakevdp/PythonDataScienceHandbook
### Status of Notebooks

#### Currently running Jupyter processes

<table>
<thead>
<tr>
<th>Terminals</th>
<th>Notebooks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="shutdown.png" alt="Shutdown" /></td>
</tr>
<tr>
<td><code>~ </code> terminals/1</td>
<td><img src="shutdown.png" alt="Shutdown" /></td>
</tr>
</tbody>
</table>

Notebooks:

- `notebooks/01.00-IPython-Beyond-Normal-Python.ipynb` (Python 2) ![Shutdown](shutdown.png)
Installing Packages

```
$ pip install antigravity
Collecting antigravity
  Downloading antigravity-0.1.zip
Building wheels for collected packages: antigravity
  Running setup.py bdist_wheel for antigravity ... done
  Stored in directory: /opt/app-root/.cache/pip/wheels/57/a0/83/f04be786fed18f1715187e33f99755d7a32279320124f0c065
Successfully built antigravity
Installing collected packages: antigravity
Successfully installed antigravity-0.1
```
Good Points

• Save notebooks/data locally.

• Python virtual environments.
  • Select Python version you want.
  • Install required Python packages.
Bad Points

• Operating system differences.
• Python distribution differences.
• Python version differences.
• Package index differences.
  • PyPi (pip) vs Anaconda (conda)
• Effort to setup and maintain.
Docker Images

https://github.com/jupyter/docker-stacks
Running Docker Image

```bash
docker run -it --rm -p 8888:8888 \ jupyter/minimal-notebook
```
Good Points

• Pre-created images.
  • Bundled operating system packages.
  • Known Python distribution/vendor.
  • Bundled Python packages.
• Docker images are read only.
• Run in an isolated environment.
• Don’t need to maintain the image.
Bad Points (1)

- More effort to customise experience.
  - Build a custom Docker image to extend.
  - Install extra packages each time you run it.
- Images can be very large.
  - Multiple Python versions.
  - Packages that you do not need.
Bad Points (2)

• Access to and saving your notebooks/data.
  • Need to mount persistent storage volumes.
  • Configure a contents manager for external storage.
• Ensuring access is done securely.
Interactive coding in your browser

Free, in the cloud, powered by Jupyter

https://notebooks.azure.com/
CLOUD DATALAB
An easy to use interactive tool for data exploration, analysis, visualization and machine learning

TRY IT FREE

Powerful Data Exploration

Cloud Datalab is a powerful interactive tool created to explore, analyze, transform and visualize data and build machine learning models on Google Cloud Platform. It runs on Google Compute Engine and connects to multiple cloud services easily so you can focus on your data science tasks.

https://cloud.google.com/datalab/
Good

- Somebody else looks after everything.
- Designed for use by multiple users at the same time.

Bad

- Shared resource.
- Outside of your control.
  - Reliability/Customisation/Software versions.
- Information security.
- Vendor lockin.
JupyterHub

https://jupyterhub.readthedocs.io
Good

- Can customise however you want.
  - Modify code for service.
  - Use custom images.
- Many deployment options.

Bad

- Dedicated infrastructure.
- Effort to understand and set it up.
- Effort to keep it running.
binder
(beta)

Turn a GitHub repo into a collection of interactive notebooks

Have a repository full of Jupyter notebooks? With Binder, open those notebooks in an executable environment, making your code immediately reproducible by anyone, anywhere.

Build and launch a repository

GitHub repo or URL

GitHub repository name or link

Git branch, tag, or commit

master

Path to a notebook file (optional)

Path to a notebook file (optional) File

http://mybinder.org/
• Ability to customise the image.

• Still owned and managed by someone else.
BinderHub

Important:

BinderHub is under active development and subject to breaking changes.

Getting started

The primary goal of BinderHub is creating custom computing environments that can be used by many remote users. BinderHub enables an end user to easily specify a desired computing environment from a GitHub repo. BinderHub then serves the custom computing environment at a URL which users can access remotely.

This guide assists you, an administrator, through the process of setting up your BinderHub deployment and helps you connect and configure the following things:

- A cloud provider such as Google Cloud, Microsoft Azure, Amazon EC2, and others
- Kubernetes to manage resources on the cloud
- Helm to configure and control Kubernetes
- Docker to use containers that standardize computing environments
- A BinderHub UI that users can access to specify GitHub repos they want built
- BinderHub to generate Docker images using the URL of a GitHub repository
- A Docker registry (such as gcr.io) that hosts container images
- JupyterHub to deploy temporary containers for users

https://binderhub.readthedocs.io
Good

• Can install it yourself.

Bad

• Re-inventing the wheel?
# Cloud Services

<table>
<thead>
<tr>
<th>Legacy</th>
<th>IaaS</th>
<th>CaaS</th>
<th>PaaS</th>
<th>SaaS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td>Functions</td>
<td>Functions</td>
<td>Functions</td>
<td>Functions</td>
<td>Functions</td>
</tr>
<tr>
<td>Application</td>
<td>Application</td>
<td>Application</td>
<td>Application</td>
<td>Application</td>
</tr>
<tr>
<td>Runtime</td>
<td>Runtime</td>
<td>Runtime</td>
<td>Runtime</td>
<td>Runtime</td>
</tr>
<tr>
<td>Containers (???)</td>
<td>Containers (???)</td>
<td>Containers</td>
<td>Containers</td>
<td>Containers</td>
</tr>
<tr>
<td>Virtualization</td>
<td>Virtualization</td>
<td>Virtualization</td>
<td>Virtualization</td>
<td>Virtualization</td>
</tr>
<tr>
<td>Operating System</td>
<td>Operating System</td>
<td>Operating System</td>
<td>Operating System</td>
<td>Operating System</td>
</tr>
<tr>
<td>Hardware</td>
<td>Hardware</td>
<td>Hardware</td>
<td>Hardware</td>
<td>Hardware</td>
</tr>
</tbody>
</table>

- **Customer Managed Unit of Scale**
- **Abstracted by Vendor**
PaaS - Source to Image

**BUILD APP**
(OpenShift)

**BUILD IMAGE**
(OpenShift)

**DEPLOY**
(OpenShift)
Deploy a Notebook
Build and Deployment Options

Choose from web frameworks, databases, and other components to add content to your project.

**Python**

- **JupyterHub Builder**
  - Template for creating customised JupyterHub images.
  - Namespace: user0
  - Select

- **JupyterHub**
  - Template for deploying a JupyterHub instance.
  - Namespace: user0
  - Select

- **JupyterHub Quickstart**
  - Template for creating and deploying customised Jupyter notebook images using JupyterHub.
  - Namespace: user0
  - Select

- **Jupyter Notebook Builder**
  - Template for creating customised Jupyter Notebook images.
  - Namespace: user0
  - Select

- **Jupyter Notebook**
  - Template for deploying Jupyter Notebook images.
  - Namespace: user0
  - Select

- **Jupyter Notebook Quickstart**
  - Template for creating and deploying customised Jupyter Notebook images.
  - Namespace: user0
  - Select
Build Customised Image

Jupyter Notebook Builder

Template for creating customised Jupyter Notebook images.

Namespace: strata

Images

strata/minimal-notebook:3.5 from parameter BUILDER_IMAGE

Parameters

* NOTEBOOK_NAME
  jakevdp-notebook

* BUILDER_IMAGE
  minimal-notebook:3.5

* GIT_REPOSITORY_URL
  https://github.com/jakevdp/PythonDataScienceHandbook
Deploy JupyterHub

Jupyter Hub
Template for deploying a customised Jupyter Hub instance.

Namespace: strata

Images
- jupyterhub:latest from parameter JUPYTERHUB_IMAGE
- postgresql:9.5

Parameters
* APPLICATION_NAME
  jupyterhub

* JUPYTERHUB_IMAGE
  jupyterhub:latest

* NOTEBOOK_IMAGE
  jakevdp-notebook:latest
### Notebooks on Demand

<table>
<thead>
<tr>
<th>APPLICATION</th>
<th>jupyterhub</th>
<th><a href="https://jupyterhub-strata.apps.graham.gce.pixy.io">https://jupyterhub-strata.apps.graham.gce.pixy.io</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPLOYMENT</td>
<td>jupyterhub</td>
<td>140 MIB Memory, -- Cores CPU, -- KiB/s Network, 1 pod</td>
</tr>
<tr>
<td>DEPLOYMENT</td>
<td>jupyterhub-db, #1</td>
<td>94 MIB Memory, -- Cores CPU, 2.4 KiB/s Network, 1 pod</td>
</tr>
<tr>
<td>POD</td>
<td>jupyterhub-nb-8e75f715-2d68ac-2d4a88-2d89a7-2d31d2b6ca546e</td>
<td>3.8 MIB Memory, -- Cores CPU, -- KiB/s Network, 1 pod</td>
</tr>
</tbody>
</table>
Quotas and Limits
Why consider OpenShift?

• More accessible to developers and non technical users.
• Easier to create custom solutions for deploying Jupyter.
• Run workloads other than just Jupyter notebooks.
• Users controlled through quotas and limits on resources.
• Share infrastructure with other users in your organisation.
• Open Source platform, with option for support from Red Hat.
Jupyter on OpenShift

https://github.com/jupyter-on-openshift

Graham.Dumpleton@gmail.com
@GrahamDumpleton