JupyterHub as an OER Platform

We have been experimenting with cross-disciplinary courses at both the undergraduate and Masters level, adopting a blended learning approach: face-to-face, real-time engagements will be supported by a variety of computer-mediated learning materials. One of our important goals is to help students with little or no programming experience to become comfortable with basic data analysis and visualisation tasks. This has led us to to experiment with a framework based on iPython and JupyterHub. This approach has several attractive features:

- it offers a browser-based 'notebook' with support for rich text, executable code blocks and interactive data visualisation;
- it allows students to receive the same learning materials but experiment with them on an individual basis within a standardised environment;
- the components are all open source and actively maintained;
- since the platform is made available as a service, students are not required to download and install the necessary software; in additions we are able to ensure that all required libraries are present, together with any customisations that we want to make available.

[FRANCISCO] Use of Bokeh and Folium

Bokeh (Visualization API)

Bokeh is a python web targeted visualization library that displays D3.js like plots. The idea behind bokeh is that it provides very simple commands to do very beautiful and versatile plots outputting this in a browser renderable format. The bokeh project is extemely active on facebook and it aims to be scalable over visualizing big data sets. In terms of education here are some useful features I found whilst working on this project:

- Easy to use artists tools that allow you to draw almost anything in your plots;
- Inline js to add extra interactivity to plots;
- Widgets that also make plots interactive;
- Sync extremely well with Ipython notebook rendering beautiful and dynamic graphs ! (unlike matplotlib).
- Really nice labels and hover tools.

Some of the specific plots we worked on:

- Heatmaps;
- Scatter plot matrices;

- Scatter plots;
- Timeseries and smoothed time series;
- Binary trees.

Folium (MAP Rendering API)

Follium is a python wrapper around leaflet.js ! those of you who know what this mean and like python are probably shivering in exitement at the sound of this. leaflet.js is an extremely rich browser based map drawing tool which allows all sorts of plots analysis and visualizations over customized maps. Folium provides an extremely friendly python interface to leaflet.js similar to that of Bokeh and D3.js which allows us to create beautiful map visualizations with very trivial commands. Some of the plots we exploited were:

- Ploting data as markers that contain information labels on the map;
- Choropleth/ Voronoi Tessallation displaying how data changes accross Edinburgh.

Some of the advantages of Folium to this project are: * Smooth integration and display with ipython notebook; * Light syntax; * Reasonable diversity of displays on map.

[EWAN] Adding metadata to notebooks

Trial Service

Using Docker to power a JupyterHub service is something we were interested in due to the success of the Jessica Hamrick's JupyterHub deployment for Computational Models of Cognition. However, we couldn't simply adapt their elastic cloud setup because we don't have access to an elastic cloud. We wanted something we could run on a single server, and worry about scaling later.

Using Docker for this is still preferable for fast prototyping the configuration of the server, and keeping the configuration hierarchical. In our server configuration repository, the huge stack of scientific packages required to run the notebooks is contained in a Dockerfile, so it's possible for motivated students to run their own Jupyter instance independently on whatever computer they might like to use. The same Dockerfile is then reused for the JupyterHub server configurations. Working in this way, we can easily transfer the server to new hardware, and build upon the work we've already done.

There are two main server configurations we've been working on. The first is a centralised JupyterHub instance, in line with the standard configuration described in the JupyterHub repository. Each student has a persistent account with storage, and server instances will persist. For demonstration, we have also

built a temporary notebook server using the Jupyter tmpnb repository, which is currently running at: `http://livinglab.ngrok.io/`. This simply serves each new visitor a new Jupyter notebook instance, and removes the instance after the user is finished.

Each time a new temporary Jupyter instance is created, it uses a script in the `dds_notebooks` repository to keep notebooks up to date. Making this as seamless as possible for new users that may not be familiar with git was a concern. From the user perspective, any file that is updated remotely will be silently updated, as long as the user *hasn't edited it*. If they have, we make sure to move the edited file before updating with the remote changes. This is integrated with the static JupyterHub image as well, running as a cron job by default.

Docker volumes are used to manage user home directories in the static case, allowing for seamless changes to potentially the entire operating system. These are also used to link in datasets that the students can access at a shared location inside the container.

Potentially, this server could be extended to include programming languages beyond Python, as Jupyter is now agnostic to the language used in the kernel. And, thanks to the dockerised development work, we could run independent systems with different courses; or we are free to develop a distributed system offering data science in the cloud to the entire University.

Building a University-Wide Service

JuyterHub has a extendable authentication architecture that allows deployments to override default authentication strategies with custom Authenticator modules. Authenticators already exist for popular identity providers such as Github OAuth, Google OAuth and MediaWiki OAuth. To support institutional access at the University using a single sign-on service (EASE) based on the CoSign system, we deploy JupyerHub behind an Apache proxy server, allowing us to use existing Apache CoSign modules that are well understood and supported by the University.

The Apache server provides a public facing SSL port, which redirects requests to the JupyterHub server instance running on a private port on the same virtual machine.

If the user has not already obtained a valid JupyterHub session, the Apache proxy redirects the user to the University's EASE Single Sign-on webpage and validates the user's credentials using a secure channel. The Co-Sign service then passes the request back to the Apache proxy server, which now has access to the REMOTE_USER environment variable which was set by the CoSign service with the provided username. The Apache server uses this variable to set an addition request header and then proxies the request to the JupyterHub instance. Now that the user is authenticated and the username is available to JupyterHub

in a request header, it is possible to authenticate the user to Jupyterhub with a simple Authenticator plugin such as Magnus Hagdorn's REMOTE_USER Authenticator.