NST Physics and nbgrader

Niall McConville
PyCav Project

August 2016

Contents

1 Introduction **2**

2 Setup **3**

2.1 JupyterHub . . . 3

2.1.1 Background . . . 3

2.1.2 Raven Authentication . . . 3

2.1.3 Docker Configuration . . . 4

2.1.4 The JupyterHub Config . . . 7

2.2 nbgrader . . . 9

2.2.1 Background . . . 9

2.2.2 Installation . . . 9

2.2.3 Directory Structure . . . 9

2.2.4 JupyterHub Integration . . . 10

2.2.5 Teaching System Integration . . . 10

2.3 Server . . . 10

3 Courses **11**

3.1 Courses . . . 11

3.1.1 Directory Structure . . . 11

3.1.2 The nbgrader Config . . . 11

3.2 Exercises . . . 14

3.2.1 Creating Exercises . . . 14

3.2.2 Assigning Exercises . . . 15

3.2.3 Collecting Exercises . . . 15

3.2.4 Marking Exercises . . . 16

3.2.5 Giving Feedback . . . 16

4 Conclusion **17**

Chapter 1

Introduction

nbgrader[1] is a system which allows for the use of Jupyter[2] notebooks as a form of assessment material.

Lecturers (or at least, their lackeys[3]) can create a set of exercises contained within these notebooks. They can be distributed to students using the JupyterHub[4] server.

From there, students can complete the computational exercises and return them (hopefully completed) to the lecturers. The lecturers can autograde the exercises and manually mark aspects which do not lend themselves easily to a machine marking them (such as graphs).

This document outlines how the PyCav project set up nbgrader for use within the Physics courses within the Natural Sciences Tripos.

Another source of documentation for nbgrader can be found here: `http://nbgrader.readthedocs.io/`.

[1] `https://github.com/jupyter/nbgrader`
[2] `http://jupyter.org/`
[3] `https://pycav.github.io/about/`
[4] `https://github.com/jupyterhub/jupyterhub`

Chapter 2

Setup

2.1 JupyterHub

2.1.1 Background

JupyterHub is a system which allows the hosting of a multi-user server which allows users to log in to and create Jupyter notebooks without hosting a server for themselves.

Docker[1] is effectively a sandboxing system which allows JupyterHub to create single user Jupyter servers inside a set of isolated environments.

The idea for this project was to set up an environment where students could experiment with Python without having to install any software locally. The above were selected as they were able to provide this environment.

2.1.2 Raven Authentication

All Cambridge students and staff have access to a crsid. This allows them to authenticate using the Raven[2] service.

To extend this functionality to JupyterHub, an authentication plugin jupyterhub-raven-auth[3] was written. It was used throughout the project.

[1] `https://www.docker.com/`
[2] `https://raven.cam.ac.uk/`
[3] `https://github.com/PyCav/jupyterhub-raven-auth`

2.1.3 Docker Configuration

The isolation provided by Docker is useful for running an assessment environment. It allows for caps to be placed on usage, especially in the instances of runaway code. Students are unable to view other student's files in a way that goes beyond setting access rights. It also allows for the simple addition of read only volumes, which we have exploited to share Demonstrations for everyone to see.

Docker images are built from Dockerfiles (compare with Makefiles). These images are called by the Dockerspawner in the JupyterHub config. The Dockerspawner creates *containers* which take up file space. It is possible to mount volumes in an NFS setup, using the {username} filter.

In the Dockerfile provided below, one can see the extent of the customisation we provide. Notably, nbgrader is installed in each container.

Docker containers (and their 'real' mounted volumes) should be continually backed up. We (as of August 22, 2016) have not considered what the total size of such a system would be. For the 'Computational Models' course at Berkley, CA a 3 TB NFS was used for storage[4].

We have also set up the containers to execute the 'start-singleuser.sh' shell script. This contains code to create a new user, whose username matches the crsid of the individual logged into the JupyterHub. This is required as nbgrader will use this username in the filenames of submitted coursework.

[4] `https://github.com/compmodels/jupyterhub-deploy`

Dockerfile:

```
# Build as jupyterhub/singleuser
# Run with the DockerSpawner in JupyterHub

FROM jupyterhub/singleuser

MAINTAINER jordan <jo357@cam.ac.uk>

EXPOSE 8888

USER root
RUN echo "deb http://ftp.debian.org/debian jessie-backports main" \
    >> /etc/apt/sources.list
RUN apt-get -y update
RUN apt-get -t jessie-backports  -y install ffmpeg
RUN pip3 install vpython
RUN pip3 install pycav
RUN pip3 install nbgrader
RUN nbgrader extension install
RUN nbgrader extension activate

# Prep. to replace the jovyan user with the crsid
RUN userdel jovyan
ENV SHELL /bin/bash

ADD pycav-start.sh /srv/pycav/pycav-start.sh

# Execute this script on startup
CMD ["sh", "/srv/pycav/pycav-start.sh"]
```

start-singleuser.sh file:

```
#!/bin/bash

set -e
if getent passwd $JPY_USER > /dev/null ; then
 echo "$JPY_USER exists"
else
 echo "Creating user $JPY_USER (9002)"
 useradd -u 1000 -s $SHELL $JPY_USER
fi

notebook_arg=""
if [ -n "${NOTEBOOK_DIR:+x}" ]
then
    notebook_arg="--notebook-dir=${NOTEBOOK_DIR}"
fi

sudo touch /home/jovyan/work/.nbgrader.log
sudo chmod 666 /home/jovyan/work/.nbgrader.log

sudo -E PATH="${CONDA_DIR}/bin:$PATH" -u $JPY_USER jupyterhub-singleuser \
 --port=8888 \
 --ip=0.0.0.0 \
 --user=$JPY_USER \
 --cookie-name=$JPY_COOKIE_NAME \
 --base-url=$JPY_BASE_URL \
 --hub-prefix=$JPY_HUB_PREFIX \
 --hub-api-url=$JPY_HUB_API_URL \
 ${notebook_arg} \
 $@
```

2.1.4 The JupyterHub Config

```
"""
jupyterhub_config.py

The pycav project jupyterhub config
"""

c = get_config()

"""
Imports

os : for accessing absolute paths
pycav_tis : for interfacing with the Teaching Information System
"""
import os
from pycav_tis import tis

# Setup for the TiS module
# TODO: Replace DictReader with SQL system when appropriate
tis_config = '/home/public/tis_config'
tis_csv = '/home/public/tis.csv'
tis_conn = tis.pycavTisDictReader(tis_csv)

c.NotebookApp.open_browser = False

# Enable Logging
c.JupyterHub.log_level = 'DEBUG'
c.JupyterHub.port = 8000

# SSL
c.JupyterHub.ssl_key = '/etc/letsencrypt/live/pycav.ovh/privkey.pem'
c.JupyterHub.ssl_cert = '/etc/letsencrypt/live/pycav.ovh/fullchain.pem'

# Cookies
c.JupyterHub.proxy_auth_token='you-need-to-set-this-see-the-docs'
```

```
c.JupyterHub.cookie_secret_file = '/home/jordan/jupyterhub_cookie_secret'

# Users
c.JupyterHub.db_url = '/home/jordan/jupyterhub.sqlite'
c.JupyterHub.admin_access = True

c.Authenticator.admin_users = tis_conn.get_admins()
c.Authenticator.whitelist = tis_conn.get_users()

# Ucam Authentication
from raven_auth.raven_auth import RavenAuthenticator
c.JupyterHub.authenticator_class = RavenAuthenticator
c.RavenAuthenticator.description.value = "pyCav"
c.RavenAuthenticator.long_description = "The pyCav Jupyterhub server."
c.RavenAuthenticator.login_logo = '/home/public/py_cav.jpg'

# Docker
from dockerspawner import DockerSpawner
c.Spawner.debug = True
c.JupyterHub.spawner_class = DockerSpawner
#c.DockerSpawner.container_prefix = pycav
c.DockerSpawner.read_only_volumes={'/home/public/demos':
'/home/jovyan/work/demos',
'/home/public/crsidify':'/srv/crsidify'}
c.DockerSpawner.volumes={'/srv/nbgrader':'/srv/nbgrader'}
c.DockerSpawner.extra_create_kwargs.update({
    'command': 'sh /srv/crsidify/start-singleuser.sh'
})
c.DockerSpawner.container_image = "jordanosborn/pycav"

import netifaces
docker0 = netifaces.ifaddresses('docker0')
docker0_ipv4 = docker0[netifaces.AF_INET][0]
c.JupyterHub.hub_ip = docker0_ipv4['addr']
```

2.2 nbgrader

2.2.1 Background

nbgrader is a tool which allows for the assignment and grading of Jupyter notebooks. It can operate in numerous ways, the simplest of which is the manual distribution and collection of notebook assignments. The PyCav project uses a combination of nbgrader and JupyterHub to automate various aspects of the process.

The documentation for nbgrader can be found here: `http://nbgrader.readthedocs.io/`.

2.2.2 Installation

The nbgrader was setup in two places,

1. The pycav server (the local server, where the JupyterHub is hosted).
2. Within the Docker containers.

The installation on the local server was for the utilisation of nbgrader's *assign*, *collect*, and *formgrade* applications. These are to be run by the teaching staff, and will be discussed in a later chapter.

The installation inside the Docker containers was for nbgrader's Jupyter extension, which calls nbgrader's *fetch* and *submit* applications. These will be run by the extension (and so not directly by students).

Upon installation within a Docker image, a file called '.nbgrader.log' is created within the '/home/jovyan/-work/' directory. As mentioned in the previous chapter, this *must* have write access for the user or else the nbgrader functionality will not work.

2.2.3 Directory Structure

One can posit that there are effectively two sides to nbgrader directories: course directories and the distribution directory. These are both present on the local server.

Central to the automation of distribution is the *exchange* folder. Typically this is stored in the directory '/srv/nbgrader/exchange', although one can specify alternative directories. The exchange directory **must** be world readable and writable.

Although any folder can be specified as the exchange folder, when mounting it to a Docker container, it is simplest to mount it to the container directory such that it appears as '/srv/nbgrader/exchange'. The default behaviour of nbgrader is to check this directory, with further (possibly complicated) configuration required to change this.

Course directories provide a structure for the courses that can be provided. They will be elaborated upon in the next chapter.

2.2.4 JupyterHub Integration

2.2.5 Teaching System Integration

Currently a work in progress. There will be a module which allows nbgrader to interface with the TiS.

2.3 Server

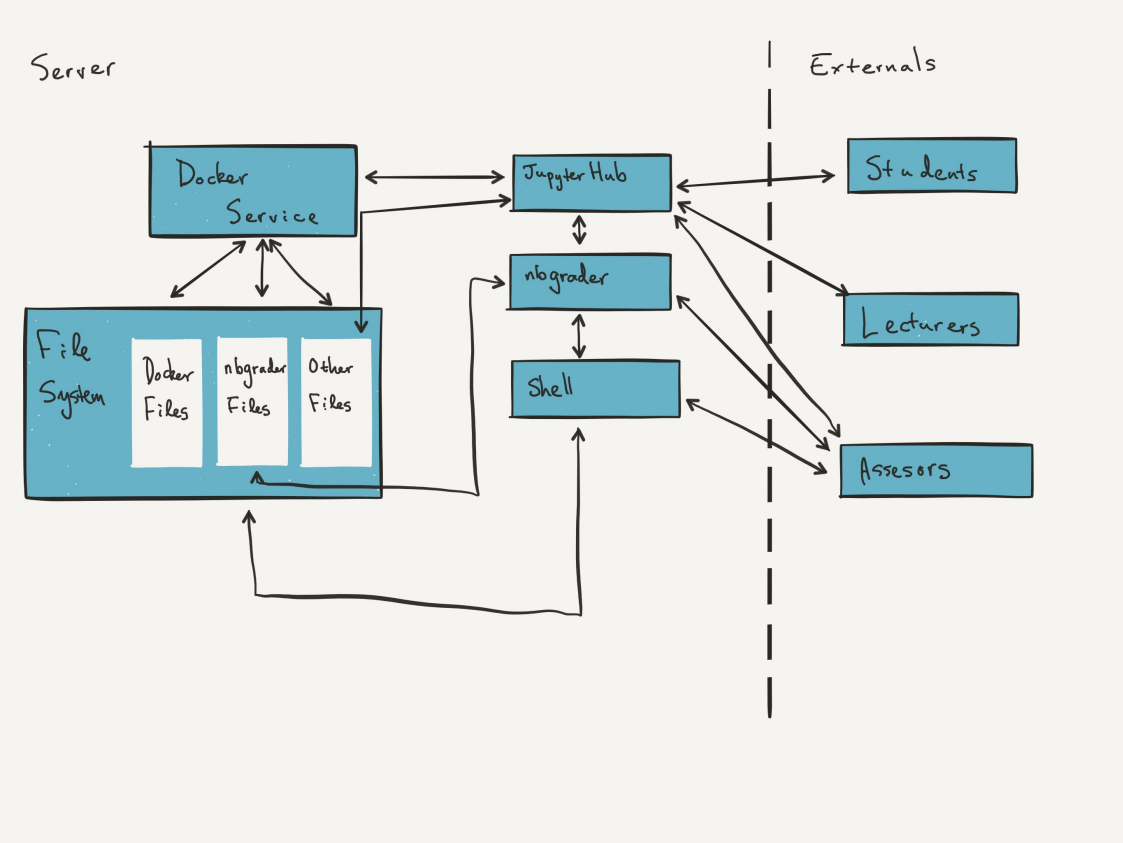

Figure 2.1: The interactions between the various bits of software in relation to the users and the servers. Note that Assessors should be able to access software beyond the JupyterHub, for example, the nbgrader formgrade interface (which in turn relies on JupyterHub for authentication). Not depicted: The nbgrader installation **inside** the Docker containers.

Chapter 3

Courses

3.1 Courses

3.1.1 Directory Structure

The primary folder of any course is the {course} directory folder. To create any course, you must start with one of these folders. It contains a few files (some of which are created when nbgrader is run for the first time),

1. {course}/gradebook.db
2. {course}/nbgrader_config.py

The gradebook stores information about the types of assignment which are pertinent to the course. The nbgrader_config will be elaborated upon in a later section.

nbgrader commands should be run from the {course} directory.

Courses contain assignments, which contain exercises.

In order to add assignments to a course, you should create a {course}/source directory, with folders corresponding to each assignment (each assignment can contain multiple notebooks).

3.1.2 The nbgrader Config

The nbgrader config file specifies the following information,

1. The course id (used to lookup from the TiS).
2. The students who are taking the course.

3. The graders responsible for the course.

4. The Formgrader configurations (ip, port, JupyterHub integration)

A TiS plugin was/is being developed that would allow for the students and graders dictionaries to be populated. This stops unwanted individuals from being able to submit their work (which would waste grading time). Currently there are no developments which would allow for pushing autograded entries to the TiS. This might be desirable, but will of course have to be weighed up against the pros and cons. A positive is that it (pending sampling for error) would reduce the possibility of incorrect grades being communicated to the markbooks. However, automation of such a system could be quite 'blind' and could limit accountability.

```
import os
from pycav_tis import tis

c = get_config()

# TODO: When creating a new course, change this to the course_id on the TiS
# May have to write a dictionary to look this up because nbgrader automatically converts a
course_id = 'pycav-test'

# TiS
#tis_config = ''
tis_csv = '/home/public/tis.csv'
tis_conn = tis.pycavTisDictReader(tis_csv, course_id)

# Generic nbgrader configs
c.NbGrader.course_id = course_id
#c.TransferApp.exchange_directory = "/home/public/pycav-nbgrader/exchange"
c.NbGrader.db_assignments = [dict(name="ex1")]
c.NbGrader.db_students = tis_conn.get_students()

# Options that are specific to formgrader & integrating it with JupyterHub

c.FormgradeApp.ip = "127.0.0.1"
c.FormgradeApp.port = 9000
c.FormgradeApp.authenticator_class = "nbgrader.auth.hubauth.HubAuth"

#
import netifaces
docker0 = netifaces.ifaddresses('docker0')
docker0_ipv4 = docker0[netifaces.AF_INET][0]

# This is the actual URL or public IP address where JupyterHub is running (by
# default, the HubAuth will just use the same address as what the formgrader is
# running on -- so in this case, 127.0.0.1). If you have JupyterHub behind a
# domain name, you probably want to set that here.
```

```
# TODO: Convert this into some sort of jupyterhub or shared config
c.HubAuth.hub_base_url = "https://pycav.ovh:8000"
c.HubAuth.hub_port = 8001
c.HubAuth.hubapi_port = 8081
c.HubAuth.hubapi_address = docker0_ipv4['addr']

# Call the TiS to get the graders for this particular course_id
c.HubAuth.graders = tis_conn.get_graders()

c.HubAuth.hubapi_token = os.environ['JPY_API_TOKEN']
```

3.2 Exercises

3.2.1 Creating Exercises

For a full guide on creating exercises: `http://nbgrader.readthedocs.io/en/stable/user_guide/creating_and_grading_assignments.html`.

In short, exercises are Jupyter Notebooks which have been customised using the nbgrader Jupyter extension. This will affix metadata to the notebooks.

Before creating an exercise, one must install the nbgrader Jupyter extension[1].

Exercises contain four different types of cell,

1. *Manually Graded Cells*: This lets nbgrader know that a human will grade these cells. Points are allocated to these cells.
2. *Read-only Cells*: (Ideally, although in practice it doesn't work) Cells which students can read but not write to, for example, the exercises themselves[2].
3. *Autograded Answer Cells*: These cells do not get awarded points when creating an exercise. In the full guide on creating exercises, one can indicate certain tags which allow you to write an answer, which will be scrubbed from the notebook upon assignment.
4. *Autograder Cells*: These cells are assigned points. Typically answers are tested by using python's assert (which will quietly pass tests). A failure is quantified by these cells by their throwing an error. Graphs are difficult to autograde and should probably be reserved for manual testing. These tests

[1] `http://nbgrader.readthedocs.io/en/stable/user_guide/installation.html`

[2] Effectively the problem lies in the fact that there is no functionality within nbgrader to enforce this behaviour. There does, however, exist a Jupyter nbextension for enforcing read only cells using Javascript. It is installed using our Dockerfiles.

can be viewed and modified by students, but a checksum system will prevent them from submitting modified cells[3]

It is critical to assign points correctly to cells. Failure to do so correctly will result in errors upon assignment, or autograding for students.

When finished, place your assignments in the folder folder {course}/source/{assignment_id}.

Make sure that the assignment directory is recorded in the nbgrader_config.py.

3.2.2 Assigning Exercises

To begin the process of sending out assignments (which contain the exercises), go to the {course} directory and execute the following command,

```
nbgrader assign {assignment_id}
```

Where {assignment_id} is the name of the exercise folder {course}/source/{assignment_id}, containing the exercises to be assigned.

This will create a folder {release} in the course directory. One could now, if they wished, distribute these files manually.

Once this command has been run, the notebooks in the assignment directory are scrubbed of inputs as well as of answers if the correct specification has been used.

The system we have developed for the Cavendish uses another step of the nbgrader system. In order to distribute the assignemnts using the JupyterHub, one should execute the next command,

```
nbgrader release {assignment_id}
```

This will copy the assignment files from the release directory to '/srv/nbgrader/exchange/{course}/outbound/{assignment_id} folder. This directory is mirrored into docker containers and students can grab their assignments using the nbgrader extension. Students can submit multiple times, which will be stored in the corresponding 'inbound' directory, although only the most recent submission will be graded.

3.2.3 Collecting Exercises

When you want to collect the exercises, run the following command in the {course} directory,

[3]There are two things to note here. One is that the checksum is actually distributed as a JSON field inside the notebooks. Therefore the cells are prone to an attack where the original checksum is substituted with that of a modified autograder cell. nbgrader will not throw an error in this case. I think that the way around this is to store the checksum with a the gradebook.db and throw an error (or at least, flag for manual marking). The second note is that visibile autograder tests may limit the type of functions that could be tested. One could try to describe this as a system of 'working to an answer' rather than towards one. To get around this, it might be worth compiling autograder tests using Cython and importing the tests in these cells.

```
nbgrader collect {assignment_id}
```

This will copy the submitted notebooks from the inbound directory to the {course} directory.

3.2.4 Marking Exercises

There are two steps to marking exercises/assignments.

The first is to run the autograder, this **must** be done first. It will check whether or not students are signed up for the course and it will print an error and skip students who it does not recognise.

```
nbgrader autograde {assignment_id}
```

Once this is done, the formgrader, a web interface for grading notebooks can be run using,

```
nbgrader formgrade
```

This starts a tornado server which interfaces with the JupyterHub (for redirects & authentication only, it does not run in a docker container).

Exercises can then be marked and points awarded.

Only those who are recorded as markers for a particular course are allowed to access the formgrader.

3.2.5 Giving Feedback

Once marked, the graded notebooks can be compiled into HTML files which can be sent back to students to give feedback.

To do this, run in the {course} directory,

```
nbgrader feedback {assignment_id}
```

This will store the feedback in the {course}/feedback directory. A script has been written which will copy these files into '/srv/nbgrader/feedback/' and the appropriate directory has been mirrored into the Docker containers (to stop students viewing each other's feedback).

Chapter 4

Conclusion

A brief guide to the JupyterHub and nbgrader system setup by the PyCav team is presented. The advantages and pitfalls are also discussed. The structure of the server and how the various setups are tied together. A guide is also given as to how to setup and assess courses - however, this is presented like a framework, with the reader expected to get a feeling for the shape of the nbgrader system, rather than to illuminate it completely.

Maintenance is expected to be simple - especially if the existing fleet of experienced *nix users remain with the Cavendish. Security aspects of nbgrader are also highlighted, and could be fixed by a future PyCav team if this isn't done within the current project.

In terms of preventing students from tailoring answers to those given in autograder cells, it is my suggestion that Cython be used in order to provide a layer of obfuscation for autograder tests in the course fields. This would also be useful as it would make the testing cells more concise. In order to avoid creating autograder tests which are difficult for the students to interpret once their tests have failed, I recommend that methods should be given rather literal (and one might expect, quite long) names so as to not obstruct self-learning through knowledge of error.

Conversion of the Part II Computational Exercises course would be an obvious place to start, with the exercise sheet being converted into notebooks.

In summary, this system should be helpful for the Cavendish in any quest to streamline the teaching of computational resources (including, but not limited to assessment). I hope it finds some success in future.