Building Python modules on CircleCI OS X instances

Recently I've been building a couple of supplimental build tools in Python for enhancing the development process of the iOS application I work on. I've been trying to make these tools to be appealing for other developers to use and integrate into their systems as well. I see having CI and unit tests as a core part of this goal. I started using CircleCI for my continuous integration environment for these projects. This was working out very well until I started working on some code that relies on a couple of OS X specific APIs. Due to that requirement I was unable to continue using the Linux platform for testing this code. Luckily CircleCI provides OS X instances for open source Mac and iOS projects. I put in a request and was granted access to using OS X instances for building this Python tool. However, I immediately ran into issues with getting the tool to be installed and have my tests run. CircleCI doesn't provide engineering support for free accounts, so I had to work all this out on my own, hopefully you can use this as a guide to also be able to support building Python on OS X build instances on CircleCI.

Background

The Python tool I am writing relies on the pyobjc framework. This allows Python to make calls into Cocoa and the various other frameworks provided by Apple on OS X. This comes pre-installed on OS X with the system Python version, 2.7.10. Since an official sunset date has been set for Python 2, the code is written to also support Python 3. Since Python 3 doesn't come as part of the default OS X installation, this adds another requirement to the build environment that is needed. In addition to the Python packages that are used by this tool, all of the tooling that is needed to run the unit tests and additional checks that gets validated as part of the build. This ultimately generates the following list of requirements:

- virtualenv
- pyenv
- tox
- tox-pyenv (tox plugin to work with pyenv)
- Python 3
- pyobjc (for Python 3)

- coverage.py
- Python 2 (installed separate from the system Python version by pyenv)

In addition to this list, there are a few more things that I use:

- danger
- codeclimate-test-reporter
- pylint

While all of these packages and tools can be installed easily on the Ubuntu instances that are provided by CircleCI, the setup process to accomodate this was not as easy for OS X. The rest of this post is a guide as to how I was able to set up an environment that allowed me to build and run the unit tests for a Python CLI application that uses OS X specific APIs. It took me a whole day and a couple dozen attempts at this before I was able to get it working, and hopefully can someone else at least that much time in the future.

Xcode Project File

The configuration of CircleCI instances of OS X seems to assume that you are going to be building something using the Xcode build system. While this is true for the majority of iOS or OS X software, that isn't true if you are relying on existing systems on OS X (such as interfacing with Cocoa frameworks through pyobjc), as I am doing here. To ensure that the CircleCI job will not immediately fail, you must include an Xcode project file with at least one valid target in you repo to not cause the job to fail immediately. This target doesn't have to do anything, as the setup and test execution will be over-ridden in the `circle.yml` file to run the Python specific test framework instead.

CircleCI Instance Configuration

Machine

As per CircleCI's documentation, they do not support the ability to dictate specific language versions on OS X. So, to handle the install process of Python and the necessary components needed, the following commands will need to be run as part of the `pre:` section of the `machine:` configuration in the `circle.yml` file.

```
machine:
  pre:
    - export PATH=/usr/local/bin:$PATH:/Users/distiller/Library/Python/
    - pip install --user --ignore-installed --upgrade virtualenv
    - ln -s $HOME/Library/Python/2.7/bin/virtualenv /usr/local/bin/virt
    - cd "$(brew --repository)" && git fetch && git reset --hard origin
    - brew update
```

1. First step involves exporting the necessary paths needed for the rest of the build. This requires that the brew install path and the user's Python install path is included.
2. Next, for the installation of `virtualenv` into the user's Python path. This is necessary as `virtualenv` will be invoked as an inferred build step later in the build process.
3. After installation, create a symlink from the installed executable for `virtualenv` into brew's install path. This is needed to ensure that it will be seen by the system for later execution.
4. This step may not be required for all, but I was running into an issue that prevented the `brew update` command from being run without causing a failure. As a result this command was taken from an issue on the Homebrew GitHub repo to resolve the error.
5. Finally, perform a update to the `brew` packages list. I found this necessary to ensure that the packages I need are available.

Dependencies

To install the necessary dependencies, I am using a target in my Makefile to invoke all the necessary commands and install what is needed to properly perform a build and execute the tests. For this I am overriding any inferred behavior on Circle's part for installation with the following:

```
dependencies:
  override:
    - make install-deps
    - pyenv local 2.7.10 3.5.1
```

1. This invokes the Makefile target that will install all of the components that are needed.
2. This performs some extra setup with pyenv to be done in preparation for using `tox` as part of running the unit tests.

The target in the Makefile will perform the following commands:

```
# brew commands
$ brew install pyenv
$ brew install python3

# pip commands
$ pip install coverage --user
$ pip install tox --user
$ pip install tox-pyenv --user
$ pip install codecoverage-test-reporter --user
$ pip install pylint --user
$ pip install pyobjc-core --user
$ pip install pyobjc-framework-Cocoa --user

# pip3 commands
$ pip3 install coverage
$ pip3 install tox
$ pip3 install tox-pyenv
$ pip3 install codecoverage-test-reporter
$ pip3 install pylint
$ pip3 install pyobjc-core
$ pip3 install pyobjc-framework-Cocoa

# pyenv commands
$ pyenv install 2.7.10
$ pyenv install 3.5.1

# gem commands
$ gem install danger
```

There are a couple of important things to take away here that are specific to both the way that the OS X instances on CircleCI are setup and to how I have configured `tox`

1. User vs System Installation of Python Packages:

You will notice that all of the Python 2 packages (installed by `pip`) are invoked with the `--user` flag, whereas the Python 3 packages (installed by `pip3`) are not. The significance is that since Python 2 and `pip` come as part of OS X, we are leveraging the existing install to act as the default version of Python to use and not require installing again. Since Python 3 is being installed by Homebrew, it will be able to install packages into the default location.

1. Duplicate Installation of Python:

After installing all of the packages needed, I am invoking `pyenv` to install both version `2.7.10` and `3.5.1` to be installed and registered for Python environments. This is necessary for use with the `tox-pyenv` plugin for `tox`

that allows for tests to be executed in an array of different versions of Python. This enables two types of Python environments, the “Host” version of Python and the “Guest” version of Python. I recommend this approach when using `tox` so that you can install all of the dependencies into the “Host” version of Python (the system version or one installed by Homebrew), and inherit them into the “Guest” version of Python (the versions installed by `pyenv`). The “Guest” versions of Python are used to run the unit tests in by `tox`. This separation of installed packages makes it easier to deal with on a CircleCI instance. To do this, you will have to enable the `sitepackages` option in your `tox.ini` configuration file.

1. Invokation of `gem install` without the `--user` flag:

One thing to note is while the system version of Python is used on the OS X build instances, that is not true of Ruby. I am not sure of the exact configuration of which version of Ruby or RubyGems is used, but passing the `--user` flag to install a Gem caused it to be installed to a location outside of the defined `PATH` variable, and thus unable to be executed.

Test

As done with the dependencies, we are going to force an override to the system to run our own set of commands as part of the test phase of the CircleCI job. This will allow you to execute the Python specific testing framework instead of the job inferring that `xcodebuild` should be invoked.

```
test:
  override:
    - make ci
```

This Makefile target will:

1. Invoke `tox` to run the unit tests registered as the test suite in my `setup.py` file.
2. After the tests get run, this will run `pylint` to analyze the source code and find any defects with it.
3. Then it will execute the generation of a report of the test coverage. This is to determine that the unit tests are covering all of the code paths that exist in the source.
4. Finally it will run `danger` to see if this run was from an active pull request. If so, then `danger` will report information about the results of the test and build back to the pull request so that the contributor gets feedback on their changes beyond a pass/fail from the CI.

Beyond that, the behavior of the uploading of artifacts and test reporting works the same way that it does on the Linux instances of CircleCI. I was using `make` to wrap most of the heavy lifting around executing various commands with building my code, I would recommend usage of it, or another tool to make the various commands you may need to run easier to handle rather than listing them all in the `circle.yml` file. The major take-away I had from this experience was that, unlike the Linux instances, you should expect to install almost everything yourself on the OS X instances with CircleCI. Take advantage of Homebrew and whatever other package manager you need to install what you need to perform a build.
