

# How to Test and Deploy a Python Web App

An Opinionated Tour of Testing Tools



Bear (Mike Taylor)  
@codebear

Site Reliability Engineer at CircleCI

Using Python to build, test and deploy applications at:  
&Yet  
Mozilla  
Seismic  
Open Source Applications Foundation



This talk will not be

- a deep dive into any one area of testing
- a debate on which style, framework or methodology is better

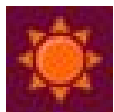
This talk will be

- Opinionated
- A list of what tools are available
- A collection of resources available
- Useful (hopefully)

# Tenki - A Weather Service



Flask Web site & API that returns the weather  
<https://github.com/bear/tenki>

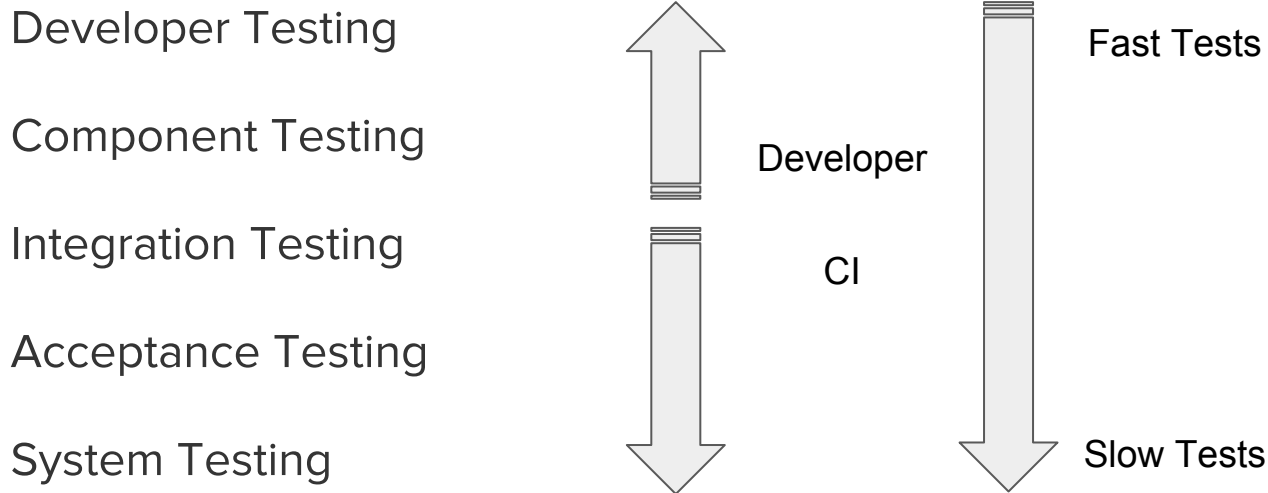


OpenWeatherMap  
<http://openweathermap.org/api>



PyOWM - Python wrapper for the  
OpenWeatherMap API  
<https://github.com/csparpa/pyowm>

# Python Web App Testing Path



# Developer Testing



- Must be *very* fast
- Started via Makefile
- pytest
- Must be fast
- pre-commit hooks
- PyEnv -- **NO System Python**
- Able to run specific test(s)
- Did I mention fast

# Developer Testing



- Must be *very* fast
- Started via Makefile
- pytest
- Must be fast
- pre-commit hooks
- PyEnv -- **NO System Python**
- Able to run specific test(s)
- Did I mention fast



# Developer Testing



- Must be *very* fast
- Started via Makefile
- pytest
- Must be fast
- pre-commit hooks
- PyEnv -- **NO System Python**
- Able to run specific test(s)
- Did I mention fast





# Developer Testing



- Must be *very* fast
- all commands run from a Makefile
- PyEnv -- **NO System Python**
- pytest
- pre-commit hooks
- Did I mention fast



```
clean:
    python manage.py clean

lint: clean
    pycodestyle --config={toxiniidir}/setup.cfg

test: clean
    python manage.py test

integration:
    python manage.py integration

coverage: lint
    @coverage run --source=tenki manage.py test
    @coverage html
    @coverage report

info:
    @python --version
    @pip --version
    @virtualenv --version

ci: info test integration coverage
    @CODECOV_TOKEN=`cat .codecov-token` codecov

all: clean integration coverage

install-hook:
    git-pre-commit-hook install --force --plugins json --plugins yaml \
        --plugins flake8 --flake8_ignore E111,E124,E126,E201,E202,E221,E241,E302,E501,N802,N803
```

```
import sys
from flask.ext.script import Manager, Server
from flask.ext.script.commands import Command, ShowUrls, Clean
from tenki import create_app

def test(marker="not web and not integration"):
    test_args = ['--strict', '--verbose', '--tb=long', 'tests', '-m', marker]
    import pytest
    errno = pytest.main(test_args)
    sys.exit(errno)

class Test(Command):
    def run(self):
        self.test_suite = True
        test()

class Integration(Command):
    def run(self):
        self.test_suite = True
        test(marker="integration")

class WebTests(Command):
    def run(self):
        self.test_suite = True
        test(marker="web")

app = create_app('tenki.settings.%sConfig' % env.capitalize())
manager = Manager(app)
manager.add_command("test", Test())
manager.add_command("integration", Integration())
manager.add_command("webtest", WebTests())
```



**pyenv** -- <https://pypi.python.org/pypi/pyenv>

```
bear@opus ~/circleci/tenki
$ pyenv virtualenv 2.7.10 tenki27
New python executable in
/usr/local/var/pyenv/versions/2.7/envs/tenki27/bin/python2.7
...
```

```
bear@opus ~/circleci/tenki
$ pyenv local tenki27
pyenv-virtualenv: deactivate
pyenv-virtualenv: activate tenki
```

```
(tenki27)
bear@opus ~/circleci/tenki
$ git clone git@github.com:bear/tenki.git .
```

```
(tenki27)
bear@opus ~/circleci/python_testing/tenki (master %)
$ make update-all
```

**pytest** -- <http://pytest.org/latest/>



- runs on Posix/Windows, Python 2.6-3.5, PyPy
- Mature testing framework that has no boilerplate,
- Doesn't require special syntax for assertions,
- Tests can be written as xUnit classes or as functions,
- pytest can run tests written using nose, unittest, and doctest.
- Marking test functions with attributes,
- Skip and xfail

# git pre-commit -- <https://pypi.python.org/pypi/git-pre-commit-hook>



built-in plugins for:

- validate json files
- validate Python-code with flake8
- validate Python-code with frosted
- validate .rst files with restructuredtext\_lint
- validate .ini files with configparser
- validate .yaml files with PyYAML
- validate .xml files with xml.etree.ElementTree
- check filesize



<http://trixie-j-lulamoon.deviantart.com/art/You-shall-not-pass-424148297>

```
[check-manifest]
```

```
ignore =  
    circle.yml
```

```
[pycodestyle]
```

```
ignore = E111,E124,E126,E201,E202,E221,E241,E302,E501
```

```
[tool:pytest]
```

```
markers =
```

```
    web: tests that require chrome webdriver
```

```
    integration: tests that require mocking or external services
```

# Component Testing



## Docker Compose

- nginx
- uwsgi
- robcherry/docker-chromedriver:latest

**NOTE** - Docker is not the only way to do WebUI or Component testing. Some consider it heavy-handed in that it requires developers to maintain a tech stack (which perversely re-introduces “worked on my stack” issues.)



```
uwsgi:
  build: .
  dockerfile: docker-uwsgi
web:
  build: .
  dockerfile: docker-nginx
  links:
    - uwsgi
  expose:
    - 8000
  ports:
    - "8000:8000"
chromedriver:
  image: robcherry/docker-chromedriver:latest
  links:
    - web
  ports:
    - "4444:4444"
  environment:
    CHROMEDRIVER_WHITELISTED_IPS: ""
```

**docker-cleanup:**

```
docker rm $(docker ps -q -f 'status=exited')  
docker rmi $(docker images -q -f "dangling=true")
```

**docker-build:**

```
docker-compose build  
docker-compose pull  
docker-compose rm -f
```

**docker-start:**

```
docker-compose up -d
```

**webtest: docker-start**

```
$(eval DRIVER_IP := $(shell ./wait_for_ip.sh))  
DOCKER_IP=$(DOCKER_IP) python manage.py webtest  
docker-compose stop
```

```
upstream app {  
    server uwsgi:8001;  
}  
  
server {  
    listen      8000    default_server;  
    server_name _;  
    charset     utf-8;  
  
    location / {  
        uwsgi_pass    app;  
        include        /etc/nginx/uwsgi_params;  
    }  
}
```



```
# bundle application as a tarball
BDATE=`date +%Y%m%d` git archive -o tenki_${BDATE}.tar.gz branch

# start uwsgi service
uwsgi --socket :8001 --chdir /opt/tenki --module uwsgi-app --callable
application

#!/usr/bin/env python
# uwsgi-app.py
import os
from tenki import create_app

env = os.environ.get('TENKI_ENV', 'prod')
application = create_app('tenki.settings.%sConfig' % env.capitalize())

if __name__ == "__main__":
    application.run()
```

# Deploy



Google Cloud and AppEngine

<https://circleci.com/docs/deploy-google-app-engine>

```
gcloud -q preview app deploy app.yaml --promote --version=staging
```

# Resources



Tox -- Run tests using different Python versions

Coverage.py -- Measure code coverage during test runs

Mock -- Replace code with mock objects

<https://pages.18f.gov/testing-cookbook/python/>

<http://docs.python-guide.org/en/latest/writing/tests/>

<https://pythonhosted.org/testing/>

# Review



Created a Developer environment that enforced good patterns

lint, unit tests, pre-commit checks, local testing

Created a component/integration test environment using Docker

This allows us to have developer and QA tests be identical

Created a Makefile to coordinate start, stop and test runs

Again, allows us to have the identical paths for Dev, QA, CI, CD

Hopefully learned something new!

# Questions?



CircleCI Support

<https://discuss.circleci.com/>

Example Repo

<https://github.com/bear/tenki/>