Quality of Life through Unit Testing

PyCon Malaysia 2015 Workshop

WHAT we wish to accomplish today?

1. Understand Unit Test in improving quality of life
2. Practice Unit Testing using python 3
3. Integrate Unit Testing as part of workflow

Materials

GitHub https://github.com/kiawin/pyconmy2015

```
git clone git@github.com:kiawin/pyconmy2015.git
vagrant up
vagrant ssh
```

Sian Lerk Lau

linkedin.com/in/sianlerk
sianlerk.lau@onapp.com | kiawin@gmail.com

Software Engineer

Volunteer

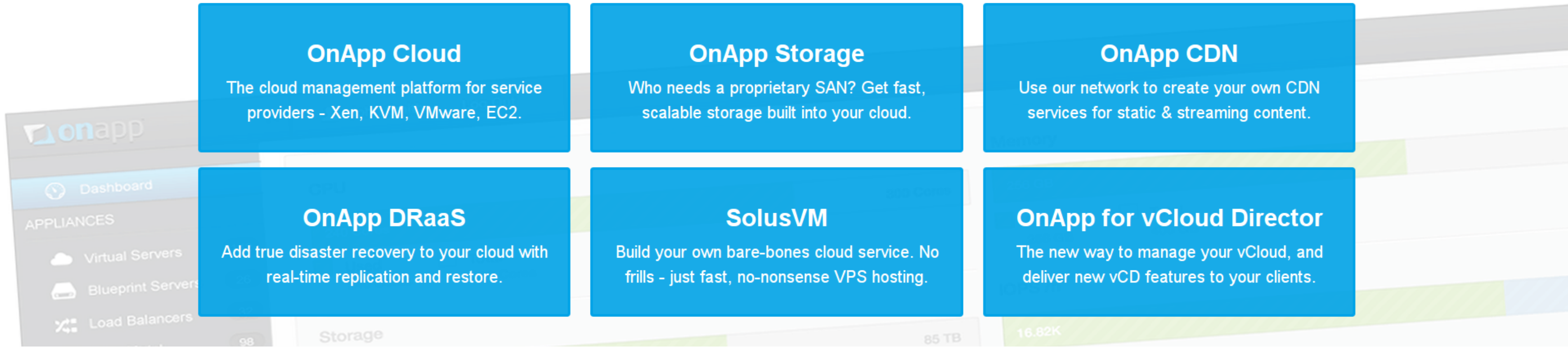
Educator-in-Exile

onapp™

onapp

https://twitter.com/OnApp - https://facebook.com/OnApp

What we do...



OnApp Cloud
The cloud management platform for service providers - Xen, KVM, VMware, EC2.

OnApp Storage
Who needs a proprietary SAN? Get fast, scalable storage built into your cloud.

OnApp CDN
Use our network to create your own CDN services for static & streaming content.

OnApp DRaaS
Add true disaster recovery to your cloud with real-time replication and restore.

SolusVM
Build your own bare-bones cloud service. No frills - just fast, no-nonsense VPS hosting.

OnApp for vCloud Director
The new way to manage your vCloud, and deliver new vCD features to your clients.

1. Understand Unit Test in improving quality of life

Quality of Life

is determined by

Quality of Code

Quality of Life

(of one)

is determined by

(the)

Quality of Code

(that one wrote)

Quality of Life

(of one)

is determined by

(the)

Quality of Code

(that someone wrote)

WHAT is unit?

A unit of work is a **single logical functional use case** in the system that can be **invoked by some public interface** (in most cases). A unit of work can span a single method, a whole class or multiple classes working together to achieve **one single logical purpose** that **can be verified**.

- The Art of Unit Test, Roy Osherove

WHAT is a good unit test

- Able to be fully automated
- Has full control over all the pieces running (Use mocks or stubs to achieve this isolation when needed)
- Can be run in any order if part of many other tests
- Runs in memory (no DB or File access, for example)
- Consistently returns the same result (You always run the same test, so no random numbers, for example. save those for integration or range tests)
- Runs fast
- Tests a single logical concept in the system
- Readable
- Maintainable
- Trustworthy (when you see its result, you don’t need to debug the code just to be sure)

2. Practice Unit Testing using python 3

```
vagrant up
vagrant provision
vagrant ssh
cd /vagrant/src
```

# Module / Guide

https://docs.python.org/3.4/library/unittest.html#module-unittest

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

2. Practice Unit Testing Trick #1 Doc-less Reference

```
vagrant@archlinux:/vagrant$ python
Python 3.4.3 (default, Mar 25 2015, 17:13:50)
[GCC 4.9.2 20150304 (prerelease)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import unittest
>>> dir(unittest)
['BaseTestSuite', 'FunctionTestCase', 'SkipTest', 'TestCase', 'TestLoader',
'TestProgram', 'TestResult', 'TestSuite', 'TextTestResult', 'TextTestRunner',
'_TextTestResult', '__all__', '__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__path__', '__spec__', '__unittest',
'case', 'defaultTestLoader', 'expectedFailure', 'findTestCases',
'getTestCaseNames', 'installHandler', 'loader', 'main', 'makeSuite',
'registerResult', 'removeHandler', 'removeResult', 'result', 'runner', 'signals',
'skip', 'skipIf', 'skipUnless', 'suite', 'util']
```

2. Practice Unit Testing Trick #1 Doc-less Reference

```
vagrant@archlinux:/vagrant$ python
Python 3.4.3 (default, Mar 25 2015, 17:13:50)
[GCC 4.9.2 20150304 (prerelease)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import unittest
>>> dir(unittest.TestCase)
['__call__', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
'__init__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '__weakref__', '_addExpectedFailure', '_addSkip',
'_addUnexpectedSuccess', '_baseAssertEqual', '_classSetupFailed', '_deprecate',
'_diffThreshold', '_feedErrorsToResult', '_formatMessage',
'_getAssertEqualityFunc', '_truncateMessage', 'addCleanup', 'addTypeEqualityFunc',
'assertAlmostEqual', 'assertAlmostEquals', 'assertCountEqual',
'assertDictContainsSubset', 'assertDictEqual', 'assertEqual', 'assertEquals',
'assertFalse', 'assertGreater', 'assertGreaterEqual', 'assertIn', 'assertIs',
'assertIsInstance', 'assertIsNone', 'assertIsNot', 'assertIsNotNone',
'assertLess', 'assertLessEqual', 'assertListEqual', 'assertLogs',
'assertMultiLineEqual', 'assertNotAlmostEqual', 'assertNotAlmostEquals',
'assertNotEqual', 'assertNotEquals', 'assertNotIn', 'assertNotIsInstance',
'assertNotRegex', 'assertRaises', 'assertRaisesRegex', 'assertRaisesRegexp',
```

2. Practice Unit Testing Trick #1 Doc-less Reference

```
vagrant@archlinux:/vagrant$ python
Python 3.4.3 (default, Mar 25 2015, 17:13:50)
[GCC 4.9.2 20150304 (prerelease)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import unittest
>>> import inspect
>>> inspect.getargspec(unittest.TestCase.assertEqual)
ArgSpec(args=['self', 'first', 'second', 'msg'], varargs=None, keywords=None,
defaults=(None,))
```

Ref: http://stackoverflow.com/questions/218616/getting-method-parameter-names-in-python

2. Practice Unit Testing Trick #2 Import Module

```
vagrant@archlinux:/vagrant/src$ python tests/calculator_tests.py
Traceback (most recent call last):
  File "tests/calculator_tests.py", line 1, in <module>
    import calculator
ImportError: No module named 'calculator'

# Solution - Set PYTHONPATH
PYTHONPATH=./ python tests/calculator_tests.py
```

Ref: http://stackoverflow.com/questions/5602559/where-is-the-python-path-set-when-i-dont-have-a-bash-profile

2. Practice Unit Testing Trick #3 Autorun Test

```
watch -n 2 "PYTHONPATH=./ python tests/calculator_tests.py"
```

```
Every 2.0s: PYTHONPATH=./ python tests/calculator_tests.py     Thu Aug 20 13:02:39 2015

F
======================================================================
FAIL: test_add (__main__.TestCalculator)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "tests/calculator_tests.py", line 14, in test_add
    self.assertEqual(c.add(1,2),2,"Alamak?")
AssertionError: 3 != 2 : Alamak?

----------------------------------------------------------------------
Ran 1 test in 0.002s

FAILED (failures=1)
```

2. Practice Unit Testing Trick #3 Autorun Test

`watch -n 2 "PYTHONPATH=./ python tests/calculator_tests.py"`

```
Every 2.0s: PYTHONPATH=./ python tests/calculator_tests.py            Thu Aug 20 13:11:44 2015

.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

2. Practice Unit Testing Examples

1. simple calculator
2. simple assertion
3. use setUp
4. use tearDown
5. mock / patch

3. Integrate Unit Testing as part of workflow

- Validate your work - Integration
- Part of development flow - TDD

Our Way

- **Unit tests** as part of development process
- Continuous integration using Jenkins
- **Behavioral tests** to perform functional, integration and regression tests on applications.
- **Performance tests** based on defined metrics

We’re hiring!

- **System Admins** as integral role in managing and develop tools for our ecosystem
- **Software Developers** as engineering role in creating bleeding edge applications for our ecosystem

Wonderful things we use

Python, Java, Ruby, Lua, Nginx, Wowza, Puppet, Vagrant, Docker, Debian, Cucumber, RabbitMQ, MariaDB, MongoDB, ELK, etc.

Q&A