PYTEST ALL THE THINGS

VINCENT BERNAT

KILLER FEATURES

THE ASSERT KEYWORD

- With `unittest`, you have to use provided functions:

```
class TestOperators(unittest.TestCase):
    def test_addition():
        self.assertEqual(operator.add(1, 4), 5)
        self.assertEqual(operator.add(7, 5), 12)
```

- With `pytest`, you just use the `assert` keyword:

```
def test_addition():
    assert operator.add(1, 4) == 5
```

ASSERT SMARTNESS

- **Useful** information when an assertion fails:

```
    def test_addition():
>       assert operator.add(1, 3) == 5
E       assert 4 == 5
E        +  where 4 = <built-in function add>(1, 3)
E        +    where <built-in function add> = operator.add

test1.py:3: AssertionError
```

- **Compact** representation of failures:

```
    def test_list():
>       assert range(1, 1000) == range(1, 1002)
E       assert range(1, 1000) == range(1, 1002)
E         Right contains more items, first extra item: 1000
E         Use -v to get the full diff

test1.py:6: AssertionError
```

FIXTURES

- With `unittest`, you can only have **one fixture**
- Use `setUp()` and `tearDown()` methods

```
class testInVM(unittest.TestCase):

    def setUp(self):
        self.vm = VM('Test-VM')
        self.vm.start()
        self.ssh = SSHClient()
        self.ssh.connect(self.vm.public_ip)

    def tearDown(self):
        self.ssh.close()
        self.vm.destroy()

    def test_hello(self):
        stdin, stdout, stderr = self.ssh.exec_command("echo hello")
        stdin.close()
        self.assertEqual(stderr.read(), b"")
        self.assertEqual(stdout.read(), b"hello\n")
```

FIXTURES WITH PYTEST

- Each test can have an **arbitrary number** of fixtures
- Fixtures can use **other fixtures**
- Specified with **dependency** injections

```
@pytest.yield_fixture
def vm():
    r = VM('Test-VM')
    r.start()
    yield r
    r.destroy()

@pytest.fixture
def ssh(vm):
    ssh = SSHClient()
    ssh.connect(vm.public_ip)
    return ssh

def test_hello(ssh):
    stdin, stdout, stderr = ssh.exec_command("echo hello")
    stdin.close()
    stdout.read() == b"hello\n"
```

PARAMETRIZATION

- You could use a **loop** in a test:

```
class TestOperators(unittest.TestCase):
    def test_addition():
        for a, b, result in [
                (1, 4, 5), (5, 10, 15), (10, -10, 0)]:
            self.assertEqual(operator.add(a, b), result)
```

- You could create functions **on the fly**:

```
class TestOperators(unittest.TestCase):
    @classmethod
    def _test_addition(cls, a, b, result):
        def f(self):
            self.assertEqual(operator.add(a, b), result)
        setattr(cls, "test_addition_{}".format(idx), f)

for idx, m in enumerate([
        (1, 4, 5), (5, 10, 15), (10, -10, 0)]):
    a, b, result = m
    TestOperators._test_addition(a, b, result)
```

PARAMETRIZATION WITH PYTEST

- You can use **decorators**:

```
@pytest.mark.parametrize("a, b, result", [
    (1, 3, 4),
    (8, 20, 28),
    (-4, 0, -4)])
def test_addition(a, b, expected):
    assert operator.add(a, b) == expected
```

TEST RUNNER

SELECT TESTS TO RUN

- Run only one file:

```
py.test test_operators.py
```

- Run only tests in a subdirectory:

```
py.test batman/
```

- Run only tests **matching** a string expression:

```
py.test -k with_pandas
```

- Run only tests **marked** with the `fast` marker:

```
py.test -m fast
```

ERROR HANDLING

- Stop after the **first error**:

```
py.test -x
```

- Stop after three errors:

```
py.test --max-failed=3
```

- Rerun **failed tests**:

```
py.test --last-failed
```

- Same but on a loop:

```
py.test -f
```

- Go into **PDB** on error:

```
py.test --pdb
```

DISTRIBUTED TESTS

- It's the `pytest-xdist` plugin
- Distribute on **multiple CPU**:

```
py.test -n 5
```

- Distribute on **remote hosts**:

```
py.test --dist=load --tx ssh=host1 --tx ssh=host2 --rsyncdir pkg pkg
```

- Run tests on **different platforms**:

```
py.test --dist=each --tx ssh=linux --tx ssh=windows --tx ssh=osx --rsyncd
```

EXAMPLES

TESTING A DJANGO APP

- Use `pytest-django`

```
def test_foobar(client):
    assert client.get('/foobar') == 'foobar'

def test_foobar_admin(admin_client):
    assert admin_client.get('/foobar') == 'super foobar'

@pytest.mark.parametrize("stuff, result", [
    (False, 'no stuff'),
    (True, 'stuff')])
def test_with_and_without_stuff(settings, client, stuff, result):
    settings.USE_STUFF = stuff
    assert client.get('/stuff') == result
```

TESTING A NETWORK APPLICATION

- **lldpd** is a C implementation of 802.1AB, some neighbor discovery protocol
- old school
- limited unittests
- **integration tests** with pytest using Linux namespaces

TESTING LLDPD

```
@pytest.mark.skipif('LLDP-MED' not in pytest.config.lldpd.features,
                    reason="LLDP-MED not supported")
@pytest.mark.parametrize("classe, expected", [
    (1, "Generic Endpoint (Class I)"),
    (2, "Media Endpoint (Class II)"),
    (3, "Communication Device Endpoint (Class III)"),
    (4, "Network Connectivity Device")])
def test_med_devicetype(lldpd, lldpcli, namespaces, links,
                        classe, expected):
    links(namespaces(1), namespaces(2))
    with namespaces(1):
        lldpd("-r")
    with namespaces(2):
        lldpd("-M", str(classe))
    with namespaces(1):
        out = lldpcli("-f", "keyvalue", "show", "neighbors", "details")
        assert out['lldp.eth0.lldp-med.device-type'] == expected
```

DEMO

TESTING VM DEPLOYMENT

- This is an advertising/subliminal slide for Exoscale, you Swiss Cloud Hosting

TESTING VM DEPLOYMENT

- Many **distributions**
- Many **disk sizes**
- Many **service offerings** (CPU/memory)
- Many **availability zones**
- This seems a job for **pytest**

USE COMMAND-LINE OPTIONS

```
default = {
    'jobtimeout': 60,
    'distributions': ['Ubuntu 14.04'],
    'sizes': ['50'],
    'serviceofferings': ['tiny']
}
def pytest_addoption(parser):
    parser.addoption("--zone", action="append", default=[],
                     help="list of zone to test")
    parser.addoption("--distribution", action="append", default=[],
                     help="list of distributions to test")
    parser.addoption("--size", action="append", default=[],
                     help="list of disk sizes to test")
    parser.addoption("--serviceoffering", action="append", default=[],
                     help="list of service offerings to test")
def pytest_generate_tests(metafunc):
    for f in ['zone', 'distribution', 'size', 'serviceoffering']:
        if f in metafunc.fixturenames:
            metafunc.parametrize(f,
                                 getattr(metafunc.config.option, f) or
                                 default['{}s'.format(f)],
                                 scope='module')
```

EXCERPTS

- To get a VM, we need a template ID:

```
@pytest.fixture(scope='module')
def templateid(cloudstack, zoneid, distribution, size):
    templates = cloudstack.listTemplates(templatefilter='featured',
                                         zoneid=zoneid)['template']
    templates = [t for t in templates
                 if re.search(r'\b{}\b'.format(re.escape(distribution)),
                              t['name']) and
                 int(t['size'] / 1024 / 1024 / 1024) == int(size)]
    return templates[0]['id']
```

- Fixture to create a VM:

```
@pytest.yield_fixture(scope='module')
def vm(cloudstack, serviceofferingid, securitygroupid, templateid, zoneid
    v = cloudstack.deployVirtualMachine(
        serviceofferingid=serviceofferingid, templateid=templateid,
        zoneid=zoneid, securitygroupids=[securitygroupid],
        name="pytest-{}".format(something())
    yield v
    cloudstack.destroyVirtualMachine(id=v['id'])
```

MORE EXCERPTS

- We want an SSH connection to the VM

```
@pytest.fixture(scope='module')
def runningvm(request, cloudstack, vm):
    wait_for_vm(cloudstack, vm, 10)
    wait_for_ssh(vm, 30)
    return vm
@pytest.yield_fixture(scope='module')
def sshvm(runningvm):
    with SSHClient() as client:
        client.set_missing_host_key_policy(AutoAddPolicy())
        client.connect(runningvm['ipaddress'],
                       username="root", timeout=10,
                       password=runningvm['password'])
        yield client
```

TESTING FOR DISK SIZE

- Now, we can run real tests on the VM content:

```
def test_disk_size(sshvm, size):
    stdin, stdout, stderr = sshvm.exec_command(
        "df --block-size=1G / | tail -1 | awk '{print $2}'")
    stdin.close()
    assert stderr.read() == b""
    realsize = int(stdout.read().strip().decode('ascii'))
    intsize = int(size)
    assert abs(realsize - intsize) <= intsize * 0.05
```

DEMO

```
py.test-3 -v test_vm.py \
   --zone=ch-gva-2 --zone=ch-dk-2 \
   --size=50 \
   --distribution='Debian 8' \
```

TEST A WEB PAGE

- Use **Selenium** to drive a browser

```
import pytest
from selenium import webdriver
@pytest.yield_fixture(scope='module')
def browser():
    d = webdriver.Chrome(executable_path='/usr/lib/chromium/chromedriver'
    yield d
    d.quit()
```

TEST A WEB PAGE

- First example

```
from selenium.webdriver.support.ui import WebDriverWait as wait
from selenium.webdriver.support import expected_conditions as EC
@pytest.mark.parametrize("search", [
    "Django Python Meetup at Lausanne",
    "exoscale", "cats"])
def test_google(browser, search):
    browser.get('https://www.google.ch')
    input = browser.find_element_by_name("q")
    input.send_keys(search)
    input.submit()
    wait(browser, 10).until(EC.title_contains("Google-Suche"))
    assert browser.title == '{} - Google-Suche'.format(search)
```

TEST A WEB PAGE

- Second example

```
@pytest.fixture
def logged_browser(browser):
    browser.get('https://portal.exoscale.ch')
    email = browser.find_element_by_name("email")
    email.send_keys("vbe+batman@exoscale.ch")
    password = browser.find_element_by_name("password")
    password.send_keys("alfred0")
    password.submit()
    wait(browser, 10).until(EC.title_contains('Exoscale Console'))
    return browser
def test_account_login(logged_browser):
    login = logged_browser.find_element_by_class_name("ellipsis-whoami")
    assert login.text == "vbe+batman@exoscale.ch"
```

QUESTIONS?