Testinfra test your salt infrastructure

Philippe Pepiot <phil@philpep.org>

Author and maintainer of testinfra

Software engineer and devops @

Logilab

Testinfra

- Test the actual state of your server configured by salt, ansible, puppet, chef and so on.
- In python!
- Tests are written in python too!
- https://testinfra.readthedocs.io
- https://github.com/philpep/testinfra
- Licence Apache 2

Infrastructure as code

- yaml and jinja are code
- We manage the code in DVCS (git, mercurial)
- How to test it?

Test and staging servers

- Good solution for a company
- Doesn't offer good concurrency when there are many developpers.
- Not an option for open source project with external contributors (salt formulas, ansible roles, puppet modules)

Virtualization

- Run the deployment in a ephemeral test environment

Tools

- qemu https://www.qemu.org
- docker https://docker.com
- lxc https://linuxcontainers.org
- `$your_favorite_container_tool`
- Openstack / EC2 / GCE / K8S

Test process

- Run a VM or a container
- Deploy it with your config management tool
- Verify
- Destroy the VM or the container

Tools

- Vagrant https://www.vagrantup.com
- Test-kitchen http://kitchen.ci
- Molecule https://molecule.readthedocs.io

CI

- Submit pull requests
- Review
- Test them automatically
- Jenkins https://jenkins.io
- Gitlab CI
- Travis https://travis-ci.org (we can run docker containers in travis !)

Verify the deployment

- monitoring is good, but comes too late, we want to test before code is actually merged.
- As a maintainer of infrastructure code, testing PR is boring, red / green status helps a lot.
- Your infrastructure code is an API, testing it is a good way to keep it stable enough and make users confident with upgrades.

Testinfra

- First version in 2015
- In python with `pytest` integration

```
def test_apache_installed(host):
    if host.system_info.distribution == 'redhat':
        pkgname = 'httpd'
    else:
        pkgname = 'apache2'
    assert host.package(pkgname).is_installed
```

pytest

- All function named `test_...` are tests
- The `host` object is a `pytest` fixture defined in testinfra
- Lot of features (not covered here) https://docs.pytest.org

Running the tests

```
$ py.test mytest.py
mytest.py::test_apache_installed[local://] PASSED
```

```
$ py.test mytest.py
[...]
mytest.py::test_apache_installed[local://] FAILED
[...]
  def test_apache_installed(host):
      if host.system_info.distribution == 'redhat':
          pkgname = 'httpd'
      else:
          pkgname = 'apache2'
>     assert host.package(pkgname).is_installed
E     AssertionError: assert False
E      +  where False = <package apache2>.is_installed
```

Testinfra

- testinfra runs commands locally or on a remote host
- write assertions on the actual state of the server
- some system abstractions (services, packages)

Connection backends

- The `host` API is available through `testinfra.get_hosts()`
- Supports different connection types (local, paramiko, docker, salt, ansible, kubectl, winrm)
- With some connection options (user, use sudo, ansible inventory to use).
- Supports testing multiple hosts through pytest test parameterization!

```
$ py.test --host=paramiko://user@host
test.py::test_postgres[pramiko://user@host] PASSED

$ py.test --host=docker://boring,docker://wozniak
test.py::test_postgres[docker://boring] PASSED
test.py::test_postgres[docker://wozniak] PASSED

$ py.test --host 'salt://web*'
```

Run commands

```
>>> import testinfra
>>> host = testinfra.get_host('ssh://somehost')
>>> host.run('echo foo;echo bar >/dev/stderr; exit 42')
CommandResult(rc=42, stdout='foo', stderr='bar')
```

Write assertions

```
>>> assert host.run('echo foo').stdout == 'bar'
Traceback (most recent call last):
  [...]
AssertionError
```

Modules

package

- apt, rpm, *BSD

```
>>> host.package('nginx').is_installed
True
>>> host.package('nginx').version
'1.2.1-2.2+wheezy3'
```

service

- Systemd, Upstart, Sysv, *BSD

```
>>> host.service('nginx').is_enabled
True
>>> host.service('nginx').is_running
False
```

Modules

https://testinfra.readthedocs.io/en/latest/modules.html

ansible, file, group, interface, mount_point, package, pip_package, process, puppet_resource, facter, salt, service, socket, sudo, supervisor, sysctl, system_info, user

```
def test_postgres(host):
    postgresql = host.service('postgresql')
    assert postgresql.is_enabled
    assert postgresql.is_running

    with host.sudo('postgres'):
        assert host.check_output(
            "psql -tAc 'show shared_buffers'"
        ) == '256MB'

    assert sum([p.pmem for p in
                host.process.filter(user='postgres')
               ]) < 80
    assert len(host.socket('tcp://5432').clients) < 50
```

salt module

```
>>> host.salt("pkg.version", "nginx")
'1.6.2-5'
>>> host.salt("state.sls", "nginx.ng")
{'pkg_|-nginx_install_|-nginx_|-installed': {
  {'changes': {'nginx': {'new': '1.10.3-1+deb9u1', 'old':
   'comment': 'The following packages were installed/upda
   'duration': 42310.523,
   'name': 'nginx',
   'result': True}},
 'file_|-nginx_server_enabled_dir_|-/etc/nginx/sites-enab
   'comment': 'Directory /etc/nginx/sites-enabled is in t
   'duration': 3.436,
   'name': '/etc/nginx/sites-enabled',
   'pchanges': {},
   'result': True}}
```

Test multi-hosts

```
import uuid
import testinfra

def test_nfs_responsive():
    srv0 = testinfra.get_host('salt://minion0')
    srv1 = testinfra.get_host('salt://minion1')
    uid = str(uuid.uuid4())
    srv0.check_output('echo %s > /nfs/test.tmp', uid)
    assert srv1.file('/nfs/test.tmp').content == uid
```

Reusing tests for monitoring

- Infrastructure tests can be close to monitoring checks
- With the option `--nagios` testinfra turns into a nagios compatible plugin
- Share your test code between CI and production monitoring

```
$ testinfra -q --nagios my_test.py
TESTINFRA OK - 1 passed, 0 failed, 0 skipped in 0.02 secc
```

Test-kitchen

- testinfra can be used as a test-kitchen verifier

```
verifier:
  name: shell
  command: testinfra --host="paramiko://
  ${KITCHEN_USERNAME}@${KITCHEN_HOSTNAME}:${KITCHEN_PORT}
  ?ssh_identity_file=${KITCHEN_SSH_KEY}"
  --junit-xml "junit-${KITCHEN_INSTANCE}.xml"
  "test/integration/${KITCHEN_SUITE}"
```

Vagrant

```
$ vagrant ssh-config > .vagrant/ssh-config
$ py.test --hosts=default \
    --ssh-config=.vagrant/ssh-config tests/
```

Some test suites examples

- https://github.com/ceph/ceph-ansible
- https://github.com/freedomofpress/securedrop
- https://github.com/saltstack-formulas/docker-formula
- https://github.com/saltstack-formulas/elasticsearch-formula
- https://github.com/saltstack-formulas/kibana-formula

Others projects

- serverspec (BDD, ruby)
- goss (json spec file, local only): https://github.com/aelsabbahy/goss
- inspec for chef https://www.inspec.io/

Roadmap

- run tests asynchronously
- run background commands

```
with host.run('journalctl -f') as output:
    testmyapp()
assert 'some log' in output.stdout
```

- Having a stable API for maintaining external modules

Example

- write tests for https://github.com/saltstack-formula/nginx-formula
- using docker and testinfra

Dockerfile

```
FROM debian:stretch

RUN apt-get update && apt-get -y install salt-minion \
    systemd-sysv net-tools
RUN rm /etc/systemd/system/multi-user.target.wants/salt*
ADD test-minion.conf /etc/salt/minion.d/minion.conf
ADD nginx /srv/nginx
RUN salt-call --retcode-passthrough --hard-crash \
    state.sls nginx.ng
CMD ["/sbin/init"]
```

test-minion.conf

```
file_roots:
  base:
    - /srv
file_client: local
log_level: info
```

test_nginx_formula.py

```
import os
import subprocess

import pytest
import requests
import testinfra

ROOT = os.path.dirname(__file__)


@pytest.fixture
def host():
    subprocess.check_call([
        'docker', 'build', '-t', 'test-nginx-formula',
        ROOT])
    docker_id = subprocess.check_output([
        'docker', 'run', '-d', '--privileged',
        'test-nginx-formula',
    ]).decode().strip()
    yield testinfra.get_host('docker://' + docker_id)
    subprocess.check_output([
        'docker', 'rm', '-f', docker_id])
```

test_nginx_formula.py

```
def test_nginx(host):
    assert host.package('nginx').is_installed
    assert host.service('nginx').is_running
    assert host.socket('tcp://80').is_listening
    ip = host.interface('eth0').addresses[0]
    response = requests.get('http://{}'.format(ip))
    assert response.status_code == 200
```

test_nginx_formula.py

```
def test_state_indempotence(host):
    summary = host.salt('state.sls', 'nginx.ng')
    assert all(s['result'] is True
               and s['changes'] == {}
               for s in summary.values())
```

test_nginx_formula.py

```
def test_pillars(host):
    ip = host.interface('eth0').addresses[0]
    response = requests.get('http://{}'.format(ip))
    assert response.status_code == 200
    assert response.headers['server'] == 'nginx/1.10.3'
    host.salt('state.sls', ['nginx.ng', (
        "pillar={'nginx': {'ng': {'server': "
        "{'config': {'http': {'server_tokens': 'off'}"
        "}}}}}")])
    response = requests.get('http://{}'.format(ip))
    assert response.status_code == 200
    assert response.headers['server'] == 'nginx'
```

Questions ?

http://slides.logilab.fr/2018/testinfra_cfgmgmtcamp_2018.pdf