Infrastructure and Linux package testing with Molecule/TestInfra

Tomislav Plavčić

Percona

Topics

- Infrastructure as Code
- Testing IaC / Frameworks
- Molecule
- TestInfra
- Examples / Demo

Infrastructure as Code

IaC

PERCONA

Infrastructure as code benefits

Infrastructure as Code (IaC) is a method to provision and manage IT infrastructure through the use of source code, rather than through standard operating procedures and manual processes.

You're basically treating your servers, databases, networks, and other infrastructure like software.

Benefits:

- Speed and simplicity
- Configuration consistency
- Minimization of risk
- Cost savings
- Increased efficiency in software development

PERCONA

Infrastructure as code tools

Open source tools:

- Chef
- Puppet
- Ansible (RedHat)
- Terraform (HashiCorp, multi-cloud)

Cloud specific tools:

- CloudFormation (AWS)
- Google Cloud Deployment Manager
- Azure Resource Manager

PERCONA

Differences

- Configuration Management vs Provisioning
- Procedural vs Declarative
- Master vs Masterless
- Agent vs Agentless (Pull vs Push)

Ansible

- Maintained by RedHat.
- Playbooks written in YAML files.

- Configuration management
- Procedural/Declarative (Hybrid)
- Masterless
- Agentless (Push)

Infrastructure as code testing

PERCONA

Why test infrastructure as code?

- It changes
- It is code
- Make changes with more confidence
- Faster troubleshooting
- Continuous compliance and security standards

Frameworks

Ansible / Molecule / TestInfra stack

- Molecule (python)
  - Tool for testing Ansible roles. By default uses Testinfra for verification (from v3 Ansible is default).
- TestInfra (python)
  - Unit tests in Python to test actual state of your servers configured by management tools.

Chef / TestKitchen / InSpec stack

- Test Kitchen (ruby)
  - Test Kitchen is an integration tool for developing and testing infrastructure code and software on isolated target platforms.
- InSpec (ruby)
  - Chef InSpec is a free and open-source framework for testing and auditing your applications and infrastructure.

- ServerSpec (ruby)
  - ServerSpec is an extension the ruby RSpec framework.

PERCONA

Frameworks

- Goss (go)
  - YAML based serverspec alternative tool for validating a server's configuration.
  - Allows the user to generate tests from the current system state.
- Container Structure Tests (go)
  - github: GoogleContainerTools/container-structure-test
  - Can be used to check the output of commands in an image, as well as verify metadata and content of the filesystem.
  - Tests can be run either through a standalone binary, or through a Docker image.

PERCONA

Our use case

We provide MySQL, MongoDB, PostgreSQL and some other tools to the users in form of:

- Debian/Ubuntu packages (apt repo)
- RedHat/CentOS packages (yum repo)
- Docker images (dockerhub repo)
- binary tarballs

Testing packages before they become part of user infrastructure including some smoke tests for features.

Checking test environments and setup.

PERCONA

Molecule

PERCONA

Molecule

- https://github.com/ansible-community/molecule
- Designed to aid in the development and testing of Ansible roles.
- Uses Ansible to manage instances to operate on and supports any provider Ansible supports.
- Maintained by Ansible community.

Testing with multiple:

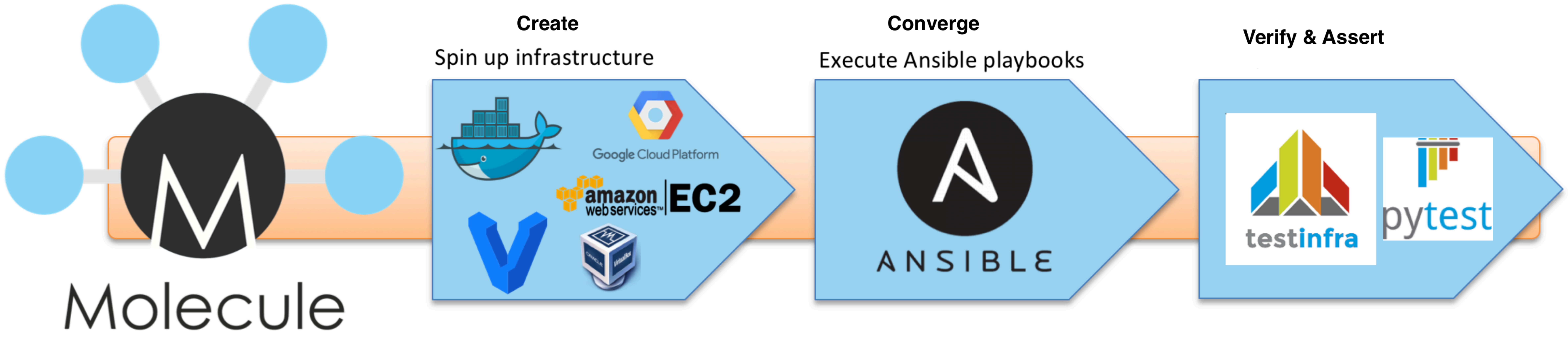
- instances
- operating systems and distributions
- virtualization providers
- test frameworks
- testing scenarios

Install

Pip is the only supported installation method.

```
apt install python python-pip
python -m venv ~/python-venv/python-molecule
source ~/python-venv/python-molecule/bin/activate
pip install molecule 'molecule[docker]'
```

Workflow



source: https://blog.codecentric.de/en/2018/12/test-driven-infrastructure-ansible-molecule/

Drivers supported

- DigitalOcean
- Docker
- EC2
- GCE
- Hetzner Cloud
- Linode
- LXC
- LXD
- Openstack
- Podman
- Vagrant
- Azure

PERCONA

Project structure

```
.
├── README.md
├── defaults
│   └── main.yml
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── molecule
│   ├── default                    → Default scenario
│   │   ├── cleanup.yml
│   │   ├── molecule.yml           → Main config file
│   │   ├── playbook.yml
│   │   └── tests                  → TestInfra tests
│   │       └── test_default.py
│   └── ec2                        → Alternative scenario
│       ├── molecule.yml
│       └── playbook.yml
├── tasks
│   └── main.yml                   → Role playbook
└── vars
    └── main.yml
```

molecule.yml

Main config file which defines:

- dependency manager (default: galaxy)
- driver provider (default: docker)
- lint command
- platforms definition
- provisioner (ansible only)
- scenario definition
- verifier framework (from v3 default is: Ansible)

molecule.yml example

```
dependency:
  name: galaxy
driver:
  name: vagrant
  provider:
    name: virtualbox
lint:
  name: yamllint
platforms:
  - name: new-project-server
    box: ubuntu/bionic64
    instance_raw_config_args:
      - "vm.network 'forwarded_port', guest: 80, host: 8088"
provisioner:
  name: ansible
  lint:
    name: ansible-lint
verifier:
  name: testinfra
  lint:
    name: flake8
```

```
test_sequence:
  - lint
  - destroy
  - dependency
  - create
  - prepare
  - converge
  - verify
  - cleanup
  - destroy
```

PERCONA

Basic commands

```
# create new role
molecule init role my-new-role1 --verifier-name testinfra --driver-name docker

# create instances
molecule create

# list created instances
molecule list

# run ansible playbook
molecule converge

# run verifier assertions
molecule verify

# destroy created instances
molecule destroy

# run the whole create/converge/verify/destroy workflow
molecule test
```

TestInfra

Testinfra overview

Write unit tests in Python to test actual state of your servers configured by management tools like Salt, Ansible, Puppet, Chef...

Testinfra aims to be a Serverspec equivalent in python and is written as a plugin to the powerful Pytest test engine.

Has "--junit-xml" parameter for exporting and loading test results into jenkins.

Installation:

```
pip install testinfra
```

PERCONA

Modules

Testinfra modules are provided through the host fixture, declare it as arguments of your test function to make it available within it.

Example modules:

- file
- group
- package
- environment
- process
- service
- user
- socket
- docker

PERCONA

Test examples

```
def test_passwd_file(host):
    passwd = host.file("/etc/passwd")
    assert passwd.contains("root")
    assert passwd.user == "root"
    assert passwd.group == "root"
    assert passwd.mode == 0o644


def test_nginx_is_installed(host):
    nginx = host.package("nginx")
    assert nginx.is_installed
    assert nginx.version.startswith("1.2")


def test_nginx_running_and_enabled(host):
    nginx = host.service("nginx")
    assert nginx.is_running
    assert nginx.is_enabled
```

PERCONA

Connection backends

- local (default when no hosts provided, commands are run locally in a subprocess under the current user)
- paramiko (python implementation of ssh protocol)
- docker
- podman
- ssh
- salt
- ansible
- kubectl
- openshift
- winrm
- lxc/lxd

PERCONA

Examples / Demo

PERCONA

About me

- Tomislav Plavčić
- QA Engineer at Percona
- @tplavcic
- tomislav.plavcic@percona.com

Thank You!

PERCONA