Molecule Documentation

Release 2.17.0

AUTHORS.rst

Aug 14, 2018

Contents

Molecule is designed to aid in the development and testing of Ansible roles. Molecule provides support for testing with multiple instances, operating systems and distributions, virtualization providers, test frameworks and testing scenarios. Molecule is opinionated in order to encourage an approach that results in consistently developed roles that are well-written, easily understood and maintained.

Molecule uses Ansible playbooks to exercise the role and its associated tests. Molecule supports any provider[1] that Ansible supports.

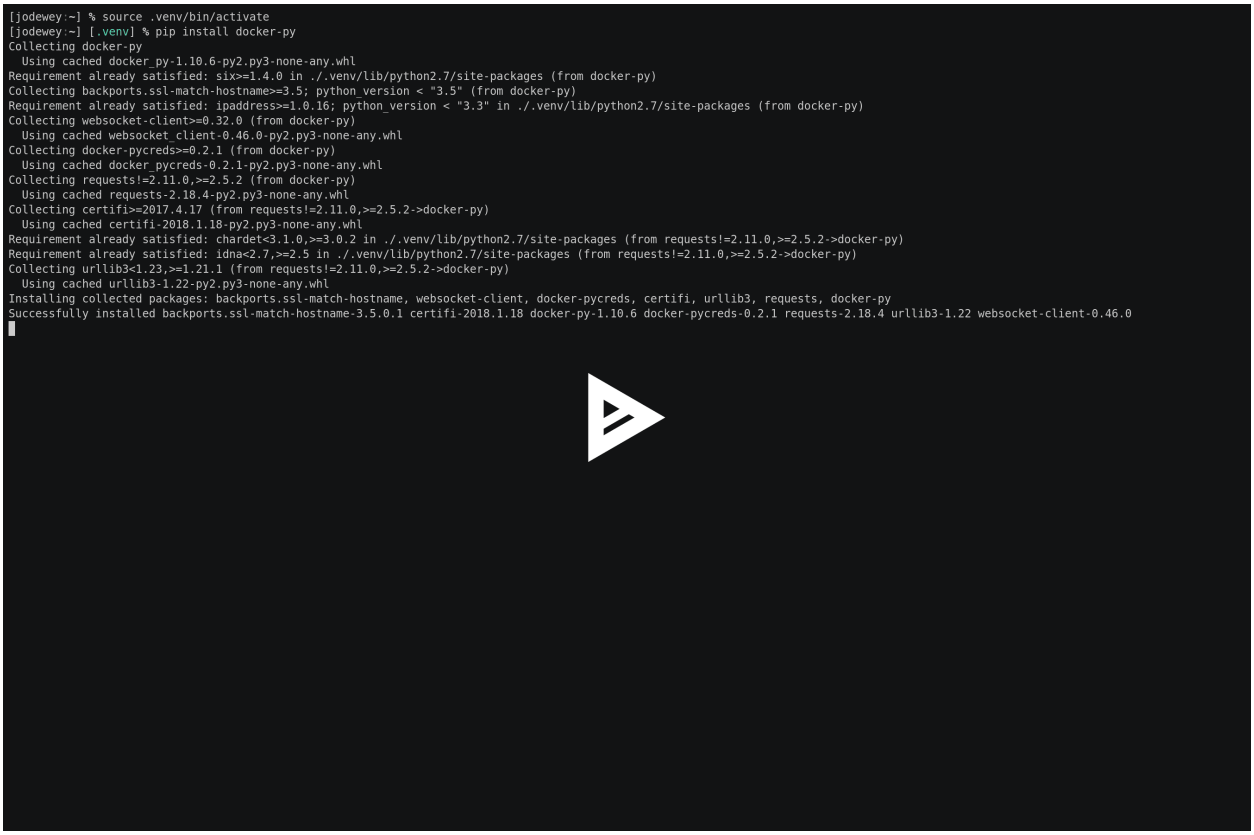
[1] Providers can be bare-metal, virtual, cloud or containers. If Ansible can use it, Molecule can test it. Molecule simply leverages Ansible's module system to manage instances.

CHAPTER 1

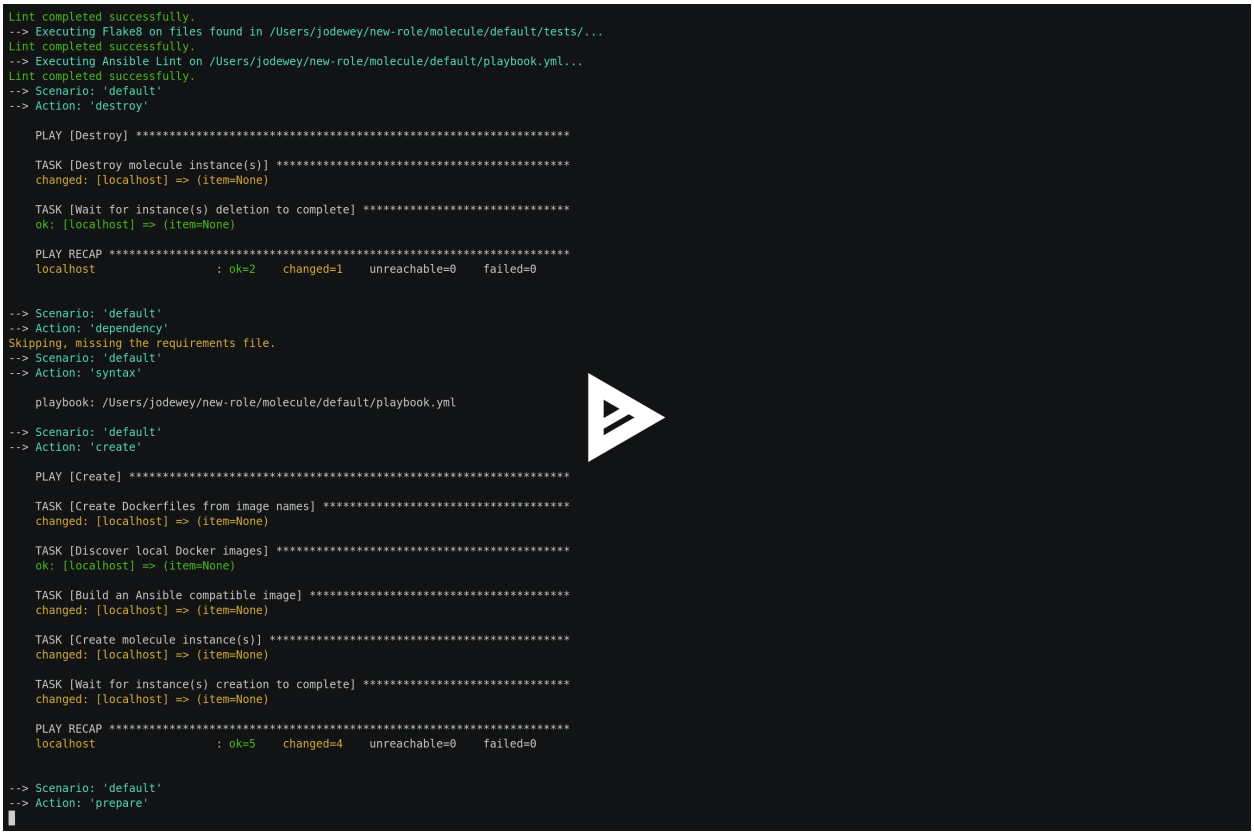
Quick Start

1.1 Installing

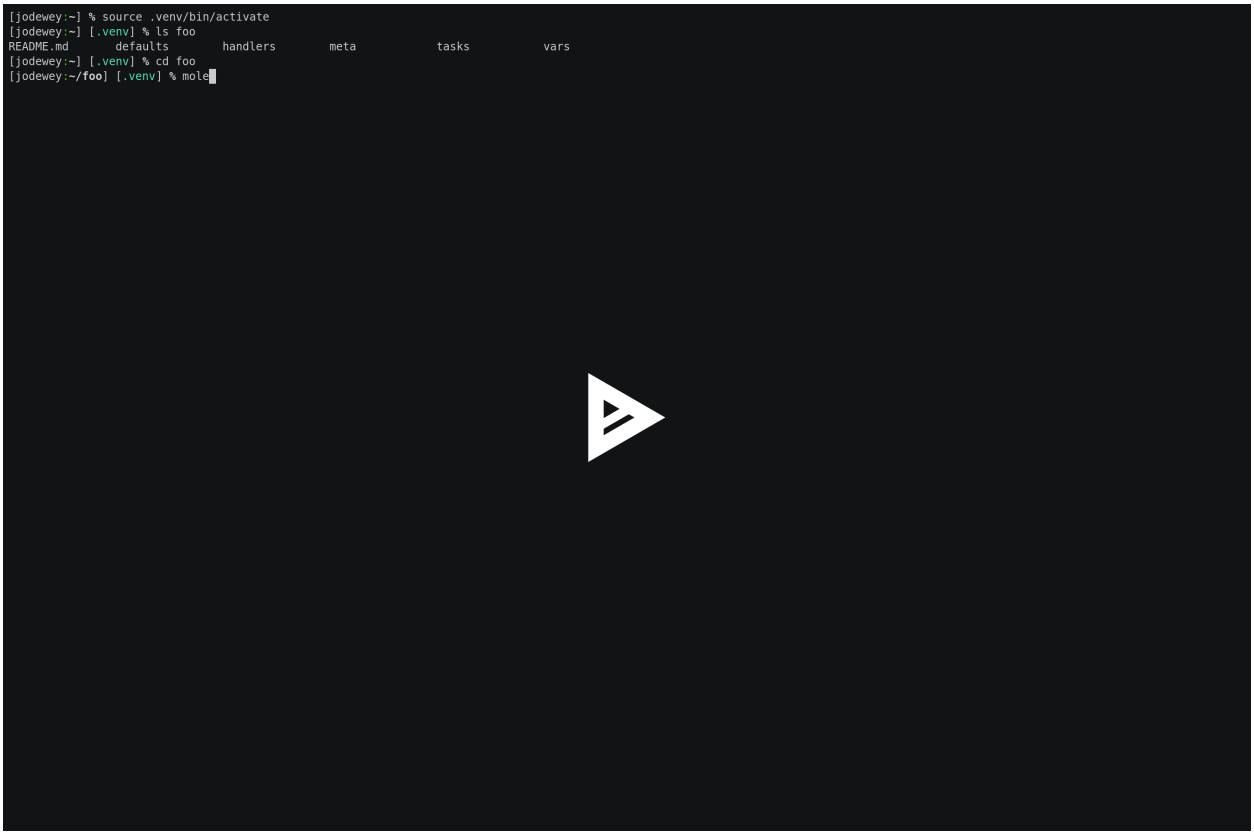
1.2 Creating a new role



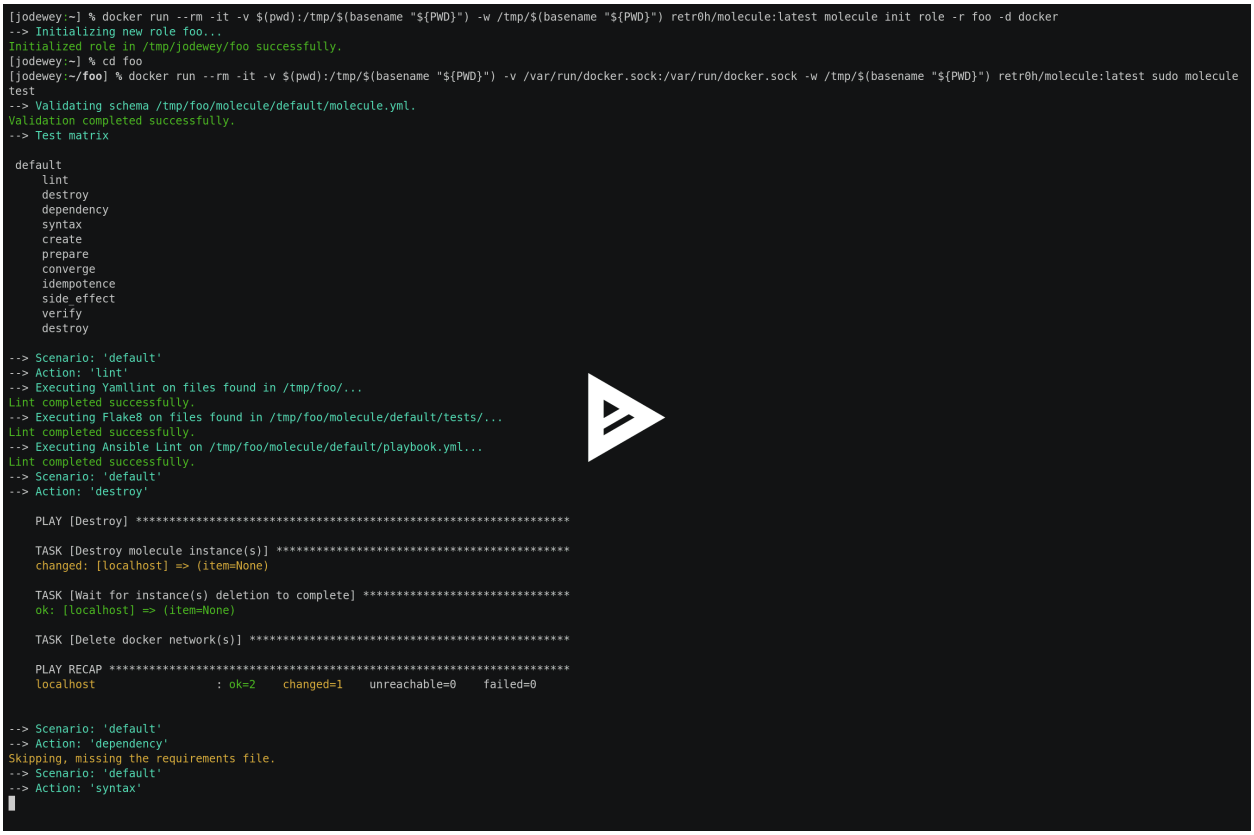
1.3 Testing a new role



1.4 Testing an existing role



1.5 Docker in Docker



CHAPTER 2

Documentation

https://molecule.readthedocs.io/

CHAPTER 3

Contact

3.1 IRC

Join us in the #molecule-users channel on freenode.

3.2 Forums

- molecule-users
- molecule-dev

CHAPTER 4

Ansible Support

Molecule requires Ansible version 2.2 or later.

CHAPTER 5

License

MIT

CHAPTER 6

Contents:

6.1 Installation

This document assumes the developer has a basic understanding of python packaging, and how to install and manage python on the system executing Molecule.

6.1.1 Requirements

Depending on the driver chosen, you may need to install additional OS packages. See *INSTALL.rst*, which is created when initializing a new scenario.

- Ansible >= 2.2
- Python 2.7
- Python >= 3.6 with Ansible >= 2.4

CentOS 7

```
$ sudo yum install -y epel-release
$ sudo yum install -y gcc python-pip python-devel openssl-devel libselinux-python
```

Ubuntu 16.x

```
$ sudo apt-get update
$ sudo apt-get install -y python-pip libssl-dev
```

6.1.2 Pip

Pip is the only supported installation method.

Requirements

Depending on the driver chosen, you may need to install additional python packages. See the driver's documentation or *INSTALL.rst*, which is created when initializing a new scenario.

Install

Install Molecule:

```
$ sudo pip install molecule
```

6.1.3 Source

Due to the rapid pace of development on this tool, you might want to install it in "development mode" so that new updates can be obtained by simply doing a `git pull` in the repository's directory.

Requirements

CentOS 7

```
$ sudo yum install -y libffi-devel git
```

Ubuntu 16.x

```
$ sudo apt-get install -y libffi-dev git
```

Install

```
$ cd /path/to/molecule/checkout
$ pip install -U -e .
```

6.2 Usage

6.2.1 Check

class molecule.command.check.**Check**

molecule check
Target the default scenario.

molecule check --scenario-name foo
Targeting a specific scenario.

molecule --debug check
Executing with *debug*.

molecule `--base-config base.yml check`
Executing with a *base-config*.

molecule `--env-file foo.yml check`
Load an env file to read variables from when rendering molecule.yml.

6.2.2 Converge

Converge will execute the sequence necessary to converge the instances.

class molecule.command.converge.**Converge**

molecule `converge`
Target the default scenario.

molecule `converge --scenario-name foo`
Targeting a specific scenario.

molecule `converge -- -vvv --tags foo,bar`
Providing additional command line arguments to the *ansible-playbook* command. Use this option with care, as there is no sanitation or validation of input. Options passed on the CLI override options provided in provisioner's *options* section of *molecule.yml*.

molecule `--debug converge`
Executing with *debug*.

molecule `--base-config base.yml converge`
Executing with a *base-config*.

molecule `--env-file foo.yml converge`
Load an env file to read variables from when rendering molecule.yml.

6.2.3 Create

class molecule.command.create.**Create**

molecule `create`
Target the default scenario.

molecule `create --scenario-name foo`
Targeting a specific scenario.

molecule `create --driver-name foo`
Targeting a specific driver.

molecule `--debug create`
Executing with *debug*.

molecule `--base-config base.yml create`
Executing with a *base-config*.

molecule `--env-file foo.yml create`
Load an env file to read variables from when rendering molecule.yml.

6.2.4 Dependency

class molecule.command.dependency.**Dependency**

molecule dependency
Target the default scenario.

molecule dependency --scenario-name foo
Targeting a specific scenario.

molecule --debug dependency
Executing with *debug*.

molecule --base-config base.yml dependency
Executing with a *base-config*.

molecule --env-file foo.yml dependency
Load an env file to read variables from when rendering molecule.yml.

6.2.5 Destroy

class molecule.command.destroy.**Destroy**

molecule destroy
Target the default scenario.

molecule destroy --scenario-name foo
Targeting a specific scenario.

molecule destroy --all
Target all scenarios.

molecule destroy --driver-name foo
Targeting a specific driver.

molecule --debug destroy
Executing with *debug*.

molecule --base-config base.yml destroy
Executing with a *base-config*.

molecule --env-file foo.yml destroy
Load an env file to read variables from when rendering molecule.yml.

6.2.6 Idempotence

class molecule.command.idempotence.**Idempotence**
Runs the converge step a second time. If no tasks will be marked as changed the scenario will be considered idempotent.

molecule idempotence
Target the default scenario.

molecule idempotence --scenario-name foo
Targeting a specific scenario.

molecule --debug idempotence
Executing with *debug*.

molecule --base-config base.yml idempotence
Executing with a *base-config*.

molecule --env-file foo.yml idempotence
Load an env file to read variables from when rendering molecule.yml.

6.2.7 Init

class molecule.command.init.role.**Role**

molecule init role --role-name foo
Initialize a new role.

class molecule.command.init.scenario.**Scenario**

molecule init scenario --scenario-name bar --role-name foo
Initialize a new scenario.

cd foo; molecule init scenario --scenario-name bar --role-name foo
Initialize an existing role with Molecule:

class molecule.command.init.template.**Template**

molecule init init template --url https://example.com/user/cookiecutter-repo
Initialize a new role from a Cookiecutter URL.

6.2.8 Lint

class molecule.command.lint.**Lint**

molecule lint
Target the default scenario.

molecule lint --scenario-name foo
Targeting a specific scenario.

molecule --debug lint
Executing with *debug*.

molecule --base-config base.yml lint
Executing with a *base-config*.

molecule --env-file foo.yml lint
Load an env file to read variables from when rendering molecule.yml.

6.2.9 List

class molecule.command.list.**List**

molecule list
Target the default scenario.

molecule `list --scenario-name foo`
Targeting a specific scenario.

molecule `list  --format plain`
Machine readable plain text output.

molecule `list  --format yaml`
Machine readable yaml output.

molecule `--debug list`
Executing with *debug*.

molecule `--base-config base.yml list`
Executing with a *base-config*.

molecule `--env-file foo.yml list`
Load an env file to read variables from when rendering molecule.yml.

6.2.10 Login

class molecule.command.login.**Login**

molecule `login`
Target the default scenario.

molecule `login --scenario-name foo`
Targeting a specific scenario.

molecule `login --host hostname`
Targeting a specific running host.

molecule `login --host hostname --scenario-name foo`
Targeting a specific running host and scenario.

molecule `--debug login`
Executing with *debug*.

molecule `--base-config base.yml login`
Executing with a *base-config*.

molecule `--env-file foo.yml login`
Load an env file to read variables from when rendering molecule.yml.

6.2.11 Matrix

Matrix will display the subcommand's ordered list of actions, which can be changed in scenario configuration.

class molecule.command.matrix.**Matrix**

molecule `matrix subcommand`
Target the default scenario.

molecule `matrix --scenario-name foo subcommand`
Targeting a specific scenario.

molecule `--debug matrix subcommand`
Executing with *debug*.

molecule `--base-config base.yml matrix subcommand`
Executing with a *base-config*.

molecule `--env-file foo.yml matrix subcommand`
Load an env file to read variables from when rendering molecule.yml.

6.2.12 Prepare

class molecule.command.prepare.**Prepare**

molecule `prepare`
Target the default scenario.

molecule `prepare --scenario-name foo`
Targeting a specific scenario.

molecule `prepare --driver-name foo`
Targeting a specific driver.

molecule `prepare --force`
Force the execution fo the prepare playbook.

molecule `--debug prepare`
Executing with *debug*.

molecule `--base-config base.yml prepare`
Executing with a *base-config*.

molecule `--env-file foo.yml prepare`
Load an env file to read variables from when rendering molecule.yml.

6.2.13 Side Effect

class molecule.command.side_effect.**SideEffect**
This action has side effects and not enabled by default. See the provisioners documentation for further details.

molecule `side-effect`
Target the default scenario.

molecule `side-effect --scenario-name foo`
Targeting a specific scenario.

molecule `--debug side-effect`
Executing with *debug*.

molecule `--base-config base.yml side-effect`
Executing with a *base-config*.

molecule `--env-file foo.yml side-effect`
Load an env file to read variables from when rendering molecule.yml.

6.2.14 Syntax

class molecule.command.syntax.**Syntax**

molecule syntax
Target the default scenario.

molecule syntax --scenario-name foo
Targeting a specific scenario.

molecule --debug syntax
Executing with *debug*.

molecule --base-config base.yml syntax
Executing with a *base-config*.

molecule --env-file foo.yml syntax
Load an env file to read variables from when rendering molecule.yml.

6.2.15 Test

Test will execute the sequence necessary to test the instances.

class molecule.command.test.**Test**

molecule test
Target the default scenario.

molecule test --scenario-name foo
Targeting a specific scenario.

molecule test --all
Target all scenarios.

molecule test --destroy=always
Always destroy instances at the conclusion of a Molecule run.

molecule --debug test
Executing with *debug*.

molecule --base-config base.yml test
Executing with a *base-config*.

molecule --env-file foo.yml test
Load an env file to read variables from when rendering molecule.yml.

6.2.16 Verify

class molecule.command.verify.**Verify**

molecule verify
Target the default scenario.

molecule verify --scenario-name foo
Targeting a specific scenario.

molecule --debug verify
Executing with *debug*.

molecule --base-config base.yml verify
Executing with a *base-config*.

molecule --env-file foo.yml verify
Load an env file to read variables from when rendering molecule.yml.

6.3 Examples

A good source of examples are the scenario functional tests.

6.3.1 Docker

Molecule can be executed via an Alpine Linux container by leveraging dind (Docker in Docker). Currently, we only build images for the latest version of Ansible and Molecule. In the future we may break this out into Molecule/ Ansible versioned pairs. The images are located on Docker Hub.

To test a role, change directory into the role to test, and execute Molecule as follows.

```
docker run --rm -it \
    -v '$(pwd)':/tmp/$(basename "${PWD}"):ro \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -w /tmp/$(basename "${PWD}") \
    retr0h/molecule:latest \
    sudo molecule test
```

6.3.2 Monolith Repo

Molecule is generally used to test roles in isolation. However, it can also test roles from a monolith repo.

```
$ tree monolith-repo -L 3 --prune
monolith-repo
├── library
│   └── foo.py
├── plugins
│   └── filters
│       └── foo.py
└── roles
    ├── bar
    │   └── README.md
    ├── baz
    │   └── README.md
    └── foo
        └── README.md
```

The role initialized with Molecule (baz in this case) would simply reference the dependant roles via it's *playbook.yml* or meta dependencies.

Molecule can test complex scenarios leveraging this technique.

```
$ cd monolith-repo/roles/baz
$ molecule test
```

Molecule is simply setting the *ANSIBLE_** environment variables. To view the environment variables set during a Molecule operation pass the `--debug` flag.

```
$ molecule --debug test

DEBUG: ANSIBLE ENVIRONMENT
---
ANSIBLE_CONFIG: /private/tmp/monolith-repo/roles/baz/molecule/default/.molecule/
↪ansible.cfg
ANSIBLE_FILTER_PLUGINS: /Users/jodewey/.pyenv/versions/2.7.13/lib/python2.7/site-
↪packages/molecule/provisioner/ansible/plugins/filters:/private/tmp/monolith-repo/
↪roles/baz/plugins/filters:/private/tmp/monolith-repo/roles/baz/molecule/default/.
↪molecule/plugins/filters
ANSIBLE_LIBRARY: /Users/jodewey/.pyenv/versions/2.7.13/lib/python2.7/site-packages/
↪molecule/provisioner/ansible/plugins/libraries:/private/tmp/monolith-repo/roles/baz/
↪library:/private/tmp/monolith-repo/roles/baz/molecule/default/.molecule/library
ANSIBLE_ROLES_PATH: /private/tmp/monolith-repo/roles:/private/tmp/monolith-repo/roles/
↪baz/molecule/default/.molecule/roles
```

Molecule can be customized any number of ways. Updating the provisioner's env section in *molecule.yml* to suit the needs of the developer and layout of the project.

```
provisioner:
  name: ansible
  env:
    ANSIBLE_$VAR: $VALUE
```

6.3.3 Systemd Container

The docker daemon was designed to provide a simple means of starting, stopping and managing containers. It was not originally designed to bring up an entire Linux system or manage services for such things as start-up order, dependency checking, and failed service recovery.[1]

To start a service which requires systemd, configure *molecule.yml* with a systemd compliant image, capabilities, volumes, and command as follows.

```
platforms:
  - name: instance
    image: solita/ubuntu-systemd:latest
    command: /sbin/init
    capabilities:
      - SYS_ADMIN
    volumes:
      - /sys/fs/cgroup:/sys/fs/cgroup:ro
```

The developer can also opt to start the container with extended privileges.

Important: Use caution when using *privileged* mode.[2][3]

```
platforms:
  - name: instance
    image: solita/ubuntu-systemd:latest
```

(continues on next page)

[1] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/managing_containers/using_systemd_with_containers

[2] https://blog.docker.com/2013/09/docker-can-now-run-within-docker/

[3] https://groups.google.com/forum/#!topic/docker-user/RWLHyzg6Z78

(continued from previous page)

```
    privileged: True
    command: /sbin/init
```

6.3.4 Vagrant Proxy Settings

One way of passing in proxy settings to the Vagrant provider is using the vagrant-proxyconf plugin and adding the vagrant-proxyconf configurations to ~/.vagrant.d/Vagrantfile.

To install the plugin run:

```
$ vagrant plugin install vagrant-proxyconf
```

On linux add the following Vagrantfile to ~/.vagrant.d/Vagrantfile.

```
Vagrant.configure("2") do |config|
  if Vagrant.has_plugin?("vagrant-proxyconf")
    config.proxy.http     = ENV['HTTP_PROXY']
    config.proxy.https    = ENV['HTTP_PROXY']
    config.proxy.no_proxy = ENV['NO_PROXY']
  end
end
```

6.3.5 Sharing Across Scenarios

Playbooks and tests can be shared across scenarios.

```
$ tree shared-tests
shared-tests
├── molecule
│   ├── centos
│   │   └── molecule.yml
│   ├── resources
│   │   ├── playbooks
│   │   │   ├── Dockerfile.j2
│   │   │   ├── create.yml
│   │   │   ├── destroy.yml
│   │   │   ├── playbook.yml
│   │   │   └── prepare.yml
│   │   └── tests
│   │       └── test_default.py
│   ├── ubuntu
│   │   └── molecule.yml
│   └── ubuntu-upstart
│       └── molecule.yml
```

Tests can be shared across scenarios. In this example the *tests* directory lives in a shared location and *molecule.yml* is points to the shared tests.

```
verifier:
name: testinfra
directory: ../resources/tests/
lint:
  name: flake8
```

6.4 Configuration

class molecule.config.**Config**

Molecule searches the current directory for *molecule.yml* files by globbing *molecule/*/molecule.yml*. The files are instantiated into a list of Molecule `Config` objects, and each Molecule subcommand operates on this list.

The directory in which the *molecule.yml* resides is the Scenario's directory. Molecule performs most functions within this directory.

The `Config` object has instantiated *Dependency*, *Driver*, *Lint*, *Platforms*, *Provisioner*, *Verifier*, *Scenario*, and **State_** references.

6.4.1 Variable Substitution

class molecule.interpolation.**Interpolator**

Configuration options may contain environment variables. For example, suppose the shell contains *VERIFIER_NAME=testinfra* and the following molecule.yml is supplied.

```
verifier:
  - name: ${VERIFIER_NAME}
```

Molecule will substitute *$VERIFIER_NAME* with the value of the *VERIFIER_NAME* environment variable.

> **Warning:** If an environment variable is not set, Molecule substitutes with an empty string.

Both *$VARIABLE* and *${VARIABLE}* syntax are supported. Extended shell-style features, such as *${VARIABLE-default}* and *${VARIABLE:-default}* are also supported.

If a literal dollar sign is needed in a configuration, use a double dollar sign (*$$*).

Molecule will substitute special *MOLECULE_* environment variables defined in *molecule.yml*.

Important: Remember, the *MOLECULE_* namespace is reserved for Molecule. Do not prefix your own variables with *MOLECULE_*.

A file may be placed in the root of the project as *env.yml*, and Molecule will read variables when rendering *molecule.yml*. See command usage.

6.4.2 Dependency

Testing roles may rely upon additional dependencies. Molecule handles managing these dependencies by invoking configurable dependency managers.

Ansible Galaxy

class molecule.dependency.ansible_galaxy.**AnsibleGalaxy**

Ansible Galaxy is the default dependency manager.

Additional options can be passed to *ansible-galaxy install* through the options dict. Any option set in this section will override the defaults.

Note: Molecule will remove any options matching '^[v]+$', and pass *-vvv* to the underlying *ansible-galaxy* command when executing *molecule –debug*.

```
dependency:
  name: galaxy
  options:
    ignore-certs: True
    ignore-errors: True
    role-file: requirements.yml
```

The dependency manager can be disabled by setting *enabled* to False.

```
dependency:
  name: galaxy
  enabled: False
```

Environment variables can be passed to the dependency.

```
dependency:
  name: galaxy
  env:
    FOO: bar
```

Gilt

class molecule.dependency.gilt.**Gilt**

Gilt is an alternate dependency manager.

Additional options can be passed to *gilt overlay* through the options dict. Any option set in this section will override the defaults.

```
dependency:
  name: gilt
  options:
    debug: True
```

The dependency manager can be disabled by setting *enabled* to False.

```
dependency:
  name: gilt
  enabled: False
```

Environment variables can be passed to the dependency.

```
dependency:
  name: gilt
  env:
    FOO: bar
```

Shell

class molecule.dependency.shell.**Shell**

Shell is an alternate dependency manager. It is intended to run a command in situations where Ansible Galaxy and Gilt don't suffice.

The *command* to execute is required, and is relative to Molecule's project directory when referencing a script not in $PATH.

Note: Unlike the other dependency managers, *options* are ignored and not passed to *shell*. Additional flags/subcommands should simply be added to the *command*.

```
dependency:
  name: shell
  command: path/to/command --flag1 subcommand --flag2
```

The dependency manager can be disabled by setting *enabled* to False.

```
dependency:
  name: shell
  command: path/to/command --flag1 subcommand --flag2
  enabled: False
```

Environment variables can be passed to the dependency.

```
dependency:
  name: shell
  command: path/to/command --flag1 subcommand --flag2
  env:
    FOO: bar
```

6.4.3 Driver

Molecule uses Ansible to manage instances to operate on. Molecule supports any provider Ansible supports. This work is offloaded to the *provisioner*.

The driver's name is specified in *molecule.yml*, and can be overriden on the command line. Molecule will remember the last successful driver used, and continue to use the driver for all subsequent subcommands, or until the instances are destroyed by Molecule.

Important: The verifier must support the Ansible provider for proper Molecule integration.

The driver's python package requires installation.

Azure

class molecule.driver.azure.**Azure**

The class responsible for managing Azure instances. Azure is *not* the default driver used in Molecule.

Molecule leverages Ansible's azure_module, by mapping variables from *molecule.yml* into *create.yml* and *destroy.yml*.

```
driver:
  name: azure
platforms:
  - name: instance
```

```
$ sudo pip install 'ansible[azure]'
```

Change the options passed to the ssh client.

```
driver:
  name: azure
  ssh_connection_options:
    -o ControlPath=~/.ansible/cp/%r@%h-%p
```

Important: Molecule does not merge lists, when overriding the developer must provide all options.

Provide the files Molecule will preserve upon each subcommand execution.

```
driver:
  name: azure
  safe_files:
    - foo
```

Delegated

class molecule.driver.delegated.**Delegated**

The class responsible for managing delegated instances. Delegated is *not* the default driver used in Molecule.

Under this driver, it is the developers responsibility to implement the create and destroy playbooks. *Managed* is the default behaviour of all drivers.

```
driver:
  name: delegated
```

However, the developer must adhere to the instance-config API. The developer's create playbook must provide the following instance-config data, and the developer's destroy playbook must reset the instance-config.

```
- address: ssh_endpoint
  identity_file: ssh_identity_file
  instance: instance_name
  port: ssh_port_as_string
  user: ssh_user
```

Molecule can also skip the provisioning/deprovisioning steps. It is the developers responsibility to manage the instances, and properly configure Molecule to connect to said instances.

```
driver:
  name: delegated
  options:
    managed: False
    login_cmd_template: 'docker exec -ti {instance} bash'
    ansible_connection_options:
      connection: docker
platforms:
  - name: instance-docker
```

```
$ docker run \
    -d \
```

(continues on next page)

(continued from previous page)

```
    --name instance-docker \
    --hostname instance-docker \
    -it molecule_local/ubuntu:latest sleep infinity & wait
```

Use Molecule with delegated instances, which are accessible over ssh.

Important: It is the developers responsibility to configure the ssh config file.

```
driver:
  name: delegated
  options:
    managed: False
    login_cmd_template: 'ssh {instance} -F /tmp/ssh-config'
    ansible_connection_options:
      connection: ssh
      ansible_ssh_common_args -F /path/to/ssh-config
platforms:
  - name: instance-vagrant
```

Provide the files Molecule will preserve post *destroy* action.

```
driver:
  name: delegated
  safe_files:
    - foo
```

Docker

class molecule.driver.docker.**Docker**

The class responsible for managing Docker containers. Docker is the default driver used in Molecule.

Molecule leverages Ansible's docker_container module, by mapping variables from *molecule.yml* into *create.yml* and *destroy.yml*.

```
driver:
  name: docker
platforms:
  - name: instance
    hostname: instance
    image: image_name:tag
    pull: True|False
    registry:
      url: registry.example.com
      credentials:
        username: $USERNAME
        password: $PASSWORD
        email: user@example.com
    command: sleep infinity
    privileged: True|False
    security_opts:
      - seccomp=unconfined
    volumes:
      - /sys/fs/cgroup:/sys/fs/cgroup:ro
```

(continues on next page)

(continued from previous page)

```
    tmpfs:
      - /tmp
      - /run
    capabilities:
      - SYS_ADMIN
    exposed_ports:
      - 53/udp
      - 53/tcp
    published_ports:
      - 0.0.0.0:8053:53/udp
      - 0.0.0.0:8053:53/tcp
    ulimits:
      - nofile:262144:262144
    dns_servers:
      - 8.8.8.8
    networks:
      - name: foo
      - name: bar
    docker_host: tcp://localhost:12376
    env:
      FOO: bar
```

Ansible < 2.6

```
$ sudo pip install docker-py
```

Ansible >= 2.6

```
$ sudo pip install docker
```

Provide the files Molecule will preserve upon each subcommand execution.

```
driver:
  name: docker
  safe_files:
    - foo
```

EC2

class molecule.driver.ec2.**EC2**

The class responsible for managing EC2 instances. EC2 is *not* the default driver used in Molecule.

Molecule leverages Ansible's ec2_module, by mapping variables from *molecule.yml* into *create.yml* and *destroy.yml*.

```
driver:
  name: ec2
platforms:
  - name: instance
```

```
$ sudo pip install boto
$ sudo pip install boto3
```

Change the options passed to the ssh client.

```
driver:
  name: ec2
  ssh_connection_options:
    -o ControlPath=~/.ansible/cp/%r@%h-%p
```

Important: Molecule does not merge lists, when overriding the developer must provide all options.

Provide the files Molecule will preserve upon each subcommand execution.

```
driver:
  name: ec2
  safe_files:
    - foo
```

GCE

class molecule.driver.gce.**GCE**

The class responsible for managing GCE instances. GCE is *not* the default driver used in Molecule.

GCE is somewhat different than other cloud providers. There is not an Ansible module for managing ssh keys. This driver assumes the developer has deployed project wide ssh key.

Molecule leverages Ansible's gce_module, by mapping variables from *molecule.yml* into *create.yml* and *destroy.yml*.

```
driver:
  name: gce
platforms:
  - name: instance
```

```
$ sudo pip install apache-libcloud
```

Change the options passed to the ssh client.

```
driver:
  name: gce
  ssh_connection_options:
    -o ControlPath=~/.ansible/cp/%r@%h-%p
```

Important: Molecule does not merge lists, when overriding the developer must provide all options.

Provide the files Molecule will preserve upon each subcommand execution.

```
driver:
  name: gce
  safe_files:
    - foo
```

LXC

class molecule.driver.lxc.**LXC**

The class responsible for managing LXC containers. LXC is *not* the default driver used in Molecule.

Molecule leverages Ansible's lxc_container module, by mapping variables from *molecule.yml* into *create.yml* and *destroy.yml*.

```
driver:
  name: lxc
```

```
$ sudo pip install lxc-python2
```

Provide the files Molecule will preserve upon each subcommand execution.

```
driver:
  name: lxc
  safe_files:
    - foo
```

LXD

class molecule.driver.lxd.**LXD**

The class responsible for managing LXD containers. LXD is *not* the default driver used in Molecule.

Molecule leverages Ansible's lxd_container module, by mapping variables from *molecule.yml* into *create.yml* and *destroy.yml*.

```
driver:
  name: lxd
```

Provide the files Molecule will preserve upon each subcommand execution.

```
driver:
  name: lxd
  safe_files:
    - foo
```

Openstack

class molecule.driver.openstack.**Openstack**

The class responsible for managing OpenStack instances. OpenStack is *not* the default driver used in Molecule.

Molecule leverages Ansible's openstack_module, by mapping variables from *molecule.yml* into *create.yml* and *destroy.yml*.

```
driver:
  name: openstack
platforms:
  - name: instance
```

```
$ sudo pip install shade
```

Change the options passed to the ssh client.

```
driver:
  name: openstack
  ssh_connection_options:
    -o ControlPath=~/.ansible/cp/%r@%h-%p
```

Important: Molecule does not merge lists, when overriding the developer must provide all options.

Provide the files Molecule will preserve upon each subcommand execution.

```
driver:
  name: openstack
  safe_files:
    - foo
```

Vagrant

class molecule.driver.vagrant.**Vagrant**

The class responsible for managing Vagrant instances. Vagrant is *not* the default driver used in Molecule.

Molecule leverages Molecule's own *Molecule Vagrant Module*, by mapping variables from *molecule.yml* into *create.yml* and *destroy.yml*.

Important: This driver is alpha quality software. Do not perform any additonal tasks inside the *create* playbook. Molecule does not know about the Vagrant instances' configuration until the *converge* playbook is executed.

The *create* playbook boots instances, then the instance data is written to disk. The instance data is then added to Molecule's Ansible inventory on the next execution of *molecule.command.create*, which happens to be the *converge* playbook.

This is an area needing improvement. Gluing togher Ansible playbook return data and molecule is clunky. Moving the playbook execution from *sh* to python is less than ideal, since the playbook's return data needs handled by an internal callback plugin.

Operating this far inside Ansible's internals doesn't feel right. Nor does orchestrating Ansible's CLI with Molecule. Ansible is throwing pieces over the wall, which Molecule is picking up and reconstructing.

```
driver:
  name: vagrant
platforms
  - name: instance-1
    instance_raw_config_args:
      - "vm.network 'forwarded_port', guest: 80, host: 8080"
    interfaces:
      - auto_config: true
        network_name: private_network
        type: dhcp
    config_options:
      synced_folder: True
    box: debian/jessie64
    box_version: 10.1
    box_url: http://repo.example.com/images/postmerge/debian.json
    memory: 1024
    cpus: 1
    provider_options:
      gui: True
    provider_raw_config_args:
      - "customize ['modifyvm', :id, '--cpuexecutioncap', '50']"
```

(continues on next page)

(continued from previous page)

```
    provider_override_args:
      - "vm.synced_folder './', '/vagrant', disabled: true, type: 'nfs'"
    provision: True
```

```
$ sudo pip install python-vagrant
```

Change the provider passed to Vagrant.

```
driver:
  name: vagrant
  provider:
    name: parallels
```

Change the options passed to the ssh client.

```
driver:
  name: vagrant
  ssh_connection_options:
    -o ControlPath=~/.ansible/cp/%r@%h-%p
```

Important: Molecule does not merge lists, when overriding the developer must provide all options.

Provide the files Molecule will preserve upon each subcommand execution.

```
driver:
  name: vagrant
  safe_files:
    - foo
```

Molecule Vagrant Module

Molecule manages Vagrant via an internal Ansible module. The following belongs in the appropriate create or destroy playbooks, and uses the default provider.

Supported Providers:

- VirtualBox (default)
- VMware (vmware_fusion, vmware_workstation and vmware_desktop)
- Parallels
- Libvirt (requires vagrant-libvirt plugin)

Create instances.

```
- hosts: localhost
  connection: local
  tasks:
    - name: Create instances
      molecule_vagrant:
        instance_name: "{{ item }}"
        platform_box: ubuntu/trusty64
        state: up
      with_items:
```

(continues on next page)

(continued from previous page)

```
        - instance-1
        - instance-2
```

Destroy instances.

```
- hosts: localhost
  connection: local
  tasks:
    - name: Destroy instances
      molecule_vagrant:
        instance_name: "{{ item }}"
        platform_box: ubuntu/trusty64
        state: destroy
      with_items:
        - instance-1
        - instance-2
```

Halt instances (shutdown without destroy).

```
- hosts: localhost
  connection: local
  tasks:
    - name: Halt instances
      molecule_vagrant:
        instance_name: "{{ item }}"
        platform_box: ubuntu/trusty64
        state: halt
      with_items:
        - instance-1
        - instance-2
```

Create instances with interfaces.

```
- hosts: localhost
  connection: local
  tasks:
    - name: Create instance with interfaces
      molecule_vagrant:
        instance_name: instance-1
        instance_interfaces:
          - auto_config: true
            network_name: private_network
            type: dhcp
          - auto_config: false
            network_name: private_network
            type: dhcp
          - auto_config: true
            ip: 192.168.11.3
            network_name: private_network
            type: static
        platform_box: ubuntu/trusty64
        state: destroy
```

Create instances with additional provider options.

```
- hosts: localhost
  connection: local
```

(continues on next page)

(continued from previous page)

```
  tasks:
    - name: Create instances
      molecule_vagrant:
        instance_name: "{{ item }}"
        platform_box: ubuntu/trusty64
        provider_name: virtualbox
        provider_memory: 1024
        provider_cpus: 4
        provider_raw_config_args:
          - "customize ['modifyvm', :id, '--cpuexecutioncap', '50']"
        provider_options:
          gui: true
        provision: true
        state: up
      with_items:
        - instance-1
        - instance-2
```

Create instances with additional instance options.

```
- hosts: localhost
  connection: local
  tasks:
    - name: Create instances
      molecule_vagrant:
        instance_name: "{{ item }}"
        platform_box: ubuntu/trusty64
        instance_raw_config_args:
          - "vm.network 'forwarded_port', guest: 80, host: 8080"
        state: up
      with_items:
        - instance-1
        - instance-2
```

Create instances on a remote Libvirt node using default create/destroy templates.

```
- hosts: localhost
  connection: local
  tasks:
    - name: Create instances
      molecule_vagrant:
        instance_name: "{{ item }}"
        platform_box: ubuntu/trusty64
        provider_raw_config_args:
          - 'host = "remote-node.example.com"'
          - 'connect_via_ssh = "True"'
          - 'username = "sshuser"'
          - 'driver = "kvm"'
          - 'cpu_mode = "host-passthrough"'
        state: up
      with_items:
        - instance-1
        - instance-2
```

6.4.4 Lint

Molecule handles project linting by invoking configurable linters.

Yaml Lint

class molecule.lint.yamllint.**Yamllint**

Yamllint is the default projet linter.

Yamllint is a linter for YAML files. In addition to checking for syntax validity, also checks for key repetition, and cosmetic problems such as lines length, trailing spaces, and indentation.

Additional options can be passed to *yamllint* through the options dict. Any option set in this section will override the defaults.

```
lint:
  name: yamllint
  options:
    config-file: foo/bar
```

The project linting can be disabled by setting *enabled* to False.

```
lint:
  name: yamllint
  enabled: False
```

Environment variables can be passed to lint.

```
lint:
  name: yamllint
  env:
    FOO: bar
```

6.4.5 Platforms

class molecule.platforms.**Platforms**

Platforms define the instances to be tested, and the groups to which the instances belong.

```
platforms:
  - name: instance-1
```

Multiple instances can be provided.

```
platforms:
  - name: instance-1
  - name: instance-2
```

Mapping instances to groups. These groups will be used by the *Provisioner* for orchestration purposes.

```
platforms:
  - name: instance-1
    groups:
      - group1
      - group2
```

Children allow the creation of groups of groups.

```
platforms:
  - name: instance-1
    groups:
      - group1
      - group2
    children:
      - child_group1
```

6.4.6 Provisioner

Molecule handles provisioning and converging the role.

Ansible

class molecule.provisioner.ansible.**Ansible**

Ansible is the default provisioner. No other provisioner will be supported.

Molecule's provisioner manages the instances lifecycle. However, the user must provide the create, destroy, and converge playbooks. Molecule's *init* subcommand will provide the necessary files for convenience.

Molecule will skip tasks which are taged with either *molecule-notest* or *notest*.

Important: Reserve the create and destroy playbooks for provisioning. Do not attempt to gather facts or perform operations on the provisioned nodes inside these playbooks. Due to the gymnastics necessary to sync state between Ansible and Molecule, it is best to perform these tasks in the converge playbook.

It is the developers responsiblity to properly map the modules's fact data into the instance_conf_dict fact in the create playbook. This allows Molecule to properly configure Ansible inventory.

Additional options can be passed to *ansible-playbook* through the options dict. Any option set in this section will override the defaults.

Important: Options do not affect the create and destroy actions.

Note: Molecule will remove any options matching '^[v]+$', and pass *-vvv* to the underlying *ansible-playbook* command when executing *molecule –debug*.

Molecule will silence log output, unless invoked with the *–debug* flag. However, this results in quite a bit of output. To enable Ansible log output, add the following to the *provisioner* section of *molecule.yml*.

```
provisioner:
  name: ansible
  log: True
```

The create/destroy playbooks for Docker and Vagrant are bundled with Molecule. These playbooks have a clean API from *molecule.yml*, and are the most commonly used. The bundled playbooks can still be overriden.

The playbook loading order is:

1. provisioner.playbooks.$driver_name.$action
2. provisioner.playbooks.$action

3. bundled_playbook.$driver_name.$action

```
provisioner:
  name: ansible
  options:
    vvv: True
  playbooks:
    create: create.yml
    converge: playbook.yml
    destroy: destroy.yml
```

Share playbooks between roles.

```
provisioner:
  name: ansible
  playbooks:
    create: ../default/create.yml
    destroy: ../default/destroy.yml
    converge: playbook.yml
```

Multiple driver playbooks. In some situations a developer may choose to test the same role against different backends. Molecule will choose driver specific create/destroy playbooks, if the determined driver has a key in the playbooks section of the provisioner's dict.

Important: If the determined driver has a key in the playbooks dict, Molecule will use this dict to resolve all provisioning playbooks (create/destroy).

```
provisioner:
  name: ansible
  playbooks:
    docker:
      create: create.yml
      destroy: destroy.yml
    create: create.yml
    destroy: destroy.yml
    converge: playbook.yml
```

Important: Paths in this section are converted to absolute paths, where the relative parent is the $scenario_directory.

The side effect playbook executes actions which produce side effects to the instances(s). Intended to test HA failover scenarios or the like. It is not enabled by default. Add the following to the provisioner's *playbooks* section to enable.

```
provisioner:
  name: ansible
  playbooks:
    side_effect: side_effect.yml
```

Important: This feature should be considered experimental.

The prepare playbook executes actions which bring the system to a given state prior to converge. It is executed after create, and only once for the duration of the instances life.

This can be used to bring instances into a particular state, prior to testing.

```
provisioner:
  name: ansible
  playbooks:
    prepare: prepare.yml
```

Environment variables. Molecule does its best to handle common Ansible paths. The defaults are as follows.

```
ANSIBLE_ROLES_PATH:
  $ephemeral_directory/roles/:$project_directory/../
ANSIBLE_LIBRARY:
  $ephemeral_directory/library/:$project_directory/library/
ANSIBLE_FILTER_PLUGINS:
  $ephemeral_directory/plugins/filters/:$project_directory/filter/plugins/
```

Environment variables can be passed to the provisioner. Variables in this section which match the names above will be appened to the above defaults, and converted to absolute paths, where the relative parent is the $scenario_directory.

Important: Paths in this section are converted to absolute paths, where the relative parent is the $scenario_directory.

```
provisioner:
  name: ansible
  env:
    FOO: bar
```

Modifying ansible.cfg.

```
provisioner:
  name: ansible
  config_options:
    defaults:
      fact_caching: jsonfile
    ssh_connection:
      scp_if_ssh: True
```

Important: The following keys are disallowed to prevent Molecule from improperly functioning. They can be specified through the provisioner's env setting described above, with the exception of the *privilege_escalation*.

```
provisioner:
  name: ansible
  config_options:
    defaults:
      roles_path: /path/to/roles_path
      library: /path/to/library
      filter_plugins: /path/to/filter_plugins
    privilege_escalation: {}
```

Roles which require host/groups to have certain variables set. Molecule uses the same variables defined in a playbook syntax as Ansible.

```
provisioner:
  name: ansible
  inventory:
    group_vars:
      foo1:
        foo: bar
      foo2:
        foo: bar
        baz:
          qux: zzyzx
    host_vars:
      foo1-01:
        foo: bar
```

An alternative to the above is symlinking. Molecule creates symlinks to the specified directory in the inventory directory. This allows ansible to converge utilzing its built in host/group_vars resolution. These two forms of inventory management are mutually exclusive.

Note: The source directory linking is relative to the scenario's directory.

The only valid keys are *group_vars* and *host_vars*. Molecule's schema validator will enforce this.

```
provisioner:
  name: ansible
  inventory:
    links:
      group_vars: ../../../inventory/group_vars/
      host_vars: ../../../inventory/host_vars/
```

Override connection options:

```
provisioner:
  name: ansible
  connection_options:
    ansible_ssh_user: foo
    ansible_ssh_common_args: -o IdentitiesOnly=no
```

Lint

Molecule handles provisioner linting by invoking configurable linters.

class molecule.provisioner.lint.ansible_lint.**AnsibleLint**

Ansible Lint is the default role linter.

Ansible Lint checks playbooks for practices, and behaviour that could potentially be improved.

Additional options can be passed to *ansible-lint* through the options dict. Any option set in this section will override the defaults.

```
provisioner:
  name: ansible
  lint:
    name: ansible-lint
    options:
      exclude:
```

(continues on next page)

(continued from previous page)

```
        - path/exclude1
        - path/exclude2
      x: ["ANSIBLE0011,ANSIBLE0012"]
      force-color: True
```

The *x* option has to be passed like this due to a bug in Ansible Lint.

The role linting can be disabled by setting *enabled* to False.

```
provisioner:
  name: ansible
  lint:
    name: ansible-lint
    enabled: False
```

Environment variables can be passed to lint.

```
provisioner:
  name: ansible
  lint:
    name: ansible-lint
    env:
      FOO: bar
```

6.4.7 Scenario

Molecule treats scenarios as a first-class citizens, with a top-level configuration syntax.

class molecule.scenario.**Scenario**

A scenario allows Molecule test a role in a particular way, this is a fundamental change from Molecule v1.

A scenario is a self-contained directory containing everything necessary for testing the role in a particular way. The default scenario is named *default*, and every role should contain a default scenario.

Any option set in this section will override the defaults.

```
scenario:
  name: default
  create_sequence:
    - create
    - prepare
  check_sequence:
    - destroy
    - create
    - prepare
    - converge
    - check
    - destroy
  converge_sequence:
    - create
    - prepare
    - converge
  destroy_sequence:
    - destroy
  test_sequence:
    - lint
```

(continues on next page)

(continued from previous page)

```
    - destroy
    - dependency
    - syntax
    - create
    - prepare
    - converge
    - idempotence
    - side_effect
    - verify
    - destroy
```

6.4.8 Verifier

Molecule handles role testing by invoking configurable verifiers.

Goss

class molecule.verifier.goss.**Goss**

Goss is not the default test runner.

Goss is a YAML based serverspec-like tool for validating a server's configuration. Goss is *not* the default verifier used in Molecule.

Molecule executes a playbook (*verify.yml*) located in the role's *scenario.directory*. This playbook will copy YAML files to the instances, and execute Goss using a community written Goss Ansible module bundled with Molecule.

Additional options can be passed to *goss validate* by modifying the verify playbook.

The testing can be disabled by setting *enabled* to False.

```
verifier:
  name: goss
  enabled: False
```

Environment variables can be passed to the verifier.

```
verifier:
  name: goss
  env:
    FOO: bar
```

Change path to the test directory.

```
verifier:
  name: goss
  directory: /foo/bar/
```

Important: Due to the nature of this verifier. Molecule does not perform options handling in the same fashion as Testinfra.

Lint

The Goss verifier does not utilize a linter.

Inspec

class molecule.verifier.inspec.**Inspec**

Inspec is not the default test runner.

InSpec is Chef's open-source language for describing security & compliance rules that can be shared between software engineers, operations, and security engineers. Your compliance, security, and other policy requirements become automated tests throughout all stages of the software delivery process. Inspec is *not* the default verifier used in Molecule.

Molecule executes a playbook (*verify.yml*) located in the role's *scenario.directory*. This playbook will copy test files to the instances, and execute Inspec locally over Ansible. Molecule executes Inspec over an Ansible transport, in an attempt to provide Inspec support across all Molecule drivers.

Additional options can be passed to *inspec exec* by modifying the verify playbook.

The testing can be disabled by setting *enabled* to False.

```
verifier:
  name: inspec
  enabled: False
```

Environment variables can be passed to the verifier.

```
verifier:
  name: inspec
  env:
    FOO: bar
```

Change path to the test directory.

```
verifier:
  name: inspec
  directory: /foo/bar/
```

Important: Due to the nature of this verifier. Molecule does not perform options handling in the same fashion as Testinfra.

Lint

class molecule.verifier.lint.rubocop.**RuboCop**

RuboCop is not the default verifier linter.

RuboCop is a linter for ruby files.

Additional options can be passed to *rubocop* through the options dict. Any option set in this section will override the defaults.

```
verifier:
  name: inspec
  lint:
    name: rubocop
    options:
      auto-correct: True
```

Test file linting can be disabled by setting *enabled* to False.

```
verifier:
  name: inspec
  lint:
    name: rubocop
    enabled: False
```

Environment variables can be passed to lint.

```
verifier:
  name: inspec
  lint:
    name: rubocop
    env:
      FOO: bar
```

Testinfra

class molecule.verifier.testinfra.**Testinfra**

Testinfra is the default test runner.

Additional options can be passed to *testinfra* through the options dict. Any option set in this section will override the defaults.

Note: Molecule will remove any options matching '^[v]+$', and pass *-vvv* to the underlying *py.test* command when executing *molecule –debug*.

```
verifier:
  name: testinfra
  options:
    n: 1
```

The testing can be disabled by setting *enabled* to False.

```
verifier:
  name: testinfra
  enabled: False
```

Environment variables can be passed to the verifier.

```
verifier:
  name: testinfra
  env:
    FOO: bar
```

Change path to the test directory.

```
verifier:
  name: testinfra
  directory: /foo/bar/
```

Additional tests from another file or directory relative to the scenario's tests directory (supports regexp).

```
verifier:
  name: testinfra
  additional_files_or_dirs:
    - ../path/to/test_1.py
    - ../path/to/test_2.py
    - ../path/to/directory/*
```

Lint

class molecule.verifier.lint.flake8.**Flake8**

Flake8 is the default verifier linter.

Flake8 is a linter for python files.

Additional options can be passed to *flake8* through the options dict. Any option set in this section will override the defaults.

```
verifier:
  name: testinfra
  lint:
    name: flake8
    options:
      benchmark: True
```

Test file linting can be disabled by setting *enabled* to False.

```
verifier:
  name: testinfra
  lint:
    name: flake8
    enabled: False
```

Environment variables can be passed to lint.

```
verifier:
  name: testinfra
  lint:
    name: flake8
    env:
      FOO: bar
```

6.5 Porting Guide

Porting your existing role from Molecule to Molecule v2.

6.5.1 Automatically

Molecule ships with a conversion script. This conversion script converts a molecule v1 vagrant driver config into a molecule v2 vagrant config. The script handles the creation of the initial directory structure, migration and cleanup of files.

Note: The script is fairly crude, and only supports Vagrant at the time of this writing. The authors will be adding additional functionality as additional roles are migrated to v2.

```
$ contrib/convert.py /path/to/v1/role/molecule.yml
```

6.5.2 Manually

1. In the basedir of your existing role, create a new *default* scenario. This scenario is equivalent to your existing Molecule v1 setup.

```
$ cd $role-name
$ molecule init scenario -r $role-name -s default -d $driver-name
```

2. Move existing Testinfra tests to the new scenario's test directory located at *molecule/default/tests/test_default.py*.
3. If necessary port existing Serverspec tests to Testinfra. A Testinfra skeleton has been created at *molecule/default/tests/test_default.py*.
4. Port role's existing *molecule.yml* to the new format located in the scenario's directory at *molecule/default/molecule.yml*.
5. Port role's existing *playbook.yml* to the new location in the scenario's directory at *molecule/default/playbook.yml*.
6. Cleanup

```
$ rm -rf .molecule/
$ rm -rf molecule.yml
$ rm -rf playbook.yml
$ rm -rf tests/
```

6. Test

```
$ molecule test
```

6.6 Testing

Molecule has an extensive set of unit and functional tests. Molecule uses Tox Factors to generate a matrix of python x Ansible x unit/functional tests. Manual setup required as of this time.

6.6.1 Dependencies

Tests will be skipped when the driver's binary is not present.

Install the test framework Tox.

```
$ pip install tox
```

Install the remaining requirements in a venv (optional).

```
$ pip install -r test-requirements.txt -r requirements.txt
```

6.6.2 Full

Run all tests, including linting and coverage reports. This should be run prior to merging or submitting a pull request.

```
$ tox
```

6.6.3 Unit

Run all unit tests with coverage.

```
$ tox -t unit
```

6.6.4 Functional

Run all functional tests.

Note: The functional tests are a work in progress. They need better structure and reuse.

```
$ tox -t functional
```

Run all functional tests targeting the docker driver.

```
$ tox -t functional -- -v -k docker
```

Delegated

Run all the functional delegated tests.

```
$ ansible-playbook -i test/resources/playbooks/delegated/inventory \
  test/resources/playbooks/delegated/create.yml
$ tox -t functional -- --delegated -v -k delegated
$ ansible-playbook -i test/resources/playbooks/delegated/inventory \
  test/resources/playbooks/delegated/destroy.yml
```

6.6.5 Formatting

The formatting is done using YAPF.

```
$ tox -e format
```

6.6.6 Linting

Linting is performed by Flake8.

```
$ tox -e $(tox -l | grep lint | paste -d, -s -)
```

6.6.7 LXC

Follow the steps detailed in the Vagrantfile below.

```
$ cd test/functional/lxc
$ vagrant up
```

6.6.8 LXD

Follow the steps detailed in the Vagrantfile below.

```
$ cd test/functional/lxd
$ vagrant up
```

6.6.9 Continuous integration

Travis CI

Travis is a CI platform, which can be used to test Ansible roles.

A `.travis.yml` testing a role named foo1 with the Docker driver.

```
---
sudo: required
language: python
services:
  - docker
before_install:
  - sudo apt-get -qq update
install:
  - pip install molecule
  # - pip install required driver (e.g. docker, python-vagrant, shade, boto, apache-
↪libcloud)
script:
  - molecule test
```

A `.travis.yml` using Tox as described below.

```
---
sudo: required
language: python
services:
  - docker
before_install:
  - sudo apt-get -qq update
install:
  - pip install tox-travis
```

(continues on next page)

(continued from previous page)

```
script:
  - tox
```

Gitlab CI

Gitlab includes its own CI. Pipelines are usually defined in a `.gitlab-ci.yml` file in the top folder of a repository, to be ran on Gitlab Runners.

Here is an example setting up a virtualenv and testing an Ansible role via Molecule. User-level pip is cached and so is the virtual environment to save time. And this is run over a runner tagged *pip36* and *docker*, because its a minimal CentOS 7 VM installed with pip36 from IUS repository and docker.

```
---
stages:
  - test

variables:
  PIP_CACHE_DIR: "$CI_PROJECT_DIR/.pip"

cache:
  paths:
    - .pip/
    - virtenv/

before_script:
  - pip3.6 install virtualenv
  - virtualenv virtenv
  - source virtenv/bin/activate

molecule:
  stage: test
  tags:
    - pip36
    - docker
  script:
    - docker -v
    - python -V
    - pip install ansible molecule docker
    - ansible --version
    - molecule --version
    - molecule test
```

Jenkins Pipeline

Jenkins projects can also be defined in a file, by default named *Jenkinsfile* in the top folder of a repository. Two syntax are available, Declarative and Scripted. Here is an example using the declarative syntax, setting up a virtualenv and testing an Ansible role via Molecule.

```
pipeline {

  agent {
    // Node setup : minimal centos7, plugged into Jenkins, and
    // git config --global http.sslVerify false
```

(continues on next page)

(continued from previous page)

```
    // sudo yum -y install https://centos7.iuscommunity.org/ius-release.rpm
    // sudo yum -y install python36u python36u-pip python36u-devel git curl gcc
    // git config --global http.sslVerify false
    // sudo curl -fsSL get.docker.com | bash
    label 'Molecule_Slave'
  }

  stages {

    stage ('Get latest code') {
      steps {
        checkout scm
      }
    }

    stage ('Setup Python virtual environment') {
      steps {
        sh '''
          export HTTP_PROXY=http://10.123.123.123:8080
          export HTTPS_PROXY=http://10.123.123.123:8080
          pip3.6 install virtualenv
          virtualenv virtenv
          source virtenv/bin/activate
          pip install --upgrade ansible molecule docker
        '''
      }
    }

    stage ('Display versions') {
      steps {
        sh '''
          source virtenv/bin/activate
          docker -v
          python -V
          ansible --version
          molecule --version
        '''
      }
    }

    stage ('Molecule test') {
      steps {
        sh '''
          source virtenv/bin/activate
          molecule test
        '''
      }
    }

  }

}
```

Tox

Tox is a generic virtualenv management, and test command line tool. Tox can be used in conjunction with Factors and

Molecule, to perform scenario tests.

To test the role against multiple versions of Ansible.

```
[tox]
minversion = 1.8
envlist = py{27}-ansible{20,21,22}
skipsdist = true

[testenv]
passenv = *
deps =
    -rrequirements.txt
    ansible20: ansible==2.0.2.0
    ansible21: ansible==2.1.2.0
    ansible22: ansible==2.2.0.0
commands =
    molecule test
```

To view the factor generated tox environments.

```
$ tox -l
py27-ansible20
py27-ansible21
py27-ansible22
```

Detox

Detox is a distributed version of Tox which can be used to make efficient use of multiple CPUs. Detox will run tox environment tests in parallel.

Detox takes the same arguments and configuration as tox, however Molecule must be made aware of the parallel testing by setting a MOLECULE_EPHEMERAL_DIRECTORY environment variable per environment.

```
[tox]
minversion = 1.8
envlist = py{27}_ansible{23,24}
skipsdist = true

[testenv]
deps =
    -rrequirements.txt
    ansible23: ansible==2.3
    ansible24: ansible==2.4
commands =
    molecule test
setenv =
    MOLECULE_EPHEMERAL_DIRECTORY={envname}
```

If you are utilizing the Openstack driver you will have to make sure that your `envname` variable does not contain any invalid characters, particularly `-`.

You also must include the MOLECULE_EPHEMERAL_DIRECTORY variable in the `molecule.yml` configuration file.

```
---
dependency:
```

(continues on next page)

(continued from previous page)

```
  name: galaxy
driver:
  name: docker
lint:
  name: yamllint
platforms:
  - name: instance1-$MOLECULE_EPHEMERAL_DIRECTORY
    image: mariadb
  - name: instance2-$MOLECULE_EPHEMERAL_DIRECTORY
    image: retr0h/centos7-systemd-ansible:latest
    privileged: True
    command: /usr/sbin/init
provisioner:
  name: ansible
  lint:
    name: ansible-lint
scenario:
  name: default
verifier:
  name: testinfra
  lint:
    name: flake8
```

6.7 Contributing

- We are interested in various different kinds of improvement for Molecule; please feel free to raise an Issue if you would like to work on something major to ensure efficient collaboration and avoid duplicate effort.
- Create a topic branch from where you want to base your work.
- Check for unnecessary whitespace with `git diff --check` before committing.
- Make sure you have added tests for your changes.
- Run all the tests to ensure nothing else was accidentally broken.
- Reformat the code by following the formatting section below.
- Submit a pull request.

6.7.1 Installing

Installation from source or package.

6.7.2 Testing

Perform all of the *Full* testing steps prior to submitting a PR.

6.7.3 Ansible Modules

This project uses the following Ansible modules, and Gilt to manage them.

- Ansible Goss

To bring in updated upstream modules. Update *gilt.yml* and execute the following:

```
$ gilt overlay
```

6.8 Development

- Please read the *Contributing* guidelines.

6.8.1 Branches

- The `master` branch is stable. Major changes should be performed elsewhere.

6.8.2 Release Engineering

Pre-release

- Edit the *History*.
- Follow the *Testing* steps.

Release

Molecule follows Semantic Versioning.

Tag the release and push to github.com

```
$ git tag 2.x.x
$ git push --tags
```

Upload to PyPI

- Build and upload to PyPI.

```
$ make -f build/Makefile build
$ make -f build/Makefile push
$ make -f build/Makefile clean
```

Upload to Docker Hub

- Build and upload to Docker Hub.

```
$ make -f build/Makefile docker-build
$ make -f build/Makefile docker-push
```

Post-release

- Comment/close any relevant Issues.
- Announce the release in *#molecule-users*.

6.8.3 Roadmap

- See Issues on Github.com.

6.9 History

6.9.1 2.17

- Correct .env file interpolation.
- Fixes Tox link in docs.
- Adds tox-tags to test-requirements.txt.
- Expose config.project_directory as env var.
- Update Matrix usage.rst.
- Update ci.rst with Jenkinsfile example.
- Support passing arbitrary keys to vm.network.
- Pin wheel version to 0.30.0.
- Add git to docker DIND container.
- Added inspec download for Ubuntu 14.04.
- Added env to docker.
- Accept a single option to the matrix subcommand.
- Knob to change Ansible *no_log*.
- Bumped testinfra to 1.14.1 due to testinfra bug.
- Remove upgrade from Dockerfile.
- Bumped requirements.txt.
- Corrected provider_override_args.
- Add docker python and rubocop dependencies.
- Added python 3.7 support.

6.9.2 2.16

- Add feature for auto bumping docker image tag.
- Fixed Docker provider not using DOCKER_HOST environmental variable.
- Updates to the Ansible provisioning playbook for docker and vagrant for missing options.
- Documentation : dependencies on centos and docker driver clarifications.

- Added matrix subcommand.
- added pull: yes|no param to Docker executor.
- Added Gitlab CI example.
- Add information about the action which failed.
- Support Ansible 2.6.
- Corrected schema due to #1344.
- Prevalidator should enforce allowed options.
- Add support for multiple distributions to inspec verifier.
- Update InSpec to version 2.2.20.
- Update ansible-lint to version 3.4.23.
- Create unique keypair to allow parallel executions with OpenStack driver.
- Requirements update.
- Update the Dockerfile for work with az client and rubocop.

6.9.3 2.15

- Removed docker credential regexp validation.
- Added rsync to Docker image.
- Docker create playbooks: add tmpfs & security_opts docker_container parameters.
- Moved default scenario to a const.
- Pre-validate Molecule special variables.
- Added env file.
- Corrected command syntax.
- Delegated driver acts as managed.

6.9.4 2.14

- Add pre-validation.
- **MOLECULE_** special variables available in molecule.yml.
- Log Vagrant stdout to a file in MOLECULE_EPHEMERAL_DIRECTORY.
- Reintroduce base config merging.
- Corrected unit tests to work with tox.
- Add verifier mutually exclusive checking.
- UTF-8 issue in idempotence.
- Made prepare playbook optional.
- Bundle common playbooks.
- Added Goss linter.
- Disallow verifier.options with Goss and Inspec.

Important Changes

- **MOLECULE_** special variables available in molecule.yml.
- Molecule introduces a new CLI option *–base-config*, which is loaded prior to each scenario's *molecule.yml*. This allows developers to specify a base config, to help reduce repetition in their molecule.yml files. The default base config is ~/.config/molecule/config.yml.
- Prepare playbook no longer needs to exist, unless using it.
- Molecule bundles Docker and Vagrant create/destroy playbooks.

6.9.5 2.13.1

- Enable Ansible 2.4 support with py36.

6.9.6 2.13

- Allow the destroying of remote libvirt instances.
- Bumped testinfra version for Ansible 2.5.1 compatibility.
- Added RuboCop as Inspec's linter.
- Minor fixes to Goss verifier playbook.
- Update documentation for verify and idempotency checks.
- Added Inspec verifier.
- Support Void Linux when using Docker driver.
- Converge with built in Molecule skip tags.
- Render inventory links relative to scenario dir.
- Disallow null provider.env values.
- Log vagrant errors.
- Enable py36 support for Ansible 2.5.
- Retry downloading goss 3 times.
- Delegated driver should report unknown on *molecule list*.
- Correct Docker container terminal sizing.
- Bumped Ansible 2.4 minor version in tox.
- Add docker_host attribute to templates to allow talking to a remote docker daemon.
- Across-the-board requirements update.
- Add parameter for Vagrant provider override.
- Add 'halt' option to Vagrant module.

Important Changes

- Python 3.6 support.
- Added Inspec verifier.
- Added RuboCop linter for Inspec.

Breaking Changes

- Render inventory links relative to scenario dir instead of ephemeral dir. Unfortunately, this was a side effect of #1218.

6.9.7 2.12.1

- Disable pytest caching plugin.

Important Changes

- No longer need to *.gitignore* the *.pytest_cache/* directory.

6.9.8 2.12

- Ensure prune properly removes empty dirs.
- Allow verify playbook to be shared.
- Added cookiecutter tests.
- Moved temporary files to $TMPDIR.
- Added and tested Ansible 2.5 support.
- Remove include tasks from driver playbooks.
- Set *delete_fip = yes* for os_server resources.
- Relaxed schema validation for which allows unknown keys in *molecule.yml*.
- Corrected AnsibleLint *-x* example.
- Added dind support and docs.
- Exclude .venv directory from yamllint.
- Move Molecule playbook vars into host inventory.
- Switch functional tests to pytest.raises.

Important Changes

- Molecule writes temporary files to *$TMPDIR* hashed as *molecule/$role_name/$scenario_name/*. Temporary files are no longer written to *$scenario_directory/.molecule/*.
- No longer need to *.gitignore* the *.molecule/* directory.

Breaking Changes

- Users of the Goss verifier will need to change their *verifier.yml* playbook to *verify.yml*.

6.9.9 2.11

- Correct verbose flag options with *–debug*.
- Bumped Ansible 2.4 and 2.3 minor versions.
- Reimplemented schema validation with Cerberus.
- Bumped version of jinja2.
- Move merge_dicts into util.
- Forward port Molecule v1 shell dependency manager.
- Vagrantfile cleanup.
- Ability to log into a Docker registry.

Important Changes

- Reimplemented schema validation with Cerberus. The Molecule file is thoroughly validated. This may result in validation errors if the developer's *molecule.yml* is doing something unusual.
- Cleaned up the Vagrantfile, and allow the developer to change options on the base Vagrant config object.

Breaking Changes

- Changed Vagrant's *molecule.yml raw_config_args* to *provider_raw_config_args* for differentiating *instance_raw_config_args*.

6.9.10 2.10.1

- Correct Vagrant to automatically insert a keypair.
- Corrected synced_folders usage.

6.9.11 2.10

- Properly skipping Vagrant speedup keys in provider.
- Allow Vagrant to automatically insert a keypair.
- Correct molecule_vagrant.py bug where *provider_options* would cause Vagrant to fail if keys from #1147 were provided.
- Fix line length in cookie cutter README.

Important Changes

- PR #1147 reduced Vagrant create time, which disabled Vagrant from automatically inserting a keypair. Molecule's default is now changed back to Vagrant's default of True, which may reduce the speed of Vagrant create as fixed by #1147.

6.9.12 2.9

- Bumped yamllint version.
- Namespaced Docker registry.
- Reduce create time with Vagrant driver.
- Replace >>> with $ in documentation.
- Moved prune to run after destroy.
- Fix confusion between exposed and published ports in docker create playbook.
- Add basic support for libvirt in Vagrant driver.
- Ignore psutil on cygwin platform.
- Corrected ability to set multiple x options in provisioner's lint.
- Disallow privilege_escalation via schema.
- Validate schema for invalid ansible config options.
- Adding provision option for Vagrant driver.

Important Changes

- These changes do not impact existing projects. However, if one was using the old syntax, and upgraded create.yml, changes would be required. The Docker driver's registry has been moved to a key named *url* under *registry*.

```
driver:
  name: docker
platforms:
  - name: instance
    image: image_name:tag
    registry:
      url: registry.example.com
```

- Fix confusion between exposed and published ports in docker create playbook.

```
driver:
  name: docker
platforms:
  - name: instance
    image: image_name:tag
    exposed_ports:
      - "53/udp"
      - "53/tcp"
    published_ports:
      - "0.0.0.0:8053:53/udp"
      - "0.0.0.0:8053:53/tcp"
```

6.9.13 2.8.2

- Corrected ansible args.

6.9.14 2.8.1

- Reverted, release does not exist.

6.9.15 2.8

- Improved quickstart video.
- Ability to specify a custom registry to Docker driver.
- Add a link to talk demo.
- Corrected incorreclty fixed bug when tags provided to provisioner.
- Corrected dependency scenario functional tests.
- Corrected incorrectly fixed bug when providing provisioner lint options.
- Regexp support in additional_files_or_dirs.
- Add custom nameserver to Docker container.
- Add network create and destroy support to Docker driver.

Breaking Changes

- The verifier's *additional_files_or_dirs* option is relative to the test directory, as opposed to the scenario directory.
- The verifier's *additional_files_or_dirs* option now supports regexp. Molecule will add additional files or directories, only when the glob succeeds. Directories must be appended with the regexp to match, further details in the verifier's documentation.

6.9.16 2.7

- Ability to set a ulimit for the Docker driver.
- Switching log_driver from none to json-file to for compatibility with Ansible 2.2.
- Default to always destroy strategy.
- Support linked_clone for Vagrant 2.X.
- Bump tree-format to 0.1.2.
- Correct starting container on Docker edge by changing log_driver to none.
- Make psutil installation platform-dependent.

6.9.17 2.6

- Path searching to check ephemeral dir first.
- Update Goss verifier.yml.
- Bump ansible-lint version.
- Added example for setting Vagrant proxy settings for Linux.
- Never destroy instances if –destroy-never requested.
- Variable Molecule Ephemeral Directory.
- Added systemd example.

6.9.18 2.5

- Ignore provisioner.options when in the create/destroy provisioner.
- Switched Docker driver to a portable default command.
- Parallel instance management.
- Added Azure driver.
- Corrected testinfra SystemInfo tests.
- Execute *dependency* on check and converge sequence.
- Updated Docs usage of dependency role-file instead of requirements_file.
- Cleaned up YAML syntax.
- Execute linting first in test sequence.
- Support expose_ports option in docker driver.

6.9.19 2.4

- Corrected missing code block inside documentation.
- Bump ansible-lint version.
- Added yamlint to init scenario.
- Correct env path qualification.
- Add sudo package to Fedora section of Dockerfile template.
- Correct ANSIBLE_ROLES_PATH path component.
- Allow re-run of prepare playbook.

6.9.20 2.3

- Report friendly error message when interpolation fails.
- Added a new line after printing matrix.
- Added molecule header to generated Dockerfiles.
- Check supported python and ansible versions when executing Molecule.

- Sanitize user provided config options.
- Sanitize user provided env options.
- Added shell friendly env output

6.9.21 2.2.1

- Ensure setup is run for prepare to correct ssh connection failures.

6.9.22 2.2

- Ability to execute a prepare playbook post create.
- Log deprecation warning when missing prepare.yml.
- Support Ansible 2.4.
- Revert "Add support import data from original ansible.cfg".
- Changed testinfra command to py.test.

6.9.23 2.1

- Add a destroy strategy to the *test* action.
- Delegated driver may or may not manage instances.

6.9.24 2.0.4

- Fix Dockerfile for Fedora.

6.9.25 2.0.3

- Generate host/group vars when host vars missing.

6.9.26 2.0.2

- Pass the provisioner's env to the verifier.

6.9.27 2.0.1

- Corrected init scenario validation.

6.9.28 2.0

- Major overhaul of Molecule.

Important Changes

- Ansible playbooks to manage instances.
- Vagrant is managed through a custom Ansible module bundled with Molecule.
- Addition of Scenarios.
- Addition of a Delegated Driver to test instances managed outside of Molecule.
- Promoted Goss Verifier to a supported verifier.
- Added GCE Driver, EC2 Driver, LXC Driver, LXD Driver , and OpenStack Driver native Molecule drivers.

Breaking Changes

- Not compatible with Molecule v1 style config.
- Demoted serverspec support entirely.
- Does not support all of the Molecule v1 functionality or flexibility, in favor of simplicity and consistency throughout.
- Ansible 2.2 and 2.3 support only.
- See Molecule v1 to v2 Porting Guide.
- Molecule no longer defaults to passing the *–become* flag to the *ansible-playbook* command.
- Roles are linted with Yamllint vs v1's custom linter.

6.9.29 1.25.1

- Update ansible-lint for Ansible 2.4 compatibility.

6.9.30 1.25

- Display output when *idempotence* fails.
- Changed basebox to ubuntu/trusty64 for molecule init.
- Allow disable_cache parameter for Docker containers enhancement.
- Update goss verifier.
- Add a 'private' parameter in OpenStack driver.

6.9.31 1.24

- Support Ansible 2.3.

6.9.32 1.23.3

- Clean up {group,host}_vars on destroy.

6.9.33 1.23.2

- Globally disable cowsay, since it impacts the idempotence check.

6.9.34 1.23.1

- Added ungrouped hosts under all.

6.9.35 1.23

- Prescriptive ansible.cfg defaults.
- Ansible v2 has deprecated ansible_ssh_{host,port,user}.
- Docker driver: use POSIX shell and support more linux package systems.
- Add quotes around ansible_ssh_private_key_file format.
- Ansible 1.9 No longer supported.

6.9.36 1.22

- Handling of networks with Docker driver.

6.9.37 1.21.1

- Corrected None RepoTags bug with docker driver.

6.9.38 1.21

- No longer skip setting hostname with Vagrant's libvirt provider.
- Openstack: Allow using ssh keys from ssh-agent.
- Obtain driver from state file if set.
- Updated to Goss 0.3.0.
- Remove terminal warnings while running apt.
- Support for new docker sdk.
- Updated doc for docker driver links.

Breaking Changes

- The *docker-py* pip package has been deprecated in favor of *docker*.

6.9.39 1.20.3

- Version bump, network interuption while uploading package to pypi.

6.9.40 1.20.2

- Correct testinfra tests discovered twice.

6.9.41 1.20.1

- Correct too many authentication failures error.

6.9.42 1.20

- Expose network configuration to docker driver.
- Openstack: Performance improvements for multiinstance setups.
- Do not require a project_config when a local_config is present.
- Corrected molecule.yml's group_vars/host_vars.

Breaking Changes

- The *host_vars* and *group_vars* section of molecule.yml no longer accepts a list, rather a dict similar to Ansible's vars usage.

6.9.43 1.19.3

- Openstack: Use configured ssh key.

6.9.44 1.19.2

- Properly handle testinfra verbose flag setting.

6.9.45 1.19.1

- Add raw_config_args option to providers.

6.9.46 1.19

- Convert vagrantfile from relying on jinja.

6.9.47 1.18.1

- Make Openstack ssh timeout configurable.

6.9.48 1.18

- Fix availability timeout in Openstack driver.
- Do not alter users known_hosts file in Openstack driver.
- Allow using environment variables in molecule.ym.
- Make ansible.cfg settings configurable through molecule.yml.
- Add multiple network support in Openstack driver.
- Add links functionality to Docker driver.
- Switched options from 'sudo' to 'become'.

6.9.49 1.17.3

- Create test skeleton with *molecule init* when initializing a role in current directory.

6.9.50 1.17.2

- Fix unittests to allow ls to be in both /usr/bin and /bin.
- Force raw_env_vars to string for *ansible-playbook*.

6.9.51 1.17.1

- Correct functional tests.
- Correct locale issues with print class of methods.
- Correct ansible-lint exit error when role dependency is in newer dictionary format.
- Pass env to *ansible-lint*.

6.9.52 1.17

- Cleanup sphinx doc generation.
- Bumped testinfra requirement which drops the now useless installation of which in centos and fedora images.
- Made OpenStack's ip pool configurable.
- Corrected Docker's overlayfs for RPM based distros.
- Fixed OpenStack's security_groups default for newer shade versions.
- Added missing bash completion targets.

6.9.53 1.16.1

- Removed check mode from running in test cycle.

Breaking Changes

- Molecule no longer runs in "Dry Mode" as part of *molecule test*. If one wishes to incorporate check as part of *test*, molecule.yml can be updated to include this as part of the test sequence.

6.9.54 1.16

- Slightly improved unit test coverage.
- Various doc improvements.
- Added Gilt usage to docs.
- Reimplemented info, error, debug message handling.
- Nice error message when rake and/or rubocop missing.
- Fix task determination on idempotence failure.
- Added a github issue template.
- Logging of dependency command execution.

6.9.55 1.15

- Added a shell dependency manager.
- Created a CI section to documentation with Tox details.
- Rename dependencies key to dependency.

Breaking Changes

- The galaxy override options have been moved to the *dependency* section of molecule's config. No longer support a top level *dependencies* config key. This functionality was added in 1.14, and this follow-up corrects the usage, before 1.14 was utilized.

6.9.56 1.14.1

- Fix openstack driver login and ssh key generation.

6.9.57 1.14

- Made improvements to unit/functional tests.
- Fixed Goss verifier under Ansible 2.2.
- Removed testinfra config backward compatibility.
- Broke out role dependency into a subcommand.

Breaking Changes

- The testinfra override options have been moved to the *verifier* section of molecule's config. No longer support a top level *testinfra* config key.
- The galaxy override options have been moved to the *dependencies* section of molecule's config. No longer support a *galaxy* key inside the top level *ansible* section.

6.9.58 1.13

- Implement environment handling in docker driver.
- Added vmware_workstation provider to vagrant.
- Improved overall logging, including logging of *sh* commands when debug flag set.
- Avoid images with <none> tag.
- Support and test ansible 2.2 and 2.1.2.
- Allow nested testinfra test directory structure.
- Ability to pass arbitrary ansible cli flags to *converge*.
- Added IRC info to docs.
- Return exit code from goss verifier.
- General cleanup of modules and documentation.
- Bumped requirements versions.

6.9.59 1.12.6

- Disable diff when executing idempotent check.
- Make sure ansible-lint respects the molecule ignore_paths.
- Convert readthedocs links for their .org->.io migration for hosted projects.

6.9.60 1.12.5

- Increased test coverage.
- Allow group/host vars in molecule.yml to work with ansible 1.9.
- Pass HOME to ansible-lint environment.
- Expose driver to login.
- Improved login error message messaging.

6.9.61 1.12.4

- Added a private disabled top level key. Do not use or rely on this key. Added for our molecule adoption.
- Added a coverage minimum.
- More unit and functional coverage.

6.9.62 1.12.3

- Write templates even when a custom ansible.cfg is specified.

6.9.63 1.12.2

- Removed default multiple-instances from init.

6.9.64 1.12.1

- Preserve ansible.cfg when supplying a custom one.

6.9.65 1.12

- Additional command tests.
- Changed connection to ansible_connection.
- Implemented click vs docopt. This slightly changes the CLI's semantics.
- Removed the driver python packages from installing with molecule.
- Set ssh key if specified in OpenStack driver.
- Using py.test as functional test runner.
- Added a Gemfile to `molecule init` serverspec verifier.
- Added SUSE docker driver support.
- Display the list of non-idempotent tasks with `molecule idempotence`.

Breaking Changes

- The `--debug` flag is no longer passed to the subcommand. The command and subcommand args were getting munged together, and passed to the core. They are now handled separately.
- Removed the `--debug` subcommand flag from all usage – it was never used.
- The `init` subcommand requires an optional `--role` flag vs a role argument when naming the role to initialize.
- The `init` subcommand requires a `--driver` flag when creating a driver other than vagrant.
- The `init` subcommand requires a `--verifier` flag when creating a verifier other than testinfra.
- The `login` subcommand requires a `--host` flag when more than one instance exists.
- One must install the appropriate python package based on the driver used.

6.9.66 1.11.5

- Set ssh key if specified with the OpenStack driver.
- Pass ANSIBLE_CONFIG when executing ansible-lint.

6.9.67 1.11.4

- Hide ansible-lint stacktrace on `molecule verify`.
- Corrected linked clone platform options checking.

6.9.68 1.11.3

- Handle when a container is stopped outside of molecule, when running `molecule status`.

6.9.69 1.11.2

- Preserve sudo passed in verifier options.

6.9.70 1.11.1

- Corrected bug when passing the `--platform` flag.

6.9.71 1.11

- General cleanup of core module.
- Various documentation updates.
- Pull molecule status from state file when using Vagrant driver.
- Added alpha Goss verifier support.
- Updated runtime requirements to current versions.
- Implemented `molecule check` subcommand.
- Configure verifier to be test kitchen like.
- Ability to declare multiple drivers in config.
- Implement ansible groups inheritance.

Breaking Changes

Previously molecule would execute a test framework based on the existence of a directory structure. This is no longer the case. Molecule will execute the configured suite, where *testinfra* is the default. See docs.

6.9.72 1.10.3

- Reimplemented idempotence handling. Removed the idempotence ansible callback plugin, in favor of a native implementation.

Note

There is no change in workflow. Molecule still reports if a converge was idempotent or not. However, it no longer reports which task(s) are not idempotent.

6.9.73 1.10.2

- Removed pytest-xdist from runtime deps. This allows testinfra's dependency on pytest to properly install.

6.9.74 1.10.1

- Pinned to explicit version of testinfra, due to pytest incompatabilities.

6.9.75 1.10

- Added ability to specify custom dockerfile.
- Added ability to generate and destroy temporary openstack keypair and ssh key file if they are not specified in the molecule.yml.
- Implemented Cookiecutter for `molecule init`.
- Documentation improvements.

Breaking Changes

Roles may fail to converge due to the introduction of additional verifiers.

- Added flake8 linter to testinfra verifier.
- Implemented ansible lint.

6.9.76 1.9.1

- Correct a converge –debug bug.
- Correct ansible galaxy role path.

6.9.77 1.9

- Restructured and reogranized internal code, tests, and docs.
- Added functional scenario tests.
- Improved unit tests/coverage.
- Added auto docker api version recognition to prevent api mismatch errors.
- Added fallback status for vagrant driver.
- Control over ansible galaxy options.
- Display molecule status when not created.
- Added dependency installation state, and installation step for syntax check.
- Pinned runtime requirements.
- Update login to use state data.
- Ability to target ansible groups with testinfra.
- Ability to target docker hosts with serverspec.

- Added ../../ to rolepath to fix ansible 2.1.1 default role search.
- Added docker volume mounting.
- Add support for Docker port bindings.
- Implemented a new core config class.

Breaking Changes

- Existing Testinfra tests which use the Docker driver need updating as described in 398.

6.9.78 1.8.4

- Fixed role_path with ansible 2.1.1.

6.9.79 1.8.3

- Fixed passing flags to molecule test.

6.9.80 1.8.2

- Fixed a bad reference to the molecule_dir config variable.

6.9.81 1.8.1

- Fixed a bug where molecule would fail if .molecule/ didn't already exist.

6.9.82 1.8

- Added native support for OpenStack provider.
- Fixed a bug where testinfra_dir config option wasn't being handled.
- Fixed a bug with `molecule login` where its host matching didn't work with overlapping names.

6.9.83 1.7

- It's now possible to define host_vars and group_vars in ansible section of molecule.yml.
- The –platform CLI option now supports `all`.
- Corrected issue with specifying serverspec args in molecule.yml.

6.9.84 1.6.3

- Updated config parsing so that testinfra.sudo and testinfra.debug can be set in molecule.yml.
- Demo role now pulls in correct serverspec config.

6.9.85 1.6.2

- Added inventory-file flag to `molecule check` to address Ansible 1.9.x specific issue.

6.9.86 1.6.1

- Fixed a bug preventing `molecule test` from working.
- Added a demo role for functional testing.

6.9.87 1.6

- Added –offline option to `molecule init`.
- `molecule status` now shows hosts by default.
- `molecule test` will now fail immediately when encountering an error.
- Switched to Python's logging module for displaying STDOUT, STDERR.
- Added support for libvirt provider.
- Added `molecule check` to check playbook syntax.
- Testinfra parameters can now be set as vars in molecule.yml.
- Running testinfra tests in parallel is no longer the default behaviour.

6.9.88 1.5.1

- Fixed issue with testinfra and serverspec attempting to share args.
- Added –sudo option for testinfra.
- Added tab completion support.
- Misc. Docker updates and fixes.

6.9.89 1.5

- Added support for Docker provisioner.
- Added support for group_vars.

6.9.90 1.4.2

- Made "append_platform_to_hostname" False by default.
- Testinfra tests now run in parallel.
- `init` now generates testinfra tests by default.
- Testinfra env vars (including ssh) are now consistent with what is passed to ansible-playbook.

6.9.91 1.4.1

- Fixed a bug where testinfra_dir wasn't being used.
- Changed append_platform_to_hostname to default to False.

6.9.92 1.4

- Updated `init` to install role dependencies from Ansible Galaxy.
- Now using DocOpt subcommands to dispatch commands internally.
- Updated `login` command to take no hostname (for single instances) and partial hostnames.
- Improved visibility when running (and not running) tests.
- Can now pass multiple instances of –tags for specifying more than one tag.
- Can now pass –destroy flag to `test` with various options suitable for use in CI.
- Numerous small bug fixes.

6.9.93 1.3

- Added very basic support for the vagrant-triggers plugin.

6.9.94 1.2.4

- Fixed a bug introduced in 1.2.3 preventing `init` from working.

6.9.95 1.2.3

- Fixed a bug where `destroy` would fail on VMs that hadn't been created. Caused errors running `test`.
- Moved rubocop, rake, and testinfra into validators. Added tests.
- Moved ansible-playbook logic out of core, commands and into a dedicated class. Added tests.
- Provisioner logic moved to its own class outside of core.

6.9.96 1.2.2

- Added a CLI option for the `list` command to make the output machine readable.
- Refactored commands.py to be more conducive to dispatch from DocOpt (#76).
- Fixed issue #82 where callback plugin path wasn't being properly merged with user-defined values.
- Fixed issue #84 where `molecule init` would produce a molecule.yml that contained trailing whitespace.
- Fixed issue #85 preventing user-defined serverspec directory from being used.

6.9.97 1.2.1

- Updated idempotence plugin path to be appended to existing plugin path rather than overwriting it.
- Fixed case where idempotence plugin would crash when unable to read response dictionary.

6.9.98 1.2

- Added support for Vagrant 1.8's linked_clone option.
- Updated idempotence test to use an Ansible callback plugin that will print failed tasks.
- Path to templates can now be relative to a user's home directory.
- box_url in Vagrantfile is no longer set if box_version is defined.
- Uses the latest version of python-vagrant.

6.9.99 1.1.3

- Fixed a bug where inventory wasn't getting created on a new converge.
- Linting now targets a specific list of file extensions.
- Hostname created during `init` is now sanitized.
- Creattion of python cache directory is now disabled by default.

6.9.100 1.1.2

- Fixed a bug where calling `create` separately from `converge` wasn't generating an inventory file.

6.9.101 1.1.1

- Cleaned up state file management logic to be more concise, functional for other purposes.
- Removed –fast flag from converge in favor of using state file for fast converging.
- Instance hostname is now printed during serverspec runs.
- Fixed a bug where loading template files from absolute paths didn't work.

6.9.102 1.1

- Added support for static inventory where molecule can manage existing sites, not just vagrant instances.
- Added support for skipping instance/inventory creation during `molecule converge` by passing it –fast. MUCH faster.

6.9.103 1.0.6

- Fixed a bug preventing vagrant raw_config_args from being written to vagrantfile template.
- Cleaned up error messaging when attempting to *molecule login* to a non-existent host.
- Added release engineering documentation.
- Moved commands into a separate module.
- Switched to using yaml.safe_load().
- Added more debugging output.

6.9.104 1.0.5

- Added support for Vagrant box versioning. This allows teams to ensure all members are using the correct version in their development environments.

6.9.105 1.0.4

- Fixed a bug where specifying an inventory script was causing molecule to create it.
- config_file and inventory_file specified in ansible block are now treated as overrides for molecule defaults.

6.9.106 1.0.3

- Updated format of config.yml and molecule.yml so they use the same data structure for easier merging. In general it's more clear and easy to understand.
- Defaults are now loaded from a defaults file (YAML) rather than a giant hash. Maintaining data in two formats was getting tiresome.
- Decoupled main() from init() in Molecule core to make future tests easier.
- Removed mock from existing tests that no longer require it now that main() is decoupled.
- Moved all config handling to an external class. Greatly simplified all logic.
- Added tests for new config class.
- Cleaned up all messages using format() to have consistent syntax.
- Fixed status command to not fire unless a vagrantfile is present since it was triggering vagrant errors.
- Renamed _init_new_role() to init() to be consistent with other commands.
- Fixed incorrect messaging in _print_valid_providers().
- Fixed edge case in vagrantfile template to make sure we always have default cpus/memory set for virtualbox instances.
- Leveraged new config flexibility to clean up old hack for *molecule init*.
- Fixed utility test for deep_merge that was failing.
- Made print_line two different functions for stdout and stderr.
- Updated print functions to be Python 3 ready.
- Moved template creation into a generic function.

- Test all the (moved) things.
- Updated image assets.
- Removed aio/mcp naming from docs and templates.

6.9.107 1.0.2

- Switched to deep merging of config dicts rather than using update().

6.9.108 1.0.1

- Fixed trailing validator, and broke out into a module.

6.9.109 1.0

- Initial release.

6.10 Credits

6.10.1 Development Leads

- John Dewey (@retr0h)

6.10.2 Core Committers

- Adam Brown <adambr2@cisco.com>
- Erik Nadel <einadel@wpi.edu>

6.10.3 Contributors

```
$ git shortlog -s | cut -c8-
```

6.11 FAQ

6.11.1 Why does Molecule make so many shell calls?

Ansible provides a python API. However, it is not intended for direct consumption. We wanted to focus on making Molecule useful, so our efforts were spent consuming Ansible's CLI.

Since we already consume Ansible's CLI, we decided to call additional binaries through their respective CLI.

Note: This decision may be reevaluated later.

6.11.2 Why does Molecule only support Ansible versions 2.2 and later?

- Ansible 2.2 is the first good release in the Ansible 2 lineup.
- The modules needed to support the drivers did not exist pre 2.2 or were not sufficient.

6.11.3 Why are playbooks used to provision instances?

Simplicity. Ansible already supports numerous cloud providers. Too much time was spent in Molecule v1, re-implementing a feature that already existed in the core Ansible modules.

6.11.4 Have you thought about using Ansible's python API instead of playbooks?

This was evaluated early on. It was a toss up. It would provide simplicity in some situations and complexity in others. Developers know and understand playbooks. Decided against a more elegant and sexy solution.

6.11.5 Why are there multiple scenario directories and molecule.yml files?

Again, simplicity. Rather than defining an all encompassing config file opted to normalize. Molecule simply loops through each scenario applying the scenario's molecule.yml.

Note: This decision may be reevaluated later.

6.11.6 Are there similar tools to Molecule?

- Ansible's own Testing Strategies
- ansible-test (abandoned?)
- RoleSpec

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

C

D

E

F

G

I

L

M

O

P

R

S

T

V

Y