Testing Ansible Playbooks in Your Kitchen

Safe, repeatable testing with Test Kitchen, Docker, Podman, Vagrant, and QEMU

Robby Callicotte

LinuxFest Northwest 2026

About Me

- Robby Callicotte, Senior Software Engineer, Redhat
- 20+ years in IT Infrastructure management, DevOps, Systems Engineering
- Fedora packager, Member of EPEL Steering Committee, CentOS Alt Images SIG

@robbycl2v@fosstodon.org

@rcallicotte:fedora.im

robby@redhat.com

The Problem

"It worked on my machine...
then it burned down the server."

-- every engineer at some point

Why Test Your Playbooks?

- Infrastructure code is still code
- Manual testing is slow, error-prone, and scary
- Production *should not* be your test environment
- Playbooks grow in complexity over time
- Teams need confidence to ship changes

The Ansible Testing Landscape

Tool	What It Does	Level
ansible-lint	Static analysis & best practices	Linting
yamllint	YAML syntax validation	Linting
Molecule	Role testing framework	Integration
Test Kitchen	Full playbook testing with real VMs/containers	Integration
--check mode	Dry-run your playbook	Smoke test

Enter Test Kitchen

Your local lab for Ansible experiments

What is Test Kitchen?

- Originally built for Chef — works great with Ansible, Puppet, Saltstack
- Spins up real instances (containers or VMs)
- Runs your playbook against them
- Verifies the result with InSpec, ServerSpec or other verifiers (bash[pytest], busser)
- Tears everything down when done

The Test Kitchen Workflow

- Create — Spin up a fresh instance
- Converge — Run your Ansible playbook
- Verify — Run tests to check the result
- Destroy — Tear it all down cleanly

One Command To Rule Them All

```
# Run the full lifecycle in one shot:
$ kitchen test

# Or step through manually for debugging:
$ kitchen create
$ kitchen converge
$ kitchen verify
$ kitchen login      # Log in and poke around
$ kitchen destroy
```

Setting Up Your Kitchen

Prerequisites, drivers, and configuration

Prerequisites

- Ruby — Test Kitchen is a Ruby gem
- Bundler — Dependency management
- Ansible — Installed on your workstation
- A driver — Docker, Vagrant, or QEMU (and dependencies)
- kitchen-ansible — The Ansible provisioner plugin

Gemfile

Gemfile

```
source 'https://rubygems.org'

gem 'test-kitchen'
gem 'kitchen-ansible'
gem 'kitchen-docker'        # or kitchen-vagrant
gem 'kitchen-inspec'        # verifier


$ bundle install
```

.kitchen.yml — The Heart of It

.kitchen.yml

```
driver:
  name: docker

provisioner:
  name: ansible_playbook
  playbook: site.yml
  hosts: all

verifier:
  name: inspec

platforms:
  - name: almalinux-9
  - name: centos-stream-10
  - name: debian-13

suites:
  - name: default
```

Choosing Your Driver

Docker, Podman, Vagrant, or QEMU

Driver Comparison

Driver	Speed	Isolation	Systemd	Best For
Docker	Fast	Medium	Tricky	Quick feedback loops
Podman	Fast	Medium	Better	Rootless, SELinux
Vagrant	Slower	Full VM	Yes	Full OS fidelity
QEMU	Slower	Full VM	Yes	Cross-arch testing

Docker / Podman Driver

.kitchen.yml (Docker)

```
driver:
  name: docker
  use_sudo: false
  privileged: true          # needed for systemd
  run_command: /sbin/init   # boot systemd

platforms:
  - name: ubuntu-24.04
    driver:
      image: ubuntu:24.04
      platform: ubuntu
```

Vagrant / QEMU Driver

.kitchen.yml (Vagrant)

```
driver:
  name: vagrant
  provider: libvirt        # or virtualbox

platforms:
  - name: almalinux-9
    driver:
      box: generic/alma9
      customize:
        memory: 1024
        cpus: 2
```

Writing Verification Tests

Proving your playbook did what you think it did

Pytest-Testinfra: Infrastructure as Test Code

- Human-readable test framework for infrastructure (It's Python!!)
- Tests what your system looks like after Ansible runs
- Checks packages, services, files, ports, users, etc.
- Tests live in test/integration/default/

Pytest-Testinfra Test Examples

test/integration/default/test_webserver.py

```
# Is nginx installed?
def test_nginx_is_installed(host):
    assert host.package(“nginx”).is_installed

# Is it running and enabled?
def test_nginx_is_running_enabled(host):
    assert host.service(“nginx”).is_running
    assert host.service(“nginx”).is_enabled

# Does the config contain our server name?
def test_nginx_server_name(host):
    assert host.file(“/etc/nginx/nginx.conf”).contains(“myserver”)
```

Project Layout

```
my-ansible-project/
  .kitchen.yml
  .kitchen.local.yml # local dev instance (place in .gitignore)
  site.yml                # your playbook
  roles/
    webserver/
      tasks/main.yml
      handlers/main.yml
      templates/nginx.conf.j2
  test/
    integration/
      default/
        test_webserver.py  # Pytest tests
```

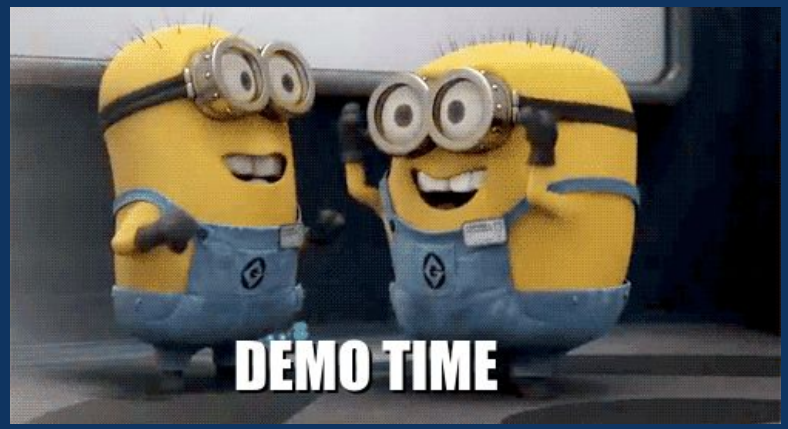
What a Successful Run Looks Like

```
$ kitchen test
-----> Starting Test Kitchen
-----> Testing <webserver-c10s>
-----> Creating <webserver-c10s>...
       PLAY [Deploy static web page] ************************************************

       TASK [Gathering Facts] *******************************************************
       ok: [localhost]
-----> Setting up <webserver-c10s>...
       Finished setting up <webserver-c10s> (0m0.00s).
-----> Verifying <webserver-c10s>...

============================== 8 passed in 1.56s ===============================
       Finished verifying <webserver-c10s> (0m1.86s).
-----> Destroying <webserver-c10s>...
      Finished destroying <webserver-c10s> (0m0.54s).
       Finished testing <webserver-c10s> (0m47.96s).
-----> Test Kitchen is finished. (0m48.35s)
```

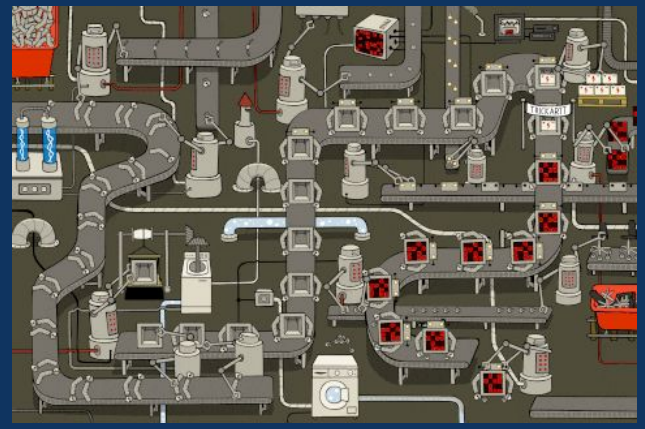
Live Demo



https://github.com/c4t3l/demo-kitchen-ansible

CI/CD Integration

Automating your kitchen in the pipeline



GitHub Actions Example

`.github/workflows/test-ansible.yml`

```yaml
name: Test Ansible Playbooks
on: [push, pull_request]

jobs:
  kitchen-ci:
    runs-on: [self-hosted, kitchen]
    steps:
      - name: Checkout Code
        uses: actions/checkout@v4
      - name: Run Kitchen test
        env:
          MY_SECRET: ${{ secrets.mypass }}
        run: kitchen test -d
```

CI Tips & Best Practices

- Use Docker/Podman in CI (if appropriate) — faster than VMs
- Run ansible-lint and yamllint first as a fast gate
- Use kitchen test --concurrency=N for parallelism
- Cache Ruby gems and Docker/Podman/System images
- Run on PRs — catch errors before merge
- Consider a nightly full matrix for broader coverage

Common Pitfalls

- Systemd in containers — need privileged mode + /sbin/init
- Networking differences — containers vs VMs vs production
- Missing Python — minimal images may lack Python
- Slow feedback loops — use kitchen converge to iterate
- Ignoring idempotency — run converge twice, expect zero changes

Testing Idempotency

.kitchen.yml

```
provisioner:
  name: ansible_playbook
  playbook: site.yml
  idempotency_test: true   # runs playbook twice!
```

Second run should show changed=0.

If not, your playbook isn't truly idempotent —
and that's a bug waiting to happen.

Key Takeaways

- Infrastructure code deserves the same testing rigor as app code
- Test Kitchen gives you real environments, not mocks
- Start simple: one platform, a few Pytest tests
- Build up to multi-platform matrix testing in CI
- Test behavior and outcomes, not Ansible internals
- The goal: confidence to ship changes

Resources

- kitchen.ci — Test Kitchen documentation
- github.com/neillturner/kitchen-ansible — Ansible provisioner
- docs.pytest.org/en/stable/ — Pytest documentation
- testinfra.readthedocs.io/en/latest/index.html — Pytest plugin
- docs.ansible.com — Ansible documentation
- ansible.readthedocs.io/projects/lint — ansible-lint
- osinside.github.io/kiwi/ — kiwi OS image builder

Questions?

Thank you! Don't burn down your servers.