Test-Kitchen Demo

Tom Duffield

tom@getchef.com
@tomduffield

Introduction

Test Kitchen is...

- "a test harness tool to execute your configured code on one or more platforms in isolation."
- a development utility that provides quick, iterative feedback on code by converging on and running tests against a running machine.
- a tool that helps speed up your infrastructure QA cycle.
- an open source project maintained primarily by Fletchor Nichol.
- distributed as a RubyGem

The Old Way vs The New Way

- The Old Way
  - Manually test against a live development environment.

For Infracoders...
Dev is Your Prod

The Old Way vs The New Way

- The Old Way
  - Manually test against a live development environment.
  - Test manually against a one Vagrant box at a time.
  - Distribute test code to your node as part of your cookbook and use a Chef Handler like *Chef MiniTest Handler* to execute tests.
- The New Way
  - Test multiple platforms in parallel.
  - Keep test code separate from your cookbook.
  - Integration with Continuous Integration / Continuous Delivery.

Test-Kitchen Components

Driver	Plugin code responsible for creating the (virtual) machine used to test your code.
Provisioner	Built-in code responsible for executing your infrastructure code on the machine.
Platform	List of operation systems on which we want to run our code using the specified Driver.
Suite	Defines the scope of the test including what policy to apply (Chef run_list, Puppet Manifest, Ansible Playbook, etc) and which tests to run.
Busser	RubyGem plugin configured as part of the Suite that executes tests on your instance.

Getting Started

To get started quickly, simply install the necessary gems.

```
$ gem install test-kitchen
```

Once the gem is installed, initialize a working directly. This can either be a chef-repo, a cookbook, or any other directory.

```
$ kitchen init
      create  .kitchen.yml
      create  test/integration/default
         run  gem install kitchen-vagrant from "."
Successfully installed kitchen-vagrant-0.14.0
1 gem installed
```

The output of this initialization command is:

- a configuration template (`.kitchen.yml`)
- a skeletal framework for tests (`test/integration/default`)
- the installation of the default Driver (`kitchen-vagrant`)

Kitchen YAML

```yaml
driver: vagrant
provisioner: chef_zero

platforms:
- name: ubuntu-12.04
  driver_config:
    box: vagrant-ubuntu-12.04
    box_url: http://files.vagrantup.com/precise64.box
    require_chef_omnibus: true
  run_list:
  - recipe[apt]

suites:
- name: default
  run_list:
  - recipe[postfix]
  - recipe[mysql::server]
  - recipe[ghost::database]
  - recipe[ghost::default]
  - recipe[ghost::nginx]
  attributes:
    mysql:
      bind_address: "127.0.0.1"
      server_root_password: "foobar"
      server_repl_password: "foobar"
      server_debian_password: "foobar"
```

Sharing Tests

The artifacts created by `kitchen init` are designed to be stored in source control with the rest of your automation code.

Don't Share Everything

```
$ git init
Initialized empty Git repository in /some/local/directory
```

```
$ kitchen init
   identical  .kitchen.yml
      create  .gitignore
      append  .gitignore
      append  .gitignore
         run  gem install kitchen-vagrant from "."
Successfully installed kitchen-vagrant-0.14.0
1 gem installed
```

If `kitchen init` is run inside of a git repository, it will configure your `.gitignore` file to ignore the `.kitchen` directory and `.kitchen.local.yml`

- `.kitchen` directory is where all the run-time data is stored (including running `.vmdks`)
- `.kitchen.local.yml` is helpful when there are private values (passwords, secret keys) and overrides to the default `.kitchen.yml`.

Demo

Kitchen Command Guarantee

The designers of Test Kitchen have taken great care to ensure that exit codes are always appropriate.

- Test Kitchen will always exit with code **0** if all operations were successful.
- Test Kitchen will always exit with non-zero if any part of the operation was unsuccessful.

This exit code behavior is fundamental for its use in CI/CD pipelines.

kitchen list

- `Instance` is how'll you'll refer to the VM
  - `{suite}-{platform}` with punctuation taken out
- `Driver` is the Driver that will be used to control the `Instance`
- `Provisioner` is the Provisioner that will be used to converge the `Instance`
- `Last Action` shows the status of the `Instance`
  - `Not Created`
  - `Created` - `Instance` is running but has not been converged.
  - `Converged` - `Provisioner` has successfully run on the `Instance`.
  - `Verified` - `kitchen verify` has returned with no errors.

kitchen test

1. Destroys the instance if it exists
    - `Cleaning up any prior instances of <default-ubuntu-1204>`
2. Creates the instance
    - `Creating <default-ubuntu-1204>`
3. Converges the instance
    - `Converging <default-ubuntu-1204>`
4. Sets up Busser and runner plugins on the instance
    - `Setting up <default-ubuntu-1204>`
5. Verifies the instance by running Busser tests
    - `Verifying <default-ubuntu-1204>`
6. Destroys the instance
    - `Destroying <default-ubuntu-1204>`

Thanks for listening