CHEF®
Progress®

Accelerate Test-driven Development with Chef Workstation and Test Kitchen

Test your Chef code locally and in the cloud

Date 06/2021

Contents

Introduction

Test Kitchen, one of the free tools that ships with Chef Workstation, makes automation testing a snap by enabling you to test your code and policies on a variety of enterprise-grade systems in containers, VMs and public cloud instances.

As with any software development, it's important to test your Chef code in environments that match your development servers. These steps are sometimes ignored or given cursory attention because it can be time-consuming to spin up test machines and then test configurations. In this guide, I'll explain how to use Test Kitchen with Docker, Vagrant (with VirtualBox) and in AWS with maximum speed and minimum pain. You can also use Test Kitchen with Azure, Digital Ocean, Google and other cloud vendors; OpenStack; Hyper-V and others.

Chef + Test Kitchen: Better Together

Test Kitchen allows you to apply your Chef Infra cookbooks and recipes to a variety of machines and immediately verify the results with Chef InSpec. Chef Infra configures systems the way you want, and Chef InSpec ensures those systems are properly deployed and remain in the states you want.

For example, if you want to ensure NGINX is installed and ports 80 and 443 are open, Chef Infra will make that happen and Chef InSpec will check the results. On one or two nodes, this is handy, but when you're managing thousands of different Linux (and Windows) machines, it's a must – and allows you to perform tests in your Test Kitchen development environment that produce the same results in production.

Start with Chef Workstation

To take full advantage of Test Kitchen, download Chef Workstation, the drop-in replacement for the now-deprecated ChefDK. Chef Workstation includes `chef`, `knife`, `kitchen`, `inspec`, `cookstyle` and `hab` – all the command-line tools you need to develop Chef solutions.

Download the package for your Linux, Mac or Windows laptop at https://downloads.chef.io/tools/workstation. Once installed, execute the following commands from your CLI to allow Chef Workstation to use the embedded Ruby and other packages it needs to run. For example, on Linux or Mac:

```
$ echo 'eval "$(chef shell-init bash)"' >> /home/$USER/.bash_profile

$ echo 'export PATH="/opt/chef-workstation/embedded/bin:$PATH"' >>
/home/$USER/.configuration_file
```

The rest of this guide assumes you have generated a local Chef Repo with `chef generate repo <my-repo>`. Within that repo directory is a cookbooks folder, where you'll create your examples.

```
$ cd ~/chef-repo/cookbooks
```

Create and Test with Dokken

Use the `chef generate cookbook` command to create a new cookbook, adding the `-k dokken` flag. This will automatically generate a `kitchen.yml` file as part of your cookbook that can be used to create and run special Docker containers for testing. The `kitchen.yml` file tells Test Kitchen what to do, from setting the target-system driver, provisioner, transport and InSpec verifier. It also contains entries for your target platforms (CentOS, RedHat, SUSE, Ubuntu, etc.) and suites that can run additional cookbook recipes or InSpec verification tests.

The `Policyfile.rb` is automatically created when you generate your cookbook, and includes the run list – the recipe or recipes intended to run on your nodes.

```
name 'cowsay'

default_source :supermarket

run_list 'cowsay::default'

cookbook 'cowsay', path: '.'
```

In the example in this Chef Guide, you'll create a simple cookbook that installs Cowsay and shows you each node's IP address in an ASCII image whenever you login:

```
 _______________
< 172.31.22.66  >
 ---------------
   \
    \
        .--.
       |o_o |
       |:_/ |
      //   \ \
     (|     | )
    /'\_   _/`\
    \___)=(___/
```

To get started, create the Cowsay cookbook, adding the `-k` flag to generate the Dokken `kitchen.yml` file:

```
$ cd ~/chef-repo/cookbooks

$ chef generate repo cowsay -k dokken
```

This command creates a default `kitchen.yml` file with two test platforms, Ubuntu 20.04 and CentOS 8. Let's change the second platform entry to test on CentOS 7. You can also enter different or additional platforms to suit your needs:

```
---
driver:
  name: dokken
  use_sudo: false
  privileged: true

provisioner:
  name: dokken

transport:
  name: dokken
```

```
verifier:
  name: inspec

platforms:
  - name: ubuntu-20.04
    driver:
      image: dokken/ubuntu-20.04
      pid_one_command: /bin/systemd
      intermediate_instructions:
        - RUN /usr/bin/apt-get update
  - name: centos-7
    driver:
      image: dokken/centos-7
      pid_one_command: /usr/lib/systemd/system


suites:
  - name: default
    verifier:
      inspec_tests:
        - test/integration/default
```

Dokken uses special containers that add back in some of the capabilities common to full virtual machines, such as `systemd`. Though you have not created a recipe file, you can run `kitchen list` to see the Docker nodes Test Kitchen intended to create for you:

```
Instance             Driver  Provisioner  Verifier  Transport  Last Action    Last Error
default-ubuntu-2004  Dokken        Inspec    Dokken     <Not Created>  <None>
default-centos-7     Dokken        Inspec    Dokken     <Not Created>  <None>
```

You can use the **Instance** names later to log in to your Docker container nodes.

Create a Multi-platform Cowsay Recipe

Since you're looking to test this small cookbook on different platforms, you'll create a recipe that works slightly differently on each. Ubuntu 20.04 is part of the Debian family, and CentOS 7 is part of the Red Hat Enterprise Linux (RHEL) family, so you'll use the built-in Chef `platform_family` resource to ensure the installation works on both despite their differences.

CentOS needs the `epel-release` repository, which includes Cowsay, so using `platform_family?('rhel')` will add that repo for the CentOS 7 node, but not for the Ubuntu 20.04 node. Next, use the `package` resource to install Cowsay (Chef knows how to use `apt` and `yum` to do that), and then have Chef write a line to the profile that will make Cowsay show the login graphic with the IP address. Finally, the recipe uses the `bash` resource to create a symbolic link to the `cowsay` binary, which is located in different places on Ubuntu and CentOS:

```
# On RHEL platforms, add the epel-release repo
if platform_family?('rhel')
  yum_package 'epel-release'
elsif platform_family?('debian')
  apt_update 'update'
end

# Install the cowsay package on all platforms
package 'cowsay'

# Have cowsay show the IP address on login for all platforms
bash 'run_cowsay' do
  code <<-EOS
  echo "hostname -I | /usr/bin/cowsay -f tux" >> /etc/profile
  EOS
  not_if 'grep -q cowsay /etc/profile'
end

# On Debian platforms, create a symbolic link to cowsay
if platform_family?('debian')
  link '/usr/bin/cowsay' do
    to '/usr/games/cowsay'
  end
end
```

Note that the `bash` resource includes the `not_if` option, which ensures the `hostname -I` command is written to the `/etc/profile` file just once. Without this, a new line will be added to the file each time the Chef client runs.

Create a Simple Test to Verify the Node's State

With the recipe set, create a simple `./test/integration/default/default_test.rb` file Test Kitchen will use to verify the state of your nodes. These are simple tests to ensure Cowsay is installed and the `/etc/profile` file contains the command you added:

```
describe package('cowsay') do
  it { should be_installed }
end

describe file('/etc/profile') do
  its('content') { should match /cowsay/ }
end
```

There are no platform differences here. The recipe took care of them for you. To do a full InSpec test, you can separately convert these simple tests into an InSpec profile, which would look like this:

```
control 'cowsay-installed' do
  impact 0.7
  title 'Ensure cowsay is installed'
  describe package('cowsay') do
      it { should be_installed }
  end
end
```

```
control 'cowsay-profile' do
  impact 0.7
  title 'Ensure cowsay runs at login'
  describe file(/etc/profile) do
      its('content') { should match /cowsay/ }
  end
end
```

With the recipe and simple tests complete, you're ready to use Test Kitchen to run, converge and verify the content of your nodes. Since this is running in Docker, you'll need **docker.io** installed on your system.

Create and Converge Cowsay with Test Kitchen

Test Kitchen can do each step separately, starting with creating the raw Docker OS image containers, Ubuntu 20.04 and CentOS 7 in this example:

```
$ kitchen create
```

The speed of the creation will vary based on whether you already have the images available locally for Docker to use. When the creation is done, run `kitchen list` again to see that the two systems are indeed available and running. You can also run `docker ps` to see the running containers.

```
Instance             Driver  Provisioner  Verifier  Transport  Last Action  Last Error
default-ubuntu-2004  Dokken  Dokken       Inspec    Dokken     Created      <None>
default-centos-7     Dokken  Dokken       Inspec    Dokken     Created      <None>
```

Next, deploy your code to these container nodes with the `converge` command. Converging applies your code to your container nodes using the Chef Infra client.

```
$ kitchen converge
```

This will take a minute or so, and you'll see output in the terminal that shows the steps Chef is applying from the `recipe/default.rb` file. You're not asking Chef to do much work here, so this will run pretty quickly. If you want to converge just one node (and not both), you can use the instance name with the command, such as:

```
$ kitchen converge default-centos-7
```

Verify the results

When the converge completes, your test node containers are up and running and should now have Cowsay installed, and the profile should be edited the way you want. Run the following:

```
$ kitchen verify
```

This fires the Chef client on each node and the results appear in the terminal, repeated for each platform:

```
  System Package cowsay
     ✓ is expected to be installed
  File /etc/profile
     ✓ content is expected to match /cowsay/
```

This is also a good time to check that the profile on each node is edited and working as expected. Use `kitchen login <instance-name>` to confirm the results:

```
$ kitchen login default-ubuntu-2004
```

```
$ kitchen login default-centos-7
```

If you have just a single node, you can shortcut this command with just `$ kitchen login`.

Clean Up After-the Fact

To delete your images and clean up your Docker environment, run the following to terminate and remove your containers:

```
$ kitchen destroy
```

If you want to skip some of the keystrokes, you can simply run `kitchen test` to run all these steps at once. This simple command destroys any existing instances or containers, creates new ones, converges your code, verifies the results, and destroys them.

Try Test Kitchen Using Vagrant and AWS

Test Kitchen includes drivers for a number of system types, including Vagrant and AWS. The Vagrant driver enables you to spin up test nodes in a desktop virtualization environment like VirtualBox. Vagrant is the default driver, so if you created your Cowsay cookbook without the `-k dokken` flag, you would get a `kitchen.yml` file that looks like this:

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero

verifier:
  name: inspec

platforms:
  - name: ubuntu-20.04
  - name: centos-8

suites:
  - name: default
    verifier:
      inspec_tests:
        - test/integration/default
    attributes:
```

To run this Cowsay example using VirtualBox, just change the default **centos-8** platform entry to **centos-7** and run it. No need to make any changes to your recipe or test files.

The same is true if you want to run your test nodes in the public cloud, such as AWS or Azure. In the next step, you'll create a `kitchen.yml` file that includes a few details not present in either the Dokken (Docker) or Vagrant versions, namely your AWS credentials.

In order to make this work, you'll need an existing AWS account, AWS IAM credentials, and have the aws-cli client installed on your laptop. The AWS CLI enables simple, secure connections between your laptop and the AWS cloud.

Create an AWS IAM user

To create an AWS IAM user, visit the AWS dashboard, click on **Users** and then **Add User**. Create a user with **Programmatic access** and add or create a group with **PowerUserAccess**. When, prompted, download the .csv credentials file. It stores your **Access key ID** and **Secret access key** you'll need in the next step.

Set up AWS credentials

Back on your laptop, run `aws configure` to set the various credentials. When prompted, enter your AWS IAM **Access key ID** and **Secret access key** and enter a default AWS region where your test nodes will be created, such as **us-east-1**. You'll also need to look up availability zones for your region, such as **a**, **b**, etc. with the following command:

```
$ aws ec2 describe-availability-zones --region us-east-1
```

You should also have an AWS ssh key pair handy. If you haven't already, create one for the AWS region in which you're working, such as *us-east-1*. You'll be able to use this pair to shell into your test nodes.

Create an AWS-specific kitchen.yml file

With the connection between your laptop and AWS set, you can now edit your Cowsay `kitchen.yml` file to work with it.

You'll notice a couple key differences, starting with the driver, which is **ec2**, to work with AWS. Next, add an `aws_ssh_key_id` value that matches the name of your ssh key pair. The **region** and **availability zone** should match what you configured with **aws-cli**. The `instance_type` is the type of AWS machine on which you want to test. This example uses a `t2.micro` instance, which is small and available at low cost.

In the **transport** section of your `kitchen.yml` file, add an `ssh_key` entry and the path to your key's .cer or .pem file. Ensure the permissions on the file are `600` for security purposes.

The **verifier** section uses `inspec`, and you may need to install a Chef Ruby gem to make this work. Just run this command as a regular user:

```
$ chef gem install kitchen-verifier-awspec
```

Since you're running both Ubuntu and CentOS instances, you also need to add different default usernames for each. The default for Ubuntu is **ubuntu** and the default for CentOS is **centos**. These can be easily set in the **platforms** section of your `kitchen.yml` file using both the **box** and **transport.username** options.

Your complete `kitchen.yml` file will look something like this:

```
---
driver:
  name: ec2
  aws_ssh_key_id: my-chef-aws
  region: us-east-1
  availability_zone: a
  instance_type: t2.micro
  associate_public_ip: true
  interface: dns

transport:
  ssh_key: /aws/my-chef-aws.cer
  connection_timeout: 10
  connection_retries: 5

verifier:
  name: inspec

platforms:
  - name: ubuntu-20.04
    driver:
      box: ubuntu-20.04
      transport.username: ubuntu

  - name: centos-7
    driver:
      box: centos-7
      transport.username: centos

suites:
  - name: default
    verifier:
      inspec_tests:
```

```
        - test/integration/default
    attributes:
```

Do an initial test of this file with the following:

```
$ kitchen list
```

When you're satisfied, run the following as you did for the Dokken example. All the commands are the same as before:

```
$ kitchen create
```

```
$ kitchen converge
```

```
$ kitchen verify
```

Once all the steps are complete, ssh into your AWS instances by running the same Test Kitchen login command you used for Vagrant or Dokken:

```
$ kitchen login default-centos-7
```

```
 _________________
< 172.31.23.133  >
 -----------------
   \
    \
        .--.
       |o_o |
       |:_/ |
      //   \ \
     (|     | )
    /'\_   _/`\
    \___)=(___/
[centos@ip-172-31-23-133 ~]$
```

Success! Before issuing a `kitchen destroy` command, try making some changes to your recipe and re-running `kitchen converge` to see the results. Each converge will update the contents of your test instances and `kitchen verify` will confirm them.

Next Steps

See all the capabilities of Test Kitchen and learn more about Chef Infra and Chef InSpec and more by visiting chef.io today.

Resources:

Test Kitchen documentation – https://docs.chef.io/workstation/kitchen/

Chef Infra Best Practices eBook - https://d1l5pp53ux74mz.cloudfront.net/docs/default-source/ebook/chef-infra-automation-best-practices.pdf

Chef Twitch Stream - https://www.twitch.tv/chefsoftware

Chef Infra – https://www.chef.io/products/chef-infra

Chef InSpec – https://www.chef.io/products/chef-inspec

For more information on purchasing Chef products, please contact sales@chef.io.

ABOUT CHEF AND PROGESS

Progress (NASDAQ: PRGS) provides the best products to develop, deploy and manage high-impact business applications. Acquired in October 2020, Chef extends Progress offerings in DevOps and DevSecOps, with market-leading, modern infrastructure, compliance, and application by automation. With Progress, you can accelerate the creation and delivery of strategic business applications, automate the process which you configure, deploy and scale those apps, and make your critical data and content more accessible and secure— leading to competitive differentiation and business success. Over 1,700 independent software vendors, 100,000+ enterprise customers, and a three-million-strong developer community rely on Progress to power their applications. Learn about Progress at www.progress.com or +1-800-477-6473.

http://www.chef.io