LEARN CHEF

ChefSpec ¶

Use ChefSpec to simulate the convergence of resources on a node:

- Runs the chef-client on a local machine
- Uses chef-zero or chef-solo
- Is an extension of RSpec, a behavior-driven development (BDD) framework for Ruby
- Is the fastest way to test resources and recipes

ChefSpec is a framework that tests resources and recipes as part of a simulated chef-client run. ChefSpec tests execute very quickly. When used as part of the cookbook authoring workflow, ChefSpec tests are often the first indicator of problems that may exist within a cookbook.

Run ChefSpec ¶

ChefSpec is packaged as part of the Chef development kit. To run ChefSpec:

```
$ chef exec rspec
```

Unit Tests ¶

RSpec is a behavior-driven development (BDD) framework that uses a natural language domain-specific language (DSL) to quickly describe scenarios in which systems are being tested. RSpec allows a scenario to be set up, and then executed. The results are compared to a set of defined expectations.

ChefSpec is built on the RSpec DSL.

Syntax ¶

The syntax of RSpec-based tests should follow the natural language descriptions of RSpec itself. The tests themselves should create an English-like sentence: “The sum of one plus one equals two, and not three.” For example:

```
describe '1 plus 1' do
  it 'equals 2' do
    a = 1
    b = 1
    sum = a + b
    expect(sum).to eq(2)
    expect(sum).not_to eq(3)
  end
end
```

where:

- `describe` creates the testing scenario: `1 plus 1`
- `it` is a block that defines a list of parameters to test, along with parameters that define the expected outcome
- `describe` and `it` should have human readable descriptions: “one plus one equals two”
- `a`, `b`, and `sum` define the testing scenario: `a` equals one, `b` equals one, the `sum` of one plus equals two
- `expect()` defines the expectation: the sum of one plus one equals two—`expect(sum).to eq(2)`—and does not equal three–`expect(sum).not_to eq(3)`
- `.to` tests the results of the test for true; `.not_to` tests the result of the test for false; a test passes when the results of the test are true

context ¶

RSpec-based tests may contain `context` blocks. Use `context` blocks within `describe` blocks to define “tests within tests”. Each `context` block is tested individually. All `context` blocks within a `describe` block must be true for the test to pass. For example:

```
describe 'math' do
  context 'when adding 1 + 1' do
    it 'equals 2' do
      expect(sum).to eq(2)
    end
  end

  context 'when adding 2 + 2' do
    it 'equals 4' do
      expect(sum).to eq(4)
    end
  end
end
```

where each `context` block describes a different testing scenario: “The sum of one plus one to equal two, and also the sum of two plus two to equal four.” A `context` block is useful to handle platform-specific scenarios. For example, “When on platform A, test for foo; when on platform B, test for bar.” For example:

```
describe 'cookbook_name::recipe_name' do

  context 'when on Debian' do
    it 'equals 2' do
      a = 1
      b = 1
      sum = a + b
      expect(sum).to eq(2)
    end
  end

  context 'when on Ubuntu' do
    it 'equals 2' do
      expect(1 + 1).to eq(2)
    end
  end

  context 'when on Windows' do
    it 'equals 3' do
      expect(1 + 2).to eq(3)
    end
  end
```

```
end
```

let ¶

RSpec-based tests may contain `let` statements within a `context` block. Use `let` statements to create a symbol, assign it a value, and then use it elsewhere in the `context` block. For example:

```
describe 'Math' do
  context 'when adding 1 + 1' do
    let(:sum) { 1 + 1 }

    it 'equals 2' do
      expect(sum).to eq(2)
    end
  end

  context 'when adding 2 + 2' do
    let(:sum) do
      2 + 2
    end

    it 'equals 4' do
      expect(sum).to eq(4)
    end
  end
end
```

where:

- The first `let` statement creates the `:sum` symbol, and then assigns it the value of one plus one. The `expect` statement later in the test uses `sum` to test that one plus one equals two
- The second `let` statement creates the `:sum` symbol, and then assigns it the value of two plus two. The `expect` statement later in the test uses `sum` to test that two plus two equals four

Require ChefSpec ¶

A ChefSpec unit test must contain the following statement at the top of the test file:

```
require 'chefspec'
```

Examples ¶

The ChefSpec repo on github has an impressive collection of examples. For all of the core chef-client resources, for guards, attributes, multiple actions, and so on. Take a look at those examples and use them as a starting point for building your own unit tests. Some of them are included below, for reference here.

file Resource ¶

Recipe

```
file '/tmp/explicit_action' do
  action :delete
end

file '/tmp/with_attributes' do
  user 'user'
  group 'group'
  backup false
  action :delete
end

file 'specifying the identity attribute' do
  path   '/tmp/identity_attribute'
 action :delete
end
```

Unit Test

```
require 'chefspec'

describe 'file::delete' do
  let(:chef_run) { ChefSpec::Runner.new.converge(described_recipe) }

  it 'deletes a file with an explicit action' do
    expect(chef_run).to delete_file('/tmp/explicit_action')
    expect(chef_run).to_not delete_file('/tmp/not_explicit_action')
  end

  it 'deletes a file with attributes' do
    expect(chef_run).to delete_file('/tmp/with_attributes').with(backup: false)
    expect(chef_run).to_not delete_file('/tmp/with_attributes').with(backup: true)
  end

  it 'deletes a file when specifying the identity attribute' do
    expect(chef_run).to delete_file('/tmp/identity_attribute')
  end
end
```

template Resource ¶

Recipe

```
template '/tmp/default_action'

template '/tmp/explicit_action' do
  action :create
end

template '/tmp/with_attributes' do
  user 'user'
  group 'group'
  backup false
end

template 'specifying the identity attribute' do
  path '/tmp/identity_attribute'
end
```

Unit Test

```
require 'chefspec'

describe 'template::create' do
  let(:chef_run) { ChefSpec::Runner.new.converge(described_recipe) }

  it 'creates a template with the default action' do
    expect(chef_run).to create_template('/tmp/default_action')
    expect(chef_run).to_not create_template('/tmp/not_default_action')
  end

  it 'creates a template with an explicit action' do
    expect(chef_run).to create_template('/tmp/explicit_action')
  end

  it 'creates a template with attributes' do
    expect(chef_run).to create_template('/tmp/with_attributes').with(
      user: 'user',
      group: 'group',
      backup: false,
    )

    expect(chef_run).to_not create_template('/tmp/with_attributes').with(
      user: 'bacon',
      group: 'fat',
      backup: true,
    )
  end

  it 'creates a template when specifying the identity attribute' do
    expect(chef_run).to create_template('/tmp/identity_attribute')
  end
end
```

package Resource ¶

Recipe

```
package 'explicit_action' do
  action :remove
end

package 'with_attributes' do
  version '1.0.0'
  action :remove
end

package 'specifying the identity attribute' do
  package_name 'identity_attribute'
  action :remove
end
```

Unit Test

```
require 'chefspec'

describe 'package::remove' do
  let(:chef_run) { ChefSpec::Runner.new.converge(described_recipe) }
```

```
  it 'removes a package with an explicit action' do
    expect(chef_run).to remove_package('explicit_action')
    expect(chef_run).to_not remove_package('not_explicit_action')
  end

  it 'removes a package with attributes' do
    expect(chef_run).to remove_package('with_attributes').with(version: '1.0.0')
    expect(chef_run).to_not remove_package('with_attributes').with(version: '1.2.3')
  end

  it 'removes a package when specifying the identity attribute' do
    expect(chef_run).to remove_package('identity_attribute')
  end
end
```

chef_gem Resource ¶

Recipe

```
chef_gem 'default_action'

chef_gem 'explicit_action' do
  action :install
end

chef_gem 'with_attributes' do
  version '1.0.0'
end

chef_gem 'specifying the identity attribute' do
  package_name 'identity_attribute'
end
```

Unit Test

```
require 'chefspec'

describe 'chef_gem::install' do
  let(:chef_run) { ChefSpec::Runner.new.converge(described_recipe) }

  it 'installs a chef_gem with the default action' do
    expect(chef_run).to install_chef_gem('default_action')
    expect(chef_run).to_not install_chef_gem('not_default_action')
  end

  it 'installs a chef_gem with an explicit action' do
    expect(chef_run).to install_chef_gem('explicit_action')
  end

  it 'installs a chef_gem with attributes' do
    expect(chef_run).to install_chef_gem('with_attributes').with(version: '1.0.0')
    expect(chef_run).to_not install_chef_gem('with_attributes').with(version: '1.2.3')
  end

  it 'installs a chef_gem when specifying the identity attribute' do
    expect(chef_run).to install_chef_gem('identity_attribute')
  end
end
```

directory Resource ¶

Recipe

```
directory '/tmp/default_action'

directory '/tmp/explicit_action' do
  action :create
end

directory '/tmp/with_attributes' do
  user 'user'
  group 'group'
end

directory 'specifying the identity attribute' do
  path '/tmp/identity_attribute'
end
```

Unit Test

```
require 'chefspec'

describe 'directory::create' do
  let(:chef_run) { ChefSpec::Runner.new.converge(described_recipe) }

  it 'creates a directory with the default action' do
    expect(chef_run).to create_directory('/tmp/default_action')
    expect(chef_run).to_not create_directory('/tmp/not_default_action')
  end

  it 'creates a directory with an explicit action' do
    expect(chef_run).to create_directory('/tmp/explicit_action')
  end

  it 'creates a directory with attributes' do
    expect(chef_run).to create_directory('/tmp/with_attributes').with(
      user: 'user',
      group: 'group',
    )

    expect(chef_run).to_not create_directory('/tmp/with_attributes').with(
      user: 'bacon',
      group: 'fat',
    )
  end

  it 'creates a directory when specifying the identity attribute' do
    expect(chef_run).to create_directory('/tmp/identity_attribute')
  end
end
```

Guards ¶

Recipe

```
service 'true_guard' do
  action  :start
```

```
  only_if { 1 == 1 }
end

service 'false_guard' do
  action :start
  not_if { 1 == 1 }
end

service 'action_nothing_guard' do
  action :nothing
end
```

Unit Test

```
require 'chefspec'

describe 'guards::default' do
  let(:chef_run) { ChefSpec::Runner.new.converge(described_recipe) }

  it 'includes resource that have guards that evalute to true' do
    expect(chef_run).to start_service('true_guard')
  end

  it 'excludes resources that have guards evaluated to false' do
    expect(chef_run).to_not start_service('false_guard')
  end

  it 'excludes resource that have action :nothing' do
    expect(chef_run).to_not start_service('action_nothing_guard')
  end
end
```

include_recipe Method ¶

Recipe

```
include_recipe 'include_recipe::other'
```

Unit Test

```
require 'chefspec'

describe 'include_recipe::default' do
  let(:chef_run) { ChefSpec::Runner.new.converge(described_recipe) }

  it 'includes the `other` recipe' do
    expect(chef_run).to include_recipe('include_recipe::other')
  end

  it 'does not include the `not` recipe' do
    expect(chef_run).to_not include_recipe('include_recipe::not')
  end
end
```

Multiple Actions ¶

Recipe

```
service 'resource' do
  action :start
end

service 'resource' do
  action :nothing
end
```

Unit Test

```
require 'chefspec'

describe 'multiple_actions::sequential' do
  let(:chef_run) { ChefSpec::Runner.new(log_level: :fatal).converge(described_recipe) }

  it 'executes both actions' do
    expect(chef_run).to start_service('resource')
  end

  it 'does not match other actions' do
    expect(chef_run).to_not disable_service('resource')
  end
end
```

For more information ... ¶

For more information about ChefSpec:

- ChefSpec Github Repo