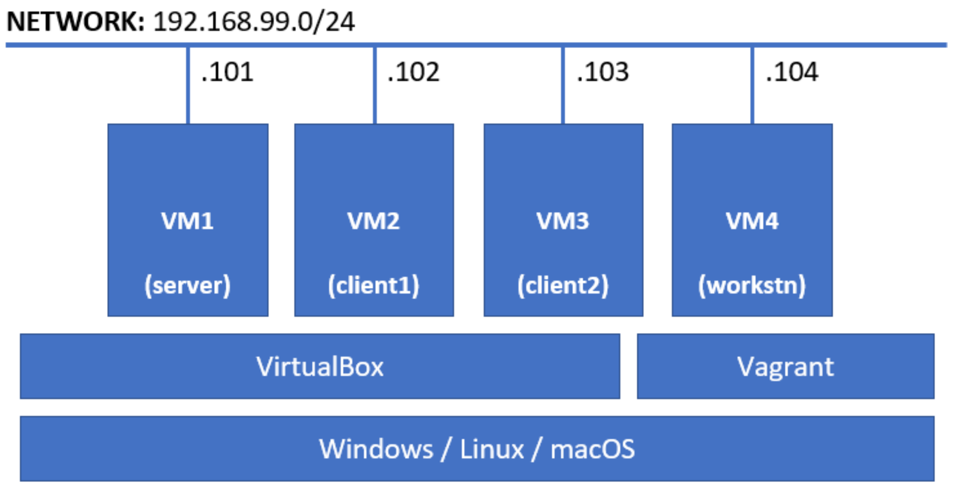
Practice M4: Chef

For this practice, our lab environment will look like this



We are going to use mostly **CentOS Stream 9** boxes and at least one **Debian**-based box

All configurations and supplementary files are provided as a ZIP archive and can be downloaded from the module section in the official site

Part 1

First, bring up the environment using the provided **Vagrantfile**

Install Chef Server

The procedure is described here: https://docs.chef.io/server/install_server/

Preparation

Before we start, we should take care for a few preparation steps

This set of tasks **must be repeated on the client nodes as well**

On CentOS

Install and activate **NTP** solution (for example **Chrony**):

```
sudo dnf install -y chrony
sudo systemctl enable chronyd
sudo systemctl start chronyd
```

Set **SELinux** to permissive mode for the current session

```
sudo setenforce permissive
```

And for the next boot

```
sudo sed -i 's\=enforcing\=permissive\g' /etc/sysconfig/selinux
```

On Debian/Ubuntu

Install **NTP** server (it will be automatically enabled and started):

```
sudo apt-get install -y ntp
```

Actual Installation

In order to install a stand-alone **Chef server** we must follow these steps

On CentOS

Download the package

```
wget -P /tmp https://packages.chef.io/files/stable/chef-server/15.6.2/el/8/chef-server-core-15.6.2-1.el8.x86_64.rpm
```

Install the package

```
sudo rpm -Uvh /tmp/chef-server-core-15.6.2-1.el8.x86_64.rpm
```

On Debian/Ubuntu

In order to install a stand-alone **Chef server** on **Debian/Ubuntu 18.04+**, we must follow these steps:

Download the package

```
wget -P /tmp https://packages.chef.io/files/stable/chef-server/15.6.2/ubuntu/22.04/chef-server-core_15.6.2-1_amd64.deb
```

Install the package:

```
sudo dpkg -i /tmp/chef-server-core_15.6.2-1_amd64.deb
```

Post Installation

Configure the services

```
sudo chef-server-ctl reconfigure
```

When asked to accept the licenses, answer with **yes**

After a while, around 5 minutes, the installation will complete

Create administrator user

```
sudo chef-server-ctl user-create chefadmin Chef Admin chefadmin@do2.lab 'Password1' --filename /home/vagrant/chefadmin.pem
```

Create an organization

```
sudo chef-server-ctl org-create demo-org 'Demo Org.' --association_user chefadmin --filename /home/vagrant/demoorg-validator.pem
```

One more post-installation step is necessary – ports **80/tcp** and **443/tcp** must be open

If we are running **firewalld**, then we can execute the following

```
sudo firewall-cmd --add-port=80/tcp --permanent
sudo firewall-cmd --add-port=443/tcp --permanent
sudo firewall-cmd --reload
```

More details on the firewall configuration here: https://docs.chef.io/server/server_firewalls_and_ports/

Post Installation (additional)

We can install an additional component like the web management console

*Please note that this addon is being deprecated in favor of **Chef Automation** product*

To install the web management console, execute

```
sudo chef-server-ctl install chef-manage
```

Then reconfigure the server with

```
sudo chef-server-ctl reconfigure
```

And then the configure the management tools with

```
sudo chef-manage-ctl reconfigure
```

Now, we can open a browser tab and navigate to **https://192.168.99.101**

Use the credentials you created earlier – user **chefadmin** with password set to **Password1**

Nothing to see here, yet

Install Chef Workstation

Switch to the **workstation** machine

Preparation

Execute all the pre-requisite tasks (**NTP** and **SELinux** if running on CentOS) first

Installation

Full process is described here: https://docs.chef.io/workstation/install_workstation/

On CentOS

Download the package

```
wget -P /tmp https://packages.chef.io/files/stable/chef-
workstation/23.4.1032/el/8/chef-workstation-23.4.1032-1.el8.x86_64.rpm
```

Install the package

```
sudo rpm -Uvh /tmp/chef-workstation-23.4.1032-1.el8.x86_64.rpm
```

Most likely we will need a git client installed, so let's do it

```
sudo dnf install -y git
```

On Debian/Ubuntu

Download the package

```
wget -P /tmp https://packages.chef.io/files/stable/chef-
workstation/23.4.1032/ubuntu/22.04/chef-workstation_23.4.1032-1_amd64.deb
```

Install the package

```
sudo dpkg -i /tmp/chef-workstation_23.4.1032-1_amd64.deb
```

Most likely we will need a git client installed, so let's do it

```
sudo apt-get install -y git
```

Post Installation

We can check that the installation went fine with

```
chef -v
```

Now, we must further configure the workstation

Check the **Ruby** environment

```
which ruby
```

Switch to the **Ruby** version provided by **Chef**

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

Modify the **PATH** variable:

```
echo 'export PATH="/opt/chef-workstation/embedded/bin:$PATH"' >> ~/.bash_profile &&
source ~/.bash_profile
```

Check that ruby is setup correctly

```
which ruby
```

Local Working Environment

Next step is to create the local working environment

There are multiple ways to achieve this, but if we have installed the web console, then we can do the following:

Login to the console and go to **Administration**, then select the organization (**demo-org**), click on **Starter Kit**

Finally, click on **Download Starter Kit**

When asked, confirm by clicking on **Proceed**

Now move the archive to the workstation machine

```
scp chef-starter.zip vagrant@192.168.99.104:.
```

Then return on the workstation machine

And unzip the file (you may need to install the unzip tool)

```
unzip chef-starter.zip
```

Check the extracted files with tree (you may need to install it)

```
tree .
```

Then go to the repository directory

```
cd chef-repo
```

Now get the certificates from the server (based on the configuration in the **.chef** folder)

```
knife ssl fetch
```

List all known nodes

```
knife client list
```

Optionally add the workstation to the server

```
knife bootstrap 192.168.99.104 -N workstation -U vagrant -P vagrant --sudo
```

Accept the licenses and then confirm the host authenticity

If we list again all known nodes

```
knife client list
```

We will notice that the workstation is there

Install Chef Nodes

Installation process on the nodes can be done in multiple ways

The simplest and easiest method is to use the **knife bootstrap** option

In order to use this method, we must open a session to the **workstation** and execute a set of steps

Make sure we are in the **chef-repo** folder created earlier

To join the first client, we must execute

```
knife bootstrap 192.168.99.102 -N client-1 -U vagrant -P vagrant --sudo
```

Confirm the host authenticity

And a similar command for the second client

```
knife bootstrap 192.168.99.103 -N client-2 -U vagrant -P vagrant --sudo
```

Again, confirm the host authenticity

Check if the node is recognized by the server

```
knife client show client-2
```

Get the list of all nodes

```
knife client list
```

Go to the web interface and check if all the nodes are there

Local Test

Let's imagine for a moment that we have just the workstation and no server or nodes

We can create and execute configurations locally

For this we will use the **chef-solo** utility

Make sure that you are in the home folder of the current user

Create a folder

```
mkdir cookbooks
```

And navigate to it

```
cd cookbooks
```

If you haven't configured the git client yet (as this is new machine it is not configured), do it by executing

```
git config --global user.email "<YOUR-EMAIL-ADDRESS>"
git config --global user.name "<YOUR-NAME>"
```

Then, create an empty recipe with the following command

```
chef generate cookbook test
```

If this is the first time you execute the chef utility, you should accept the license

Now, we have our first and empty cookbook

Check the file hierarchy with

```
tree test
```

Plenty of files there. We are interested in the **default.rb** file as it will contain the body of our first cookbook

Open it for editing

```
vi test/recipes/default.rb
```

And enter the following

```
file '/tmp/file.txt' do
  content 'This file is created with Chef'
end
```

Save and close the file

When executed it will create a simple text file

Let's see it in action

Return to the home folder

Create a simple configuration file named **solo.rb** (the name could be different) with the following content

```
file_cache_path "/home/vagrant/cache"
cookbook_path "/home/vagrant/cookbooks"
```

It will instruct chef-solo where to look for files

Now, create another one. This time, name it **solo.json** (the name could be different) and place there the following

```
{
  "run_list": [ "recipe[test]" ]
}
```

This creates a list of recipes to be executed. In our case, it is a single recipe coming from a single cookbook

Now, let's execute it

```
chef-solo -c solo.rb -j solo.json
```

After a while it will finish. Let's check the file

```
cat /tmp/file.txt
```

Yes, it is there and contains what we wanted

If we re-execute the recipe, no changes will be made

Now, change the file by adding extra content to it

```
echo 'some more text' >> /tmp/file.txt
```

Let's test again what will happen if we run the recipe but this time in dry-run (why-run) mode

```
chef-solo -c solo.rb -j solo.json -W
```

It will be applied as our target file has been changed

Check the file

```
cat /tmp/file.txt
```

It is not changed

Now, execute it in normal mode

```
chef-solo -c solo.rb -j solo.json
```

If we check again, the file will be with its original content

Ad-hoc Actions

We can use **chef-run** instead of **chef-solo**

Remove the **/tmp/file.txt** file

```
rm /tmp/file.txt
```

Now, being in the **cookbooks/test** folder, execute the following

chef-run localhost ./recipes/default.rb --user vagrant --password vagrant

When it finishes, check for the **/tmp/file.txt** file

```
cat /tmp/file.txt
```

It should be there

The same can be executed against remote machine

We can even skip the recipe part and specify directly what we want to be executed

```
chef-run localhost file /tmp/test.txt content='ad-hoc test' --user vagrant --password vagrant
```

Then check again

```
cat /tmp/test.txt
```

Again, this can be executed against a remote host

Starter Recipe/Cookbook

Let's make our first steps in the kitchen but this time utilizing the whole infrastructure

Log on to the workstation (if you closed the session) and go to the **cookbooks** folder in the repository folder

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

Execute the tree command if available (if not install it)

```
tree .
```

It appears that there is a **starter** cookbook with one recipe (**default.rb**) in it, let's explore it

```
cat starter/recipes/default.rb
```

Now, let's run it locally

```
chef-client --local-mode --override-runlist starter
```

It works. We should be able to see the message ***"INFO: Welcome to Chef Infra, Sam Doe!"***

Where this message came from?

If we open the **starter/recipes/default.rb** file

```
vi starter/recipes/default.rb
```

We will see the **log** resource reading from some kind of variable (we will come back to this later)

Now, let's open the **starter/attributes/default.rb** file

```
vi starter/attributes/default.rb
```

And change the name from **Sam Doe** to **Joe Black**

Save and close the file

Run again the cookbook locally

```
chef-client --local-mode --override-runlist starter
```

Now, the greeting is different

In order other nodes to be able to run this cookbook, we must upload it to the server

```
knife cookbook upload starter
```

Now, we can ask for the list of available cookbooks

```
knife cookbook list
```

Let's assign the cookbook to **client #1**

```
knife node run_list add client-1 "recipe[starter]"
```

Now, go to **client #1** and execute

```
sudo chef-client
```

You should see the message from the previous step

To see the same on the other client, we must return to the **workstation** and assign the cookbook to it

```
knife node run_list add client-2 "recipe[starter]"
```

Then go to **client #2** and execute

```
sudo chef-client
```

The message from the previous step will appear again

Part 2

We continue with the same environment from ***Part 1***

Make sure that all machines are up and running and you are on the **workstation**

If the git client is not set up, you must do it

```
git config --global user.email "<YOUR-EMAIL-ADDRESS>"
git config --global user.name "<YOUR-NAME>"
```

A Cookbook with Recipe(s)

Let's create one simple cookbook, that will put two text files on the target station

Go to the **cookbooks** folder of the repository

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

And execute

```
chef generate cookbook demo_cookbook
```

Explore the newly created set of folders and files

```
tree demo_cookbook
```

Open the **demo_cookbook/recipes/default.rb** for editing

```
vi demo_cookbook/recipes/default.rb
```

And type in the following

```
file "#{ENV['HOME']}/readme.txt" do
  content 'Hello from Chef!'
end
file '/tmp/readme.txt' do
  content 'Chef was here as well :)'
end
```

Save the file and exit

The firs block will create a file in the home folder of the executing user and the second in the **/tmp** folder

Execute it locally in test mode with

```
chef-client --local-mode --override-runlist demo_cookbook -W
```

Indeed, the changes would be just as expected

Now, execute it in normal mode

```
chef-client --local-mode --override-runlist demo_cookbook
```

Check the contents of both files

```
cat ~/readme.txt
cat /tmp/readme.txt
```

Remove one of them and re-execute the cookbook

```
rm ~/readme.txt
chef-client --local-mode --override-runlist demo_cookbook
```

Just the missing file gets re-created

Now, let's add one more recipe in the cookbook

This time we will create two users

We need one preparation step – generate an encrypted password

```
openssl passwd -1 "Password1"
```

Copy the resulting string

Create a new file **users.rb**

```
vi demo_cookbook/recipes/users.rb
```

And enter the following

```
user 'demo-user-1' do
  comment 'Demo user #1'
  manage_home true
  shell '/bin/bash'
  password '<RESULT-FROM-OPENSSL>'
end
```

```
user 'another user' do
  username 'demo-user-2'
  comment 'Demo user #2'
  manage_home true
  shell '/bin/bash'
  password '<RESULT-FROM-OPENSSL>'
end
```

Pay attention to the additional **username** clause in the second block

Save and exit

Now execute locally together with the previous recipe

```
sudo chef-client --local-mode --override-runlist demo_cookbook,demo_cookbook::users
```

The extended version of the above would be (skip it)

```
sudo chef-client --local-mode --override-runlist
demo_cookbook::default,demo_cookbook::users
```

Check that users are there

```
tail /etc/passwd
```

If we want, we can upload the cookbook

```
knife cookbook upload demo_cookbook
```

And eventually assign it (in fact the two recipes) as a run list to the **client #2**

```
knife node run_list add client-2 "recipe[demo_cookbook], recipe[demo_cookbook::users]"
```

If you added the starter cookbook on this node, you would see a list of three recipes

Should you want, you can remove the extra one with this command

```
knife node run_list remove client-2 "recipe[starter]"
```

Now, go to **client #2** and execute

```
sudo chef-client
```

We should see both the two files (one of them will be in **/root**) and the two users

```
sudo cat /root/readme.txt
cat /tmp/readme.txt
tail /etc/passwd
```

Attributes

Being on the **workstation**, and in the **cookbooks** repository folder (**chef-repo**), create an empty cookbook

chef generate cookbook attrib

Navigate to the folder (**attrib**) and check the resulting files with

```
tree .
```

We may clean up a bit

```
rm -rf compliance
```

```
rm -rf test
```

Create a folder

```
mkdir attributes
```

And create a file to hold them

```
vi attributes/default.rb
```

Enter the following information

```
default['greeting'] = 'Joe Black'
```

Save and close the file

Open the **recipes/default.rb** file for editing

vi recipes/default.rb

And enter the following

```
log "Test #1: Hello, #{node["greeting"]}!" do
  level :info
end
```

Save and close the file

Test the cookbook locally

```
chef-client --local-mode --override-runlist attrib
```

Everything looks like we expect

Now, open the recipe again

```
vi recipes/default.rb
```

And add this block at the end

```
log "Test #2: Hello, #{node.default["greeting"]}!" do
  level :info
end
```

Save and close the file

Test the cookbook locally

```
chef-client --local-mode --override-runlist attrib
```

No changes, everything looks like we expect

It appears that ***node[attribute]*** is the same as ***node.default[attribute]*** (at least for now)

Now, add new attribute file

```
vi attributes/additional.rb
```

And enter the following

```
normal['greeting'] = 'Joe Blue'
```

Save and close the file

Test the cookbook locally

```
chef-client --local-mode --override-runlist attrib
```

Hmm, something changed

The attribute value that was defined as ***normal*** overwrote the one defined as ***default***

Now, open the recipe again

```
vi recipes/default.rb
```

And add this block at the end to display the ***normal*** value separately

```
log "Test #3: Hello, #{node.normal["greeting"]}!" do
  level :info
end
```

Save and close the file

Test the cookbook locally

```
chef-client --local-mode --override-runlist attrib
```

Okay, now we see and have a basic understanding of attributes and their precedence

Note that the basic set of attribute files could be generated with

```
chef generate template PATH_TO_COOKBOOK ATTRIBUTE_NAME
```

Which in our case (if we are in the **~/chef-repo/cookbooks** folder) should be

```
chef generate attribute attrib default
```

Templates

Being on the **workstation**, navigate to **~/chef-repo/cookbooks** folder

Create a new cookbook

```
chef generate cookbook tmpl
```

Navigate to the folder (**tmpl**) and check the resulting files with

```
tree .
```

We may clean up a bit

```
rm -rf compliance
rm -rf test
```

This time, we will use a command to prepare the additional folders

First, create an attribute file

```
chef generate attribute . default
```

Now, open the file for editing

```
vi attributes/default.rb
```

And add the following

```
default['new_files']['/tmp/file1'] = 'AAAA'
default['new_files']['/tmp/file2'] = 'BBBB'
```

```
default['new_files']['/tmp/file3'] = 'CCCC'
```

Save and close the file

Let's create the recipe

```
vi recipes/default.rb
```

We want to create a number of files coming from the attribute list we just created

Of course, we can have three separate calls to the **file resource**, but we do not want to go this way

Instead, we will **loop over the list**

Enter the following

```
node['new_files']&.each do |name, val|
  file "#{name}" do
    action :create
    content "#{val}"
  end
end
```

Save and close the file

Test the cookbook locally

```
chef-client --local-mode --override-runlist tmpl -W
```

It appears that it will work

Imagine that we want to produce a report of what we did and store it on every node

We can do this by using a template

Now, we can prepare the template folder and file

The command general structure is like this

chef generate template *PATH_TO_COOKBOOK TEMPLATE_NAME*

Assuming that we are in the cookbook's folder (**~/chef-repo/cookbooks/tmpl**) we can execute

```
chef generate template . report
```

If we check the folder structure again, we will notice the **templates/report.erb** folder and file

Open it for editing

```
vi templates/report.erb
```

And enter the following

```
Report for node <%= node['fqdn'] %>
List of created files:
<% @new_files.each do |newfile| -%>
<%= newfile %>
<% end -%>
```

Save and close the file

Now extend the recipe to prepare and create the report file for every node

```
vi recipes/default.rb
```

Add the following block at the end

```
template '/tmp/report.txt' do
  source 'report.erb'
  variables(new_files: node['new_files'])
end
```

Save and close the file

Test the cookbook locally

```
chef-client --local-mode --override-runlist tmpl -W
```

It appears that it will work

Execute it in normal mode

```
chef-client --local-mode --override-runlist tmpl
```

Check the result for the **/tmp/report.txt** file

```
cat /tmp/report.txt
```

It is not the best possible output

Open the template file

```
vi templates/report.erb
```

And change it to match the following

```
Report for node <%= node['fqdn'] %>
List of created files:
<% @new_files&.each do |newfile_name, newfile_value| -%>
<%= newfile_name %> => <%= newfile_value %>
<% end -%>
```

Save and close the file

Execute again the cookbook locally

```
chef-client --local-mode --override-runlist tmpl
```

Check again the result

```
cat /tmp/report.txt
```

This time it looks much better

Files

Let's extend the previous cookbook by adding a static file that will go to the nodes unchanged

Create a folder **files**

```
mkdir files
```

Now create a file **files/index.html**

```
vi files/index.html
```

With the following content

```
<h1>Static index.html file</h1>
```

Save and close it

Open the default recipe

```
vi recipes/default.rb
```

And change it by adding one more block at the end with the following content

```
cookbook_file '/var/www/html/index.html' do
  source 'index.html'
  owner 'root'
  group 'root'
  mode '0755'
  action :create
end
```

Save and close the file

Test the cookbook locally

```
chef-client --local-mode --override-runlist tmpl -W
```

It looks just fine, if you like execute it in normal mode as well

Flexible Cookbook

Again, it is time to create another cookbook

It will install and run an **Apache** web server, and create a custom index file on both client nodes

Being on the workstation, and in the **cookbooks** repository folder (**chef-repo**), create an empty cookbook

```
chef generate cookbook apache
```

Open the default recipe for editing

```
vi apache/recipes/default.rb
```

And enter the following set of instructions

```
if node['platform_family'] == 'debian'
  vpackage = 'apache2'
else
  vpackage = 'httpd'
end
package 'Install Apache web server' do
  package_name "#{vpackage}"
end
service 'Start and Enable Apache web server' do
```

```
  service_name "#{vpackage}"
  action [ :enable, :start ]
end
file 'Create custom index.html file' do
  path '/var/www/html/index.html'
  content "<h1>Hello Chef World!</h1><br /><hr /><h5>Running on %{p}</h5>" % {p: node['platform_family']}
end
```

We can do a dry run and test it locally

```
sudo chef-client --local-mode --override-runlist apache --why-run
```

Then we can upload the cookbook

```
knife cookbook upload apache
```

Now we can attach our latest cookbook to the two client nodes

This can be done in the **Web UI** or on the terminal

Graphically is done via **Nodes > Edit Runlist**

On the terminal it is done with the following commands

```
knife node run_list add client-1 "recipe[apache]"
knife node run_list add client-2 "recipe[apache]"
```

Then go to each node and execute the **chef** client

```
sudo chef-client
```

Check the results

Of course, there is an option to start the client software as daemon

The easiest way, just for testing purposes, for the current lab and session is to execute

```
sudo chef-client -i 60 -d
```

Where **60** is the interval in seconds for the communication with the server, and **-d** is instruction to demonize the service

Should something go not according to the plan, we can execute this instead

```
sudo chef-client -i 60 -l info
```

And monitor what exactly is going on

Part 3

Let's continue with our experiments

Custom Resource

Let's turn our last cookbook to a custom resource

Being on the **workstation**, and in the **cookbooks** repository folder (**chef-repo**)

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

Create an empty cookbook

```
chef generate cookbook webserver
```

Navigate to the folder (**webserver**)

```
cd webserver
```

And check the resulting files with

```
tree .
```

We may clean up a bit (but this time with a single command)

```
rm -rf compliance test
```

We can create the required folders and files for a custom resource with a command like this

```
chef generate resource PATH_TO_COOKBOOK RESOURCE_NAME
```

As we are in the cookbook's folder, we will execute this

```
chef generate resource . site
```

Now, open the file for editing

```
vi resources/site.rb
```

And add the following

```
provides :site
if node['platform_family'] == 'debian'
  vpackage = 'apache2'
else
  vpackage = 'httpd'
end
property :homepage, String, default: "<h1>Hello #{vpackage}!</h1>"
action :create do
  package "#{vpackage}"
  service "#{vpackage}" do
    action [:enable, :start]
  end
  file '/var/www/html/index.html' do
    content new_resource.homepage
  end
end
action :delete do
  package "#{vpackage}" do
    action :remove
  end
  file '/var/www/html/index.html' do
    action :delete
  end
end
```

Save and close the file

Now open the default recipe

```
vi recipes/default.rb
```

And make sure that the content matches this

```
site 'custom-site'
```

Save and close the file

We can do a dry run and test it locally

```
sudo chef-client --local-mode --override-runlist webserver --why-run
```

It should work

We can extend the content of the default recipe to this

```
site 'custom-site' do
  action :create
end
```

It will have the same effect as the shorter version

Library

Now, let's create a helper library that we will use in a recipe

The idea it borrowed from here https://misctechmusings.com/chef-helper-library

Our recipe will create a set of files based on attributes list in a file

The helper library will be used to filter out some of the files and skip their creation

Being on the **workstation**, and in the **cookbooks** repository folder (**chef-repo**)

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

Create an empty cookbook

```
chef generate cookbook datafiles
```

Navigate to the folder (**datafiles**)

```
cd datafiles
```

And check the resulting files with

```
tree .
```

We may clean up a bit

```
rm -rf compliance test
```

First, create the attribute file with

```
chef generate attribute default
```

Open the file

```
vi attributes/default.rb
```

And add the following text

```
default['data_files']['/tmp/data1.dat'] = '23-12-2022'
default['data_files']['/tmp/data2.dat'] = '24-12-2022'
default['data_files']['/tmp/data3.dat'] = '25-12-2022'
default['data_files']['/tmp/data4.dat'] = '26-12-2022'
default['data_files']['/tmp/data5.dat'] = '27-12-2022'
default['data_files']['/tmp/data6.dat'] = '28-12-2022'
default['data_files']['/tmp/data7.dat'] = '29-12-2022'
```

Save and close the file

Next, create a file that contains the non-working days

```
vi /tmp/non-working.txt
```

And add the following

```
24-12-2022
25-12-2022
26-12-2022
27-12-2022
28-12-2022
```

Save and close the file

Let's create a folder to hold our helper library

```
mkdir libraries
```

And open an empty file there

```
vi libraries/mylib_helper.rb
```

Enter the following code

```
module MyLib
  module Helper
    def in_file?(dt)
      file = '/tmp/non-working.txt'
      if File.exist?(file)
        File.readlines(file).grep(/#{dt}/).any?
      end
    end
 end
end
```

```
Chef::Resource::File.send(:include, MyLib::Helper)
```

Save and close the file

Open the default recipe for editing

```
vi recipes/default.rb
```

Enter the following

```
node['data_files']&.each do |name, val|
  file "#{name}" do
    action :create
    content "#{val}"
    not_if { in_file?(val) }
  end
end
```

Save and close the wile

We can do a dry run and test it locally

```
sudo chef-client --local-mode --override-runlist datafiles --why-run
```

Or use the short version of the options

```
sudo chef-client -z -o datafiles -W
```

Both will result in the same – a successful execution

It appears that only files 1 and 7 will be created because the rest are for non-working days

We can either change the values, or execute the recipe in normal mode, or skip and continue

Test

Let's see how we can test our recipes not only tunning them in dry-run (or why-run) mode

We can use different drivers and target platforms but for this exercise we will use **Docker**

Make sure you have **Docker** installed on the **workstation** machine (you can refer to the files in the practice)

Next step would be to install the required development packages

On **Red Hat** based distributions this could be done with

```
sudo dnf groupinstall 'Development Tools'
```

And on **Debian** based distributions with

```
sudo apt-get install build-essential
```

Once done, we can enter the **demo_cookbook** folder

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

Here, we can create a **Gemfile**

```
vi Gemfile
```

With the following content

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

```
gem "chef"
gem "berkshelf"
gem "test-kitchen"
gem "kitchen-docker"
gem "kitchen-inspec"
```

Then, we can execute

```
bundle install
```

To install the required components

Once done, we should prepare a **kitchen.yml** file which is used to control the test environment

There is one in the folder already. Open it for editing

```
vi kitchen.yml
```

Make sure that the content of the file matches the following

```
---
driver:
  name: docker

provisioner:
  name: chef_infra

verifier:
  name: inspec

platforms:
  - name: ubuntu-22.04
    driver_config:
      image: ubuntu:22.04
      platform: ubuntu
  - name: opensuse-leap-15.4
    driver_config:
      image: opensuse/leap:15.4
      platform: opensuse

suites:
  - name: default
    verifier:
```

```
    inspec_tests:
      - test/integration/default
```

Save and close the file

With this file, we set the following:

- We will use the **Docker** driver
- Two platforms to test against – **Ubuntu** and **openSUSE**
- One test suite with **InSpec** as verifier
- And for provisioner, we will use **Chef Infra** (**chef_infra** which in older versions was **chef_zero**)

A Test Kitchen **Instance** is a combination of a **Suite** and a **Platform** as laid out in our file

We can see the instances when we ask for their list with

```
kitchen list
```

As we can see, we have two and none of them is created. We can do it with

```
kitchen create
```

After a while, our instances will be created

And if ask for the list again

```
kitchen list
```

We will see them

We can even check with the Docker CLI

```
docker container ls
```

Explore the **recipes/default.rb** file to refresh our memory what we did there

```
cat recipes/default.rb
```

It appears that we have two recipes, but we will continue with the default one

Let's send and execute the recipe on the two instances

```
kitchen converge
```

After a while, the recipe's execution will finish

Now, we either browse the feedback on the screen or we can enter in each container and investigate if everything went fine

How can we enter and check?

Execute the following command to enter the **openSUSE** one

```
kitchen login default-opensuse-leap-154
```

Once in, we can check the contents of the two files

```
cat /tmp/readme.txt
```

```
sudo cat /root/readme.txt
```

Everything seems to be fine

Close the session to the container

```
exit
```

Of course, we can do the check by preparing a test and running it against the instances

Open the sample test file for editing

```
vi test/integration/default/default_test.rb
```

Enter the following

```
# First file
describe file("/root/readme.txt") do
  it { should exist }
  its('content') { should match "Hello from Chef!" }
end
# Second file
describe file("/tmp/readme.txt") do
  it { should exist }
  its('content') { should match "Chef was here as well :)" }
end
```

Save and close the file

We are testing not if each file exists but also what it contains

Now, run the tests with

```
kitchen verify
```

One of the tests fails for one of the instances

We can see this in shorter form with this command

```
kitchen list
```

Let's enter the **Ubuntu** instance

```
kitchen login default-ubuntu-2204
```

And see where the files are

```
find / -type f -name readme.txt 2> /dev/null
```

Aha, one of the files (the one which path depends on the environment variable) does not appear where we expect

So, we should fix our test

Close the session

```
exit
```

And open the test script

```
vi test/integration/default/default_test.rb
```

And change the first block/check to match this

```
describe file("#{ENV['HOME']}/readme.txt") do
 it { should exist }
  its('content') { should match "Hello from Chef!" }
```

```
end
```

Save and close the file

Now, run the tests with

```
kitchen verify
```

Hmm, it appears that the same test fails but this time for both instances

If we check the error message, we will notice that the path where it looks for the file points to a path on the workstation (check with **echo $HOME**)

Okay, open the file again

```
vi test/integration/default/default_test.rb
```

And change the first block to match this

```
if os.debian?
  describe file("/home/kitchen/readme.txt") do
    it { should exist }
    its('content') { should match "Hello from Chef!" }
  end
elsif os.suse?
  describe file("/root/readme.txt") do
    it { should exist }
    its('content') { should match "Hello from Chef!" }
  end
end
```

Save and close the file

Now, run the tests with

```
kitchen verify
```

Finally, both tests succeeded on both instances

Let's go on the next level and test a new command

```
kitchen test
```

As we can see, it takes longer but executes all steps that we did manually

The result should be the same with the main difference that after all is done, no instances will be left working

We can check with

```
kitchen list
```

Or with

```
docker container ls -a
```

That is all for this practice 😊