# **Using Chef Solo on Linux for AWS**

**Logging in to the Amazon Web Services Console**

**Instructions**

1. To start the Lab experience, open the Amazon Console by clicking this button:

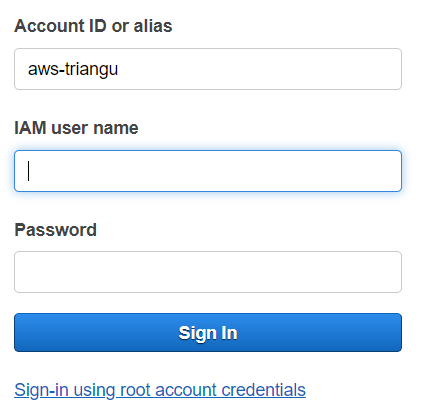
<https://aws-triangu.signin.aws.amazon.com/console>

2. Enter the following credentials created just for your Lab session, and click **Sign In**:

* **Account ID or alias**: Keep the pre-populated value
* **IAM user name**: *student{1-10}*
* **Password**: *XXXXXXXXX*
* **Signing-in AWS Management Console**

We are using URL

<https://aws-triangu.signin.aws.amazon.com/console>

After that we will be redirected to AWS Login page: 

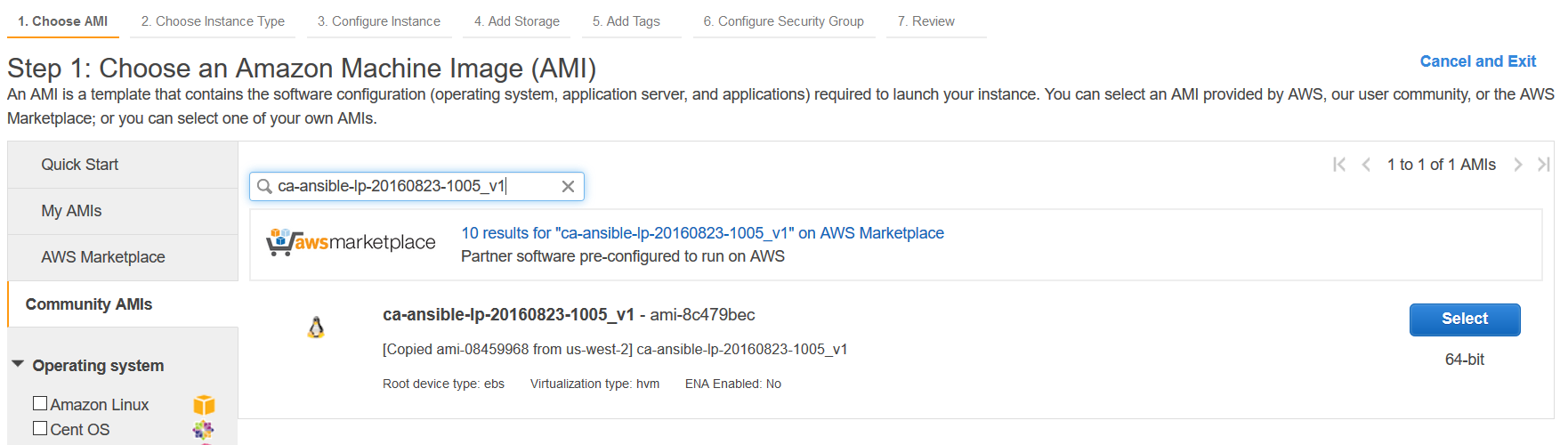
Enter IAM Username and Password here and then click “Sign In” button. AWS Console Home page will be opened:

3. Select the **US West (Oregon)** region using the upper right drop-down menu on the AWS Management Console:

Amazon Web Services is available in different regions all over the world, and the Console lets you provision resources across multiple regions. You usually choose a region that best suits your business needs to optimize your customer’s experience, but you must use the **US West (Oregon)**for this Lab.

## Launching EC2 instance - **ami-0c2aba6c**

Select EC2 service and Launch instance. Choose AWS community AMI and supply “CentOS Linux 7 x86\_64 HVM EBS 1703\_01 (**ami-0c2aba6c**)” name for search. And finish provisioning supplying respective parameters for storage and security group.



## Connect to a remote shell using an SSH connection with a User

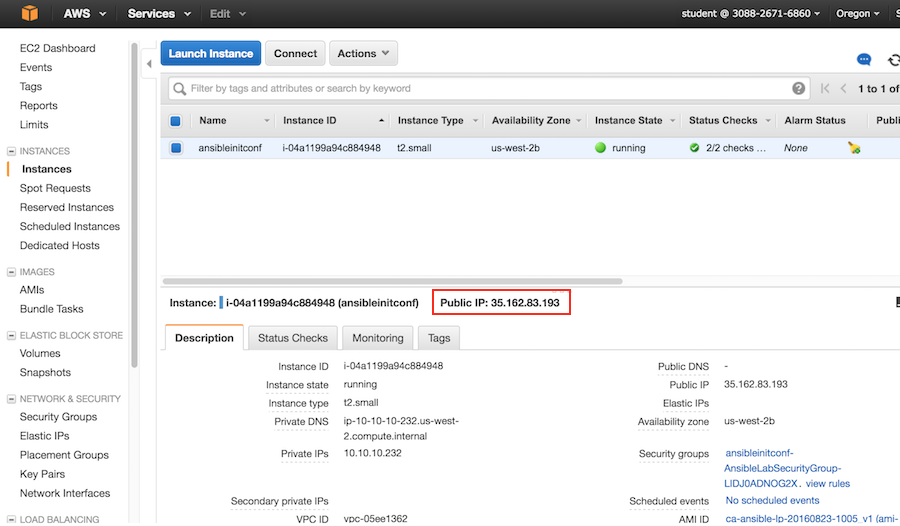
In order to manage a remote Linux server, you must employ an **SSH Client**. Secure Shell (SSH) is a cryptographic network protocol for securing data communication. It establishes a secure channel over an insecure network. Common applications include remote command-line login and remote command execution.

### Connect using Linux / Mac OS

Linux distributions and Mac OS are shipped with a fully working SSH client.

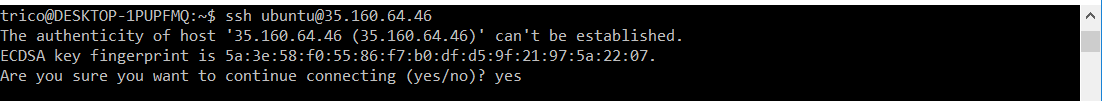
Starting a remote SSH session is easy:

* Open your **Terminal**application
* Write and run the following command: ssh **ubuntu**@**server-ip**.
  + **server-ip** is the Public IP of the EC2 server that we've automatically created for you. You can find it in the EC2 instance details. The image below shows where to find it from inside the AWS console. If the instance doesn't show up right away, you may need to wait a couple of minutes while it is provisioned.
  + **ubuntu** is the remote system user that will be used for the remote authentication

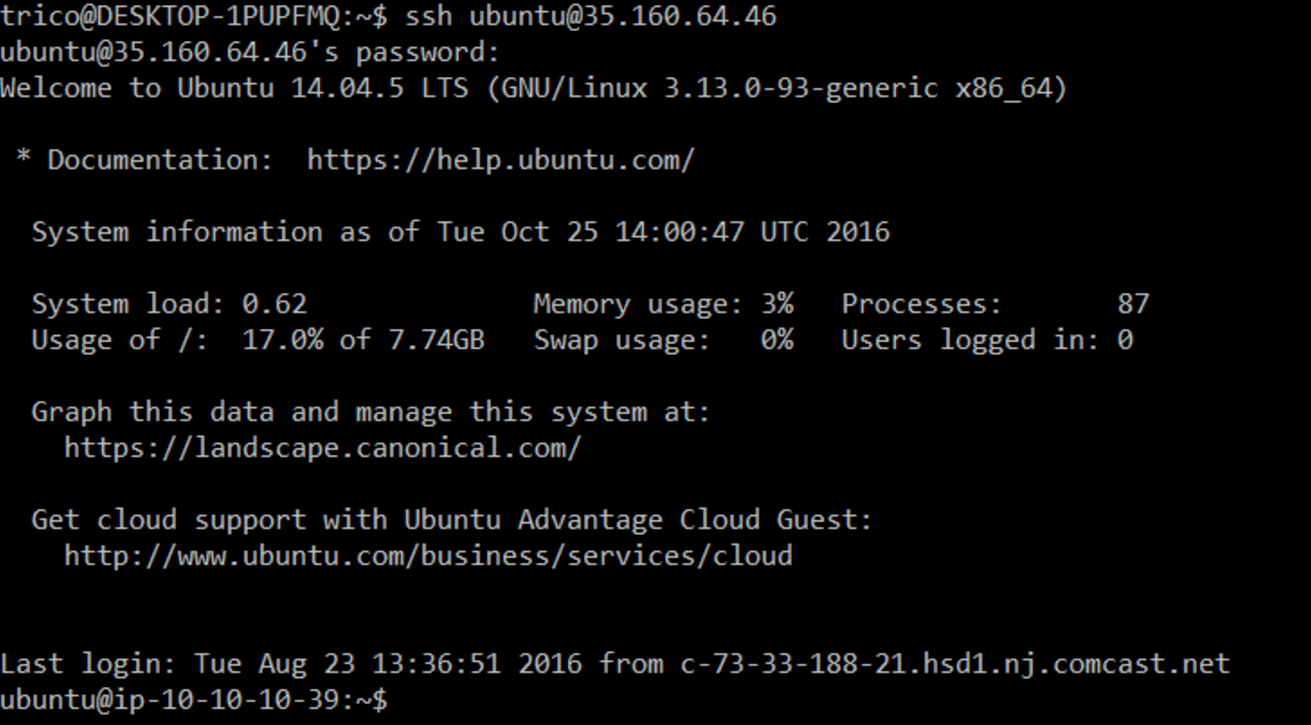


When prompted to verify that you want to connect, write the word: yes

And then press the enter/return key.



Insert the password **AnsibleLab2016** and press Enter and you will be connected:

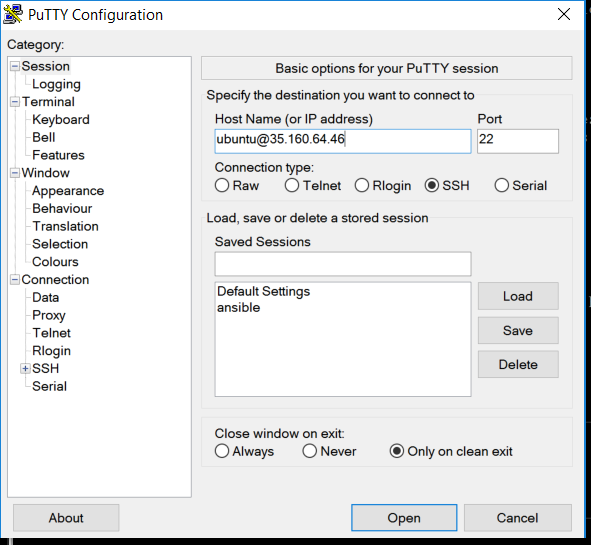


### Connect using Windows

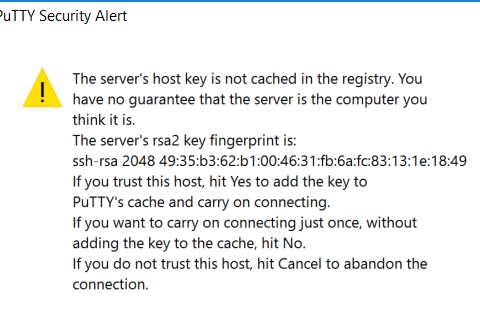
Windows has no SSH client, so you must use PuTTY and convert the PEM key to PPK using PuTTYgen.

Starting a remote SSH session using PuTTY is easy:

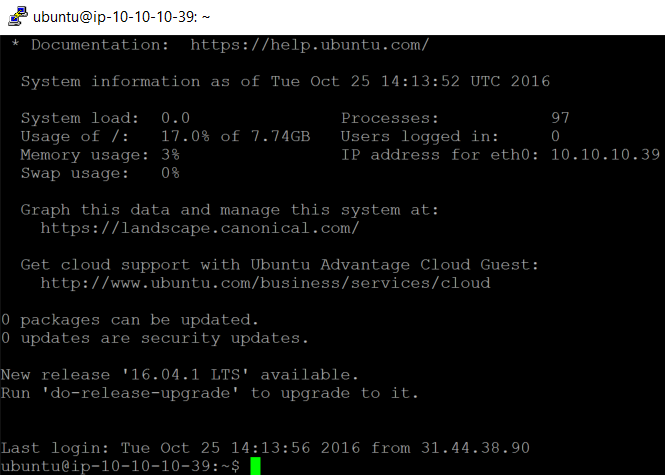
* Open PuTTY and insert ubuntu@"EC2 instance IP Address" in the Host Name field:



* Press yes if you see a putty security alert like the following:



You will be connected to the EC2 instance:



**Introduction**

chef-solo is a command that can execute Chef cookbooks without connecting to a Chef server. It uses the same Chef client that runs in Chef's client-server architecture. Chef Solo just uses the local mode of the Chef client without any server dependencies. This allows Chef Solo to use cookbooks on the local file system or via a URL instead of a centralized server. This also means that everything that would be stored on the central server needs to be stored on the node running Chef Solo.

Chef Solo was actually how Chef originally worked before support for a central server was added. You can use Chef Solo for scenarios where there is only one node to manage, when under resource constraints, and for quickly practicing with Chef. You should also know that Chef Solo doesn't support any authentication or authorization. In case you have heard about Chef Zero, Chef Solo is similar to Chef Zero in that Chef Zero also uses the local mode of chef-client.

You will install and set up Chef Solo on the virtual machine you connected to.

**Instructions**

1. For simplicity of executing the instructions in this Lab, you will sign in as the root user:

sudo su -

Chef Solo requires root permissions to run and also uses default directories that require root permisison for write operations. It is not recommended to use root for normal daily operation, but for the sake of following the Lab instructions it won't be a problem.

2. Install version 13 of Chef using the omnitruck installation script:

curl -L https://omnitruck.chef.io/install.sh | bash -s -- -v 13

This script works across many different operating systems and Linux distributions. It can be used to install the Chef client on Chef nodes and includes chef-solo. The -v option at the end tells the installer to install version 13 of Chef.

3. Verify chef-solo is installed as part of the installation:

chef-solo -v

This will report the version of Chef installed. The version will be something like:

*Chef: 13.12.14*

4. Create the default directory for the Chef Solo configuration file and change into the directory:

mkdir /etc/chef  
cd /etc/chef

5. Create a default Chef Solo configuration file using the vi text editor:

vi solo.rb

*Note*: Feel free to use a different terminal text editor if you are more familiar with one other than vi

The default configuration is located in /etc/chef/solo.rb, but know that you can pass the -c option to the chef-solo command to specify a different location. You will use the default location in this Lab to cut down on typing.

If you haven't used vi before, the following instructions will tell you everything you need to know.

6. Enter *insert* mode to start writing text in the file by pressing i.

Notice the status line at the bottom of your terminal changes to tell you that you are in insert mode:

alt

7. Enter the following into the terminal:

repo = '/etc/chef/repo'  
  
# Paths  
cookbook\_path repo + '/cookbooks'  
role\_path repo + '/roles'  
  
# Logging  
log\_level :info # Log additional info  
log\_location STDOUT # Write logs to the terminal

The configuration file specifies all of the Chef Solo settings that you need for this Lab:

* Paths to cookbooks and roles
* Log additional log info to the terminal

There are [many other settings that can be specified](https://docs.chef.io/config_rb_solo.html). You are encouraged to review them after completing the Lab.

Notice that you can use Ruby variables, like repo, and operations in the file.

 8. Press the escape key to leave insert mode and enter :wq to write the file changes to disk and quit vi.

 9. Create the directories specified in the configuration file for the cookbooks and roles:

mkdir -p repo/cookbooks repo/roles

Now you are ready to use Chef Solo.

### Instructions

1. Change into the cookbook directory and make a cookbook called chef-lab, then change into the cookbook's recipes directory:

cd repo/cookbooks  
mkdir -p chef-lab/recipes/  
cd chef-lab/recipes/

A cookbook is just a directory with specific subdirectories.

2. Start editing your first Chef recipe:

vi hello.rb

3. Enter the following after pressing i to enter insert mode:

log 'message' do  
 message 'Hello Chef'  
 level :info  
end

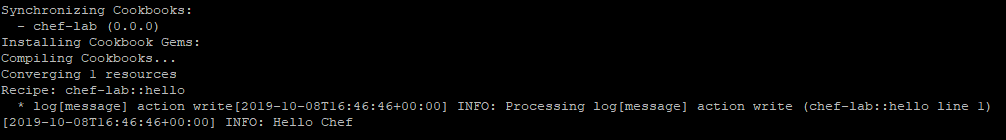
The log resource simply writes log messages. You can specify the level attribute to set the severity of the log message. As with all of the built-in resources, the [Chef documentation](https://docs.chef.io/resource_log.html) covers all of the actions and attributes.

4. Press escape to leave insert mode followed by :wq.

5. Run the recipe with Chef Solo:

chef-solo -o 'recipe[chef-lab::hello]'

The -o specifies the run list, which lists the recipes to run. You will see the following in the output:



This shows when Chef Solo is reading the **chef-lab** cookbook and converging the resource in the **hello** recipe. The line beginning with **\* log[message]** shows the default action is **write** and the last line is the message you specified written to the log.

The rest of the output is useful for understanding how Chef Solo works. You are encouraged to quickly read the other **INFO** messages.

6. Create another recipe:

vi my\_file.rb

This recipe will show how to create a file with certain properties.

7. Press i and enter:

file '/tmp/my\_file.txt' do  
 content 'This is mine'  
 owner 'centos'  
 group 'centos'  
end

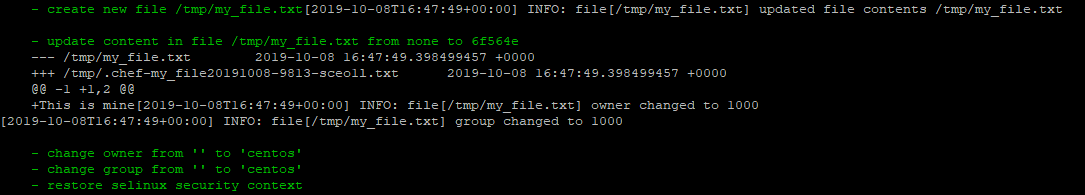
The file resource is used in this recipe. It has a default action of create. The create action will create the file if it doesn't exist, as well as modify the contents of the file if the file exists but doesn't match the specified properties.

8. Save the file by pressing escape followed by :wq.

9. Run the recipe:

chef-solo -o 'recipe[chef-lab::my\_file]'

In the output you will see a snippet similar to the following:



The output is very explicit about what it is doing to converge the file resource to the desired state declared in your recipe.  Any action that Chef Solo takes on a resource will appear in green. This output shows the file is created, the contents are updated, and finally the ownership properties are set.

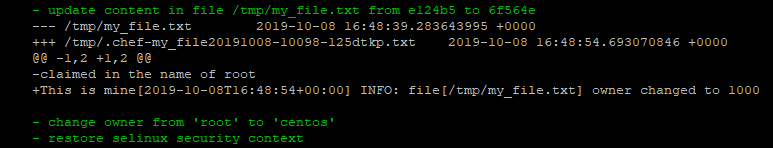
10. Change the file to diverge it from its desired stated:

echo "claimed in the name of root" > /tmp/my\_file.txt  
chown root /tmp/my\_file.txt

Now root has claimed ownership and modified the contents.

11. Run the recipe again and see how the output differs from the previous run:

chef-solo -o 'recipe[chef-lab::my\_file]'



This time Chef only had to update the content and set the owner. Also notice the diff of the file contents is shown in the middle, albeit obscured by an **owner changed** **INFO** message. The lines beginning with **-** are previous values and lines beginning with **+** show the new values. Although a simple example, this demonstrates a key benefit of using Chef over a simple command script. Chef is able to converge resources to their desired states declared in recipes.

### Instructions

1. Create a new recipe called default.rb:

vi default.rb

The default recipe is the one that is executed for a cookbook without specifying a particular recipe. The main purpose of the chef-lab cookbook is to set up your custom web server, so this recipe will be your default.

2. Enter the following into the recipe and save the file with :wq when finished:

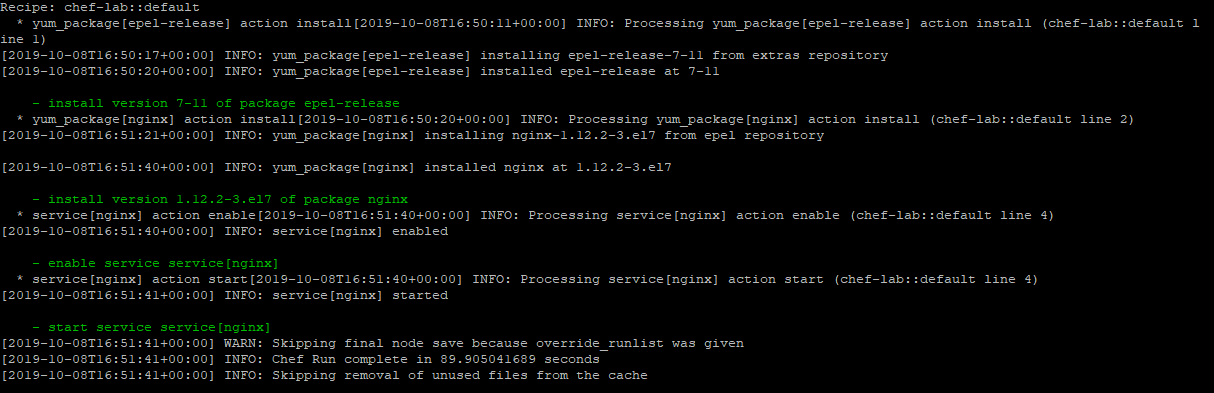
package 'epel-release' # package repository for nginx  
package 'nginx'  
  
service 'nginx' do  
 # declare actions the service supports  
 supports status: true, restart: true, reload: true  
 action [ :enable, :start ]  
end

This recipe is the starting point for your web server configuration. It installs two packages to get Nginx installed using the package resource and the default action of :install. The service resource is then used to enable Nginx on boot and start Nginx. The supports property tells Chef that Nginx supports some extra actions that you will use later.

3. As a good development practice, test the recipe before proceeding too far with a project:

chef-solo -o 'recipe[chef-lab]'

The default recipe name is implied in the run list. You will see log messages notifying you that packages are being installed, the Nginx service is enabled, and finally the service is started.



4. Test the web server is working by navigating to your virtual machine's IP address in a browser:



Tip: You can find the IP in the AWS Management Console as explained in a previous Lab Step, or from the command line by entering

curl ipecho.net/plain; echo

It will say Fedora instead of CentOS because of the source of your Nginx package. As the web page suggests, you will now configure your own web page. Keep the browser open on the web page since you will use it frequently in this Lab.

5. Create a cookbook file directory inside your chef-lab cookbook:

cd ..  
mkdir files

You now have a recipes and a files directory in your cookbook. The files directory is where you store cookbook files. Cookbook files are used with the cookbook\_file recipe resource. Using cookbook files, you can keep your recipes clean by referencing files instead of writing file content into the recipe itself.

6. Create a minimal Nginx configuration cookbook entering the following into vi files/nginx.conf:

worker\_processes 1;  
  
events {  
 worker\_connections 1024;  
}  
  
  
http {  
 include mime.types;  
 default\_type application/octet-stream;  
 sendfile on;  
  keepalive\_timeout 65;  
  
 server {  
 root /var/www;  
 listen 80 default\_server;  
 server\_name \_;  
 index index.html index.htm;  
 }  
}

The Nginx configuration specifics are out of the scope of this Lab. What's important for this Lab Step is that it looks for a web page to serve in the /var/www/html directory. You will now make the web page.

7. Save your configuration cookbook file :wq.

8. Create your web page cookbook file in vi files/index.html:

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="utf-8">  
 <title>My Chef Lab Page</title>  
 </head>  
 <body>  
 Come smell <a href=" https://learn.chef.io/" target="\_blank">what's cooking with Chef at Learn Chef Rally</a>  
 </body>  
</html>

Now that you have cookbook files, you need to modify the recipe to use them.

9. Add the following to the end of your recipes/default.rb recipe:

directory '/var/www'  
  
cookbook\_file '/var/www/index.html' do  
 action :create  
end  
  
cookbook\_file '/etc/nginx/nginx.conf' do  
 action :create  
end

This creates the /var/www directory and the two cookbook files when the recipe is executed. Chef is able to identify which cookbook file to use based on the name of the files in the cookbook's files directory. For example, it knows to use files/index.html when it sees /var/www/index.html in the cookbook\_file resource block because they are both named index.html.

10. Run Chef Solo and try to follow what it is doing as the output streams by:

chef-solo -o 'recipe[chef-lab]'

11. Refresh your browser to see the effect the recipe had:



### Instructions

1. Edit the two cookbook files in the default recipe in recipes/default.rb as follows with changes in bold:

cookbook\_file '/var/www/index.html' do  
 action :create  
 **notifies :reload, 'service[nginx]', :immediately**  
end  
  
cookbook\_file '/etc/nginx/nginx.conf' do  
 action :create  
 **notifies :reload, 'service[nginx]', :immediately**  
end

Each resource block includes a notifies property. The notifies property allows a resource to trigger an action in another resource. In this case, if either cookbook file changes, Nginx will reload its configuration to reflect the most recent changes. Reloading Nginx is faster than restarting and also keeps the server running. Sometimes a restart is required but reload is enough for this Lab.

2. Run Chef Solo:

chef-solo -o 'recipe[chef-lab]'

At the end of the outpt you see:

alt

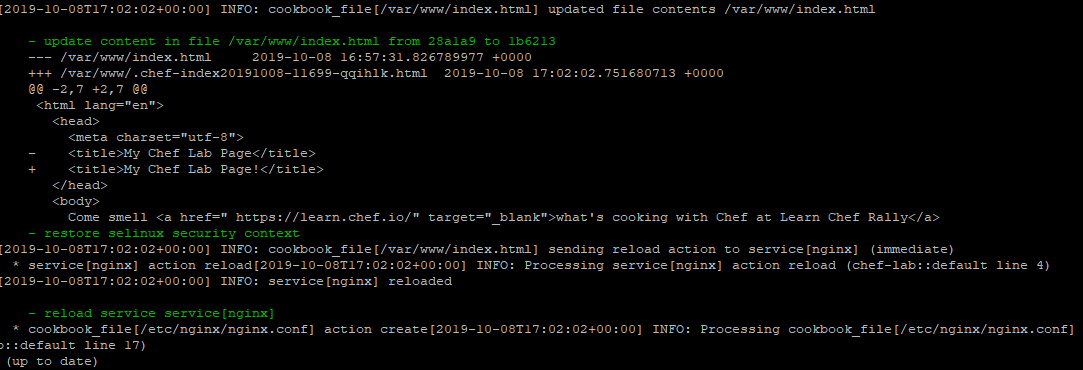
Again nothing got updated... This is because adding notifies doesn't change the desired state of the cookbook files.

 3. To test the notifies, edit files/index.html to add an exclamation mark at the end of the title

    <title>My Chef Lab Page!</title>

Now the desired state of the index.html file has changed from the one that is deployed in /var/www/index.html

4. Re-run Chef Solo and see how the notifies affects the behavior:



5. Reload the web page in your browser:



Success! With the notify properties in place, you can be assured that the Nginx server will know about file changes that affect it.

### Instructions

1. Change your recipes/default.rb to use the node['packages']['nginx\_repo'] and node['nginx']['root'] attributes:

package **node['packages']['nginx\_repo']**  
package 'nginx'  
  
service 'nginx' do  
  # declare actions the service supports  
  supports status: true, restart: true, reload: true  
  action [ :enable, :start ]  
end  
  
directory **node['nginx']['root']**cookbook\_file "**#{node['nginx']['root']}**/index.html" do  
  action :create  
  notifies :reload, 'service[nginx]', :immediately  
end  
  
cookbook\_file '/etc/nginx/nginx.conf' do  
  action :create  
  notifies :reload, 'service[nginx]', :immediately  
end

The three bold changes show how attributes are used. The node object contains attributes that you will define in an attribute file. The attributes are grouped into attributes related to packages and attributes related to nginx configuration. You are free to use whatever labels you like to organize your attributes.

 2. Make directories for your attributes and templates:

mkdir -p attributes templates/default

3. Declare all the attributes you'll use in attributes/default.rb:

default['packages']['nginx\_repo'] = 'epel-release'  
  
default['nginx']['root'] = '/var/www'  
default['nginx']['port'] = '80'  
default['nginx']['worker\_processes'] = 1  
default['nginx']['worker\_connections'] = 1024

default refers to the attribute type. default is the lowest precedence, but it is the only type you will use in this Lab. You should use default attributes as much as possible in your cookbooks. They allow users to override them with higher precedent attributes when needed.

4. Run your cookbook with Chef Solo:

chef-solo -o 'recipe[chef-lab]'

You will see that no resources were updated. This proves that the values that were hard-coded into the recipe are now being substituted using the attributes defined in the default attributes file. But, for example, if you ever wanted to change the root path for Nginx, you only need to change one attribute value. However, you need to use a template to generate the Nginx configuration file for that to be true.

5. Move the files/nginx.conf file into the templates directory:

mv files/nginx.conf templates/default/nginx.conf.erb

You will turn the static file into a template file with Embedded Ruby (ERB).

6. Replace several hard-coded values with ERB snippets for some of the hard-coded values:

worker\_processes **<%= node['nginx']['worker\_processes'] %>**;  
  
events {  
 worker\_connections **<%= node['nginx']['worker\_connections'] %>**;  
}  
  
  
http {  
 include mime.types;  
 default\_type application/octet-stream;  
 sendfile on;  
 keepalive\_timeout 65;  
  
 server {  
 root **<%= node['nginx']['root'] %>**;  
 listen **<%= node['nginx']['port'] %>** default\_server;  
 server\_name \_;  
 index index.html index.htm;  
 }  
}

The <%= and => brackets enclose the ERB code. In this case each snippet simply inserts a node attribute value in its place. Templates are much more powerful than demonstrated in this Lab. However, you can see the benefits of templates through this simple example.

7. The last instruction before being able to use the template is to change the cookbook\_file resource in recipes/default.rb for the configuration file to a template resource:

**template** '/etc/nginx/nginx.conf' do  
 action :create  
 notifies :reload, 'service[nginx]', :immediately  
end

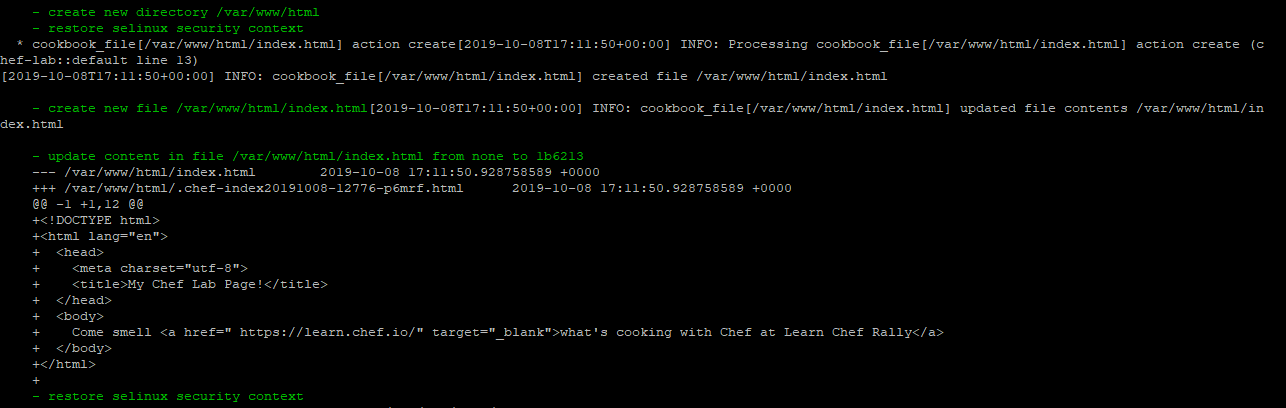
8. Run Chef Solo again and verify that no resources are updated.

This means that the template has successfully replaced the static cookbook file. Now that you have leveraged attributes and templates, configuration changes are a breeze.

9. Change the root directory for the Nginx web server content to /www by changing only the associated attribute value in attributes/default.rb:

default['nginx']['root']               = '/var/www/html'

10. Run Chef Solo one more time and observe the changes as the output streams by. Some highlights from the output are:



 11. Verify that the web page is still available by refreshing your browser.

With a bit of initial work, configuration changes become easy with Chef Solo.

### Instructions

1. Move up two levels in the directory hierarchy to the repo directory to where the roles directory is:

cd ../..

2. Create a role file for low load situations called roles/web-low.rb with the following contents:

name "web-low"  
  
default\_attributes 'nginx' => {  
 'worker\_processes' => 1,  
 'worker\_connections' => 1024  
}  
  
run\_list 'recipe[chef-lab::hello]',  
 'recipe[chef-lab]'

To handle the different load situations, the worker\_processes and worker\_connections attributes are set. Default attributes in a role have higher precedence than default attributes in an attributes file. The run\_list includes two recipes for demonstration purposes. The default recipe is the essential one, although the hello recipe doesn't hurt because it only writes a log message.

3. Copy the low role and make a high role from it:

cp roles/web-low.rb roles/web-high.rb

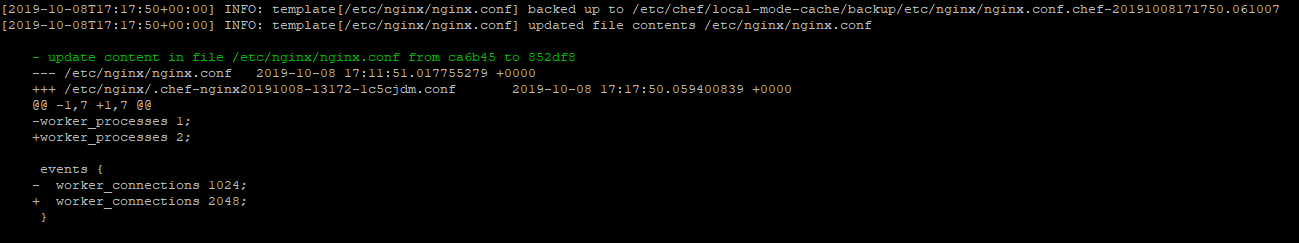
4. Edit the attributes in roles/web-high.rb to be as follows:

name "web-**high**"  
  
default\_attributes 'nginx' => {  
 'worker\_processes' => **2**,  
 'worker\_connections' => **2048**  
}  
  
run\_list 'recipe[chef-lab::hello]',  
 'recipe[chef-lab]'

The high load role doubles the number of worker\_processes and worker\_connections.

5. Use Chef Solo to put the node into the high load role:

chef-solo -o 'role[web-high]'



6. Just as easily you can change it back to web-low:

chef-solo -o 'role[web-low]'

