**Getting started with Ansible**

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

**Instructions**

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

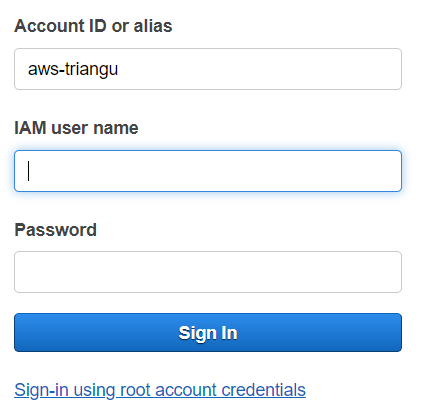
<https://aws-triangu.signin.aws.amazon.com/console>

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{01-17}*
* **Password**: *$trainig0*
* **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: 

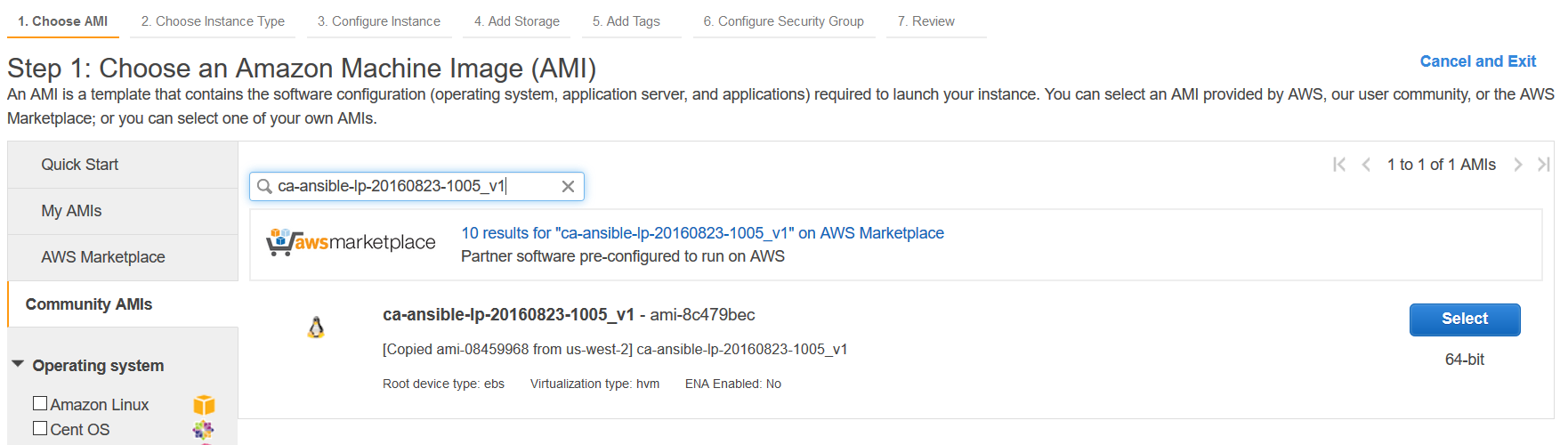
Enter IAM User name and Password here and then click “Sign In” button. AWS Console Home page will be opened:

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

Select EC2 service and Launch instance. Choose AWS community AMI and supply “ca-ansible-lp-20160823-1005\_v1 (ami-8c479bec)” 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

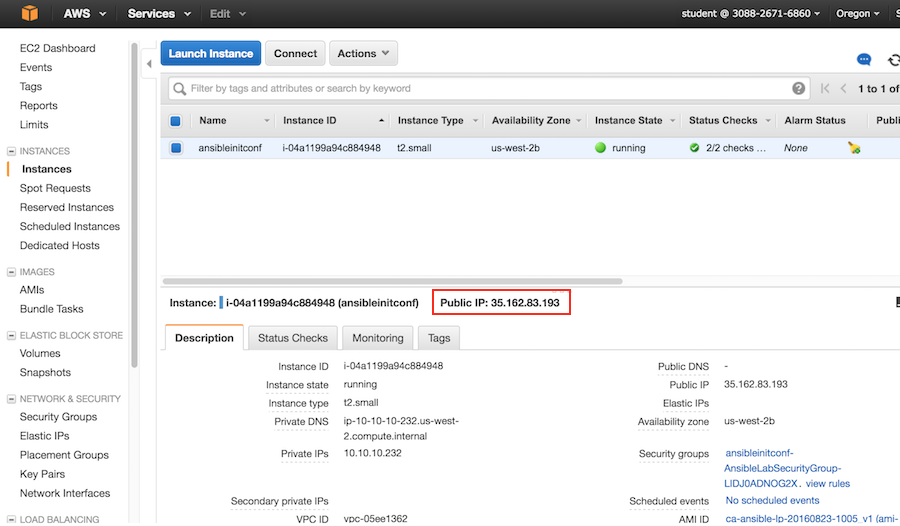
In order to manage a remote Linux server, you must employ an **SSH Client**. Secure Shell (SSH) is a cryptographic network protocol for securing data communication. It establishes a secure channel over an insecure network. Common applications include remote command-line login and remote command execution.

### Connect using Linux / Mac OS

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

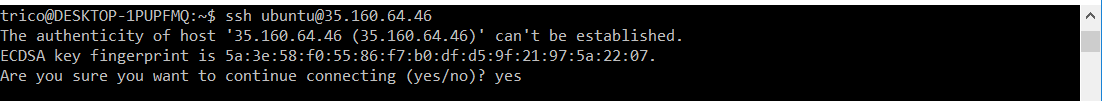
Starting a remote SSH session is easy:

* 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

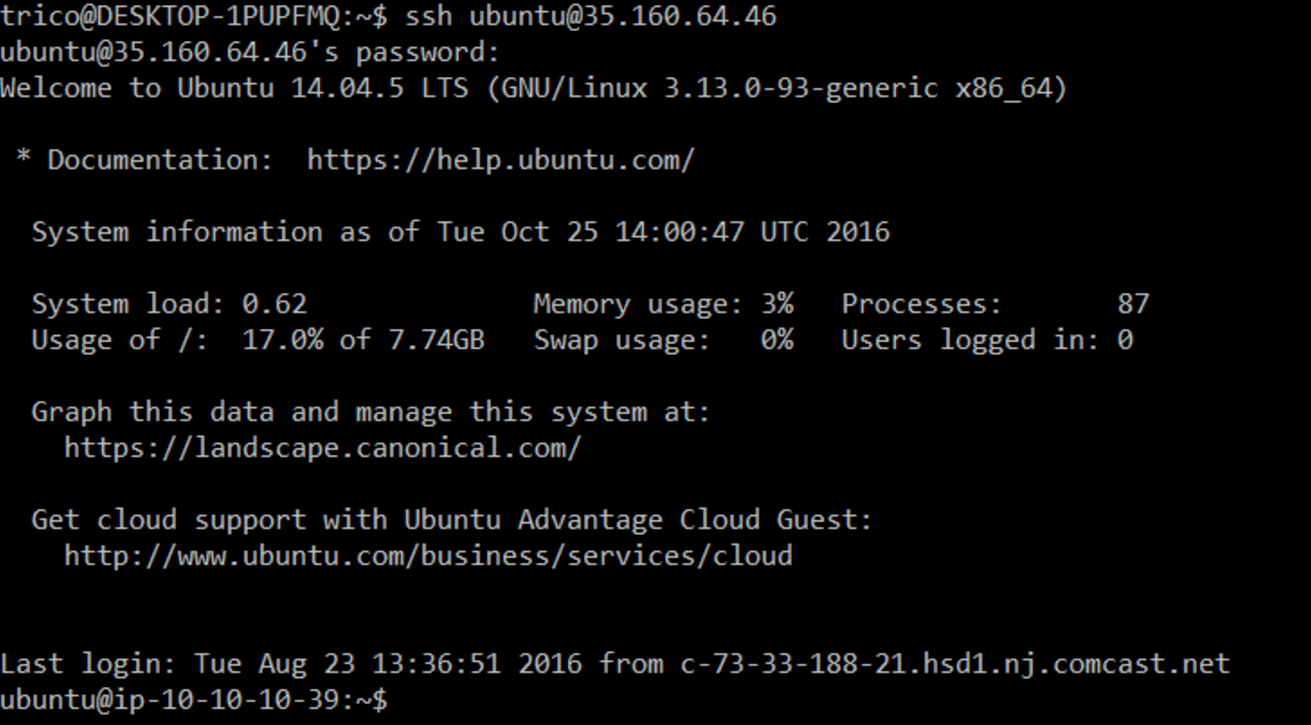


When prompted to verify that you want to connect, write the word: yes

And then press the enter/return key.



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

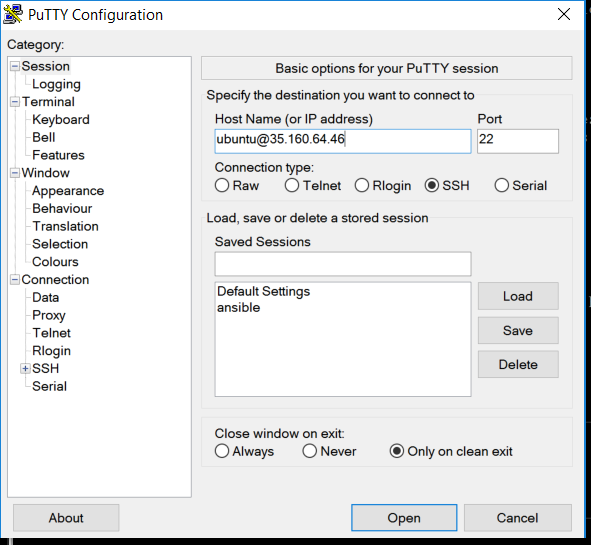


### Connect using Windows

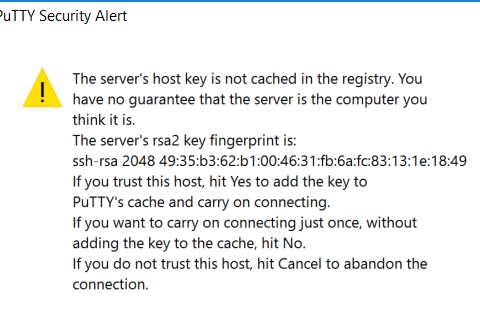
Windows has no SSH client, so you must use PuTTY and convert the PEM key to PPK using PuTTYgen.

Starting a remote SSH session using PuTTY is easy:

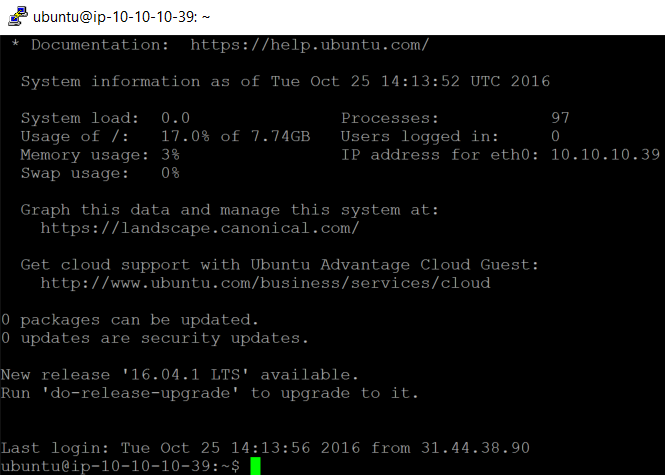
* Open PuTTY and insert ubuntu@"EC2 instance IP Address" in the Host Name field:



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



You will be connected to the EC2 instance:



## Create a Basic Ansible Playbook

Once you've connected to the Ubuntu server via SSH you can start working with Ansible. If you haven't yet connected in, go back to the previous step.

Once logged in, run the command below to confirm that Ansible has been properly installed on the system:

$ ansible --version

It should be version 2.1.0. The output should look like:

ansible 2.1.0.0  
 config file =   
 configured module search path = Default w/o overrides

You’ll start by creating the ‘lamp.yml’ playbook.  For text editors, you have a few options. Vim, nano and pico have been installed. However, if you're familiar with SCP you can use that to send local files over. However, that'll be an exercise for you to figure out. If you're new to using command line text editors, I recommend nano. So this lab will assume you're using nano.

To create the Playbook run the following command:

$ nano lamp.yml

This will open up nano and it will be empty.  So let's populate the playbook. Remember that the YAML syntax is very sensitive to whitespace. Whitespace isn't just for looks. The level of indentation matters! Also, use spaces and not tabs.

You'll want to start by installing ‘apache2’ and ‘mysql’.  These packages are both provided by ‘apt’, so use the apt module.  This demo will start off with this very basic playbook. You can copy and paste the following playbook into nano:

---

- hosts: localhost

gather\_facts: false

connection: local

become: yes

tasks:

- name: Install our packages

apt:

name: "{{ item }}"

state: present  
 update\_cache: true

with\_items:

- apache2

- mysql-server

- mysql-common

- mysql-client

Once it's in nano, you can close out by typing **CTRL+X**

And then it will ask if you want to save, press the **Y** key.

And it will then ask what the file name should be, and it should already be populated from when you opened nano. So press the **enter/return** key.

And that is how you'll save files going forward. **CTRL+X** to close out of nano, then **press Y** to save, then **press Enter** to save to the file name listed.

Before running this playbook, confirm that apache2 and mysql are \*not\* installed with the following command:

$ dpkg -l apache2 mysql-server

The output should be something like:

dpkg-query: no packages found matching apache2  
dpkg-query: no packages found matching mysql-server

Okay, now it's time to run the playbook for the first time.

So the command to call the playbook will look like this:

$ ansible-playbook lamp.yml

Note: Because the inventory file isn't configured you'll see a warning. However it doesn't matter when using Ansible to configure localhost.

After some time, the packages should install successfully. You will see this:

PLAY [all]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
TASK [Install our packages]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
changed: [localhost] => (item=[u'apache2', u'mysql-server', u'mysql-common', u'mysql-client'])  
  
PLAY RECAP   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
localhost                  : ok=1    changed=1    unreachable=0    failed=0

Run the dpkg command again to confirm that they are indeed installed:

$   dpkg -l apache2 mysql-server

Desired=Unknown/Install/Remove/Purge/Hold  
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend  
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)  
||/ Name                 Version         Architecture    Description  
+++-====================-===============-===============-=============================================  
ii  apache2              2.4.7-1ubuntu4. amd64           Apache HTTP Server  
ii  mysql-server         5.5.50-0ubuntu0 all             MySQL database server (metapackage depending

## Add Starting Services Tasks to Your Ansible Playbook

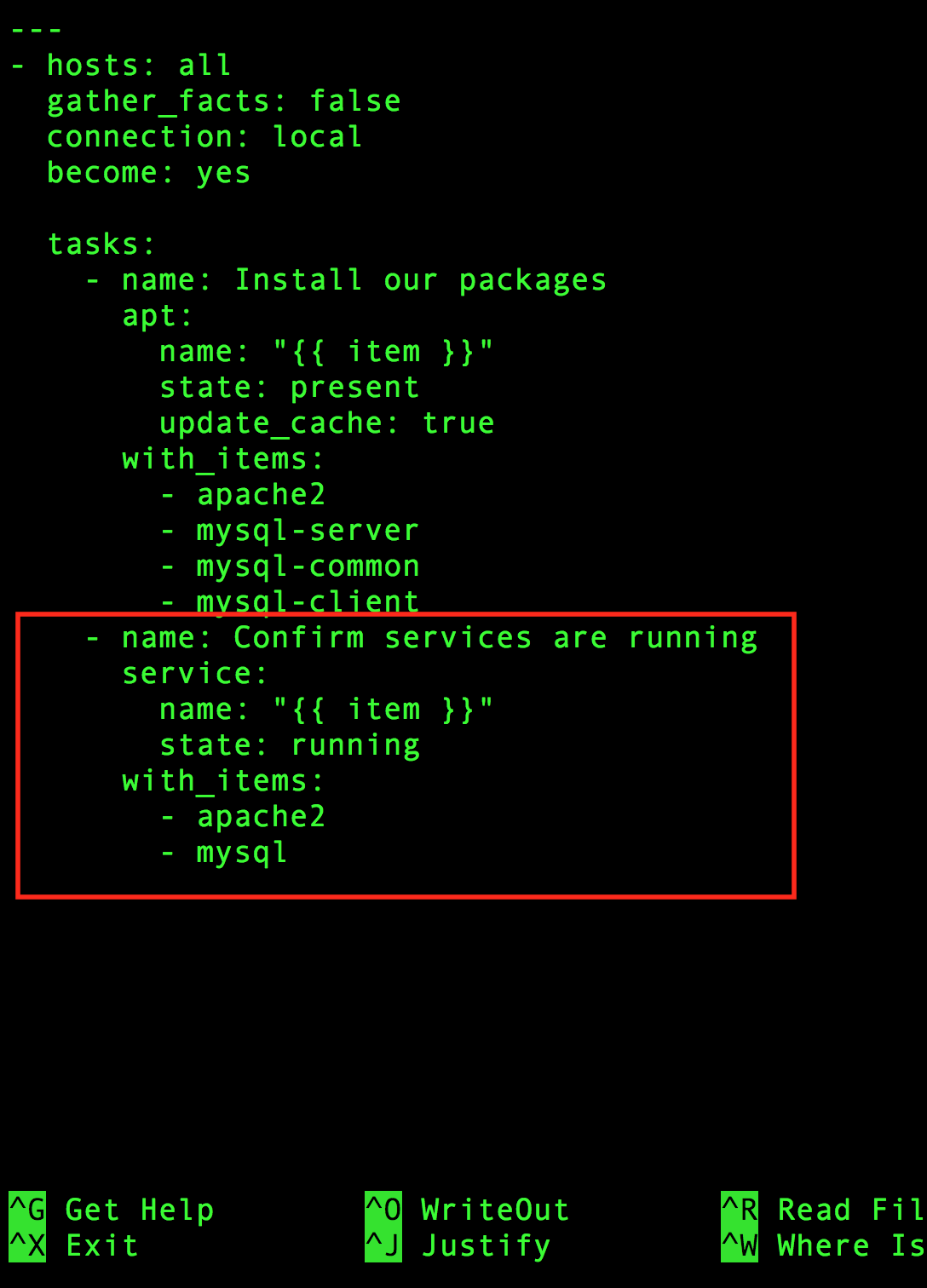
Now that you have Apache2 and MySQL installed, you'll want to make sure that they're running. Remember, the command to open the file is:

$ nano lamp.yml

To do that you need to use the service module. So, add the following to the end of the playbook:

- name: Confirm services are running  
 service:  
 name: "{{ item }}"  
 state: running  
 with\_items:  
 - apache2  
 - mysql

The final result should look like the below image. Make sure things are lined up correctly, and that the indentation is consistent:

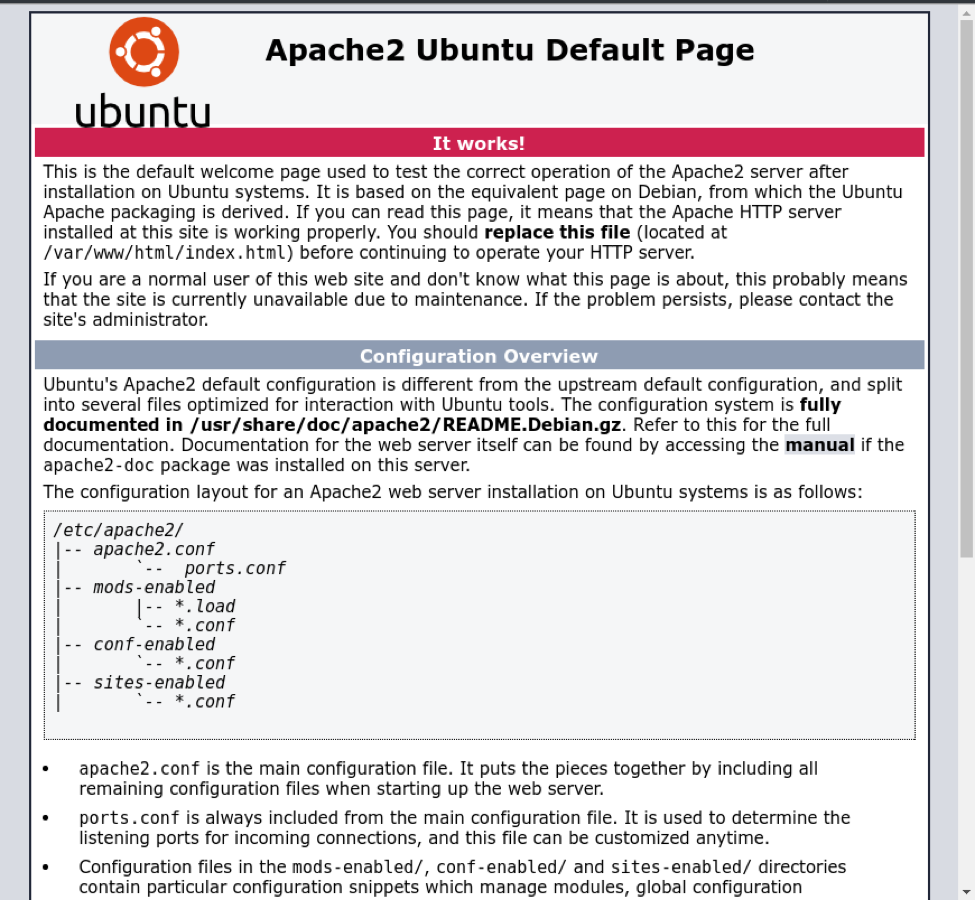
[](https://assets.cloudacademy.com/bakery/media/uploads/lab-step/blobid0-360cd120-1737-48d5-9a38-a707fcf6b72b.png)

Once you've added that, save the file and we can run the playbook again:

$ ansible-playbook lamp.yml

PLAY [all]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
TASK [Install our packages]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
ok: [localhost] => (item=[u'apache2', u'mysql-server', u'mysql-common', u'mysql-client'])  
  
TASK [Confirm services are running]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
ok: [localhost] => (item=apache2)  
ok: [localhost] => (item=mysql)  
  
PLAY RECAP   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
localhost                  : ok=2    changed=0    unreachable=0    failed=0

Now, if everything was successful we can test this out by opening a new browser tab, and pasting the public IP address into the address bar. This is the same IP address you used to connect via SSH, from the second lab step. If everything went well, then you'll see the default Apache page:



## Adding Variables to Your Ansible Playbook

Ansible allows you to use variables to make your playbooks more reusable, because you can edit the variables at runtime. So now would be a good time to show how to use them in a playbook.

The services and packages are both static so we’ll move these to the vars section of the playbook. Let's clear out everything we have in the current playbook. While there are many ways to clear out the file, you can use nano's cut command.

So, open the file in nano, make sure the cursor is on the first line, and hold CTRL+K until all the text is gone.

Then copy and paste the text below, and save the file:

---

- hosts: localhost

gather\_facts: false

connection: local

become: yes

vars:

packages:

- apache2

- mysql-server

- mysql-common

- mysql-client

services:

- apache2

- mysql

tasks:

- name: Install our packages

apt:

name: "{{ item }}"

state: present

with\_items: "{{ packages }}"

- name: Confirm services are running

service:

name: "{{ item }}"

state: running

with\_items: "{{ services }}"

Now run the playbook again to confirm the changes work, and the output should match what you had earlier. This is basically the same as before, except now you're using variables.

Okay, let's keep adding to your LAMP stack by including SSL.

By default Apache ships with only normal HTTP enabled.  To enable the HTTPS site you need to enable the modssl module and enable it on the site. This can be accomplished with these two commands, but don’t run them from the command line. You want to have Ansible do that for you:

$ a2enmod ssl

$ a2ensite default-ssl

Take these two commands and place them in the bottom of the Playbook as follows:

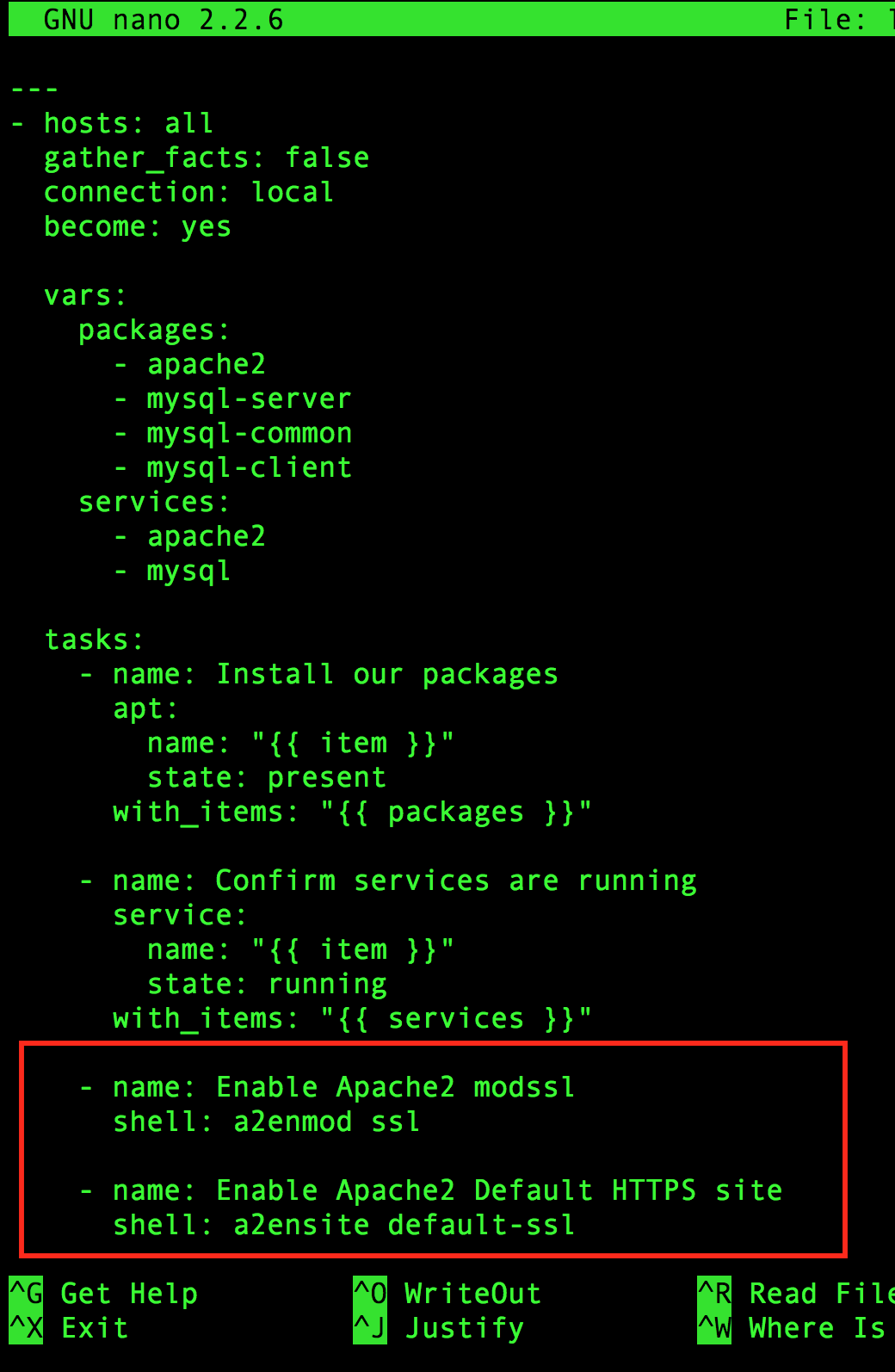
- name: Enable Apache2 modssl

shell: a2enmod ssl

- name: Enable Apache2 Default HTTPS site

shell: a2ensite default-ssl

It should look something like the image below:

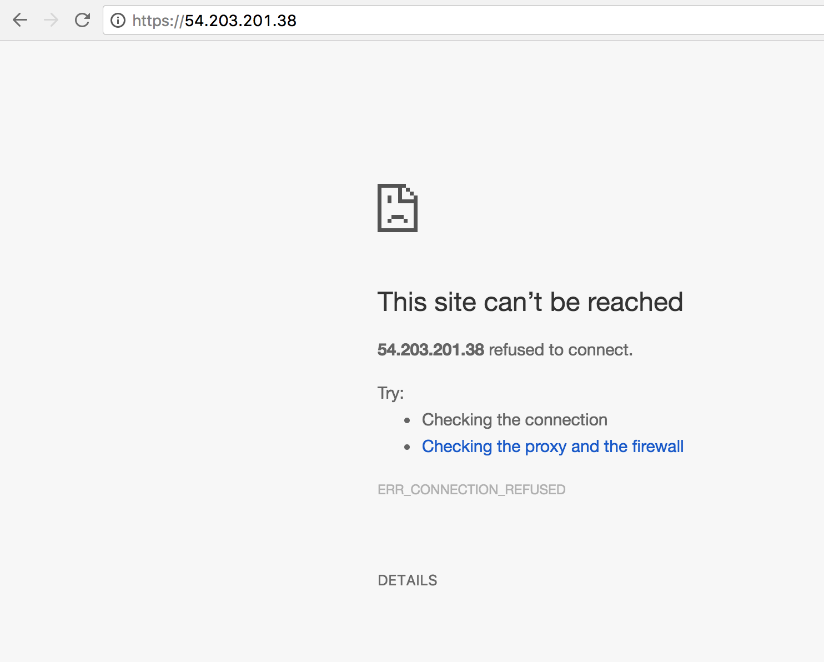


Now, run the playbook again:

$ ansible-playbook lamp.yml

PLAY [all]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
TASK [Install our packages]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
ok: [localhost] => (item=[u'apache2', u'mysql-server', u'mysql-common', u'mysql-client'])  
  
TASK [Confirm services are running]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
ok: [localhost] => (item=apache2)  
ok: [localhost] => (item=mysql)  
  
TASK [Enable Apache2 modssl]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
changed: [localhost]  
  
TASK [Enable Apache2 Default HTTPS site]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
changed: [localhost]  
  
PLAY RECAP   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
localhost                  : ok=4    changed=2    unreachable=0    failed=0

Now, if you still have the Apache default page open in your browser try editing the URL so that it uses HTTPS by adding https:// in front of the IP address. Similar to the screen shot below. Although, your IP address will be different:

[](https://assets.cloudacademy.com/bakery/media/uploads/lab-step/blobid2-a78b86ea-c638-465f-81d8-21065523c828.png)

This should fail with a refused connection. Why? After making those changes Apache2 needs to be reloaded or restarted to read in the new site and load the module.  So add a task to handle that:

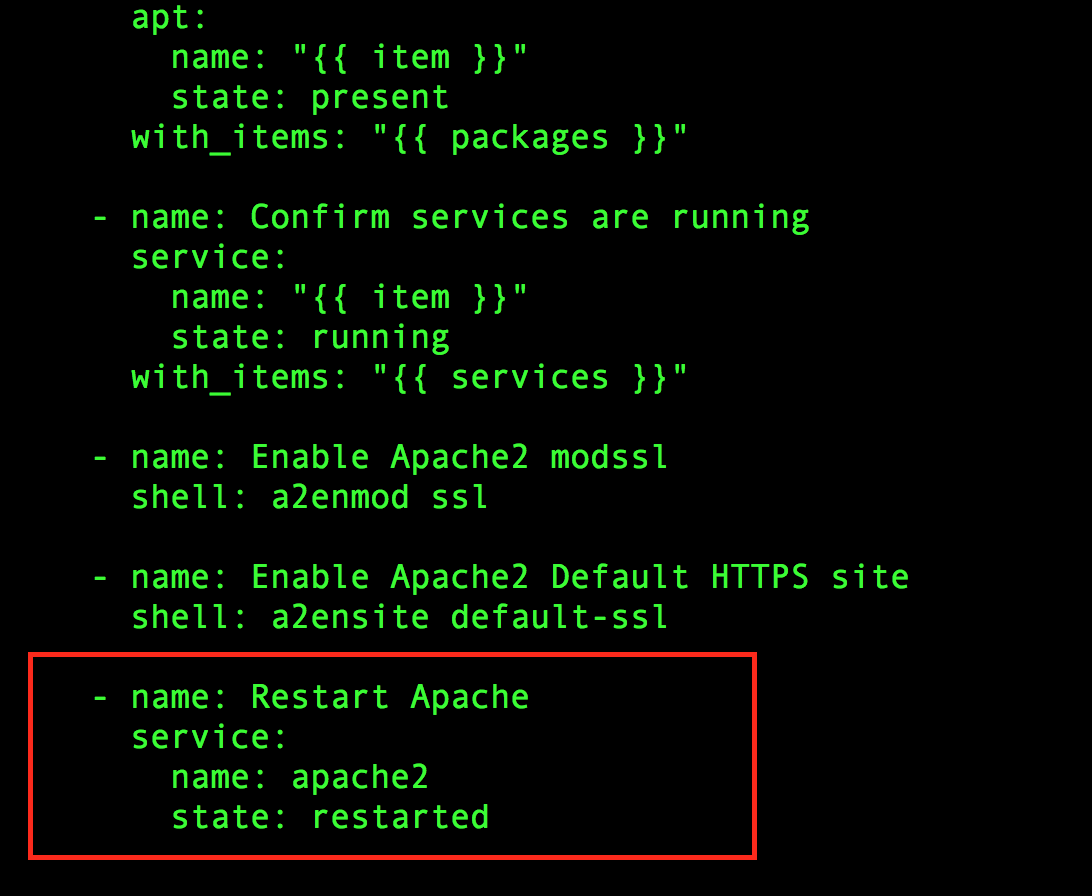
- name: Restart Apache

service:

name: apache2

state: restarted

The result should look something like the image below:

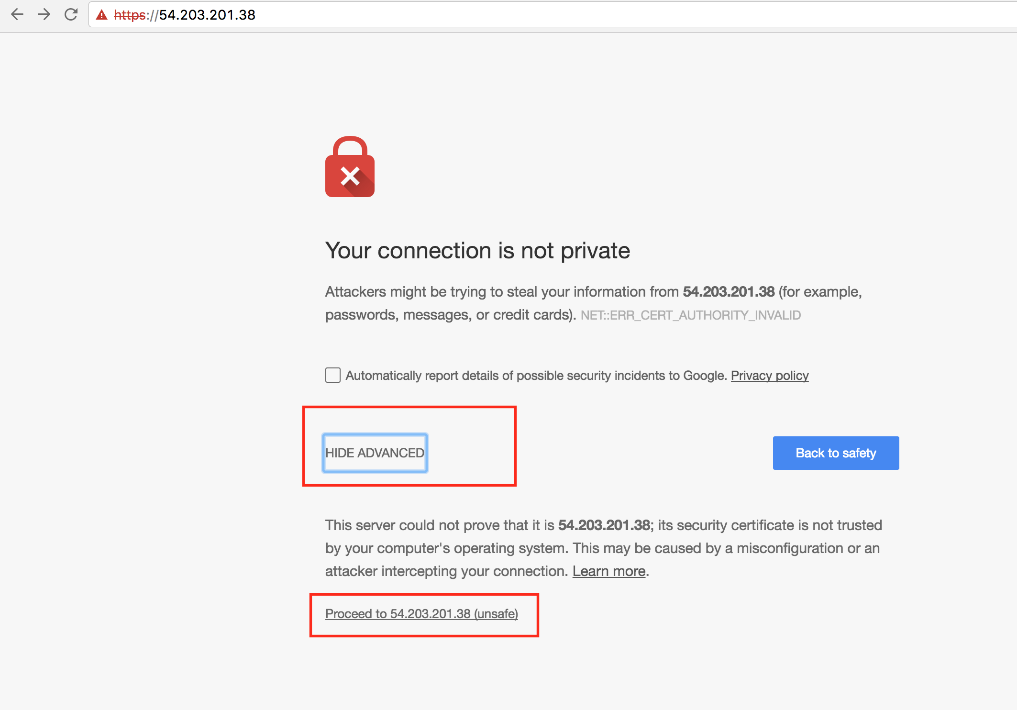


As you can see, you're using the same ‘service’ module as earlier to confirm the services are running; however, this time you're passing the state value of ‘restarted’ instead.  This will trigger the ‘service’ module to restart apache2 instead of just ensuring that it is running.  Running the playbook again will result in this:

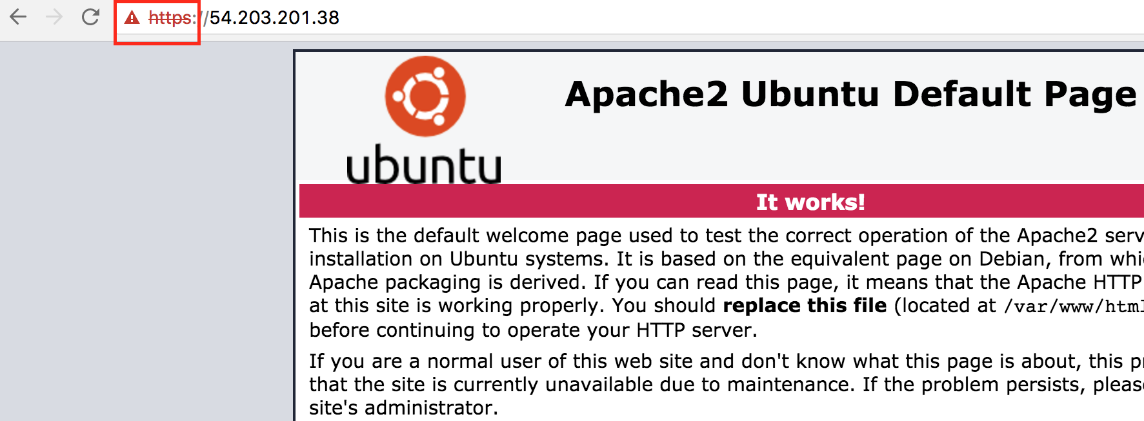
PLAY [all]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
TASK [Install our packages] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
ok: [localhost] => (item=[u'apache2', u'mysql-server', u'mysql-common', u'mysql-client'])  
  
TASK [Confirm services are running]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
ok: [localhost] => (item=apache2)  
ok: [localhost] => (item=mysql)  
  
TASK [Enable Apache2 modssl]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
changed: [localhost]  
  
TASK [Enable Apache2 Default HTTPS site]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
changed: [localhost]  
  
TASK [Restart Apache]   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
changed: [localhost]  
  
PLAY RECAP   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
localhost                  : ok=6    changed=3    unreachable=0    failed=0  

You can confirm now that the site is up and running by visiting the https version of the site again. A security prompt will present itself in your browser - accept the invalid certificate, as you know the site is indeed safe.

Below is how Chrome handles invalid cert issues. So click the advanced option and then click the **Proceed** link:



And if all goes well, you'll see the default Apache page again, running under SSL:



Also take note in that last playbook run, Ansible returned ‘changed’ for the tasks to enable modssl and the HTTPS site.  This is because they are already enabled and Ansible does not know how to determine the output from the shell command to see if a change has actually occurred.

You now have a very basic Playbook for installing Apache and MySQL and configuring the HTTPS site.

The final result of the playbook:

---

- hosts: all

gather\_facts: false

connection: local

become: yes

vars:

packages:

- apache2

- mysql-server

- mysql-common

- mysql-client

services:

- apache2

- mysql

tasks:

- name: Install our packages

apt:

name: "{{ item }}"

state: present

with\_items: "{{ packages }}"

- name: Confirm services are running

service:

name: "{{ item }}"

state: running

with\_items: "{{ services }}"

- name: Enable Apache2 modssl

shell: a2enmod ssl

- name: Enable Apache2 Default HTTPS site

shell: a2ensite default-ssl

- name: Restart Apache

service:

name: apache2

state: restarted