**Deploying webapp 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://goliaf.signin.aws.amazon.com/console>

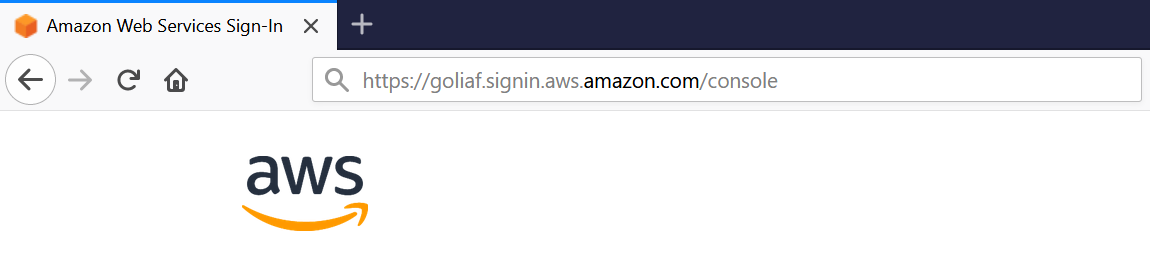
2. Enter the following credentials created just for your Lab session, and click **Sign In**:

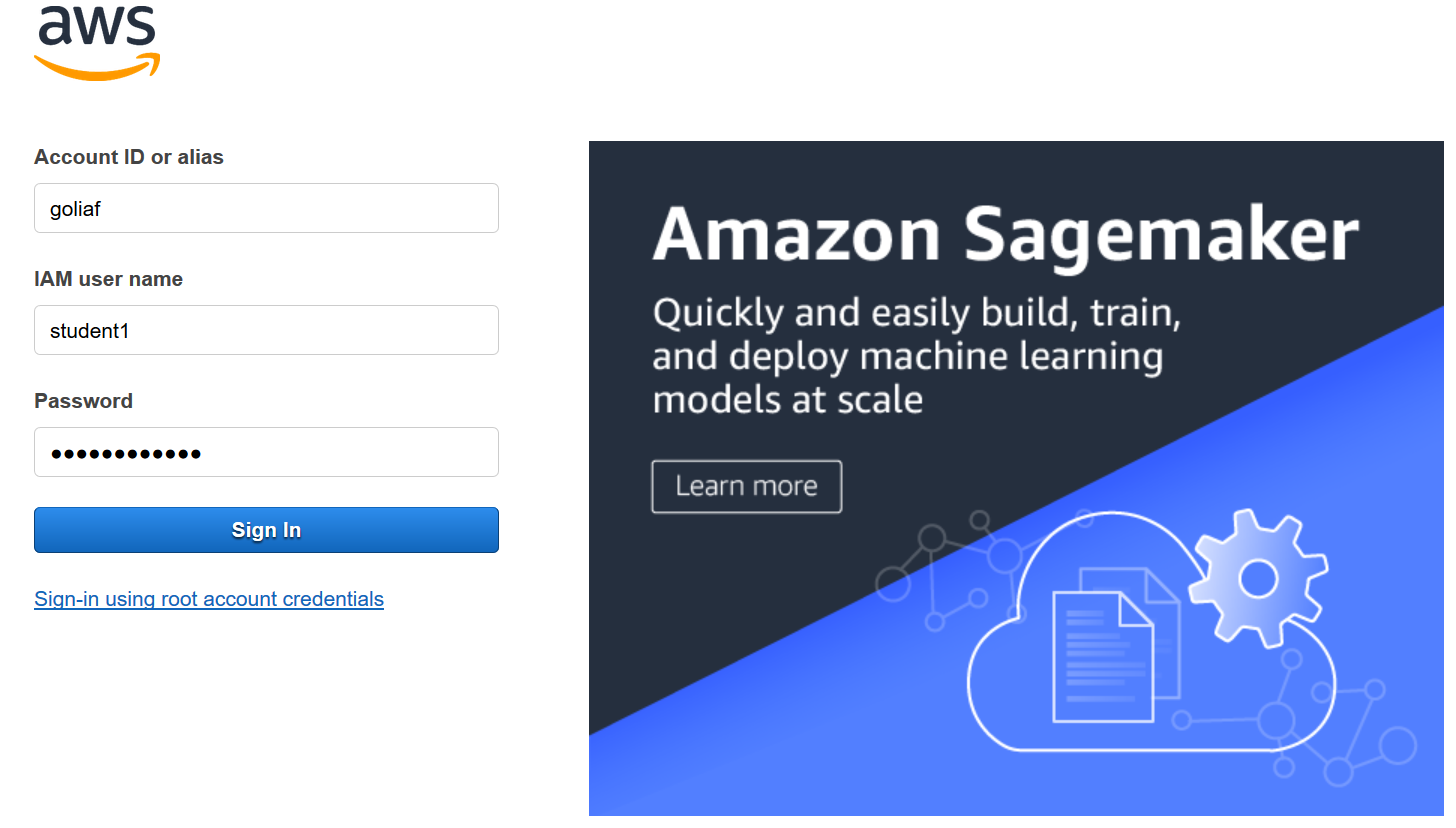
* **Account ID or alias**: Keep the pre-populated value
* **IAM user name**: *student{01-17}*
* **Password**: *$trainig0*
* **Signing-in AWS Management Console**

We are using URL

<https://goliaf.signin.aws.amazon.com/console>

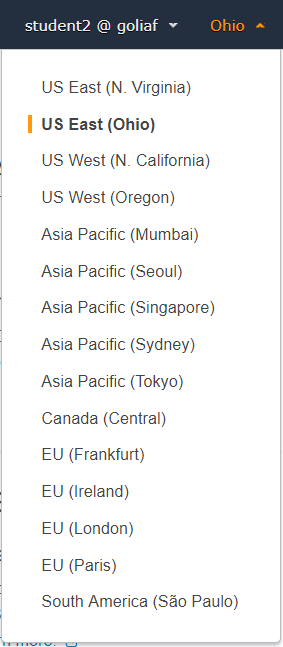
to sign-in AWS Management Console for training purpose.



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:

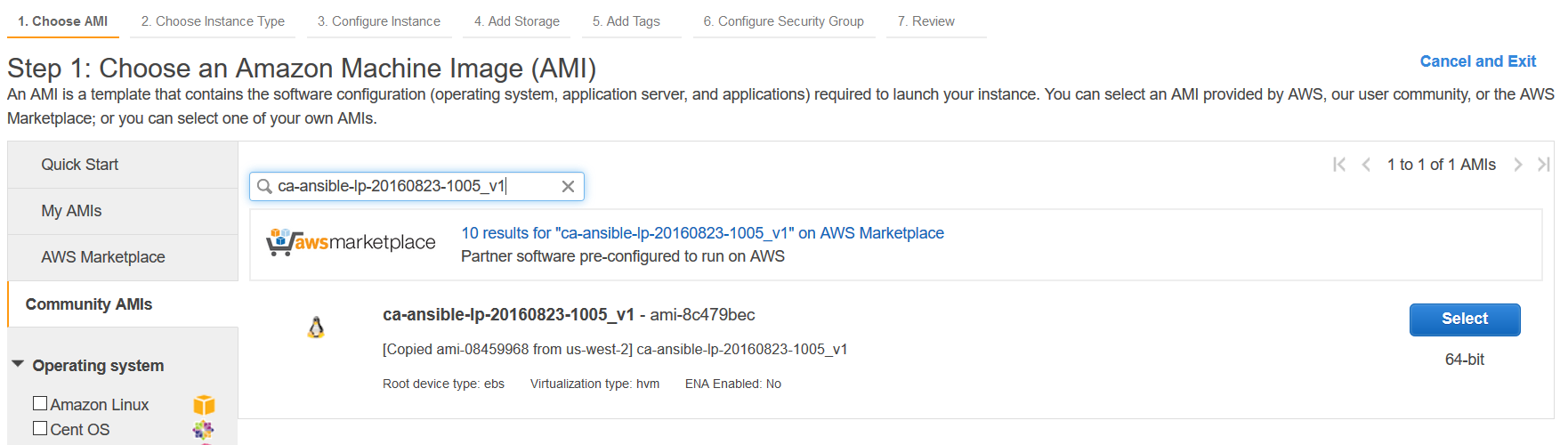
3. Select the **US East (Ohio)** region using the upper right drop-down menu on the AWS Management Console:



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 East (Ohio)**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

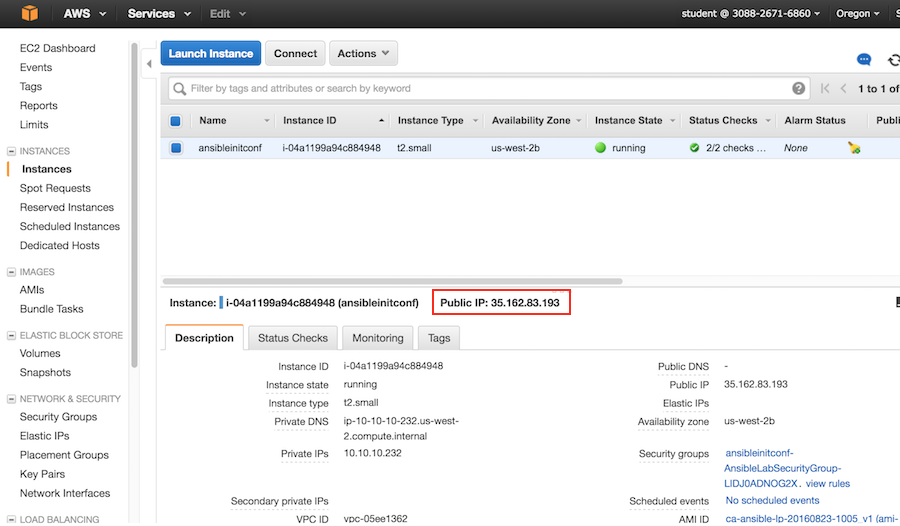
In order to manage a remote Linux server, you must employ an **SSH Client**. Secure Shell (SSH) is a cryptographic network protocol for securing data communication. It establishes a secure channel over an insecure network. Common applications include remote command-line login and remote command execution.

### Connect using Linux / Mac OS

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

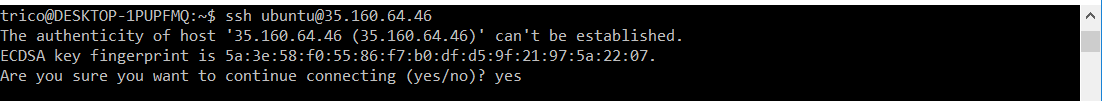
Starting a remote SSH session is easy:

* 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

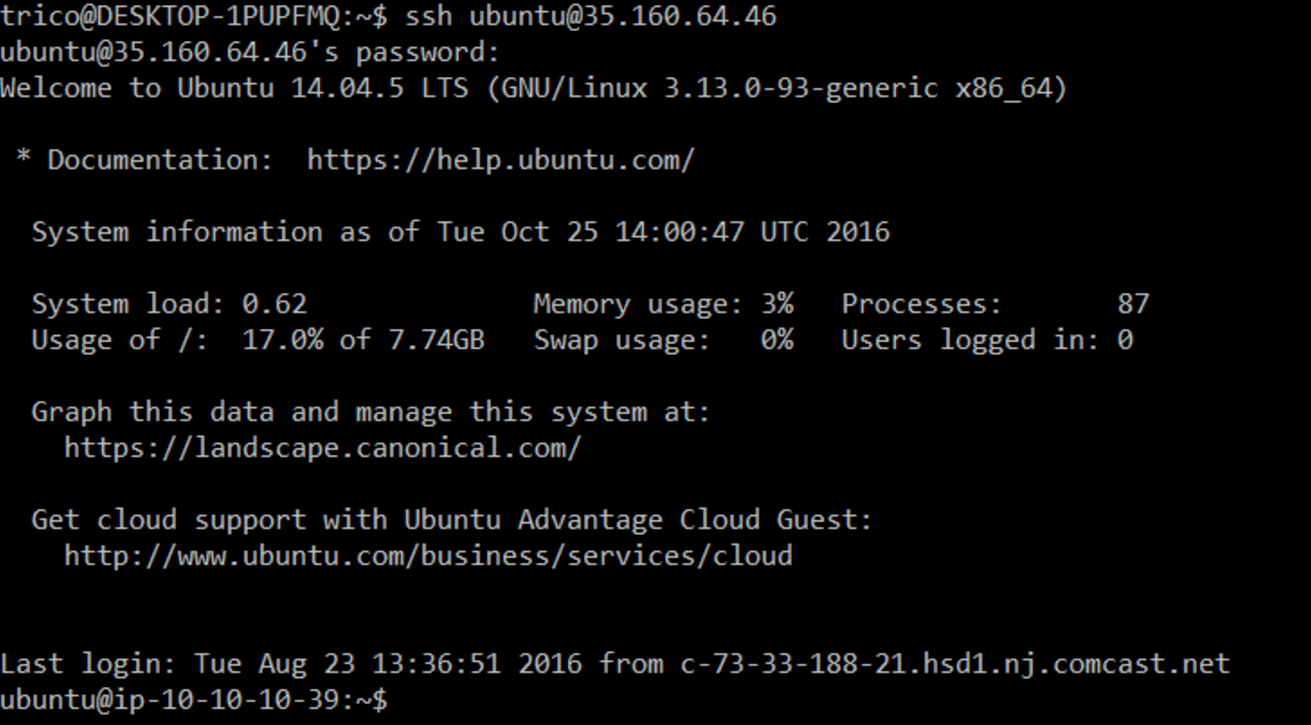


When prompted to verify that you want to connect, write the word: yes

And then press the enter/return key.

[](https://assets.cloudacademy.com/bakery/media/uploads/lab-step/blobid1-e557e56f-1008-4386-a062-2644eb0555c2.png)

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

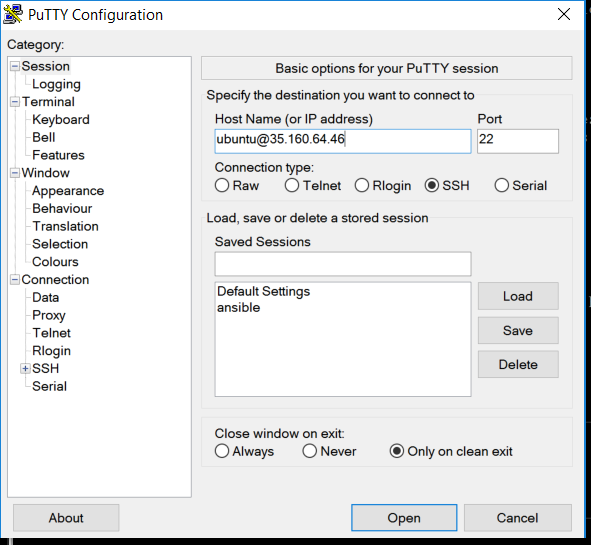


**Connect using Windows**

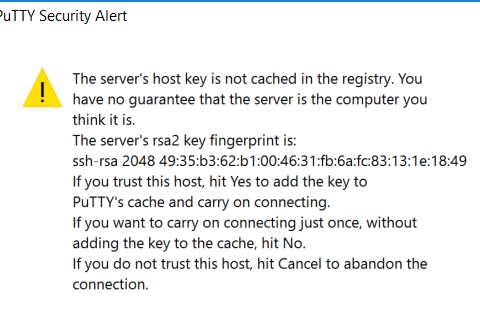
Windows has no SSH client, so you must use PuTTY and convert the PEM key to PPK using PuTTYgen.

Starting a remote SSH session using PuTTY is easy:

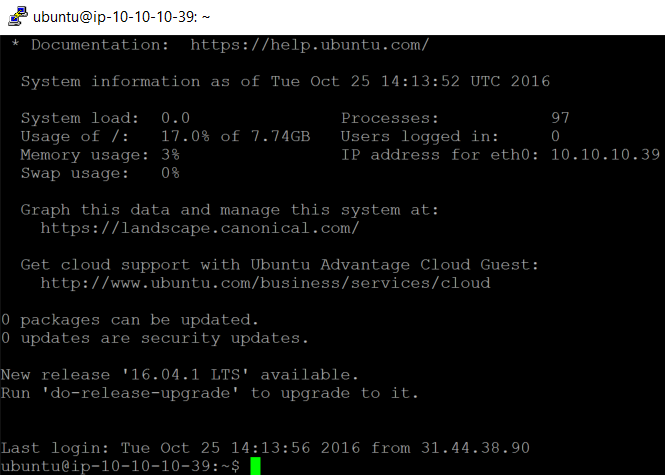
* Open PuTTY and insert ubuntu@"EC2 instance IP Address" in the Host Name field:



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



You will be connected to the EC2 instance:



## Creating an Ansible role to install a LAMP stack

In the lab “[Getting Started with Ansible](https://cloudacademy.com/amazon-web-services/labs/getting-started-ansible-48)” everything was put into one playbook. Which works, however, it makes it more difficult to reuse logic. So now it’s time to start breaking things out into roles.

First, you need to create the role directory structure. An easy way to do this is to use the “ansible-galaxy init” command. Galaxy is outside the scope of this lab, however click [here](https://galaxy.ansible.com/intro) if you want to learn more.

Starting in the home directory, create a new directory named “roles” and cd into it.

**$ cd ~ && mkdir roles && cd roles**

Then run the ansible-galaxy init command, passing in the name of the role.

**$ ansible-galaxy init lamp**

You should see the following output

- lamp was created successfully

To look at the directory structure that Galaxy created run the find command. This is a well established structure for creating roles.

**$ find lamp**

lamp  
lamp/templates  
lamp/files  
lamp/.travis.yml  
lamp/handlers  
lamp/handlers/main.yml  
lamp/tasks  
lamp/tasks/main.yml  
lamp/README.md  
lamp/tests  
lamp/tests/inventory  
lamp/tests/test.yml  
lamp/vars  
lamp/vars/main.yml  
lamp/meta  
lamp/meta/main.yml  
lamp/defaults  
lamp/defaults/main.yml

These files are just placeholders for the logic. So it’s time to start adding it. Open the main.yml file for the tasks.

**$ nano lamp/tasks/main.yml**

You should see something like the following:

---  
# tasks file for lamp

Start by removing the placeholder text. Place your cursor at the top of the file if it isn’t already and press CTRL+K until the file is empty.

Next copy the code below, paste it into the file and save it.

---  
- name: Install our packages  
  apt:  
    name: "{{ item }}"  
    state: present  
    update\_cache: true  
  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

The above logic will install any packages with apt, ensure the services are running, enable a couple Apache modules and then restart Apache. Notice there isn’t a “hosts” property; that’s because tasks are only concerned with the things they need to install; playbooks are concerned with which host to run the task on.

Now you have a role that can install the LAMP stack. However you also need a playbook to run that role. For that create a new file in your home directory.

**$ nano ~/app.yml**

 Copy the code below, paste it into the app.yml file and save it.

---  
- hosts: localhost  
  gather\_facts: false  
  connection: local  
  become: yes  
  
  vars:  
    packages:  
      - apache2  
      - mysql-server  
      - mysql-common  
      - mysql-client  
      - libapache2-mod-wsgi  
    services:  
      - apache2  
      - mysql  
  roles:  
    - lamp

The "roles" property is the key here. It tells the playbook to include your lamp role.  Ansible will know to import the main.yml file from the tasks folder of your lamp role; it's a part of the Ansible convention.

Now it’s time to test things out. Run the playbook with the following command:

**$ ansible-playbook ~/app.yml**

You should see several lines of output. However the final lines should be similar to this:

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**localhost**      : **ok=6**    **changed=3**    unreachable=0    failed=0

So far you’ve created a role that will install a LAMP stack. As you can imagine you could break it out further, and have it install the web server as its own role, as well as installing the database as a role. And the more granular, the more reusable it is.

In the next step, you’ll be removing the variables from the app.yml file and adding them as default variables in your LAMP role.

## Setting default variables and using notifications

Most of the time you’ll know what the initial variables for a role should be, and that’s why you can set the defaults.

To do that open up the app.yml file and cut the packages and services sections.

**$ nano ~/app.yml**

Cut the variables section then close and save the file. The app.yml file should look like the following:

---  
- hosts: localhost  
  gather\_facts: false  
  connection: local  
  become: yes  
  
  roles:  
    - lamp

Now add the variables to the defaults file.

**$ nano ~/roles/lamp/defaults/main.yml**

Add the below variables and save the file:

packages:  
  - apache2  
  - mysql-server  
  - mysql-common  
  - mysql-client  
services:  
  - apache2  
  - mysql

Now run the app playbook to test it.

**$ ansible-playbook ~/app.yml**

If everything is working correctly then you’ll see several lines of output. The last lines should be similar to the following:

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**localhost**      : ok=5    **changed=3**    unreachable=0    failed=0

The last change to make before deploying the application is to configure an Apache handler.  This will allow Ansible to restart Apache when a change is made that requires a service restart before it takes effect.

Start by opening the file:

**$ nano ~/roles/lamp/handlers/main.yml**

Then paste in the following:

- name: Restart Apache  
  service:  
    name: apache2  
    state: restarted

This will replace the last task in the lamp/tasks/main.yml file.

So let’s remove it. Open the file:

**$ nano ~/roles/lamp/tasks/main.yml**

Remove the “Restart Apache” task from the bottom of the file. And add a “notify” property to the final shell command. The /lamp/tasks/main.yml file should look like the following:

---  
- 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  
 **notify: Restart Apache**

With all of that saved, run the playbook again and make sure everything works.

**$ ansible-playbook ~/app.yml**

You should see a line somewhere in the output indicating that the handler ran successfully.

**RUNNING HANDLER [lamp : Restart Apache] \*\*\*\*\*\*\*\*\*\*\*\*\***

**changed: [localhost]**

What you’ve done so far is to add the default variables to the defaults/main.yml allowing anyone to use the lamp role without needing to specify the packages and services. However, allowing someone to override the defaults if they want to.

You’ve also created the a handler to allow for the Apache service to be restarted if a task calls the notifier.

In the next step you’ll be deploying a web application that will use the web server, and MySQL database the lamp role installed.

**Deploying a web application**

Now that you have the LAMP stack installed, it’s time to actually deploy an application. This application is written in Python and will fetch some data from the MySQL database that the LAMP role installed. The application is built with Flask, a lightweight web framework, and uses SQLAlchemy to interact with the MySQL database.

The application code lives on Bitbucket, and the playbook will use the git module to download it.

To complete this step you’ll need to create a new role that will contain the logic for deploying the application. It will have three task files, one for creating a database user and table, another task for downloading the application code, and the final task will configure the web server to use the application code.

The first step is to create a new role. This is the same way we created the lamp role.

**$ cd ~/roles && ansible-galaxy init webapplication**

 Next setup the default variables in the ~/roles/webapplication/defaults/main.yml file. These variables can be used anywhere in your role, including tasks, handlers and templates.

**$ nano ~/roles/webapplication/defaults/main.yml**

The file should look like the following:

---  
app\_download\_dest: /tmp/webapp  
app\_dest: /var/www/webapp  
app\_repo: https://toorroot@bitbucket.org/toorroot/ansible\_demo.git

Now you’re ready to start configuring the three task files. Start by creating the files with the following commands:

**$ touch ~/roles/webapplication/tasks/database.yml**

**$ touch ~/roles/webapplication/tasks/app.yml**

**$ touch ~/roles/webapplication/tasks/site.yml**

Next these files need to be populated before editing the ~/roles/webapplication/tasks/main.yml file to include them all.

The database.yml creates any users or tables that the application expects.

Open the database.yml file:

**$ nano ~/roles/webapplication/tasks/database.yml**

 Paste the following into the database.yml file:

- apt: name=python-mysqldb state=present  
- mysql\_user: name=appuser password=94nfsUl7 priv=\*.\*:ALL state=present  
- mysql\_db: name=appdata state=present

The app.yml file in the tasks folder is responsible for downloading the application and installing any required libraries with the Python package manager named pip.

Open the app.yml file:

**$ nano ~/roles/webapplication/tasks/app.yml**

 Paste the following into the app.yml file:

- apt: name=libmysqlclient-dev state=present  
  
- apt: name=python-pip state=present  
  
- git: repo={{app\_repo}}  
       dest="{{app\_download\_dest}}"  
  
- pip: requirements={{app\_download\_dest}}/app/requirements.txt  
  
- copy: src={{app\_download\_dest}}/app/ dest={{app\_dest}}

Notice the syntax to include variables is to include the variable name in double curly braces. These are the variables you set in the defaults.

The site.yml is responsible for configuring the web server to use the Python application. However before you can configure the YAML file you need to create a template for the Apache config file.

Create and open an Apache config template:

**$ nano ~/roles/webapplication/templates/apache.conf**

Paste in the following and save:

<VirtualHost \*>  
    ServerName **{{inventory\_hostname}}**    WSGIDaemonProcess webapp user=ubuntu group=ubuntu threads=5  
    WSGIScriptAlias / **{{app\_dest}}**/wsgi.py  
  
    <Directory **{{app\_dest}}**>  
        WSGIProcessGroup webapp  
        WSGIApplicationGroup %{GLOBAL}  
        Order deny,allow  
        Allow from all  
    </Directory>  
</VirtualHost>

Notice the use of variables again here in the template. You may recall that you set the defaults for this role and it didn’t include the “**inventory\_hostname**.” And that’s because that is an Ansible provided variable. In this case, it will say “localhost” in the rendered template. While facts are out of scope for this lab, if you were to enable fact gathering you’d have access to even more variables.

Now you’re ready to add the site.yml logic, which will use the template you just created. The idea is that the template will be processed by Ansible to replace any variables, and then it will save it to the destination you specify.

Open the site.yml file:

**$ nano ~/roles/webapplication/tasks/site.yml**

 Paste the following into the site.yml file:

- apt: name=libapache2-mod-wsgi state=present  
  
- name: Copy the apache configuration file  
  template:  
    src: "apache.conf"  
    dest: /etc/apache2/sites-available/000-default.conf  
  notify: "Restart Apache"

Now you can edit the main.yml file so it includes these three task files. By doing this you ensure that Ansible will include the tasks you just created when someone uses this role.

Open the main.yml file:

**$ nano ~/roles/webapplication/tasks/main.yml**

Paste the following into the site.yml file:

- include: database.yml  
- include: app.yml  
- include: site.yml

That completes the role, however the main playbook ~/app.yml still needs to know to run that role.

Open up the app.yml file:

**$ nano ~/app.yml**

The content should look like the following:

---  
- hosts: all  
  gather\_facts: false  
  connection: local  
  become: yes  
  
  roles:  
    - lamp  
 **- webapplication**

The next step is to run the playbook and make sure everything runs.

**$ ansible-playbook -i localhost, ~/app.yml**

Now, if everything was successful you can test this out by opening a new browser tab, and pasting the public IP address into the URL 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 simple Python application page.

[](https://assets.cloudacademy.com/bakery/media/uploads/lab-step/blobid0-08a1a9c4-2f00-44bb-ab54-6258602528fc.png)

Here's a recap of what you accomplished in this lab:

* You created a reusable role to install a LAMP stack
* You created a role to install a Python web application on top of the LAMP stack
  + You created a task to create a database table and a database user
  + You downloaded some source code from a Github repo
  + You installed some dependencies with pip - a python package manager
  + You created a template to handle the Apache config file with some variables that pull from both Ansible and the defaults for the role
  + You triggered a notification to restart Apache