Lab 1. Getting Started with Ansible

In this lab, we will begin to teach you the practical skills to cover the very fundamentals of Ansible, starting with how to install Ansible. We'll then look at node requirements and how to validate your Ansible installation.

In this lab, we will cover the following topics:

- Installing and configuring Ansible
- Understanding your Ansible installation

Lab Environment

All lab file are present at below path. Run following command in the terminal first before running commands in the lab:

`cd ~/Desktop/ansible-course/Lab_1`

Installing Ansible on Linux and FreeBSD

The following are some examples showing how you might install Ansible on Ubuntu:

- **Installing Ansible on Ubuntu**: To install the latest version of the Ansible control machine on Ubuntu, the [apt] packaging tool makes it easy using the following commands:

Ansible has been installed already using apt-get. Below instructions are for information only.

```
$ sudo apt-get update
$ sudo apt-get install software-properties-common
$ sudo apt-add-repository --yes --update ppa:ansible/ansible
$ sudo apt-get install ansible
```

- **Installing Ansible via the Python package manager (pip)**: To install Ansible via [pip], use the following command:

```
sudo pip3 install ansible
```

Now that you have installed Ansible using your preferred method, you can run the [ansible] command as before, and if all has gone according to plan, you will see output similar to the following:

```
$ ansible --version
ansible 2.9.6
  config file = None
  configured module search path = ['/Users/james/.ansible/plugins/modules',
'/usr/share/ansible/plugins/modules']
  ansible python module location =
/usr/local/Cellar/ansible/2.9.4_1/libexec/lib/python3.8/site-packages/ansible
  executable location = /usr/local/bin/ansible
  python version = 3.8.1 (default, Dec 27 2019, 18:05:45) [Clang 11.0.0 (clang-
1100.0.33.16)]
```

If you want to update your Ansible version, [pip] makes it easy via the following command:

```
$ sudo pip3 install ansible --upgrade
```

Understanding how Ansible connects to hosts

Let's focus on the INI formatted inventory. An example is shown here with four servers, each split into two groups. Ansible commands and playbooks can be run against an entire inventory (that is, all four servers), one or more groups (for example, [webservers]), or even down to a single server:

```
[webservers]
web1.example.com
web2.example.com

[apservers]
ap1.example.com
ap2.example.com
```

We will use lab environment machine for all hosts used in the lab guide.

Note:

- Check entry for each host used in the lab exists in `/etc/hosts` file: `127.0.0.1 hostname`
- Check `/etc/ansible/hosts` file and verify that above entries exist.

Let's use this inventory file along with the Ansible [ping] module, which is used to test whether Ansible can successfully perform automation tasks on the inventory host in question. The following example assumes you have installed the inventory in the default location, which is normally [/etc/ansible/hosts]. When you run the following [ansible] command, you see a similar output to this:

```
$ ansible webservers -m ping
web1.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
web2.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
$
```

Notice that the [ping] module was only run on the two hosts in the [webservers] group and not the entire inventory -- this was by virtue of us specifying this in the command-line parameters.

Verifying the Ansible installation

In this section, you will learn how you can verify your Ansible installation with simple ad hoc commands.

As we discussed in the previous section, we must also define an inventory for Ansible to run against. Another simple example is shown here:

```
[frontends]
frt01.example.com
frt02.example.com
```

Following are three simple examples that demonstrate ad hoc commands---they are also valuable for verifying both the installation of Ansible on your control machine and the configuration of your target hosts, and they will return an error if there is an issue with any part of the configuration:

- **Ping hosts**: You can perform an Ansible "ping" on your inventory hosts using the following command:

```
$ ansible frontends -i hosts -m ping
```

- **Display gathered facts**:** **You can display gathered facts about your inventory hosts using the following command:

```
$ ansible frontends -i hosts -m setup | less
```

- **Filter gathered facts**: You can filter gathered facts using the following command:

```
$ ansible frontends -i hosts -m setup -a "filter=ansible_distribution*"
```

Note: Check `/root/hosts` file and verify that host information entries exist (because `-i hosts` path is specified so ansible will look in `~/hosts` file)

For every ad hoc command you run, you will get a response in JSON format---the following example output results from running the [ping] module successfully:

```
$ ansible frontends -m ping
frontend01.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
frontend02.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Ansible can also gather and return "facts" about your target hosts---facts are all manner of useful information about your hosts, from CPU and memory configuration to network parameters, to disk geometry. These facts are intended to enable you to write intelligent playbooks that perform conditional actions---for example, you might only want to install a given software package on hosts with more than 4 GB of RAM or perhaps perform a specific configuration only on macOS hosts. The following is an example of the filtered facts from a macOS-based host:

```
$ ansible localhost -m setup -a "filter=ansible_distribution*"

localhost | SUCCESS => {
    "ansible_facts": {
        "ansible_distribution": "Ubuntu",
        "ansible_distribution_file_parsed": true,
        "ansible_distribution_file_path": "/etc/os-release",
        "ansible_distribution_file_variety": "Debian",
        "ansible_distribution_major_version": "20",
        "ansible_distribution_release": "focal",
        "ansible_distribution_version": "20.04"
    },
    "changed": false
}
```

Ad hoc commands are incredibly powerful, both for verifying your Ansible installation and for learning Ansible and how to work with modules as you don't need to write a whole playbook---you can just run a module with an ad hoc command and learn how it responds. Here are some more ad hoc examples for you to consider:

- Copy a file from the Ansible control host to all hosts in the [frontends] group with the following command:

```
$ ansible frontends -m copy -a "src=/etc/hosts dest=/root/Desktop/hosts"
```

- Create a new directory on all hosts in the [frontends] inventory group, and create it with specific ownership and permissions:

```
$ ansible frontends -m file -a "dest=/path/user1/new mode=777 owner=user1 group=user1
state=directory"
```

- Delete a specific directory from all hosts in the [frontends] group with the following command:

```
$ ansible frontends -m file -a "dest=/path/user1/new state=absent"
```

- Install the [apache2] package with [apt] if it is not already present---if it is present, do not update it. Again, this applies to all hosts in the [frontends] inventory group:

```
$ ansible frontends -m apt -a "name=apache2 state=present"
```

- The following command is similar to the previous one, except that changing [state=present] to [state=latest] causes Ansible to install the (latest version of the) package if it is not present, and update it to the latest version if it is present:

```
$ ansible frontends -m apt -a "name=demo-tomcat-1 state=latest"
```

- Display all facts about all the hosts in your inventory (warning---this will produce a lot of JSON!):

```
$ ansible all -m setup
```

Now that you have learned more about verifying your Ansible installation and about how to run ad hoc commands, let's proceed to look in a bit more detail at the requirements of the nodes that are to be managed by Ansible.

Managed node requirements

If your target hosts are lacking Python, it is usually easy to install it through your operating system's package management system. Ansible requires you to install either Python version 2.7 or 3.5 (and above) on both the Ansible control machine (as we covered earlier in this lab) and on every managed node.

- On Debian and Ubuntu systems, you would use the [apt] package manager to install Python, again specifying a version if required (the example given here is to install Python 3.6 and would work on Ubuntu 18.04):

```
$ sudo apt-get update
$ sudo apt-get install python3.6
```

The following are some examples of tasks in an Ansible playbook that you might use to bootstrap a managed node and prepare it for Ansible management:

```
- name: Bootstrap a host without python2 installed
  raw: dnf install -y python2 python2-dnf libselinux-python

- name: Run a command that uses non-posix shell-isms (in this example /bin/sh doesn't
handle redirection and wildcards together but bash does)
```

```
  raw: cat < /tmp/*txt
  args:
    executable: /bin/bash

- name: safely use templated variables. Always use quote filter to avoid injection
issues.
  raw: "{{package_mgr|quote}} {{pkg_flags|quote}} install {{python|quote}}"
```

We have now covered the basics of setting up Ansible both on the control host and on the managed nodes, and we have given you a brief primer on configuring your first connections. Before we wrap up this lab, we will look in more detail at how you might run the latest development version of Ansible, direct from GitHub.

Running from source versus pre-built RPMs

Let's get started by checking out the very latest version of the source code from GitHub:

1. You must clone the sources from the [git] repository first, and then change to the directory containing the checked-out code:

```
$ git clone https://github.com/ansible/ansible.git --recursive
$ cd ./ansible
```

2. Before you can proceed with any development work, or indeed to run Ansible from the source code, you must set up your shell environment. Several scripts are provided for just that purpose, each being suitable for different shell environments. For example, if you are running the venerable Bash shell, you would set up your environment with the following command:

```
$ source ./hacking/env-setup
```

3. Once you have set up your environment, you must install the [pip] Python package manager, and then use this to install all of the required Python packages (note: you can skip the first command if you already have [pip] on your system):

```
$ sudo pip3 install -r ./requirements.txt
```

Note that, when you have run the [env-setup] script, you'll be running from your source code checkout, and the default inventory file will be [/etc/ansible/hosts]. You can optionally specify an inventory file other than [/etc/ansible/hosts].

4. When you run the [env-setup] script, Ansible runs from the source code checkout, and the default inventory file is [/etc/ansible/hosts]; however, you can optionally specify an inventory file wherever you want on your machine. The following command provides an example of how you might do this, but obviously, your filename and contents are almost certainly going to vary:

```
$ echo "ap1.example.com" > ~/my_ansible_inventory
$ export ANSIBLE_INVENTORY=~/my_ansible_inventory
```

Once you have completed these steps, you can run Ansible exactly as we have discussed throughout this lab, with the exception that you must specify the absolute path to it. For example, if you set up your inventory as in the preceding code and clone the Ansible source into your home directory, you could run the ad hoc [ping] command that we are now familiar with, as follows:

```
$ ~/ansible/bin/ansible all -m ping
ap1.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Of course, the Ansible source tree is constantly changing and it is unlikely you would just want to stick with the copy you cloned. When the time comes to update it, you don't need to clone a new copy; you can simply update your existing working copy using the following commands (again, assuming that you initially cloned the source tree into your home directory):

```
$ git pull --rebase
$ git submodule update --init --recursive
```

That concludes our introduction to setting up both your Ansible control machine and managed nodes. It is hoped that the knowledge you have gained in this lab will help you to get your own Ansible installation up and running and set the groundwork for the rest of this course.

Summary

Ansible is a powerful and versatile yet simple automation tool, of which the key benefits are its agentless architecture and its simple installation process. Ansible was designed to get you from zero to automation rapidly and with minimal effort, and we have demonstrated the simplicity with which you can get up and running with Ansible in this lab.

In the next lab, we will learn Ansible language fundamentals to enable you to write your first playbooks and to help you to create templated configurations and start to build up complex automation workflows.

Questions

1. On which operating systems can you install Ansible? (Multiple correct answers)

A) Ubuntu

B) Fedora

C) Windows 2019 server

D) HP-UX

E) Mainframe

2. Which protocol does Ansible use to connect the remote machine for running tasks?

A) HTTP

B) HTTPS

C) SSH

D) TCP

E) UDP

3. To execute a specific module in the Ansible ad hoc command line, you need to use the [-m] option.

A) True

B) False