SWEN 438 *Special Topic:DevOps*

Laboratory 1: Ansible, Vagrant and VirtualBox

The purpose of this laboratory is to set up a working copy of Ansible, along with VirtualBox and Vagrant for creating virtual machines for experiments with configuration management. These laboratory instructions should work on any platform, but have only been tested on macOS and Linux.

Installation

This laboratory has been tested with the following versions of the required packages

Software	macOS	Ubuntu	Arch Linux
`ansible`	2.11.2	2.10.7	2.10.11
`ansible-lint`	5.1.2	4.3.7-1	5.0.12
`vagrant`	2.2.17	2.2.9	2.2.16
`virtualbox`	6.1.22	6.1.22	6.1.22

ECS Workstations

If you are using one of the ECS workstations to complete this lab then all of the required packages have already been installed. The ECS workstations use Arch Linux.

Ubuntu and Debian

These popular (and related — Ubuntu is based on Debian) Linux distributions and use `apt` (Advanced Packaging Tool) to install additional software on the system. Debian is a stable Linux distribution suitable for enterprise use but the packages available via `apt` tend to be several versions behind the latest releases, requiring manual intervention to resolve incompatibilities and dependencies when a later version of a package is installed. We will instead use the most recent release, Ubuntu 21.4 *Hirsute Hippo*, to avoid these problems, noting that Ubuntu is becoming popular for cloud computing due to its support for OpenStack https://www.openstack.org/.

Install the software necessary for this laboratory with:

```
sudo apt install ansible ansible-lint vagrant virtualbox
```

Windows 10

Windows Subsystem for Linux (WSL) https://docs.microsoft.com/en-us/windows/wsl/install-win10 *should* allow you to complete this laboratory. The default WSL Linux distribution is Ubuntu, so the instructions for Ubuntu should work under WSL.

Alternately, some students have reported successful completion of this lab using Hyper-V to create a VM with Ubuntu Server 20.04 - within which Ansible, Vagrant and VirtualBox were installed and the lab completed. For Hyper-V installation instructions see: https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/enable-hyper-v. The following command needed to be executed in the Administrator Powershell in order for the to work

```
C:\WINDOWS\System-32> Get-VM | where name -eq "virtualmachinename" | SetVMProcessor -Expose\
```

where `virtualmachinename` should be replaced with the name given to the Ubuntu VM set up with Hyper-V.

macOS

Use Homebrew https://brew.sh/ as your package manager. If you do not already have Homebrew installed then backup your system and install Homebrew following the instructions on the preceding page.

```
brew install ansible ansible-lint vagrant virtualbox
```

Directory Setup

For this first lab, because it's so simple, there are few(er) constraints. Create a directory and `cd` into it:

```
mkdir Laboratory_1 && cd Laboratory_1
```

Note: This directory will need be zipped and submitted when you have completed this lab.

VirtualBox

Note: VirtualBox creates Virtual Machine Disk (VMDK) files, some of which contain the virtual hard disk drive used in a virtual machine. Virtual hard disk drives can be large, the drives generated by this laboratory are around 2.5–3.0 GB in size. For this laboratory the VMDK files can be re-generated within a few minutes

by VirtualBox and Vagrant, so there is no reason to save, copy or backup the VMDK files. By default, VirtualBox saves these in `~/VirtualBox VMs`. If you are working on one of the School workstations then you will likely want to change the `Default Machine Folder` to `/local/scratch` (see: Using `/local/scratch` on ArchLinux Workstations). Do so by launching VirtualBox (from the command line run `virtualbox &`) and clicking on the `Preferences` icon in the GUI or pressing `Ctrl-G` (`Cmd-G` on macOS). In the `General` tab you will see the `Default Machine Folder` field.

Vagrant

On the command line of your local machine run `vagrant init ubuntu/hirsute64`. You should see:

```
$ vagrant init ubuntu/hirsute64
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
```

The box will be installed and the VM started with the command

```
vagrant up
```

You can confirm that it is running by executing

```
vagrant status
```

and confirm you can connect to the VM with `vagrant ssh`, noting that SSH keys are created and managed for you by Vagrant. You should see

```
$ vagrant ssh
Welcome to Ubuntu 21.04 (GNU/Linux 5.11.0-22-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Tue Jul 20 19:53:53 UTC 2021

  System load:  0.08              Processes:               114
  Usage of /:   5.7% of 38.71GB   Users logged in:         0
  Memory usage: 60%               IPv4 address for enp0s3: 10.0.2.15
  Swap usage:   0%                IPv4 address for enp0s8: 192.168.33.20

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

0 updates can be applied immediately.

Last login: [datestamp] from [IP address]
vagrant@ubuntu-hirsute:~$
```

Once the VM is confirmed operational, we need to make some changes. Log out of the VM (using either the `logout` command or `Ctrl-d`) to return to the command line of your local machine. Then run `vagrant halt` followed by `vagrant destroy` to remove the VMDK file, or combine them as

```
vagrant halt && vagrant destroy
```

Change the Default Virtual Machine Configuration

`vagrant init` created a default file called `Vagrantfile` in the working directory which contains some helpful documentation. We need to change some settings in this file. Open `Vagrantfile` in your preferred text editor and:

1. find the section which controls the VM memory and set `vb.memory` to 1024
2. find the section which allocates a provisioners and set the `config.vm.provision` to

```
config.vm.provision "ansible" do |ansible|
  ansible.playbook = "provisioning/playbook.yml"
end
```

Note: this seeks to run the Ansible installed on the local machine. If, for whatever reason, you don't have Ansible installed locally, then Vagrant can take care of installing Ansible on the VM and running it there, by specifying `config.vm.provision "ansible_local" ...` in the `Vagrantfile`.

Running `vagrant up` will now create *and* provision your VM, although the provisioning step will fail because `provisioning/playbook.yml` does not exist. We'll create that next.

Some useful commands:

- `vagrant up --no-provision` will create & run the VM but skip the provisioning specified in `config.vm.provision`
- `vagrant provision` will [re-]provision an already-running VM (what happens when you try this on a halted VM?)

Run `vagrant help` to see more commands, and the Vagrant Documentation https://www.vagrantup.com/docs for full details.

Ansible

Create an empty Playbook with

```
mkdir provisioning
touch provisioning/playbook.yml
```

and open it in your preferred editor.

Let's add a *very* simple task which pings the VM and shows the structure of a Playbook:

```
---
- hosts: all
  tasks:
    - name: Make sure we can connect to the VM
      ping:
```

See the Ansible documentation for a full explanation of these keywords.

Run `vagrant provision` and read Ansible's output to confirm that the `ping` succeeded.

Install PHP

We'll need PHP later, so let's make sure we know how to install it now by adding a task to the Playbook with the `tasks:` block:

```
    - name: Install PHP
      apt: name=php-cli state=present update_cache=yes
```

If you try to run this now, you will receive a permissions-related error. We need to bump our permissions by adding `become: true` at the same level as the `tasks:` block which allows us to install as the administrator. The start of your playbook should look like this:

```
---
- hosts: all
  become: true
  tasks:
    ...
```

Confirm that the playbook runs without problems by running `vagrant provision`.

The `k=v` syntax is hard to read, so rewrite the `apt` command using YAML *nested mapping* syntax:

```
      apt:
        name: php-cli
        state: present
        update_cache: yes
```

Again, confirm this installs without problems by running `vagrant provision`.

Note: Nested mappings are the required format for all tasks from this point.

Install nginx and MySQL

Using the task syntax above, add a separate task to install `nginx` using `apt`, using the following:

- Task name should be "Install nginx"
- Package name passed to `apt` must be `nginx`
- State must be `present`

Similarly, add a separate task to install `mysql`:

- Task name should be "Install MySQL"
- Package name passed to `apt` must be `mysql-server`
- State must be `present`

Confirm these install without problems by running `vagrant provision`, debugging any problems reported by Ansible.

You can also check everything is installed by connecting to your VM with `vagrant ssh` and running in sequence:

```
which php
which nginx
which mysqld
```

Iterating Over Items

It's rather inefficient to specify a task for each `apt` call, there's a lot of "boilerplate". The old, now deprecated, way to handle this was to use a `with_items` loop to iterate over a list of packages to install:

```
    - name: Install required packages
      apt:
        name: "{{item}}"
        ...
      with_items:
        - ...
        - ...
        ...
```

You may rewrite the three tasks to use a *single* task in this form. The ellipsis represent parts unchanged from before, the `with_items` block will be a YAML list of the packages you would like to install. Confirm these install without problems by running `vagrant provision`, debugging any problems reported by Ansible.

On ECS workstations, the `/etc/ansible/ansible.cfg` file indicates that `deprecation_warnings = True` is the default, and if `with_items` loop is used then a warning will be displayed:

```
[DEPRECATION WARNING]: Invoking "apt" only once while using a loop via
squash_actions is deprecated. Instead of using a loop to supply multiple items
and specifying `name: "{{ item }}"`, please use `name: ['php-cli', 'nginx',
'mysql-server']` and remove the loop. This feature will be removed from
ansible-base in version 2.11. Deprecation warnings can be disabled by setting
deprecation_warnings=False in ansible.cfg.
```

You may rewrite the `with_items` loop to use a supported syntax which will not break when the `with_items` loop is removed from ansible-base. Using the array syntax described in the warning is satisfactory, but will be difficult to read when the number of items grows. The YAML *nested mapping* syntax is preferred for consistency:

```
- name: Install required packages
  apt:
    name:
      - ...
      - ...
    state: ...
    ...
```

Note: it is acceptable to submit a working playbook with either the deprecated `with_items` loop or a modern supported syntax. There will be no disadvantage for this lab to using either syntax.

Run the `ansible-lint` command in your `Laboratory_1` directory to check your playbook (no parameters need to be passed) You may see some style violations flagged, but there will be no output if your playbook passes linting. For full marks, fix any violations flagged by `ansible-lint` .

Idempotency

Re-running the playbook by `vagrant provision` will make no changes if the packages are already installed, as would be expected from Ansible's *idempotency* paradigm. Use `vagrant halt && vagrant destroy` followed by `vagrant up` to see provisioning a "fresh" box and compare the output from Ansible when `vagrant provision` is run. This will help you understand how Ansible enforces the configuration defined in the `playbook.yml` file.

Submission and Marking

For this first lab, please zip your `Laboratory_1` directory and submit it via the ECS Submission system https://apps.ecs.vuw.ac.nz/submit/SWEN438/Laboratory_1.

Full marks will be earned for meeting the following criteria:

1. `vagrant up` creates the VM and provisions without error
2. `provisioning/playbook.yml` uses YAML *nested mapping* and `with_items` syntax
3. `ansible-lint` flags no violations
