SWEN 438 *Special Topic:DevOps*

Assignment 2: Ansible

The second assignment is to write an Ansible playbook which installs WordPress on a virtual machine, and post to your blog about your experience.

To accomplish this, you are required to work through Chapters 2, 3 and 4 of *Ansible: From Beginner to Pro*, available from the Reading List, and to commit your playbook to version control at key points during the exercise. The book gives full and detailed instructions on constructing the playbook and configuring the required software packages but, because the instructions have not been updated since the book was published, some parts are obsolete — simply copying the code from the book will not yield a functioning playbook. Instructions on how to update the code to correct the problems are given below.

This assignment builds indirectly on Laboratory 1 (which is based on *Ansible: From Beginner to Pro* Chapter 1) and it is recommended that you complete that laboratory before attempting this assignment.

GitLab Project Setup

Create a GitLab Project for this assignment in your sub-group https://gitlab.ecs.vuw.ac.nz/groups/course-work/swen438/2021/username (substitute your ECS username in the URL). This project must be named `Assignment_2`. Clone the repository to your local machine and `cd Assignment_2` to begin work on your playbook.

Additional Instructions

This section provides some additional instructions on how to complete the WordPress exercise from *Ansible: From Beginner to Pro*, as well as corrections to instructions which are now obsolete with Ansible 2.11.

Chapter 2 *The Inventory File*

Do not create a subdirectory, the root level of the repository is the correct place to begin your work.

An `inventory` file needs to be created for this exercise, as one of the **Core** requirements is that `ansible-playbook ...`, which is the typical way of running Ansible, runs without error.

Without an inventory file, Ansible will fail to execute:

```
$ ansible-playbook provisioning/playbook.yml
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that
the implicit localhost does not match 'all'

PLAY [all] *********************************************************************
skipping: no hosts matched

PLAY RECAP *********************************************************************
```

The sample inventory file in the *Running Without Vagrant* section must be re-written in YAML form for this assignment. **Note:** There is a typographical error in the IP address given in this section; the correct IP address is `192.168.33.20` and should be used in your inventory file. For more information on the use of YAML format for inventories see https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html and the *Inventory aliases* section of that page in particular.

You won't be able to test the inventory file with the command `ansible-playbook provisioning/playbook.yml` until after creating the playbook at the beginning of the next chapter, on p.32.

Chapter 3 *Installing WordPress*

To follow good practice (Marking Guide) you should commit to your repository fairly frequently, and put an annotated tag on the commit which completes logical sections. See **Completion** requirement 4 and *Appendix: Tags* for more information.

It is recommended you defer addressing `ansible-lint` https://ansible-lint.readthedocs.io/ warnings until after you have a working WordPress installation, to simplify troubleshooting and debugging.

In this chapter, obsolete `apt` module `state` parameters `installed` and `removed` are sometimes used. Use `state: present` and `state: absent` instead. See https://docs.ansible.com/ansible/latest/collections/ansible/builtin/apt_module.html for full details of `apt` module parameters and values.

The `template`, `file`, `copy` and `unarchive` modules are used in this chapter without an explicitly specified `mode` parameter. These modules will function correctly using their defaults *but* `ansible-lint` will complain:

```
risky-file-permissions: File permissions unset or incorrect
```

To resolve this in order to meet **Completion**, explicitly specify `mode: 644` for files and `mode: 755` for directories. Note that the `copy` module has a special string `preserve` which you can use if you are confident that the source file the correct permissions. Ansible and `ansible-lint` will generally tell

you if you specify the wrong permissions.

Environment Configuration

Disregard the instructions to create an `ansible-wordpress` subdirectory on p. 31, the root level of the repository, where you should already have created an `inventory` file, is the correct place to continue working and creating a `Vagrantfile`.

We will use the most recent stable release of Ubuntu, Ubuntu 21.4 *Hirsute Hippo*, available from Vagrant as `ubuntu/hirsute64` https://app.vagrantup.com/ubuntu/boxes/hirsute64, instead of the older version specified on p. 31. Your `Vagrantfile` should look like this:

```
Vagrant.configure(2) do |config|
  config.vm.box = "ubuntu/hirsute64"
  ...
```

The reason we are using a more recent version is that some packages, which were current with the older version of Ubuntu, are no longer available. Using `ubuntu/hirsute64` avoids these problems.

Installing PHP

PHP 7.4 is available with the `ubuntu/hirsute64` box, so there is no need to use a PPA (Personal Package Archive) for this exercise. Omit the "Add the ondrej PHP PPA" task.

Note: you will find that the "Update the apt cache" task must be included in the playbook, or you may receive confusing error messages about missing packages. See **Challenge 5**.

Installing MySQL

The packages `mysql-server-5.6` and `python-mysqldb` (for Python 2.7) are not available for `ubuntu/hirsute64`, use `mysql-server` and `python3-mysqldb` instead.

In the task "Update root password", you may have to add a parameter to stop `ansible-lint` from complaining about a `no-log-password` rule violation. Follow the instructions in the output and/or the `ansible-lint` documentation to resolve this linting violation.

Installing nginx

The obsolete `state=running` parameter is used with the `service` module. Use `state: started` instead. See the `service` module documentation for full details of the `state` parameter.

Tasks and Handlers

This section recommends editing the `/etc/hosts` file. You will require administrator privileges for this which will not be available to you on ECS machines. This step is optional because it only defines an alias `book.example.com` to the VM's IP address; WordPress on the VM remains available at http://192.168.33.20.

Downloading it Automatically

Even if you intend to meet **Challenge** requirements, it is *strongly* recommended that this section be deferred until after completing Chapter 4.

Configuring a WordPress Install

The "Create project folder" task uses `command: cp ...` for reasons detailed at the bottom of p. 44. The `copy` module now supports copying from a remote source location using the `remote_src` parameter, and you must use the `copy` module. Note that the `creates: /var/.../wp-settings.php` parameter is unnecessary with Ansible 2.11; compare the command module `creates` parameter with the `copy` module `force` parameter default for the reason why this is unnecessary.

Chapter 4 *Ansible Roles*

Use your username when splitting-up the monolithic playbook into roles, i.e. `username.php`, `username.nginx`, etc.

Note: `ansible-lint` will complain about `role-name` rule violations. If you are seeking to meet **Completion** or **Challenge** requirements then in this *one* instance you are allowed to create a `.ansible-lint` file at the top level of the repository with an entry to skip this rule. The output from `ansible-lint` gives excellent guidance on how to configure a skip rule.

Submission

Submit a link to your GitLab Project via the ECS Submission system https://apps.ecs.vuw.ac.nz/submit/SWEN438/Assignment_2. The purpose of this is to indicate to staff that your work is ready for marking.

Remember: you must commit your WordPress DB to your repo at the end, in order to have your blog posts preserved for marking. The process defined in the *Making a Backup* section should allow you to preserve your valuable posts... remember to commit them to version control!

Marking Guide

The core-completion-challenge structure will be used for this assignment.

- Satisfactory performance (C grade) is achieved by meeting the **Core** requirements.
- Very good performance (B+) is reached by meeting **Core** and **Completion** requirements.
- Excellent (A) and outstanding (A+) performances are achieved by meeting **Core**, **Completion** and progressively fulfilling **Challenge** requirements.

Note: some of the **Completion** and **Challenge** requirements are best addressed at the beginning of the assignment rather than at the end, e.g. configuring pre-commit in the repository or configuring commit message push rules in GitLab.

Core

These are the minimum steps to produce a working WordPress installation *without* concerns for style, consistency and good practice.

1. `.gitignore` is configured for the repository as one of the earliest commits.
2. `vagrant up --no-provision` completes without error.
3. `ansible-playbook --inventory inventory provisioning/playbook.yml` completes without Ansible warnings or errors.
4. Visiting http://192.168.33.20 in a web browser shows a correctly functioning WordPress installation displaying the default "Hello world!" blog post.
5. The repository history shows annotated tags (see: Git Basics — Taggin) for each chapter: `ch2-complete`, `ch3-complete`, `ch4-complete`.

Completion

These requirements add basic style, consistency and good practice to the core requirements.

1. pre-commit is configured for the repository as one of the earliest commits, running the following basic hooks from https://github.com/pre-commit/pre-commit-hooks: `trailing-whitespace`; `end-of-file-fixer`; `check-yaml`; `check-added-large-files` with *at most* a 10 MB threshold.
2. `ansible-lint` flags no violations; it is a requirement that a `.ansible-lint` file is configured at the repository top level to skip *only* the `role-name` rule (see the note in the Additional Instructions - Chapter 4 section, above). Apply annotated tag `lint-free` when the playbook passes linting checks.
   Note: It is recommended you defer addressing `ansible-lint` warnings until after you have a working WordPress installation, i.e. until after meeting the **Core** requirements.

3. All Ansible files (`provisioning.yml` , `main.yml` , etc.) use YAML *nested mapping* syntax. The `k=v` and `['...', '...', ]` array syntaxes are deprecated for the purposes of this assignment. Apply annotated tag `all-YAML` when this is complete.
   Note: You *may* find it simpler to use the `k=v` syntax until you have a working WordPress installation, i.e. until after meeting the **Core** requirements, and then change to YAML syntax.

4. The commit history shows logically sensible commits using annotated tags (see *Appendix: Tags* below). Commit messages can be understood by someone else (i.e. the marker). Use of commit message descriptions is encouraged. You will find the blog post "How to Write a Git Commit Message" https://chris.beams.io/posts/git-commit/ helpful.

5. The WordPress site deployed from version control shows a short blog post written by you.Apply annotated tag `completion` when this is done.
   Note: this will require you to save the modified WordPress database to the appropriate place in your provisioning directory and have it committed to version control as one of the final commits to your repository.

Challenge

These are more advanced requirements focussed on good practice, automation and reliability. It is not necessary to complete all challenge tasks. Some challenges are difficult: if you can't solve them within the time allocated don't worry, documenting your attempt and what you learned is most important.

For each challenge you attempt, write a short post in your blog reflecting on what you achieved or learned. **The blog post forms part of the assessment.**

1. Before starting on the exercise, configure git push rules with a regular expression (regexp) of your choosing, to enforce a consistent commit message format. At the same time, configure gitlint https://jorisroovers.com/gitlint/ as a pre-commit hook in `.pre-commit-config.yaml` with the same regexp as your push rule above. This allows per-commit enforcement as well as push-level enforcement. Apply annotated tag `standards` to mark the point from which commit message formatting applied. The challenge here is not technical, it is the challenge of accepting the automatic enforcement of good practice!

2. Can we eliminate some of the manual processes involved in this assignment? Some ideas:

    1. Use the `VBoxManage` command to change the default machine folder rather than doing through the VirtualBox GUI. Could this be added to the playbook or is a separate playbook required?
    2. Are there Vagrant commands https://www.vagrantup.com/docs/cli or other ways which could be used to programatically eliminate the manual editing of the `Vagrantfile` to change the VM memory size and the VM provisioner? Could this be added to the playbook or is a separate playbook required?
    3. The **Core** playbook assumes that the WordPress installation has been manually downloaded.

Implement automatic downloading, as described on p. 42 (Ch. 3). This will require significant changes to roles and handlers. Is there a reason to prefer automatic downloaded to the local machine or to the VM, and why?

Use annotated tag `no-hands` when as many manual processes as are feasible to fix have been eliminated.

3. Change the Vagrantfile so the VM runs Ubuntu 21.10 Impish Indri (development branch), i.e. `ubuntu/impish64` (https://app.vagrantup.com/ubuntu/boxes/impish64), then run `vagrant halt && vagrant destroy` followed by `vagrant up`. The playbook should run without error but does WordPress run? If you see a `502 Bad Gateway` error, find the problem and fix it. Use annotated tag `impish` when the problem is fixed
   Hint: Connect to the running VM by ssh and look at the nginx log with `sudo less /var/log/nginx/error.log`.

4. Related to the above, how can we avoid upstream updates to the Vagrant box, the various `apt` packages and the WordPress install from breaking Production? Can you implement some or all of this in your playbook? Use annotated tag `reliability` once the playbook has been made more robust.

5. **[Moderately Hard]** Modify the `Update the apt cache` task so that it will execute *once* if it has never been run before, regardless of the value of `cache_valid_time`, and thereafter observe the `cache_valid_time` setting. This will require your playbook determine whether the task has already run — is there a built-in or Ansible-recommended way to handle this situation?

   Why this is important: Vagrant updates their Ubuntu boxes https://app.vagrantup.com/ubuntu/ regularly and sooner-or-later you will receive a warning message from Vagrant:

   ```
   ==> default: Checking if box 'ubuntu/hirsute64' version '20210711.0.0' is up to
   ==> default: A newer version of the box 'ubuntu/hirsute64' for provider 'virtual
   ==> default: available! You currently have version '20210711.0.0'. The latest is
   ==> default: '20210720.0.0'. Run `vagrant box update` to update.
   ```

   Depending on your `apt` module `cache_valid_time` setting, it is possible to `vagrant box update` within the cache validity period set in the playbook... in which case you will likely receive some *very* confusing errors when you next try to provision your VM with `apt`, for example:

   ```
   fatal: [192.168.33.20]: FAILED! => {"changed": false, "msg": "No package matchin
   ```

   This problem is a *race condition*, apparently caused by Vagrant's Ubuntu boxes not shipping with a

complete `apt` package list. Executing the `Update the apt cache` task every time the playbook is run by setting the `cache_valid_time` parameter to a low value, or even removing it, will solve the problem... very inefficiently. Updating the cache on most runs is not something you want if you are running a playbook often while developing or troubleshooting!

Use the annotated tag `no-race` once you have implemented a solution to the race condition.

6. **[Hard]** Vagrant complains "guest additions on this VM do not match the installed version of VirtualBox" when it starts. It's fairly easy to suppress this warning by an entry in the `Vagrantfile`, but can we resolve this problem? There is a plugin for Vagrant called `vagrant-vbguest` https://github.com/dotless-de/vagrant-vbguest, try to resolve the guest additions mismatch by installing this. Use the annotated tag `newer-guests` to tag your implementation.

Appendix: Tags

Use the following annotated tags (see: https://git-scm.com/book/en/v2/Git-Basics-Tagging) for your git repository to show which commits complete these sections in the assignment.

Tag	Notes
`ch2-inv`	Inventory file from the *Running Without Vagrant* section of Chapter 2.
`ch3-env`	Completed environment setup. It is recommended that this tag includes the playbook created on p. 32.
`ch3-php`	Completion of the *Installing PHP* subsection.
`ch3-mysql`	Completion of the *Installing MySQL* subsection.
`ch3-nginx`	Completion of the *Installing nginx* subsection.
`ch3-hand`	Completion of the *Tasks and Handlers* subsection.
`ch3-wp`	Completion of the *Downloading WordPress* and *Configuring a WordPress Install* sections.
`ch3-complete`	Completed Chapter 3 (includes the *Making it Idempotent* section). Playbook and WordPress installation function without apparent error.
`ch4-role.php`	Completion of migration of PHP configuration to a role.
`ch4-role.mysql`	Completed migration of MySQL configuration to a role.
`ch4-role.nginx`	Completed migration of nginx configuration to a role.
`ch4-role.wp`	Completed migration of WordPress configuration to a role.

`ch4-complete`	Completed Chapter 4. Playbook, Roles and WordPress installation function without apparent error.
`lint-free`	**Completion 2:** playbook de-linted; `ansible-lint` reports no rule violations.
`all-YAML`	**Completion 3:** playbook uses YAML *nested mapping* syntax wherever possible.
`completion`	**Completion 5:** WordPress database with "completion" blog post committed to the repository, displaying correctly when a fresh VM is provisioned.
`standards`	**Challenge 1:** marks the point where commit message format standard was applied with push rules and `gitlint`.
`no-hands`	**Challenge 2:** Elimination of manual processes.
`impish`	**Challenge 3:** playbook works with Impish Indri.
`reliability`	**Challenge 4:** playbook is robust to changes/updates to the installed packages.
`no-race`	**Challenge 5:** eliminate the race condition.
`newer-guests`	**Challenge 6:** update the guest additions version on the VM to match that of VirtualBox.

It is permissible, even expected, that there will be commits between tags because there is often a reasonable amount of testing work at intermediate steps between tags.

Example: to complete the *Installing MySQL* section requires several steps and it would be wise to commit:

1. after configuring basic install (top of p. 34),
2. once everything is working at the half-way point (middle of p.35) prior to creating the `provisioning/templates/mysql` directory and contents, and
3. once everything is working at the end of the section (p. 37).

This last commit would be tagged `ch3-mysql` per the table above. There are no problems whatsoever with creating more commits if you wish, which might include bug fixes or enhancements.
