Devops UNIT-II

SETTING UP A VIRTUAL MACHINE

  - Provisioning (setting up) a virtual machine (VM) is the act of configuring a VM for a specific purpose.
  - purpose could be running an application, testing software across a different platform, or applying updates.
  - Setting up a VM requires two steps: creating and then configuring it
  - Vagrant and Ansible to build and configure a VM
- Vagrant automates the process of creating the VM, while Ansible configures the VM once it's running
- This process is similar to creating and provisioning servers in the cloud.

Why Use Code to Build Infrastructure?

- Using code to build and provision infrastructure lets you consistently, quickly, and efficiently manage and deploy applications.
- infrastructure and services to scale.
- It also can reduce operating costs
- decrease time for recovery during a disaster
- minimize the chance of configuration errors.

- two terms that relate to creating and configuring infrastructure:
- ***infrastructure as code*** *(IaC) and* ***configuration management*** *(CM).*
- *infrastructure as code is the process of using code to describe and manage infrastructure like VMs, network switches, and cloud resources such as Amazon Relational Database Service (RDS).*
- *CM is the process of configuring those resources for a specific purpose in a predictable, repeatable manner.*
- *The two tools we are using, Vagrant and Ansible, are considered IaC and CM, respectively.*
- Another benefit of treating your infrastructure as code is ease of deployment.
- Treating your infrastructure as code allows you to build reusable components, use test frameworks, and apply standard software engineering best practices.

Getting Started with Vagrant

- ***Vagrant*** is a framework that makes it easy to create and manage VMs.
- It supports multiple operating systems (OSs) that can run on multiple platforms.
- Vagrant uses a single configuration file, called a ***Vagrantfile***, to describe the virtual environment in code.
- ***Installation***
- To install Vagrant, visit Vagrant's website at .
- *https://www.vagrantup.com/downloads.html*
- Choose the correct OS and architecture for your host

- When your VM comes up, you'll also need to make sure that it has **VirtualBox's guest additions** installed on it
- *Guest additions* provide better driver support, port forwarding, and host-only networking.
- They help VM run faster and have more options available
- After finished installing Vagrant, enter the following command in the terminl to install the Vagrant plug-in for guest additions:

```
$ vagrant plugin install vagrant-vbguest
Installing the 'vagrant-vbguest' plugin. This can take a few minutes...
Fetching vagrant-vbguest-0.30.0.gem
Installed the plugin 'vagrant-vbguest (0.30.0)'!
```

Anatomy of a Vagrantfile

- A Vagrantfile describes how to build and provision a VM.
- It's best practice to use **one Vagrantfile per project** so you can add the configuration file to your project's version control and share it with your team.
- The configuration file's syntax is in the **Ruby** programming language

Vagrant File Components

Operating System

Vagrant supports many OS base images, called *boxes*, by default. You can search the list of boxes that Vagrant supports at *https://app.vagrantup.com/boxes/search/*. Once you find the one you want, set it near the top of the Vagrantfile using the `vm.box` option, as shown below:

```
config.vm.box = "ubuntu/focal64"
```

set the **vm.box** identifier to **ubuntu/focal64.**

Networking

You can configure the VM's network options for different network scenarios, like *static IP* or *Dynamic Host Configuration Protocol (DHCP)*. To do this, modify the `vm.network` option near the middle of the file:

```
config.vm.network "private_network", type: "dhcp"
```

Making VM to obtain its IP address from a private network using DHCP.

So that , it'll be easy to access resources like a web server on the VM from your local host.

Providers

A *provider* is a plug-in that knows how to create and manage a VM. Vagrant supports multiple providers to manage different types of machines. Each provider has common options like CPU, disk, and memory. Vagrant will use the provider's application programming interface (API) or command line options to create the VM. You can find a list of supported providers at *https://www.vagrantup.com/docs/providers/*. The provider is set near the bottom of the file and looks like this:

```
config.vm.provider "virtualbox" do |vb|
  vb.memory = "1024"
  vb.name = "dftd"
  --snip--
end
```

Basic Vagrant Commands

- ***vagrant up*** Creates a VM using the Vagrantfile as a guide
- ***vagrant destroy*** Destroys the running VM
- ***vagrant status*** Checks the running status of a VM
- ***vagrant ssh*** Accesses the VM over Secure Shell
- Each of these commands has additional options, enter a command and then add the --help flag for more information.
- To learn more about Vagrant's features, visit the documentation at *https://www.vagrantup.com/docs/*.
- Once you create the VM by running **vagrant up**, you'll have a core Linux system with all the OS defaults

Ansible

- *Ansible* is a CM tool used for provisioning of infrastructure like VMs.
- Ansible uses a ***declarative configuration style***, i.e. allows to describe what the desired state of infrastructure should look like.
- ***imperative configuration style***, which requires to supply all the minute details on your desired state of infrastructure.
- Ansible is a great tool for software engineers who are not well versed in system administration.
- Ansible is also open-source software and free to use.
- Ansible is written in Python

- ***Yet Another Markup Language (YAML),*** is a data serialization language
- It uses Ansible to describe complex data structures and tasks
- *https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html*.
- Ansible applies its configuration changes over *Secure Shell (SSH)*, which is a secure protocol to communicate with remote hosts.
- ***Installation***
- install Ansible so Vagrant can use it for provisioning.
- *https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html*.

Key Ansible Concepts

- **Task** A *task* is a command or action (such as installing software or adding a user) that is executed on the managed host.
- **Role** A *role* is a group of tasks and variables in a directory structure, defines a particular purpose for the server, and can be shared with other users for a common goal
- A role could configure a host to be a database server. This role would include all the files and instructions necessary to install the database application, configure user permissions, and apply seed data.
- **Playbook** A *playbook* is a collection of ordered tasks or roles that can use to configure hosts.
- **Control node** A *control node* is any Unix machine that has Ansible installed on it. You will run your playbooks or commands from a control node, and you can have as many control nodes as you like.
- **Inventory** An *inventory* is a file that contains a list of hosts or groups of hosts that Ansible can communicate with.
- **Module** A *module* encapsulates the details of how to perform certain actions across operating systems, such as how to install a software package.
- Ansible comes preloaded with many modules.

Ansible Playbook

- To configure the VM, use the Ansible playbook
- file, named ***site.yml***, is located in the ***ansible/*** directory you cloned from git repository
- Think of the playbook as an **instruction manual** on how to assemble a host.
- You can break up playbook files into different sections.
- The first section is the header, which is used to set global variables to use throughout the playbook
- Header contains few things like the name of the play, the hosts, the remote_user, and the privileged escalation method.

header

```
---
- name: Provision VM
  hosts: all
  become: yes
  become_method: sudo
  remote_user: ubuntu
--snip--
```

- each play has a ***name*** so it's easier to find and debug if things go wrong.
- could have multiple plays in a single playbook
- the **h*osts*** option is set to **all** to match any Vagrant built VMs because Vagrant will autogenerate the Ansible inventory file dynamically.
- Some operations on a host will require elevated privileges,
- so Ansible allows you to ***become***, or activate privilege escalation for, a specific user
- Since Ubuntu is used, the default user with escalated privileges is **ubuntu**.

tasks

```
--snip--
  tasks:
    #- import_tasks: chapter2/pam_pwquality.yml
    #- import_tasks: chapter2/user_and_group.yml
--snip--
```

- The built-in Ansible **import_tasks** function is loading tasks from two separate files: *pam_pwquality.yml* and *user_and_group.yml*.
- The ***import_tasks*** function allows you to organize the tasks better and avoid a large, cluttered playbook.
- Each of these files can have one or many individual tasks

Basic Ansible Commands

- Two mist frequently used commands: ***ansible*** *and* ***ansible-playbook***.
- use the **ansible** command for running ad hoc or onetime commands that you execute from the command line
- For example, to instruct a group of web servers to restart Nginx, you would enter the following command:
- $ **ansible webservers -m service -a "name=nginx state=restarted" - - become**
- This instructs Ansible to restart Nginx on a group of hosts called ***webservers***.
- Note that the mapping for the *webservers* group would belocated in the ***inventory*** file.
- The Ansible ***service*** module interacts with the OS to perform the restart.
- The service module requires some extra arguments, and they are passed with the **-a** flag.
- In this case, both the **name of the service** (nginx) and the fact that it should restart are indicated.
- You need *root* privileges to restart a service, so you'll use the **--become** flag to ask for privilege escalation

- The ***ansible-playbook*** command runs playbooks.
- This command is used by Vagrant during the provisioning phase.
- To instruct **ansible-playbook** to execute the ***aws-cloudwatch.yml*** playbook against a group of hosts called ***dockerhosts***, enter the following command in the terminal
- $ **ansible-playbook -l dockerhosts aws-cloudwatch.yml**
- The dockerhosts need to be listed in the inventory file for the command to succeed.
- if you do not provide a subset of hosts with the **-l** flag, Ansible will assume you want to run the playbook on all the hosts found in your inventory file.

Creating an Ubuntu VM

- To create the Ubuntu VM, make sure you are in the same directory as the Vagrantfile.
- This is because Vagrant needs to reference the configuration file while creating the VM.
- use the vagrant up command to create the VM
- $ **vagrant up**

The first section of the output to look at is Vagrant downloading the base image:

```
--snip--
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'ubuntu/focal64'...
--snip--
```

Here, Vagrant is downloading the ubuntu image, as expected. The image download may take a few minutes, depending on your internet connection.

- Next, Vagrant will configure a public/private key pair to provide SSH access to the VM.

```
--snip--
default: Vagrant insecure key detected. Vagrant will automatically replace
default: this with a newly generated keypair for better security.
default:
default: Inserting generated public key within guest...
default: Removing insecure key from the guest if it's present...
default: Key inserted! Disconnecting and reconnecting using new SSH key...
--snip--
```

- Vagrant stores the private key locally on the host (*.vagrant/*) and then adds the public key to the ***~/.ssh/authorized_keys*** file on the VM.
- Without these keys, you would not be able to connect to the VM over SSH.

By default, Vagrant and VirtualBox will mount a shared directory inside the VM. This shared directory will give you access to a host directory from within the VM:

```
--snip--
==> default: Mounting shared folders...
   default: /vagrant => /Users/bradleyd/devops_for_the_desperate/vagrant
--snip--
```

Finally, the following shows the Ansible provisioner output:

```
--snip--
==> default: Running provisioner: ansible...
    default: Running ❶ ansible-playbook...

PLAY [Provision VM] ***********************************
❷ TASK [Gathering Facts] *****************************
❸ ok: [default]

PLAY RECAP *********************************

--snip--
default      : ok=1    ❹ changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

This shows that the Ansible provisioner is run using the `ansible-playbook` ❶ command. Ansible logs each `TASK` ❷ and whether anything was changed on the host ❸. In this case, all the tasks are commented out, so nothing was `changed` ❹ on the VM. This output is the first place to look when gauging success or failure.

- To check the status of VM

```
$ vagrant status
Current machine states:
default    running (virtualbox)
--snip--
```

- the status of the VM is running. This means you created the VM, and it should be accessible over SSH.
- If you need more information, add the debug flag to the up command to make Vagrant increase the output verbosity:
- **vagrant up --debug**.

Summary

- installed Vagrant and Ansible to create and configure a VM.
- how to configure Vagrant using its Vagrantfile
- how to provision a VM using Ansible playbooks and tasks.
- able to create and provision any type of infrastructure, not just VMs.