SECTION 04

Configuration management with Ansible

WHAT IS CONFIGURATION MANAGEMENT AND WHY IT IS NEEDED

- We discussed earlier the need for automation to achieve a DevOps environment.
- Having applications containerized using Docker is one option of “instant provisioning”. But it’s not the only one and it’s not suitable for all cases (please refer to section 03 for more discussion).
- For those who require Vagrant as an infrastructure-provisioning tool, there should be a way to deploy or customize an application.
- So, you can find a Vagrant box with Apache installed. But what if you want an Nginx reverse proxy installed on top of that? Or, what if you need some custom users and groups configured, together with some file permissions?
- Shell scripting is an option and it is supported by Vagrant to boot up and configure the machine automatically (instant provisioning). But this may not be your best choice and here’s why.

WHY NOT USE A SHELL SCRIPT FOR THAT?

- Shell scripts are great. The Bash shell, for example has a huge set of commands and functions to automate any task.
- But when it comes to DevOps, a shell script will not offer the following:
  - Change management documentation: there needs to be a way to document even the slightest change done to a system. This will save you endless hours trying to figure out what went wrong after a series of changes occurred. You can comment your shell script, but still that is not enough as we'll see later.
  - Idempotence: this refers to the ability to run an operation several times with the same result each time. This is very difficult to be done with a shell script.

WHAT IS ANSIBLE?

- It is one of the DevOps tools concerned with configuration management. It's not the only one in the market as there are other tools like Chef, Puppet, and Salt.
- Basically, they all do the same thing. But Ansible has some strength points:
  - It's agent-less. No extra software needs to be installed on the target machines.
  - It uses Python for instructions. Python is one of the most popular scripting languages out there. It's more likely to be under the tool belt of a typical system administrator.
  - It uses SSH for connecting to the remote servers. SSH is an industry standard security protocol that all system administrators are familiar with.

INSTALLATION OPTIONS

- When it comes to its installation, Ansible has two main methods: through the OS package manager, and through pip.
- If you are familiar with Python you'd know about pip. It is the command used to search, download, and update Python packages. It can be used to install Ansible as a Python package.
- If you are using a macOS, you can use hombrew to install Ansible. This is as simple as running `brew install ansible`. If you don't have home brew you can follow this link for installing it https://docs.brew.sh/Installation.html
- If you are on Windows, it'd be better to use a virtual machine (you can use Vagrant for that) and install Ansible on it.

INSTALLING ANSIBLE ON LINUX

- I will cover installation on Ubuntu and Centos as they are well known Linux distros. Once installed, Ansible commands are the same on any platform.
- On an Ubuntu system issue the following commands to install the package:

```
sudo apt-add-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get install ansible
```

- On a Centos (or any Red Hat variant), issue the following commands:

```
sudo yum install epel-release
sudo yum install ansible
```

- If you want to install it using pip (on either system), run the following commands (make sure Python is installed):

```
sudo easy_install pip
pip install ansible
```

- In all cases, you can verify that Ansible has been correctly installed on your system by running ansible --version to print the currently installed version of the product.

THE INVENTORY FILE

- Ansible uses /etc/ansible/hosts file to know about which machines you want configured by Ansible.
- This file contains machine host names, IP addresses, or a mix of them. This depends on how the client machine (the one with Ansible installed) can communicate with target hosts.
- Hosts can be added either in named groups for easier reference, or as standalone entries.
- If you have a number of hosts for which you chose a numbered naming pattern (like web1, web2, web3…web6), you can add them on one line like web[1..6].

PASSWORD-LESS CONNECTION

- Ansible is an automation tool that works unattended. This means that there should not be an prompts that require user input, including the password prompt.
- The user account that you use to run Ansible needs to be able to connect to the target machine through SSH without being prompted for a password. This can be done as follows:
  - On the client machine, issue the following command to generate a key pair: ssh-keygen
  - Do not input a passphrase, just press enter twice to skip the prompt.
  - A pair of keys has been created in ~/.ssh/ (~ stands for your home directory). Copy the contents of id_rsa.pub file.
  - On the target machine, create a user if necessary to match the username you'll use with Ansible (for example, vagrant).
  - In the user's home directory, create a .ssh subdirectory (if it doesn't already exist). Change the permissions of the directory to be most restrictive: `chmod 700 .ssh`
  - Inside the the directory, create a file called authorized_keys and paste the text inside it
  - Ensure that you can connect to the remote host without entering a password

YOUR FIRST ANSIBLE JOB

- Ansible works in two modes: command line and playbook files. In the first mode, you pass the instructions to the remote servers as arguments and parameters to the ansible command. In the second mode, you type those instructions in a YAML file.
- We'll start by using the command line mode of Ansible to demonstrate the basics then move on to playbooks.
- Our first job will just getting the current Bash shell version. To get the Bash version, you type `bash --version` on the shell prompt.
- We need to run this command on all the servers in our inventory. Start by opening /etc/ansible/hosts and adding the IP addresses of your target machine(s) under a [servers] group.
- Now, run the following command: `ansible servers -a "bash –version"`
- The command output will list the IP address/name of each server that Ansible contacted, together with the result of the command and any text that got printed.

THE YAML FORMAT

- YAML is short for YAML Ain't Markup Language. It is used the same as JSON files: to serialize data in an easy format that is readable by both humans and machines.
- Data is stored in key/value pairs separated by colons. Items can be grouped if they have the same indentation level.
- Arrays (called dictionaries) of data are represented by the placing a dash before each item. Of course all items in an array should be on the same indentation level.
- Quoting values is optional, but it is mandatory if you are using special characters (like colons) in the values.
- YAML does not allow you to use tabs for indentation. Only spaces are allowed.

A SAMPLE YAML FILE

```
--- !clarkevans.com/^invoice
invoice: 34843
date   : 2001-01-23
bill-to: &id001
    given  : Chris
    family : Dumars
    address:
        lines: |
            458 Walkman Dr.
            Suite #292
        city     : Royal Oak
        state    : MI
        postal   : 48046
ship-to: *id001
product:
    - sku         : BL394D
      quantity    : 4
      description : Basketball
      price       : 450.00
    - sku         : BL4438H
      quantity    : 1
      description : Super Hoop
      price       : 2392.00
```

YAML can optionally start with ---

Nested key/value pairs

A list of items each is preceded by a dash. An item can has its own set of key/avalue pairs

THE WEBSERVER.YML FILE

- Our first playbook will be used to install the Apache web server on the target machine(s).
- I strongly recommend that you use a code editor (Notepad++ for Windows, Sublime Text and Textmate for macOS are excellent examples) for editing playbooks. They will ensure that you follow the correct indentation.
- Open a new file called webserver.yml and add the following text:

```
- hosts: ubuntu
  become: yes
  tasks:
    - name: Ensure that the Apache web server is installed
      apt:
        name: apache2
        state: present
    - name: Ensure that the service is running and will be started on boot
      service:
        name: apache2
        state: started
        enabled: yes
```

ANALYZING THE FILE

- The playbook started with hosts directive, preceded by a dash. The dash indicates that this is an item in array. That is because a playbook file can contain several playbooks, each is an item.
- The hosts directive specifies the target machines. It can multiple values separated by colons, or simply an asterisk * or “all” to denote all the machines in the inventory file. You can review the different patterns that you can specify on this link: http://docs.ansible.com/ansible/latest/intro_patterns.html
- become: work as root. Ansible uses the sudo command for this.
- tasks: a dictionary of tasks that will be executed. Each has its own set of key/value properties:
  - name: a description of the task
  - yum/apt: one of several modules that Ansible accepts to configure a machine. Yum is used for Red Hat machines, while apt is used for Debian ones. The full list of modules can be found on this link: http://docs.ansible.com/ansible/latest/list_of_all_modules.html
  - service: a module that controls the aspects of a service running in the background. Its properties include:
    - name: the name of the service as it appears in the OS (for example apache2 for Ubuntu and httpd for Centos).
    - state: the state that you wish this service be at. For example: started/stopped
    - enabled: whether or not you want this service to start at boot time.

USING ANSIBLE WITH VAGRANT

- In section 02, you learned that Vagrant accepts providers for instant provisioning like the shell script, Chef, Puppet, Salt, Docker, and Ansible.
- Let's see how we can use the webserver.yml playbook to make Vagrant use Ansible to configure Apache on the machine the moment you issue `vagrant up`
- Create a new Vagrant machine or destroy one of the existing ones.
- Place webserver.yml file in Vagrant machine directory.
- Open the Vagrantile and add the following to the end of it:

```
client.vm.provision "ansible" do |ansible|
  ansible.playbook = "webserver.yml"
end
```

- Issue `vagrant up` to launch the machine. Watch the log messages that get printed indicating that Ansible is being used to install and run the Apache web server.

LAB: DEPLOYING THE LAMP STACK AND CODEIGNITER

- In this lab, we'll use Ansible to deploy the LAMP stack. We'll also add the famous PHP framework CodeIgniter and ensure that it is properly configured so that developers can start working right away.
- Let's create a Vagrant machine group containing two machines: a web server and a database server.
- Create a directory called ansible-lamp. You can copy one of the Vagrantfiles you already have and paste it in the directory.
- Change the Vagrantfile by adding the following at the end:

```
config.vm.define "client" do |client|
    client.vm.hostname = "client"
    client.vm.network "private_network", ip: "192.168.33.10"
  end
config.vm.define "web" do |web|
    web.vm.hostname = "apache"
    web.vm.network "private_network", ip: "192.168.33.20"
    web.vm.network "forwarded_port", guest: 80, host: 8080
  end
  config.vm.define "db" do |db|
    db.vm.hostname = "mysql"
    db.vm.network "private_network", ip: "192.168.33.30"
  end
end
```

- We already covered that in section 02. This basically creates three Vagrant machines with different internal IP addresses: client (where Ansible will be installed), web (for the application and web server), and db (for the MySQL database).
- Issue vagrant up to launch the three machines.
- Login to client using vagrant ssh client. Install Ansible using one of the discussed methods.
- Ensure that password-less connections are enabled between the client and the other machines (web and db) by exchanging keys.
- Add the machine IP addresses to /etc/ansible/hosts in a group called [lamp].

THE APPLICATION PART

- We need Apache and PHP to be installed on the web server to host CodeIgniter.
- Using your favorite text editor, open a new file called application.yml. Make sure you save it in ansible-lamp directory so that it is visible to the machines. Write the text in webserver.yml (available in the project files directory).
- We have two new modules here:
  - yum_repository: this is used to add YUM repositories to a Red Hat variant OS. We used the name, a description, and a baseurl. We used to add the EPEL repo, which is needed to download PHP 7.
  - rpm_key: This is used to import and activate the GPG key used to securely download packages from a repository.
- We needed to add another repository, webtatic. This is also used to download some PHP 7 required packages. This time we used the YUM module to install the RPM package directly. The RPM package will take care of the GPG key and other requirements for the repo to work.
- We also used the with_items. This allows you to use an array of items and instruct the module to loop through it, working on each item individually.

CONFIGURING THE DATABASE

- We need to deploy MariaDB (a forked version of MySQL)
- Due to the large file size, you can view it directly in the project files folder accompanying this section of the class.
- Please follow the lecture with the database.yml file.