Mastering

ANSIBLE

Section 03

Exploring modules

Using Ansible without playbooks

- In the previous section, we used Vagrant to provision the machine. We created a playbook and Vagrant did the rest.
- Although playbooks are an excellent way for managing multiple machines and storing configuration instructions, you are not obliged to use them whenever you want to use Ansible.
- We've previously seen how we can use a command like `ansible servers -a 'cat /etc/hosts'` to send an ad-hoc command to all nodes under servers group.
- But you know by now that Ansible also uses *modules* to execute common tasks on the remote system. You already saw the yum module in the playbook used in last section.
- You can make ansible use a module on the command line without a playbook like the following command: `ansible servers -b -m yum -a 'name=httpd state=present'`
- As you can see, the module is yum, it is denoted by the `-m` option. Notice the use of `-b` to instruct Ansible to use elevated privileges when executing the command. Finally, we have the necessary attributes for the yum module

Preparing machine for the LAMP stack

- Now we are going to build a LAMP stack. LAMP stands for Linux Apache MySQL and PHP (sometimes Python or even Perl). This is a well known development stack among PHP developers working on Linux infrastructure.
- We already have a web server machine configured and have the Apache package installed and the service is up and running. Now we need to add PHP 7 to the server and build a second server with Mariadb (a fork of MySQL used on Centos 7 and Red Hat 7 by default) installed.
- To do this, let's first created a new machine, database. Change your Vagrantfile to adopt three machines as follows:

Hostname	IP address	Role
client	192.168.33.10	Control machine
webserver1	192.168.33.20	Web server
database	192.168.33.50	Database

Deploying Mariadb on the database server

- Let's use Ansible to deploy and configure Mariadb database package on the database server that we've just created. Login to the client machine by issuing `vagrant ssh client`.
- First, lets add a new group in /etc/ansible/hosts called db and add the ip address of the database server. Let's add another group called web for the web server(s) and add the IP address of out webserver machine.
- Before we can use Ansible we need to ensure that client can ssh to database using public/private key method. Let's add the content of `/home/vagrant/.ssh/id_rsa.pub` on the client to `/home/vagrant/.ssh/authorized_keys` on both database and webserver.
- Time to deploy Mariadb. Issue the following command to do that: `ansible db -b -m yum -a "name=mariadb-server state=present"`.
- Once done, we need to ensure that the service is started and will start automatically when the system boots. Issue the following command for this: `ansible db -b -m service -a "name=mariadb state=started enabled=yes"`

Configuring Mariadb

- Installing the package isn't sufficient to start working with Mariadb. We need to configure several aspects like setting the root password, creating an application database, user, and also following some security procedures.
- Ansible has its own module for working with MySQL/Mariadb. However, in order to use it, the `MySQL-python` package must be installed on the target machine. Let's do that: `ansible db -b -m yum -a "name=MySQL-python state=present"`
- Now we can use several Ansible `mysql_*` modules. To set the root password, issue the following command: `ansible db -b -m mysql_user -a "name=root host=localhost password=mypassword"`
- Any web application needs a database to work. Create the appdb using the following command: `ansible db -m mysql_db -a "name=appdb state=present login_user=root login_password=mypassword"`
- We also need to create a database for the application. The following command will create a new database called appdb: `ansible db -m mysql_db -a 'name=appdb state=present login_user=root login_password=mypassword'`
- Next, we need to create an application user, our application will be the CodeIgniter PHP framework: `ansible db -b -m mysql_user -a "name=codeigniter host=localhost password=cipassword priv=appdb.*:ALL state=present login_user=root login_password=mypassword"`
- If you want to double check Ansible's work, login to the database server and login to Mariadb using `mysql -u root -pmypassword`. Then inside the mysql prompt, issue the following command: `show grants for 'codeigniter'@'localhost';`

Installing PHP 7

- So far we have LAM (Linux Apache MySQL). We need to work on the P part (PHP). We'll install PHP 7 on the web server machine.
- Installing PHP 7 requires adding two third party YUM repositories: EPEL and Webtatic. If you are not using Ansible you'd have to login to the server, download the RPM package for both repositories and install them.
- However, Ansible's got it's own module for managing YUM repositories: the yum_repository module. Issue the following commands on client to install EPEL and Webtatic repositories on the webserver machine:

```
ansible web -b -m yum_repository -a "name=epel description='EPEL repository'
baseurl='https://download.fedoraproject.org/pub/epel/\$releasever/\$basearch/' gpgcheck=no
state=present"
ansible web -b -m yum_repository -a "name=webtatic description='Webtatic repository'
baseurl='https://repo.webtatic.com/yum/el7/\$basearch/' gpgcheck=no state=present"
```

- You are ready to install PHP 7. Since we need to install multiple packages to satisfy the application needs, we'll postpone this to next section to be used within a playbook. Now let's explore other interesting Ansible modules.

Adding users and groups

- One of the most repetitive tasks for system administrators is managing users and groups.
- In our case, we need to add a user and a group for coding our web application. Ansible has the `user` and `group` modules for this.
- To add a group `developers` and `cidev` to be part of this group, issue the following commands:

```
ansible web -b -m group -a "name=developers state=present"
ansible web -b -m user -a "name=cidev group=developers"
```

- The above commands are the bare minimum required for creating a user a putting it in a group. There are a lot of attributes that can further customize the process. Have a look at http://docs.ansible.com/ansible/user_module.html and http://docs.ansible.com/ansible/group_module.html for user and group module options respectively.

Working with files

- Ansible has some powerful modules to work with files and directories on remote systems. For example, you can query file attributes by using the stat module as follows: `ansible web -m stat -a 'path=/etc/hosts'`
- Notice that the output of this command is the same as the output of running stat command on /etc/hosts locally on the remote host. However, Ansible is producing this output in JSON format, which can be further parsed using command line tools (like `jq`) or within playbook (as discussed later in the course).
- For example, the output of the previous command could've been piped to jq to display only information of interest to us (permissions, user, and group ownership) as follows:
```
ansible web -m stat -a "path=/etc/hosts" | sed 's/.*SUCCESS.*=> //g' | jq '.stat | {perms: .mode, group: .gr_name, owner: .pw_name}'
```
- Some of you might argue that a much simpler method would've been `ansible web -a 'ls -l /etc/hosts'`. Yes, you're perfectly right but `jq` can do much more than just filtering output. You can have a look at this page http://webgeodatavore.com/jq-json-manipulation-command-line-with-geojson.html to find out more about `jq`.

Upload files and directories to servers

- Several tools can transfer files and directory to and from servers through SSH connections like `scp` and `rsync`. Ansible also has its own module for doing this.
- To copy a file/directory to a remote machine, you use the copy module as in the following example:
  `ansible web -m copy -a 'src=welcome.txt dest=/tmp/welcome.txt'`
- When working with directories, however, you have two options:
  - Transfer the contents of the directory to the remote destination by placing a trailing slash after the source
  - Transfer the directory itself with all its contents to the remote destination by not placing a trailing slash after the source.
- For example, the following command will create a directory called `/tmp/tools` on the remote machine, containing some files:
  `ansible web -m copy -a 'src=tools dest=/tmp/'`
- But the following command will not create a directory on /tmp; it will rather copy the files inside tools to /tmp:
  `ansible web -m copy -a 'src=tools/ dest=/tmp/'`
- And Ansible has also more advanced modules for more complex file transfer operations like `synchronize` and `rsync`.

Download files and directories from servers

- The fetch module is used by Ansible for downloading files and directories to the local machine from remote servers.
- To handle the issue of having multiple files with the same name downloaded from several servers, fetch by default creates a directory with the name (or IP address) of the remote server in which it places the downloaded file/directory.
- For example, let's download the /etc/hosts file from the database and the web server and place it in /tmp:
  `ansible all -m fetch -a 'src=/etc/hosts dest=/tmp'`
- Notice how Ansible creates a similar path on /tmp after it creates a directory with the IP address of the remote machine to avoid any possible filename collision. This behavior can be turned off by adding `flat=yes` to the attributes part of the command.

Creating and deleting files

- Ansible uses the `file` module to manage files. However, it will not create a new file if it does not already exists. So the following command to create a new file will fail: `ansible web -m file -a "dest=/home/vagrant/newfile.txt"`
- If you want to create an empty file you use the `copy` module as follows:
  `ansible web -m copy -a "content="" dest=/home/vagrant/newfile.txt"`
  and content here may or may not contain text depending on your needs.
- Once the file is there, you can change it's attributes with a command like the following: `ansible web -b -m file -a "path=/home/vagrant/newfile.txt owner=cidev group=developers mode=700"`
- The `state` parameter is by default file. But you can use to change the type of object that you want to create. For example, the following command will create a symbolic link from `/etc/hosts` and place it in `/tmp`: `ansible web -m file -a "src=/etc/hosts dest=/tmp/hosts state=link"`
- Deleting a file is as simple as setting the state to absent. The following command will delete the file that we just created: `ansible web -m file -a "path=/home/vagrant/newfile.txt state=absent"`

Working with directories

- As a matter of fact, a directory is nothing but a special type of file that may contain other files. Hence, there is no `directory` module in Ansible; the `file` module does the job.
- To create a new empty directory on a remote server, we use a command like the following: `ansible web -m file -a "path=/home/vagrant/newdir state=directory"` Notice that we used the `file` module here and not the `copy` one.
- To change the attributes of an existing directory you'd something like: `ansible web -b -m file -a "path=/home/vagrant/newdir mode=700 owner=cidev group=developers"`
- Finally, to delete a directory you set the state to absent as follows: `ansible web -b -m file -a "path=/home/vagrant/newdir state=absent"`

Cron job management

- Cron jobs are scheduled jobs that get run on the system periodically according to a standard format.
- Ansible has its own module for cron job management, cron. You can add a cron job as follows: `ansible all -b -m cron -a "name=archive-log-files hour=0 job='/opt/jobs/archive.sh'"`. This will add a cron job that will run a script `/opt/jobs/archive.sh` every day at 12:00 A.M.
- If you've worked with cron before you already know that you can also specify the minute, day, month, and day of week as extra parameters. Alternatively you can opt to ignore them to be every hour regardless of any other time restriction by placing `*` in their place.
- Ansible already takes care of that by placing `*` in place of ignored fields. Or you can explicitly state them as `minute`, `day`, `month`, and/or `weekday`.
- You can also use a number of useful parameters. The most interesting are:
  - `special_time`: instead of specifying minute, hour, day…etc you can use a shortcut like yearly, daily, monthly, weekly, or hourly. You can also instruct cron to run the job when the system reboots by using reboot.
  - `user`: by default the cron job is registered with the root user. You can change than by specifying the user parameter.
- Removing a cron job is as simple as stating its `name` and setting the `state` to `absent`. For example: `ansible all -b -m cron -a "name=archive-log-files state=absent"`