Mastering ANSIBLE

Section 04

Working with Playbooks

The ansible-playbook command options

- ansible-playbook is a powerful command. The following are the options used when you want to run the playbook as a specific user.
    - If you want to run the commands remotely using a specific user account, you can use `-u` option. For example `ansible-playbook myplaybook.yml -u cidev`
    - Sometimes you may want to use sudo access to run commands as another user (other than the login user). For this you can use `-U`.
    - If sudo-ing to another user will require a password (the login user password), you can use the `-K` option.
    - A complete example for sudo may be: `ansible-playbook myplaybook.yml -U testing1 -K`

Targeting specific hosts

- Now we need to run the playbook. But we need to run it only on the web server; we don't want PHP installed on the database server. We can either change the hosts part of the playbook to point at the web group, or we can fine grain the process by using the `--limit` flag.

- To run an Ansible playbook, you use the ansible-playbook command, passing in the name of the playbook. But to target a specific host(s) you use the `--limit` flag.

- Let's run lamp.yml on our web server machines (currently only one machine) by issuing the following command `ansible-playbook lamp.yml --limit web`

- The above command will run the playbook on all servers in the web group of /etc/ansible/hosts file (only one so far). But the --limit flag can also target ranges of machines *.development.mycompany.com or web[1..10].mycompany.com, which will target web1 till web10 of mycompany.com domain.

- Combined with `--list-hosts`, you can also determine which machines will be in the execution scope.

Playbook variables

- Adding tasks to be executed on remote servers in playbooks is a great way to orchestrate configuration management needs on you infrastructure. But sometimes you have more complex scenarios that require more than just plain tasks. Let's examine some playbook helpers that may come in handy when needed.

- First, you may need to use variables inside the playbook. Variables may include usernames, groups, packages and so on. You can add variables either at the start of the playbook and use them later, or in a separate file that can be called.

- The following YAML file contains a sample set of variables that can be called inside the main playbook file:

```
Person:
  Name: John Doe
  Job: System Administrator
  Age: 30
```

- If you want to call this variable in a playbook, you can do that using `{{ Person.* }}` where * may be Name, Job, or Age. Deeper nesting levels can be used according to your needs.

Installing PHP 7 on the web server

- We've already covered the concept of Playbooks in Ansible. Those YAML files that describe the intended state in which the target servers should be.
- We've also covered some of the most useful Ansible modules in the last section. Let's use this knowledge to install PHP 7 on the web server.
- Create a new file lamp.yml and add the following:

```
---
- hosts: all
  become: yes
  tasks:
      - name: Install PHP 7 most common packages
        yum: name={{ item }} state=present
        with_items:
            - mod_php71w
            - php71w-cli
            - php71w-common
            - php71w-gd
            - php71w-mbstring
            - php71w-mcrypt
            - php71w-mysqlnd
            - php71w-xml
```

- Perhaps the only new thing here is the items array. This allows you to execute the same module on several items by substituting `{{ item }}`, which is a variable, with each of the items in the `with_items` array.

Deploying CodeIgniter on the web server

- Download the package from https://codeload.github.com/bcit-ci/CodeIgniter/zip/3.1.4
- Unzip the file on the client machine.
- Once finished, we need to transfer all the files to `/var/www/html/`. To do that we need to use the `synchronize` module, which has at least the `src` and `dest` parameters required (same as copy but more efficient in dealing with large amounts of files).
- However, and since `synchronize` depends on the `rsync` package to be installed on the remote system, we need to ensure that this package is installed first.
- Finally, we need to enable `mod_rewrite` module in Apache. This is the module that enables Apache to serve pretty URLs (e.g. www.example.com/users/jdoe instead of www.example.com/users.php?id=jdoe)

The lamp.yml file

- Now your lamp.yml file should look as follows:

```
- hosts: web
  become: yes
  tasks:
    - name: Install PHP 7 most common packages
      yum: name={{ item }} state=present
      with_items:
        - mod_php71w
        - php71w-cli
        - php71w-common
        - php71w-gd
        - php71w-mbstring
        - php71w-mcrypt
        - php71w-mysqlnd
        - php71w-xml
    - name: Install Rsync
      yum: name=rsync state=present
    - name: Deploy and configure CodeIgniter
      synchronize:
          src: /vagrant/CodeIgniter-3.1.4/
          dest: /var/www/html/
```

Task order

- Sometimes you may need to run a specific set of tasks before the main playbook tasks get executed. Or may be you want to run them only when everything else has finished running.
- Let's say that we want another user own the application directory, `cidev` that is a member in the developers group. We need to create this use before anything else runs. We also need to give this user and group ownership of the application directory tree.
- Ansible has `pre_tasks` and `post_tasks` parameters to let you run tasks before and after the main playbook tasks respectively.
- The text for this example could not fit in the presentation. It can be found in `lamp.yml` file that accompanies this section.
- Run the playbook now and observe the order of notifications. Although the pre and post tasks are placed at the end of the playbook, pre_tasks got run first and post_tasks got run last.

Configure mod_rewrite

- In Centos 7, the `mod_rewrite` module is enabled by default. But we need to ensure that it is always enabled.
- To enable mod_rewrite, the following line should exist in `/etc/httpd/conf.modules.d/00-base.conf`: `LoadModule rewrite_module modules/mod_rewrite.so`
- To do that, we can take a copy of this file, make the appropriate changes, and upload it (using the copy module for example) to its remote location. But Ansible has an easier way for this: the `lineinfile` module.
- This module is used to add/delete/change a line in a file. The required line is identified using regular expressions.
- In our case, we can the following to the lamp.yml file to ensure that `mod_rewrite` is always enabled:

```
- name: Ensure that mod_rewrite is enabled
  lineinfile:
    path: /etc/httpd/conf.modules.d/00-base.conf
    regexp: '^.*rewrite_module.*$'
    line: 'LoadModule rewrite_module modules/mod_rewrite.so'
    state: present
```

- Run anisble-playbook again. There should be no changes to the file as the module is already enabled. If you need to test the efficiency of lineinfile, login to webserver and delete the line. Run ansible-playbook now and observe the results. Try to comment out the line (by adding a pound sign to the start of the line). Run ansible-playbook and observe the results.

Notify Apache of the changes

- Any changes made in any configuration file of Apache need to be followed by reloading the `httpd` (apache2 in Ubuntu).
- Ansible allows you to *notify* different parts of the playbook that they need to be triggered. Those are called handlers.
- Handlers are just tasks that don't get triggered automatically. Only when notified. Let's create a handler for Apache restart. Add the following `lamp.yml`:

```
  handlers:
    - name: restart Apache
      service: name=httpd state=restarted
```

- Now we can instruct the `lineinfile` module to restart Apache if (and only if) the file was changed. Add the following to the `module:`

```
      notify:
        - restart Apache
```

- Login to webserer and delete the line that loads the `mod_rewrite`. Run `ansible-playbook` and observe the results.

Database configuration revisited

- In the previous section we used ansible without playbooks to deploy and configure Mariadb. However, the database values (like users, passwords…etc.) were hardcoded. This makes an inflexible deployment. Let's use a playbook with variables to make those values changeable.

- First, we need to create a variables files. Let's call it `db_vars.yml` and add the following to it:

```
root_password: mypassword
database:
    host: 192.168.33.30
    name: appdb
    username: codeigniter
    password: cipassword
```

- Now we can use the values of this file inside our playbook. The following is the db section of lamp.yml:

```
- hosts: db
  vars_files:
    - db_vars.yml
  become: yes
  tasks:
    - name: Ensure that Maridb is installed
      yum:
        name: '{{ item }}'
        state: present
      with_items:
        - MySQL-python
        - mariadb-server
    - name: Create application database
      mysql_db:
        name: '{{ database.name }}'
        state: present
        login_user: root
        login_password: '{{ root_password }}'
    - name: Create application user
      mysql_user:
        name: '{{ database.username }}'
        password: '{{ database.password }}'
        host: '%'
        priv: '{{ database.name }}.*:ALL'
        login_user: root
        login_password: '{{ root_password }}'
        state: present
```

Create a sample table

- Now we need to create a table for CodeIgniter to use, and also add some dummy records. Ansible allows you to execute SQL commands directly on the database from script files.
- I have already created a create.sql script for creating a sample table called news. You can find this file in the project files the comes with this section.
- We need to upload this file to the database server and let Ansible run the commands inside it for us against the database. Add the following to the playbook:

```
    - name: Upload the table creation script
      copy:
        src: create.sql
        dest: /tmp/
    - name: Create an application table
      mysql_db:
        state: import
        name: '{{ database.name }}'
        target: /tmp/create.sql
        login_user: root
        login_password: '{{ root_password }}'
```

Configuring CodeIgniter db connection

- For CodeIgniter to communicate with a database, it needs to determine the necessary configuration parapets: the host, the user, and the password.
- Those values are stored inside a file called `database.php` that exists under `application/config/`
- Since we already have those values saved in a vars file, we can insert them inside the file directly and let Ansible substitute the correct values at runtime by using Jinja2 templates.
- Create a new directory on the client machine to hold your templates:
  ```
  mkdir -p templates/application/config/
  ```
- Create a `database.php.jn2` file inside the config directory and modify it to be as follows (the changed part):
  ```
  'hostname' => '{{ database.host }}',
     'username' => '{{ database.username }}',
     'password' => '{{ database.password }}',
     'database' => '{{ database.name }}',
  ```
- Now we need to instruct Ansible to transfer this template file, with the correct values to the appropriate location on webserver. Add the vars_files section to the web group of the playbook, and add the following:
  ```
      - name: Configure the application with the database settings
        template:
          src: templates/application/config/database.php.jn2
          dest: /var/www/html/application/config/database.php
  ```
- You can login to webserver and ensure that the file has the correct settings. Now we are ready to deploy a sample application on CI instead of the welcome page.

Deploying a sample PHP application on CI

- So far if you browse to the IP address of the web server you will find the default welcome page of CodeIgniter. This is fine as a proof of successful installation, but we need to prove that database connection is also operational.
- To do that, we will deploy a very minimal PHP application that contains only one module, one controller, and one view file. The files are stored on the client machine as follows:
  - `files/application/modules/News_module.php`
  - `files/application/controllers/News.php`
  - `files/application/views/news_article.php`

- We need to upload those files to their appropriate locations on webserver. We can write several copy modules for this. But we can use Ansible's loop to speed things up. Add the following to the web part of the lamp.yml playbook:

```
    - name: Deploy sample CodeIgniter files
      copy:
       src: '{{ item.src }}'
       dest: '{{ item.dest }}'
    with_items:
     - src: files/models/News_model.php
       dest: /var/www/html/application/models/
     - src: files/controllers/News.php
       dest: /var/www/html/application/controllers/
     - src: files/views/news_article.php
       dest: /var/www/html/application/views/
```

- Notice the use of with_items to make the copy module loop through all the src and destination paths. Each object in the `with_items` array can be accessed as part of the `item` parent object.