Lab: Webapp Playbook

Overview

In this lab, we will write an Ansible playbook to deploy a prewritten Guestbook application. The guestbook application is an HTTP application with two endpoints:
* GET /signatures
* POST /signatures

Users can send a request to store a message and a request to read all of the messages. If a database is not supplied, the Guestbook app will only keep signatures in memory, meaning that all signatures are lost if the application is restarted.

We will begin by deploying the application and then we will deploy and connect the app to a database to maintain our state. All of this through a playbook.

Lab: Webapp Playbook

Instructions

Read this lab like a book, all text is there for a reason!

"→" denotes an action you must take

Use your favorite editor to edit files within the console. I suggest VI, nano, or emacs.

```
White boxes with black text denote commands and file contents
```

```
Black boxes with green text denote example output
```

Task 0: Download the guestbook application

→

```
cd
git clone https://github.com/ameade/flask-guestbook
cd flask-guestbook
```

Task 1: Create a basic playbook

The repository we just cloned contains:

README.md - Instructions on how to install and run the application
playbook.yml - A partial playbook for installing the application
app.py - the application itself

Step 1: Test the playbook

The playbook provided will only ensure the application is not running. Lets test it with the inventory we created in a previous lab.

→

```
ansible-playbook -i ~/hosts playbook.yml -b
```

```
PLAY [Setup guestbook application]
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************************
*********
ok: [35.239.103.48]
ok: [34.66.187.223]
```

```
TASK [Check if application is running]
*********************************************************************************
ok: [35.239.103.48]
ok: [34.66.187.223]

TASK [Stop the application]
*********************************************************************************************
****
skipping: [35.239.103.48]
skipping: [34.66.187.223]

PLAY RECAP
*********************************************************************************************
**********************
34.66.187.223              : ok=2    changed=0    unreachable=0    failed=0    skipped=1
rescued=0    ignored=0
35.239.103.48              : ok=2    changed=0    unreachable=0    failed=0    skipped=1
rescued=0    ignored=0
```

Step 2: Update the playbook

→ Fill in the missing sections. The appropriate Ansible modules have been selected for you, refer to their documentation for allowed parameters. Use README.md to know what needs to be done. TEST YOUR PLAYBOOK AS YOU GO

APT module - https://docs.ansible.com/ansible/latest/modules/apt_module.html
pip module - https://docs.ansible.com/ansible/latest/modules/pip_module.html
git module - https://docs.ansible.com/ansible/latest/modules/git_module.html
shell module - https://docs.ansible.com/ansible/latest/modules/shell_module.html

Step 3: Test your changes

You will know you are done when you can confirm the application is working properly on **both** of your hosts.

→ This command will add a message to your guestbook, make sure you replace <IP>

```
curl -X POST http://<IP>:8080/signatures?message="hello"
```

→ This command will read all messages that have been added

```
curl http://<IP>:8080/signatures
```

Task 1: Add mysql to our playbook

Step 0: Explore persistence

You may notice that everytime you rerun your playbook, the application will lose all of it's messages. You may also notice that the application on each host has its own set of messages. This is because each application is storing its messages in its own memory, so restarting the application clears its list!

→ Write lots of messages to both hosts, execute the following for loop. Replace <IP_1> and <IP_2>

```
for i in {1..10}; do
 curl -X POST http://<IP_1>:8080/signatures?message="host 1 message "
 curl -X POST http://<IP_2>:8080/signatures?message="host 2 message "
done
```

→ Witness each host having its own messages

```
curl http://<IP_1>:8080/signatures
curl http://<IP_2>:8080/signatures
```

```
$ curl http://34.239.103.48:8080/signatures
host 1 message
host 1 message
host 1 message
host 1 message
host 1 message
host 1 message
host 1 message
host 1 message
host 1 message
host 1 message

$ curl http://35.239.103.48:8080/signatures
host 2 message
host 2 message
host 2 message
host 2 message
host 2 message
host 2 message
host 2 message
host 2 message
host 2 message
host 2 message
```

→ Rerun playbooks

```
ansible-playbook -i ~/hosts playbook.yml
```

→ See the messages are gone

```
curl http://<IP_1>:8080/signatures
curl http://<IP_2>:8080/signatures
```

Step 1: Setup a persistent database

→ Add the following play before the "Setup guestbook application" in your playbook.yml

```
- name: Setup mysql
  hosts: db
  tasks:
    # - name: Install dependencies
    #   apt:
    # - name: Allow all hosts to connect
    #   shell:
    # - name: Start the mysql
    #   service:
    # - name: Create database
    #   mysql_db:
    # - name: Create database users
    #   mysql_user:
```

This play will only run against hosts in the "db" group in your inventory. There should only be a single host in this group.

→ Fill in the missing sections. The appropriate Ansible modules have been selected for you, refer to their documentation for allowed parameters. Use README.md to know what needs to be done.

APT module - https://docs.ansible.com/ansible/latest/modules/apt_module.html
Shell module - https://docs.ansible.com/ansible/latest/modules/shell_module.html
Service module - https://docs.ansible.com/ansible/latest/modules/service_module.html
Mysql_db module - https://docs.ansible.com/ansible/latest/modules/mysql_db_module.html
Mysql_user module - https://docs.ansible.com/ansible/latest/modules/mysql_user_module.html

Don't forget you can check your playbook syntax before running it

```
ansible-playbook -i ~/hosts playbook.yml --syntax-check
```

```
playbook: playbook.yml
```


Step 2: Test your database is working

→ Install a mysql client

sudo yum install mysql

→ Connect to mysql, type the database password when prompted, Passw0rd

```
mysql -h <IP> -u db_user -p
```

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 10.1.38-MariaDB-0+deb9u1 Debian 9.8

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

→ press **ctrl-d** to exit

Task 2: Connect our app to our database

Lucky for us, the guestbook app knows how to store it's messages in a mysql database if configured. The application looks for a configuration file at '/opt/guestbook/vars.ini' If the file exists, the app will use the values provided by the file to connect to a database.

We simply need to have our playbook populate this file and put it in '/opt/guestbook/vars.ini' on our web servers.

Step 1: Create config file

→ Add the following tasks to your "Setup guestbook application" play, before the "Start the application" task

```
    - name: Create guestbook config directory
```



```
    file:
      path: "/opt/guestbook"
      state: directory
  - name: Configure application
    template:
      #TODO
```

We will be using the *template* module to pass variables into our configuration file and copy it to the correct location.
Template module - https://docs.ansible.com/ansible/latest/modules/template_module.html

→ Update the task to use vars.ini.j2 as the source template and '/opt/guestbook/vars.ini' as the destination

→ The vars.ini.j2 template expects 4 variables, add the following under 'hosts: web' in your playbook. Replace <DB_IP> with the IP of your db host

```
vars:
   db_user: "db_user"
   db_pass: "Passw0rd"
   db_name: "guestbook"
   db_host: "<DB_IP>"
```

→ **BONUS** if you define these variables in a vars file and include them in both plays, using the variables instead of hardcoded values

Step 2: Test shared data

→ Write a message to host 1

```
curl -X POST http://<IP_1>:8080/signatures?message="host 1 message "
```

→ Read the message from host 2

```
curl http://<IP_2>:8080/signatures
```

```
host 1 message
```


