ENSF 400 - Lab 5 - Ansible

Feb 27, 2024

This lab will help us understand how Ansible works with managed hosts and deploys software onto them. Code repository: `https://github.com/denoslab/ensf400-lab5-ansible`

1 GitHub Codespaces

Create an instance using GitHub Codespaces. Choose repository `denoslab/ensf400-lab5-ansible`. Install Ansible.

```
$ pipx install ansible-core
$ ansible-galaxy collection install ansible.posix
$ ansible-galaxy collection install community.general
```

2 Create SSH Secrets

Before creating managed hosts, we need to prepare an SSH key pair for authentication purposes by Ansible.

```
$ mkdir secrets
$ ssh-keygen -t rsa -N "" -C "root@0.0.0.0" -f secrets/id_rsa
$ chmod 400 secrets/id_rsa
$ cp secrets/id_rsa.pub secrets/id_rsa_container.pub
```

After this step, you will have an SSH key pair with one additional file (public key) for mounting purposes.

- `id_rsa`: This is the private key Ansible Controller will hold as the credential to access the managed hosts.
- `id_rsa.pub`: This is the public key that will be distributed to the managed hosts to authorize the Ansible Controller to login.
- `id_rsa_container.pub`: This is the public key copy to be mounted to the containers.

3 Define Managed Hosts

This step will create three slightly modified Alpine Linux containers with SSH access. Note that after these containers start, there is no additional applications installed. For the Web application to be deployed, we will use Ansible to set it up.

Now, we would like to create three hosts as containers to be managed by Ansible. Under the root directory of the repo, create `docker-compose.yml` with the following content.

```yaml
version: "3.7"

services:
  managedhost-app-1:
    build:
      context: managed-host/alpine/app1
    restart: unless-stopped
    network_mode: "bridge"
    environment:
      - SSH_ENABLE_ROOT=true
  managedhost-app-2:
    build:
      context: managed-host/alpine/app2
    restart: unless-stopped
    network_mode: "bridge"
    environment:
      - SSH_ENABLE_ROOT=true
  managedhost-app-3:
    build:
      context: managed-host/alpine/app3
    restart: unless-stopped
    network_mode: "bridge"
    environment:
      - SSH_ENABLE_ROOT=true
```

Edit `docker-compose.yml` to complete the following tasks to allow these three instances to be managed by Ansible:

1. Mount the public key from the host `secrets/id_rsa_container.pub` to the container at the path `/root/.ssh/authorized_keys`. This will authorize the Ansible controller to use its private SSH key to access the managed instances.

2. For `managedhost-app-1`, map Container port 3000 to Host port 3000 (Web). Map Container port 2223 to Host port 2223 (SSH).

3. For `managedhost-app-2`, map Container port 3000 to Host port 3001 (Web). Map Container port 2224 to Host port 2224 (SSH).

4. For `managedhost-app-3`, map Container port 3000 to Host port 3002 (Web). Map Container port 2225 to Host port 2225 (SSH).

4 Start Managed Hosts

Under the root directory of the repo, run the following command to start the managed host containers.

```
$ docker compose up -d
```

Use `docker compose ps` to check if the containers are up and running as expected.

```
$ docker compose ps
NAME                                        IMAGE
↪  COMMAND                  SERVICE             CREATED             STATUS
↪  PORTS
ensf400-ansible-lab-managedhost-app-1-1   ensf400-ansible-lab-managedhost-app-1
↪  "/entry.sh /usr/sbin..."   managedhost-app-1   7 seconds ago       Up 6
↪  seconds        0.0.0.0:2223->2223/tcp, 0.0.0.0:3000->3000/tcp
ensf400-ansible-lab-managedhost-app-2-1   ensf400-ansible-lab-managedhost-app-2
↪  "/entry.sh /usr/sbin..."   managedhost-app-2   7 seconds ago       Up 6
↪  seconds        0.0.0.0:2224->2224/tcp, 0.0.0.0:3001->3000/tcp
ensf400-ansible-lab-managedhost-app-3-1   ensf400-ansible-lab-managedhost-app-3
↪  "/entry.sh /usr/sbin..."   managedhost-app-3   7 seconds ago       Up 6
↪  seconds        0.0.0.0:2225->2225/tcp, 0.0.0.0:3002->3000/tcp
```

5 Create Inventory

Under the root directory of the repo, create an inventory file called `hosts.yml` using the following template. Note that since we are using a `bridged` network, the IP address of all managed hosts should be `0.0.0.0`. The following properties shall be set for each managed host:

- There are 3 hosts to manage: `managedhost-app-1`, `managedhost-app-2`, and `managedhost-app-3`.
- `ansible_host` is `0.0.0.0`
- `ansible_connection` is using `ssh`
- `ansible_port` should be consistent with the configuration in `docker-compose.yml`.
- `ansible_user` is `root`.
- The host group `app_group` shall include all 3 managed hosts.

```yaml
ungrouped:
  hosts:
    # TODO
app_group:
  hosts:
    # TODO
```

6 Create Ansible Config File

Under the root directory of the repo, create a file named `ansible.cfg` using the following template. This file will configure Ansible. Complete the following tasks:

- Set the `inventory` property to the inventory file you created in Section 5.

- Set the `private_key_file` property to the private RSA key file created in Section 2.

```
[defaults]

inventory = TODO
private_key_file = TODO
host_key_checking = False
action_warnings = False
```

7 Test Ansible Connectivity

Use the following command to check if all hosts are added to the inventory. Note that we will include `localhost` as we take it as the Ansible Controller.

```
$ export ANSIBLE_CONFIG=$(pwd)/ansible.cfg
$ ansible all:localhost --list-hosts
  hosts (4):
    managedhost-app-1
    managedhost-app-2
    managedhost-app-3
    localhost
```

After verifying that all hosts have been added, ping the hosts to see if they can be successfully connected.

```
$ ansible all:localhost -m ping
localhost | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
managedhost-app-3 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
managedhost-app-1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
managedhost-app-2 | SUCCESS => {
    "ansible_facts": {
```

```
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

8 Playbook Sample

The playbook sample is named `hello.yml`. It consists of 2 plays:

- First play: target all managed hosts to create a simple file.
- Second play: targets the alpine node to run NodeJS app (running inside the container on port 3000, running externally on port 3000-3002 - check docker-compose).

```
# run the playbook sample:
$ ansible-playbook hello.yml
```

After that, check if the playbook runs successfully. Use port forwarding to expose Ports 3000, 3001, and 3002 of the GitHub Codespaces instance. GitHub Codespaces will automatically create the port forwarding. In the "PORTS" tab, use the "Forwarded Address" URLs to access Ports 3000, 3001, and 3002 to verify the services.

9 Have your work checked by a TA

The TA will check the completion of the following tasks:

- SSH key pair generation.
- Completion of `docker-compose.yml`.
- Completion of the Ansible inventory file.
- Completion of `ansible.cfg`.
- Ansible connectivity to all hosts by `ping`.
- Successful deployment of the NodeJS app on all three managed hosts by accessing the web pages.

10 Cleanups

```
$ docker compose down
```