Lab 8

Delegation

During this lab you will get to play with the following ansible concepts:

- Delegating tasks (in our case to the control machine)
- Running tasks only once
- Dynamically adding hosts
- Using a different connection type (in our case: docker)
- Apply and use tags

You also get to play al lot with the openssl modules.

This lab does not really add any value to the application we are building. You will make the docker daemon running on the rest-servers accessible over TCP with TLS. You will then use this to add the docker container to the inventory and finally run a task inside the docker containers. To make it an observable task, you will check the create a file in the container.

Open the `~/course/ansible/delegate` directory and please find the following files:

- `ansible.cfg` to set the host file location, ssh username and we changed the stdout callback to yaml
- `main.yaml` the main playbook file, which includes the following tasks files:
  - `ca.yaml` to generate a CA Certificate
  - `server.yaml` to generate a server certificate for each host in the rest-servers group
  - `ca.yaml` to generate a CA Certificate for the control machine

Open the `ca.yaml` and notice we have supplied you with a list of variable for the different security artefacts that you'll create

Open the `ca.yaml` . Eventually this file needs tasks to:

- Generate a CA key
- Generate the CA CSR
- Generate the CA Certificate

We will guide you through these steps below.

Generate a CA key

Let's start with the first task to generate the CA's key

- This needs to run on the control machine `local_action`
- You will need to use the openssl_privatekey (https://docs.ansible.com/ansible/latest/modules/openssl_privatekey_module.html) module `module: openssl_privatekey`
- For the name of the TLS/SSL private key file use the `ca_key` variable
- Leave the algorithm and size default (which is RSA 4096)
- Setup a passphrase "ansible" `passphrase` and use the *aes256* to encrypt the private key `cipher`
- You also have to make sure:
  - The `main.yaml` has a global `become` , make sure this local task does not use privilege escalation `become: false`
  - This should not be run for each host, but only once `run_once: true`

You might want to test your playbook already. However you will need to install the pyOpenSSL (https://pypi.org

/project/pyOpenSSL/) python package first

```
$ pip intsall pyOpenSSL --user
```

Then run your playbook. After a successful run you should see a file named `ssl/ca-key.pem` .

Generate the CA CSR

Now let's create the `ssl/ssl/ca.csr` (using the `ca_csr` variable)

- Again this is a local action, should only be run once and as the current user
- The module to use is openssl_csr (https://docs.ansible.com/ansible/latest/modules/openssl_csr_module.html)
- Set the path using the `ca_csr` variable
- The `commonName` field `common_name` so be set to the ansible host name `inventory_hostname`
- Make sure to set the subject is a CA using Basic Constraints (https://tools.ietf.org/html/rfc5280#section-4.2.1.9) `basic_constraints` . With openssl his is set as `CA:true`
- point to the private key you have generated above `privatekey_path` (also make sure to pass the passphrase `privatekey_passphrase`)

Try to run your your playbook again. If at anytime you want to run all tasks, then rm the ssl directory's content:

```
$ rm ssl/*
```

Generate the CA Certificate

The last step in this task is to create the CA's certificate.

- This is also a local action, should only be run once and as the current user
- Use the openssl_certificate (https://docs.ansible.com/ansible/latest/modules/openssl_certificate_module.html) module
- The variable for the file name is `ca_cert`
- Specify the csr file `csr_path`
- Specify the the private key file with its passphrase
- For the provider use `selfsigned provider`

Feel free to test your playbook so far.

Now that we have a CA key pair we can use it to sign certificates for the server and the client. Let's start with the server

Open the `server.yaml` . You will go through the same steps, plus copy the files to the hosts. Create the key, csr and certifcate files on the control machine:

- Generate a unencrypted key file named using the `local_server_key` variable. `openssl_privatekey` with only `path`
- Generate the CSR. Two differences between this and the CA's are:
  - don't specify the `basic_constraints`
  - make sure to set a SAN (Subject_Alternative_Name) (https://en.wikipedia.org/wiki/Subject_Alternative_Name) `subject_alt_name` to the IP of the server (which is in our case also available with `inventory_hostname`). This is set as `IP:…` `subject_alt_name: IP:`
- Generate the Server's certificae, oly this time make sure you use `ownca` as the provider and specify the CA's certificate path, its private key and the passphrase `ownca_xxx`

As a final step copy the files to the host using the following mapping (hint use a loop):

Source	Destination

Source	Destination
`local_server_key`	`server_key`
`local_server_cert`	`server_cert`
`ca_cert`	`server_ca_cert`

Try and run your playbook. Remember you can remove the `ssl/` directory's content to force a rerun of tasks.

You should be able to use the previous task to create the following files on the control machine

- The client's private key file using the `client_key` variable
- The client's CSR file using the `client_csr` variable
- The Client's certificate using the `client_cert` variable

Test your playbook so far

Before you continue let's tag the tasks you have so far. We want to add two tags to all the SSL tasks: `ssl` and `config` . You could of course add these tags to each individual task, but is there an easier way? add to the `import_tasks`

By default the docker damon is available on a local non-networked Unix socket. We will now allow it to communicate over a HTTP socket together with TLS.

Our hosts are running centOS, which uses systemd. Therefore we will need to change the contents of the `/lib/systemd/system/docker.service` file and replace the ini key `ExecStart` to include an additional socket and tls configuration. More information about what you are about to configure is available at doc.docker.com (https://docs.docker.com/engine/security/https/).

The `/lib/systemd/system/docker.service` is an *ini*-file:

- Use the correct module to work with these kind of files `ini_file`
- The `ExecStart` option (under the `Service` section) should get the following value where we use our generate server certificate, its key and the CA's certificate to establish trust ot client certificates signed with the same CA

```
/usr/bin/dockerd -H tcp://0.0.0.0:4243 -H unix:///var/run/docker.sock --t
lsverify --tlscert  {{server_cert}} --tlscacert {{server_ca_cert}} --tlsk
ey {{server_key}}
```

- Hint: just to ensure you are not writing in the wrong file, make sure this task does not create the file if absent `cretae: false`
- Tag this task `config`

You will then need to reload and restart the docker service `systemd` module (tag this task also with `config`)

Check your playbook. This is a important step as the docker daemon will not start in case you made errors with the certificates. In case of errors you might want to clean the `ssl` directory. You might have to ssh into a machine and get the systemd log:

You might want to use your tags. Run all tasks tagged the `config` *except* those tagged 'ssl'

```
$ ansible-playbook main.yaml --tags config --skip-tags ssl
```

```
$  ssh vagrant@10.20.1.5 sudo journalctl -xe
```

Some other known issues student often have are:

- For the CA CSR: Forgetting to set the `basic_constraints` to indicate the subject is a CA `basic_constraints: "CA:true"`
- Specifying a passphrase on the server's key (we only use a passphrase on the CA's key)
- forgetting the SAN for the Server's CSR `IP:{{inventory_hostname}}

Once you've got it working we can start adding the docker containers to our inventory in the next step

Use the `main.yaml` for these steps

Now that the containers are accessible from the control machine we can add then to the inventory and directly interect with them. However before we can do that we need to make our containers have the required software installed to act as an ansible host. Do you remember what requirements there were python 2.7 or 3 (don't worry about ssh as we won't be using that as the connection type). Obviously we can't we use ansible with its apk module (https://docs.ansible.com/ansible/2.7/modules/apk_module.html). so we need to just run a `docker exec` (this allows us to run commands inside the docker container)

Install software into the container

Use the correct module to run a generic command `command` . Use it to run the following command:

```
$ docker exec -it rest-countries apk add --update python
```

Make sure to tag this task `config`

The containers are running alpine linux (https://alpinelinux.org/) therefore we are using its package manager apk (https://wiki.alpinelinux.org/wiki/Alpine_Linux_package_management)

You would normally already make sure your container's image would install python for containers which need to be accessed from ansible plays.

Add the container to the inventory

Do you remember which module to use in order to dynamically add a host to the inventory? `add_host` . As you know This module runs outside the host loop and would only run once. We therefore need to loop over the hosts in the "rest-servers" group:

- Define your task and make sure it runs over all the hosts in the "rest-servers" group `loop: "{{groups['rest-servers']}}"`
- The `name` of the host must be unique so use the loop variable to make a unique name eg., `rest-countries-`
- Now make sure to set the real name of the container using `ansible_host`
- Add the host to a group (eg., `rest-countries-containers`) `group: …`

We won't be using ssh to connect with this host, but will be using the docker connection plugin (https://docs.ansible.com/ansible/2.5/plugins/connection/docker.html)

- Make sure the docker connection plugin is used when interacting with this host `ansible_connection: docker`
- tasks won't be execute using the vagrant, but root instead `ansible_user: root`
- You'll need to add additional connection properties so that we communicate with the correct daemon and that our client certificate is used when connection `ansible_docker_extra_args` :

  ```
  --tlsverify --tlscacert={{ca_cert}} --tlscert={{client_cert}} --tlskey={{
  client_key}} -H=tcp://{{inventory_hostname}}:4243
  ```

Do not tag this task

Now we are going to run a task inside the containers. Let's write an empty file name `/HELLO_FRIEND` to each container:

- Use the correct module `file`
- Create the empty file `state: touch`

- loop over each host in the group of containers you've created (e.g, named `rest-countries-containers`)
- Because you are doing the looping, run this task only once
- don't tag this task

Run your playbook. Use your tags. How could you run only tasks without any tags?

```
$ ansible-playbook main.yaml --tags untagged
```

If you want to assure yourself these files have been written use the following ssh/docker command

```
$ ssh vagrant@10.20.1.4 "sudo docker exec rest-countries ls /"
```

Replace the ip to 10.20.1.5 to check on the second host in our group.