thisdougb / **ansible-hashivault**

Ansible roles to build a HashiCorp Vault instance

46 commits | 1 branch | 0 releases | 1 contributor

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

thisdougb Update README.md — Latest commit 9afa658 on Jul 7, 2017

roles	removed hashivault-ldap role	a year ago
vagrant	added auto pickup of fqdn for the PKI backend	a year ago
README.md	Update README.md	a year ago
ansible.cfg	typo	a year ago
bootstrap_hosts.yml	typo	a year ago
hosts	added ansible bootstrap	a year ago
vault.yml	added auto pickup of fqdn for the PKI backend	a year ago

README.md

ansible-hashivault

Vagrant and Ansible code to build a HashiCorp Vault instance for testing. Built for Centos/RHEL.

Demo screencast: https://vimeo.com/223884742

Create Vagrant Instance

Create a Vagrant instance to run the vault, see the sub-dir *vagrant* from this repo.

Ansible Bootstrap

Bootstrap the new vault instance, installing the ssh key (should match private_key_file from ansible.cfg). We also install the hosts file, picked up from /vagrant/etc_hosts on the new vault instance. When not using Vagrant you'll need to workaround this and populate the /etc/hosts file.

```
ansible $ ansible-playbook bootstrap_hosts.yml -k -K
SSH password:
SUDO password[defaults to SSH password]:

PLAY [all] *********************************************************************

TASK [install Ansible ssh key] *************************************************
changed: [vault]

TASK [install /etc/hosts on remote] ********************************************
changed: [vault]

TASK [install net-tools (netstat)] *********************************************
changed: [vault]

PLAY RECAP *********************************************************************
vault                      : ok=3    changed=3    unreachable=0    failed=0
```

Install HashiCorp Vault

Now you can go ahead and install your vault instance.

```
ansible $ ansible-playbook vault.yml
```

Quick Status Check

Sudo to root on the vault instance and check that vault is up and running:

```
laptop $ vagrant ssh
[vagrant@vault ~]$ sudo su -
[root@vault ~]# systemctl status vault
● vault.service - Hashi Corp Vault
   Loaded: loaded (/etc/systemd/system/vault.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2017-06-23 16:00:22 BST; 10min ago
 Main PID: 6873 (vault)
   CGroup: /system.slice/vault.service
           └─6873 /usr/local/bin//vault server -config /etc/vault.cfg
```

Use the vault cli to check the status:

```
[root@vault ~]# vault status
Sealed: false
Key Shares: 2
Key Threshold: 2
Unseal Progress: 0
Version: 0.6.2
Cluster Name: vault-cluster-bbdec3aa
Cluster ID: 4ac16b0f-39d2-5e8b-6030-2621184ee023

High-Availability Enabled: false
```

We can also check what backends vault has currently mounted:

```
[root@vault ~]# vault mounts
Path        Type       Default TTL  Max TTL  Description
cubbyhole/  cubbyhole  n/a          n/a      per-token private secret storage
pki/        pki        system       system   Certificate delivery for Ansible
secret/     generic    system       system   generic secret storage
sys/        system     n/a          n/a      system endpoints used for control, policy and debugging
```

We listen locally on port 8200, and using TLS on port 8201 on external interfaces:

```
[root@vault ~]# netstat -atnp | grep vault
tcp        0      0 127.0.0.1:8200          0.0.0.0:*               LISTEN      6873/vault
tcp6       0      0 :::8201                 :::*                    LISTEN      6873/vault
```

Additional Notes

The init role drops the vault keys and root token into the home directory of the ansible host user. This allows other roles to use the keys to unseal and read/write to the vault. This is fine for testing, but you will want a manual init/unseal process in production.

```
ansible $ cat ~/.hashicorp_vault_keys.json
{"keys": ["4ca011fb3c4a1ba9c97738dfc9a6424c01d3136673c5eb6ad3c86a720b39959601",
"5af792ff6f420547f65b132a1b842634e2b8ccedaa00fb76cecc6856139bf04702"],
 "keys_base64": ["TKAR+zxKG6nJdzjfyaZCTAHTE2Zzxetq08hqcgs5lZYB", "WveS/29CBUf2WxMqG4QmNOK4zO2qAPt2zsxoVhOb8EcC"],
"root_token": "8ec58cb7-712e-3454-0f
89-cd6418cc97b1"}
```

The root token (super-user access) is also dropped into the home directory of the root user on the vault instance. This, for testing, gives you easy access to the vault for playing around and troubleshooting. The vault cli looks for this file, as well as the $VAULT_ADDR environment variable.

```
[root@vault ~]# cat /root/.vault-token
8ec58cb7-712e-3454-0f89-cd6418cc97b1

[root@vault ~]# echo $VAULT_ADDR
http://127.0.0.1:8200
```

The PKI role sets up Vault as a Root CA, so it can distribute certificates. In doing so the role drops vaults issuing_ca cert onto the Ansible host. This means calling https://vault:8201 will work fine.

```
ansible $ ls -l /etc/pki/ca-trust/source/anchors/vault.crt
-rw-r--r-- 1 root root 1167 Jun 23 16:00 /etc/pki/ca-trust/source/anchors/vault.crt
```

Checking the https connection from the ansible node:

```
ansible $ wget -qO- https://vault.example.com:8201/v1/sys/health
{"initialized":true,"sealed":false,"standby":false,"server_time_utc":1498389760,"version":"0.6.2","cluster_name":"vaul
cluster-0ea8d19c","cluster_id":"fc612a4e-b0a1-bfa7-f81d-b17c994ec12e"}
```